FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Screening tool to assess the risk of falling

Alcino João Silva de Sousa

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Luís Teixeira (FEUP)

Co-Supervisor: Inês Soares (Fraunhofer Portugal AICOS)

Supervisor at Fraunhofer Portugal AICOS: Joana Silva (Fraunhofer Portugal AICOS)

June 27, 2017

Screening tool to assess the risk of falling

Alcino João Silva de Sousa

Mestrado Integrado em Engenharia Informática e Computação

June 27, 2017

Abstract

Some of our capacities and functional mechanisms tend to become impaired with aging. What kind of impact can a fall have on someone's life? Falls have major consequences over time and may even lead to institutionalization or death. They carry high costs on treatment, not only for the one that falls, but also to the public medical systems. It is possible to know who has a greater risk for falling and then, through some intervention, avoid falls. Some examples of risk factors for falls are: muscle weakness, poor gait/balance, and many others, having multiple factorial sources and many measuring formats. A good combination and evaluation of risk factors is needed to assess the risk for falling and an easy to apply, not time consuming with good accuracy and precision solution that combines multiple factors is not yet in the ideal stage of development. This work was developed in partnership with Fraunhofer Portugal AICOS. The Association has developed a wireless device with inertial sensors that allows the extraction of movement metrics during the evaluation tests of the risk factors along with balance metrics extracted with the pressure platform developed by Sensing Future, partner of the FallSensing project. In the scope of the same project, in partnership with ESTeSC – Coimbra Health School, data relative to the risk factors was collected from various persons in various contexts and conditions. All the data collected was stored on a database, including the data collected from the sensors and the information on the occurrence of falls in the following six months after the initial collection of data. The objective of the present work is to determine which are the most relevant factors and metrics from the sensors and how they can be combined to distinguish between persons that will fall in the future from the ones who won't. The approach was based on the application of Machine Learning methods, namely supervised learning classification algorithms, trying to understand which of the algorithms and combination of methods would allow a better distinction between the group of people that had fallen in the follow-up period from the remaining, having as basis the comparison of their performance metrics.

Resumo

Algumas das nossas capacidades e mecanismos funcionais tendem a ficar debilitadas com o envelhecimento. Que tipo de impacto é que uma queda pode ter na vida de alguém? As quedas acarretam consequências graves e podem levar à institucionalização ou até mesmo à morte. Para além disso, têm custos elevados, não só para a pessoa que cai, mas também para o sistema de saúde nacional. É possível saber quem tem um maior risco de queda e, através de intervenções, evitá-las. Alguns exemplos dos fatores de risco que influenciam a ocorrência de quedas são: fraqueza muscular, presença de marcha/equilíbrio anormais, entre outros, tendo por isso uma origem multifatorial e sendo medidos em escalas variadas. Uma boa combinação e avaliação destes fatores de risco são necessárias para determinar o risco de queda, mas uma solução de aplicação fácil e rápida, com boa precisão, que permita quantificar o risco de queda ainda não existe num estado ideal. A presente dissertação foi desenvolvida em parceria com a Fraunhofer Portugal AICOS. A Associação desenvolveu um dispositivo sem fios com sensores inerciais que permite extrair métricas de movimento nos testes de avaliação do risco de queda em conjunto com métricas de equilíbrio extraídas da plataforma de pressões desenvolvida pela Sensing Future, parceira do projeto Fall Sensing. No âmbito do mesmo projeto e com a parceria da ESTeSC - Coimbra Health School foram recolhidos dados relativos aos fatores de risco de várias pessoas, de vários contextos e condições. Toda a informação recolhida foi concentrada numa base de dados, incluindo a informação extraída pelos sensores bem como a informação da ocorrência de quedas nos 6 meses seguintes à recolha dos dados. O objetivo desta dissertação é determinar quais são os fatores e métricas mais relevantes dos sensores, compreender como devem ser combinados de forma a permitir a distinção entre pessoas que provavelmente vão cair num futuro próximo daquelas que, provavelmente, não vão cair. A abordagem baseou-se na aplicação de métodos de Machine Learning, nomeadamente de classificação supervisionada, na tentativa de perceber quais os algoritmos e que combinação de métodos permitem melhor distinguir o grupo de pessoas que caiu nos 6 meses seguintes das restantes, tendo como base a comparação das suas métricas de desempenho.

Acknowledgements

To Luís Teixeira, my supervisor, who guided me in a field I didn't knew, for being a true mentor that cares for the achievements and personal growth of his students,

To Inês Soares, my co-supervisor, who welcomed me to this work with great enthusiasm and was always available to encourage me,

To Joana Silva, that accompanied me at Fraunhofer Portugal and never missed a smile when asked to help,

To all Fraunhofer Portugal members that received me with major respect and welcomed me as part of the team,

To Fraunhofer Portugal for giving me the opportunity of applying my knowledge in such a humanitarian field,

To Compete2020, Portugal2020 and Fundo Europeu de Desenvolvimento Regional for co-financing the FallSensing project that allowed the execution of this dissertation,

To G.A.S. Porto that taught me the way of living my life towards serving others,

To all persons that directly or indirectly contributed to this project,

To my family and friends, that accompanied me from the beginning, that gave me such joy and confidence that allowed the fulfillment of this dissertation,

To Filipa, my girlfriend, for the patience and the never-ending care,

Alcino Sousa

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

10MWT	10 Meter Walk Test
ACC	Accuracy
ADASYN	Adaptive Synthetic Sampling
ALLKNN	All-K-Nearest Neighbor
CDST	Cost-sensitive Decision Trees
CNN	Convolutional Neural Networks
COM	Center of mass
CV	Cross-validation
DT	Decision Tree
ENN	Edited Nearest Neighbor
FN	False Negatives
FP	False Positives
FRAT	Falls Risk Assessment Tool
KNN	K-Nearest Neighbor
LR	Logistic Regression
LWL	Locally weighted learning
MIF	Mobility Interaction Fall
NB	Naive Bayes
NCR	Neighborhood Cleaning Rule
NHANHES	National Health and Nutrition Examination Survey
NN	Neural Network
OSS	One Sided Selection
PCA	Principal Component Analysis
PPA	Physiological Profile Assessment
PRE	Precision
TUG	Timed Up and Go test
REC	Recall
RENN	Repeated Edited Nearest Neighbor
RF	Random Forests
RFE	Recursive Feature Elimination
RFECV	Recursive Feature Elimination with Cross-Validation
ROC AUC	Receiver operating characteristic area under the curve
Select Fdr	Select False Discovery Rate
Select Fpr	Select False Positive Rate
Select Fwe	Select Family-wise Error Rate

ABBREVIATIONS

SMOTE	Synthetic Minority Over-sampling Technique
STS	30 Seconds Sit-to-Stand
SVM	Support Vector Machine
TL	Tomek Links
TN	True Negatives
TP	True Positives

Chapter 1

Introduction

1.1 Context

Elderly people tend to fall more often than younger people. Why does that happen? The answer seems trivial as some of our capacities and body functions tend to become impaired with aging. The world population is getting older... Can we give these persons' good life conditions? What kind of impact can a fall have on someone's life?

To start, falls are a big problem for elderly people. More than just having consequences on the moment of fall, they can have severe impact on the elderly future life. An injury, besides all the consequences that it naturally has, can send them to the hospital, reducing their activity for a long amount of time. This amount of time, in an elderly person, can have serious damage like an even more accentuated decline on their mobility. Not to mention the actual severity of the injuries on the moment of fall. It also increases their fear of falling. The elderly family is affected since they will, probably, have to be more careful with the elder. It leads to a lot more problems than it seems... It may lead to institutionalization and even death. More to this, it has lots of costs on treatment, not only for the person that falls, but also to the medical system. [APH13]

It is possible to know who are the persons that have a greater risk of falling and then, through some intervention, avoid falls. For that, to know who is more likely to fall, it is necessary to identify the factors that increase their risk of falling. These factors can be body related (intrinsic), environment related (extrinsic) or related to some activities (behavior). Some of the most important are:

- History of prior falls;
- Muscle weakness;
- Poor gait or balance;
- Visual impairment
- Arthritis

- Functional limitation
- Depression
- Use of psychotropic medications

These tend to be some of the most important factors that affected the risk of falling as some studies suggest. [APH13] [SVSC07]

There are some ways to quantify the risk factors. Using tests, for example:

- Through the **Timed Up and Go test (TUG)**: This test evaluates the time someone needs to get up from a chair, walk three meters, turn around and sit again. Through the measured time it is possible to infer about how good and normal is the mobility of the person. [WLC05]

This is a really simple test. There are many others, some are just questionnaires and interviews and other require the development of some physical activity. [APH13]

To get a real notion on the risk that one has of falling it is needed a combination of the most relevant factors, since the presence of multiple risk factors increases the risk of falling exponentially.

Some tools that evaluate multiple risk factors at a time in order to evaluate the real risk of falling have been created. There is no single tool that is recommended for use in all settings and for all populations. [SVSC07]

Some recent studies show that the incorporation of sensors in the execution of the physical tests performed by the persons can improve the quality of the results obtained. Also, recent studies started to analyze the impact that Machine Learning algorithms can have in the assessment of the risk for falling although this field needs richer studying. This is further analyzed in 2.

Fraunhofer Portugal AICOS, a research center, is working on a project called FallSensing[1]. This project is trying to enable the evaluation of multiple fall risk factors and the execution of fall prevention exercise plans. They collected, in partnership with ESTeSC - Coimbra Health School, information on risk factors from various persons (400) in various contexts. Some of the conducted tests to collect information included the use of a wearable sensor that has been developed at Fraunhofer that allowed the extraction of movement metrics and a pressure platform for balance metrics. They are also following-up the persons that participated in the initial collection of data, to understand which are the participants that have fallen after the initial information collection. This created a great opportunity to further study the appliance of Machine Learning in screening tools to assess the risk of falling.

1.2 Motivation and goals

As explained, this is a problematic question that is massively researched, as presented in 2, due to its importance. This work intended to improve this research field, analyzing the applicability

[1]http://www.fraunhofer.pt/en/fraunhofer_portugal/news/news_archive/fallsensing-_-technological-solution-for-fall-risk-screening-and.html

of different machine learning algorithms in a screening tool to assess the risk of falling understanding which could lead to better results. The algorithms were tested on the dataset collected by Fraunhofer and its partnerships, mentioned before, evaluated, perfected and compared with each other and with other existing tools through their performance metrics. The achievement of a good classification model could:

- Contribute to the development of a better assessment tool by Fraunhofer.
- Contribute to the development of a general better assessment tool that could help in the identification of people with high risk for falling.
- Contribute to the diminish of the percentage of population that suffer from this problem.
- Save money to the medical system.

The more specific objectives that can be defined for this dissertation are:

- Build a classification model capable of predicting which participants can be classified as being in the high risk for falls group (class 1) and which can be classified as being in the low risk for falls group (class 0).
- Understand which is the better classification model to apply in this context.
- Understand if the dataset includes unnecessary information for the classification, such as irrelevant tests.
- Understand if the use of sensors is profitable for the classification.
- Study several feature selection and balancing methods, proper to solve issues that occur in the construction of Machine Learning models, understanding which could apply better to this scenario.
- Study the general applicability of Machine Learning in this field.

1.3 Dissertation structure

In chapter 2 the existing assessment tools are described, starting by describing tools that are based on one test, further explaining the ones that are based on multiple tests, the improvement obtained with the use of sensors and the recent work that includes the application of Machine Learning. Later on the chapter it is taken a deeper focus on Machine Learning, briefly explaining what it is, in what fields it can be divided and several algorithms that cover its main domains of classification (the type of problem addressed here). Follows an explanation of the several feature selection methods that can be applied to the dataset, to favor the learning process, easing computational effort and improving results. Another section describes several methods that can be applied to the dataset in order to ease a problem called "class imbalance" that is derived from the different number of observations from the groups to classify. The chapter is ended by the explanation of the process of evaluating a model and by describing briefly some of the existent Machine Learning tools.

In chapter 3, it is described the dataset by first characterizing the participants of the study, describing the tests they executed and how the results are represented in the database. The dataset is then divided in a group of participants that had fallen after the collection of the information and the group that hadn't, presenting the statistical differences that exist between them. In the final part of the chapter some major issues that the dataset has are presented, normal issues that make part of a Machine Learning problem, and how they were treated in this work.

In chapter 4 it is described the approach to the problem. The tests that were made on each algorithm are explained and the general flow of a test is explained step-by-step. It is also explained specific characteristics of the application of the several algorithms.

In chapter 5 the results of the executed tests are presented, for each one of the set of experiments described in the previous chapter.

Finally, the chapter 6 presents the main conclusions obtained in this work, by the analysis of the results presented on the previous chapter. It is also explained future work that can be made in order to improve what was achieved with this work.

Chapter 2

State of the art

2.1 Introduction

In this section some of the relevant tools for risk assessment and their major advantages and disadvantages will be presented and analyzed.

This chapter goes from analyzing the evaluation of risk for falls through simple tests, explaining multiple falls risk tests and the respective tools, explaining the use of sensors in the assessment and finally the recent use of Machine Learning in this field. The final part of the chapter presents a deeper description of Machine Learning and associated algorithms ending with how to evaluate their performance and known tools that allow the application of machine learning processes.

2.1.1 Assessment tools based on one test

The necessity to correlate the results of the tests on the risk factors and the risk for falling surged in order to predict who would be the persons that had greater tendency to fall and take proper actions to avoid this negative occurrence. If a fall can be related to a specific risk factor, what is its contribution to the assessment of the risk for falling?

Usually, developed tests to assess the risk for falling use a population of participants from which is gathered their performance when executing the test. This is a sample population that is used to understand the applicability of the test. After the collection of performances, it is then started a follow-up period where the participants are accompanied in order to gather information about which of them ended-up falling or not, usually six months or one year after the collection of information. Having this, it is possible to start relating the performance of the participants with their future falls, understanding better what specific performances indicate about the risk for falling, these are also called prospective studies. Some studies are based on the number of falls in a recent past, or prior history of falls in the last year, and these studies are frequently called retrospective studies.

The Timed Up and Go test measures the time someone needs to stand up from a chair, walk three meters and return to sit again. It carries both simplicity and ease of administration since it only implies the use of a chair and a watch to count the time taken in the test. It is a clinical useful tool although it isn't clear to the point of understanding what factors justify the time taken to complete the test. [WLC05]

The Physiological Profile Assessment (PPA) is a tool that provides objective information about the contribution of vision, peripheral sensation, muscle strength, reaction time and postural sway to the risk for falling. It is an easy to apply tool in several settings. Many studies have been made with the PPA. One revealed an accuracy of 79% in a prospective study with a follow-up period of 1-year. The sample population was of 95 residents in an intermediate care hostel. Other more comprehensive study of the PPA involved 414 community dwelling women and the results obtained revealed an accuracy of 75%, it was also a study with a follow-up period of 1 year. [LMT03]

The comparison between the TUG test and PPA was also studied, to understand if the TUG could be a predictor of the PPA score. The study [WLC05] was made with 110 patients with a mean age of 79.3 years in the falls clinic of the King's College Hospital. Increased TUG test times and the presence of cognitive impairment were correlated to PPA scores.

The main conclusion was that a cut-off-point of 15 seconds in the TUG test were the acceptable maximum time that provided maximal sensitivity and acceptable specificity to identify patients with marked PPA falls risk scores. Needing more than 15 seconds to finish the test showed that the participant could be considered as having high risk for falling. This study ended up having 81% sensitivity and 39% specificity in the identification of the high falling risk group.

In an effort to understand how gait is associated with risk of falls a study was made in a cohort of community residing adults. A base line assessment was made using a computerized walkway with some embedded pressure sensors. This test is similar to the 10m-walk test (10MWT). The 10MWT measures the time, in seconds, that someone needs to walk 10 meters. Some gait variables were computed. This study was made with a follow-up period of 41 months and, in each follow-up assessment, information about the patient falls were recorded. At the end, 597 participants were eligible and their data was analyzed. The main conclusions from this study were that gait markers, independently, can strongly predict falls and a participant that had slow gait speed, below 70cm/s, had a 1.5 increased risk for falls when compared to people with normal gait speed. The gait markers can be measured with the base line test. [VHLW09]

Trying to understand the performance measures that have the greatest potential to predict falls in community-dwelling elderly adults, a study with participants from a senior's citizens center was conducted. There were 50 participants that were followed closely for 14 months after the initial screening. The screening process included several tests. The data collected from the tests was analyzed and screened with tests of collinearity. Pearson, point biserial and Phi correlations among the variables were calculated and the cut-off values for each one of them was chosen based on the best combination of sensitivity and specificity. For analysis only 11 participants were eligible because they were considered as being "true fallers", they had fallen on the follow-up period and this falls didn't derive from an accident. The main conclusion from this prospective study is that the timed floor transfer test (test that measures the time needed to move from a standing to a sitting position and back to standing in seconds) and the timed 50ft walk-test (test that measures the time taken to walk 50ft) were the only significant variables that could correctly classify the participants. A combination of these two measures leads to good sensitivity (81.8%) to predict the falls-group and great specificity (100%) to predict the no-falls group. [MOPO03]

To understand how the performance of a person in the modified Romberg test, taking in account differences by age, gender and race, can be translated in a measure for predicting falls, a study was conducted that analyzed the information on the performance of 4743 persons and compared it with their recent past history of falls, in the past year. From 2001 until 2004, the National Health and Nutrition Examination Survey (NHANHES) collected balance tests performances, in the modified Romberg test, of 5086 adults who were 40 years or older. Prior to the testing, the participants were presented to a questionnaire regarding history of falls in the previous year and history of dizziness. The Romberg test evaluates the capacity of the participant to stand unassisted under four test conditions, that test the sensory inputs that have influence on balance. The first condition implied that the participant executed the test with eyes open, on a firm surface; the second test condition implied that the participant executed the test with eyes closed, on a firm surface; the third that the participant executed the test with eyes open, on a compliant surface; the fourth and last implied that the participant executed the test with eyes closed, on a compliant surface. From the initial number of participants, 4743 were eligible for data analysis. Considering the time to failure as being the time someone needs to fail the test after initiating it, the authors used logistic regression in order to compute the odds of falling related to the different times to failure taken by the participants. This study made the connection between the time the participants took to complete the test with their past history of falls and concluded that the probability of having fallen in the past becomes extremely higher when time to failure is less than 20 seconds. Another main conclusion is that most of the participants, independently of their gender and race, cross this threshold by the time they were 60-69 years old. [ACH+11]

The main conclusions from these studies were combined in table 2.1.

These are some of the studies that tried to predict the capacity and how different performances on the basic tests for risk factors can predict the inclusion of a participant in a "High-risk for falls group" or in a "Low-risk for falls group". A lot of research has been made in this field so, due to the large amount of available research, this is not intended to be an exhaustive listing.

2.1.2 Assessment tools based on multiple tests

Having as basis that the risk of falling increases with the number of risk factors that are present [APH13], researchers started to develop tools and methods to combine multiple tests that examined multiple risk factors. A brief overview of these tools is done in this section. These tools usually assign a score to a person based on single scores that the person obtains in several single smaller tests that evaluate different risk factors.

A tool named STRATIFY that allows the assessment of risk for falling is a clinical tool that can be used in a hospital. Whenever a fall occurred in the center were this study occurred, the patient primary nurse was interviewed. From this interview were extracted a list of 21 separate pieces of information from which it was possible to understand which were the factors that most contributed

Table 2.1: Comparison of assessment tools based on one test.

Test	Population	Type	Conclusion	Sensitivity	Specificity	Compared to
10min walk Test	59	Follow-up from September 2004 until February 2008	Gait speed below 70cm/s led to 1.5 increased risk for falls	-	-	People with normal Gait and Speed
Romberg Test	4743	History of falls in the past 12 months	Time to failure below 20 seconds means a more than three-fold increase in risk for falling	-	-	-
PPA	414	Follow-up of 1 year from the initial screening	Accuracy of 75%	-	-	-
Timed Up and Go Test	110	-	Sensitivity: 81% Specificity: 39%	81%	39%	PPA
Combination of floor transfer test and Timed 50ft walk-test	50	Follow-up of 14 months after the initial screening	Floor Transfer Test: Sensitivity: 64% Specificity: 100%	64%	100%	-
			Timed 50ft walk-test: Sensitivity: 91% Specificity: 70%	91%	70%	

to the fall. In order to choose which variables to include in the assessment tool, odds ratios for all differences were calculated. The assessment tool is composed of five questions. These questions can be answered with a "yes" or "no". For each "yes" answer, the patient gets one more point in its risk for falling score. A "no" answer maintains the score. The main conclusion from the development of this tool is that, when it was applied in the clinical space were it was developed, a risk score of 2 obtained with the tool had a sensitivity of 93% and a specificity of 88% in predicting a fall in the week right after the occurrence of a fall. The tool was validated in a remote cohort of 331 patients. The results showed a sensitivity of 92% and a specificity of 68% for a risk score greater than 2. [OBS+97]

Another tool is the Falls Risk Assessment Tool (FRAT). A study that attempted to demonstrate the reliability and validity of the tool is described next. The tool is divided in eight categories, with each one of these categories receiving a specific value for each patient on evaluation. The final risk score presented is based on the sum of those values. The categories are age, confusion/agitation, elimination, history of falls, sensory impairment, activity and medications. The Committee that developed the tool decided that for a person to be considered in a high risk for falls group it needed to have a score higher than 10. This study was conducted with 89 patients in 6 medical/surgical units. One of the conclusions from this study is that these categories combined are significant. The main conclusion from this study is that this tool has sensitivity of 43% and specificity of 70% [MSH96]. It was a study made on a small sample of participants.

A tool that is quick to administer in acute care is the Hendrich II Fall Risk Model. It assesses the risk for falling in mental and emotional status, gender, dizziness and medications. It allows the intervention on specific areas of risk and a final score equal or greater than five means that the person evaluated is at a high risk for falling. This tool has a sensitivity of 74.9% and specificity of 73.9%. [Ann16]

A study that aimed at evaluating the Mobility Interaction Fall (MIF) chart, compared the results to history of falls and to staff judgment. The study was done in Sweden with 208 residents from residential care facilities, with a follow-up period of six months. The MIF chart includes information of concentration rating, a vision test result and the capacity to walk while, at the same time, interacting with another person. It is intended for use in residential facilities. The tool, in its development stage had a sensitivity of 85% and a specificity of 82%. They wanted to understand the accuracy of the MIF chart in an new sample. The time to the first fall for this population of 208 residents living in a residential care facility was analyzed. One of the conclusions of this study was that the MIF has lower sensitivity and specificity than staff judgment and history of falls and also lower scores when compared with the values obtained in its developing phase. The MIF sensitivity dropped to 43% and the specificity to 69%. [LOJNG03]

Another tool that can be used in residential care facilities is the Downton Index. A study that measured the predictive accuracy of this index at three, six and twelve months. This study was done in a residential care facility and 78 residents participated with a follow-up process of twelve months. This index can be viewed as a list of eleven items, each representing a risk for falling, each worth 1 point. The index final score is obtained through the summing up of the points of these eleven items. A score of three or more with this index is indicative of a high-risk for falling. In this study the score was calculated by a physiotherapist. The number of falls for each of the participants was stored in each follow-up procedure. With all falls included, the Downton Index got highest sensitivity value at the 3 month base prediction with a value of 95% but a low specificity of 35%. It is a study based on a small sample of 78 participants. [RLOK+03]

A web-based tool that uses a rule-based probabilistic logic program was developed, having rules for each risk factor, in order to compute the risk for falling of a person through the input of its health profile in a web browser. Its called FRAT-up. Some performance indicators were computed and the results showed that it is a comparable tool to some other validated state-of-the-art tools, having a receiver operating characteristic area under the curve (ROC AUC) of 0.642 and a Brier score of 0.174. It is a tool directed to the community dwelling population. It has the basic assumption that there is an independent contribution of the risk factors for the real, grouped, risk for falling. Although, the tool compensates the fact that the assumption not always applies through the use of synergy factors, different approaches could be investigated. Also, the tool could benefit from validation in other datasets. [CPP+15]

The table 2.2 presents an overview on the conclusions made on these studies and reported tools.

2.1.3 Assessment using sensors

To improve the results obtained from the tests and to surpass subjective scoring, sensors started to be part of the assessments on the risk factors.

A study that tried to obtain more objective data from the TUG test started to understand the different stages of the test using some sensors, namely a triaxial accelerometer that gives information about the movement on all the anteroposterior, vertical and lateral directions. To measure postural

Table 2.2: Comparison of assessment tools based on multiple tests.

Tool	Population	Type	Conclusion	Sensitivity	Specificity	Compared to
STRATIFY	331	Clinical Tool	Score of 2 or higher indicates high fall risk Applied clinically with subjective questions	93%	88%	-
FRAT	89	Clinical Tool	Score of 10 or higher indicates high fall risk. Small sample size	43%	70%	-
Hendrich II Fall Risk Model	-	Clinical Tool	Score greater than 5 indicates high fall risk	74.9%	73.9%	-
MIF	208	Applied in residential care facility	It has lower sensitivity and specificity than staff judgement and history of falls and that its development phase version	43%	69%	Staff judgement and history of falls
Downton Index	78	Applied in residential care facility	Score of three or more indicates high fall risk. Based on small sample. Best predictive values for a base prediction of 3 months.	95%	35%	-

displacement, a gyroscope sensor that enables the measuring of angular velocities was used. The use of these sensors allowed a better understanding of the different phases that compose the test. This study was made in hemiplegic persons and led to the conclusion that the use of sensors can lead to greater comprehension and enhanced results on the tests. [HYF+08]

A study that tried to move the clinical models for the assessment of risk for falling to a surrogate technique that could be used in an unsupervised environment was tested in an attempt to make the fall-risk assessment more broaden. They performed a routine of unsupervised physical tasks with the help of triaxial accelerometer for movement characterization. The performance of this model was compared with the PPA. This study was applied in 68 subjects that performed the directed routine. They found a reasonable correlation between their assessment and the validated PPA assessment with a value of p=0.81. [NRS+10]

To improve the correlation between this kind of test with triaxial accelerometry and to empower the deployment of home-monitoring systems, another study altered the way of analyzing the data, incorporating features from spectral analysis and got a better correlation of p=0.96 with the PPA. [LRW+11]

Further studies showed that the use of instrumented versions of tests, namely using accelerometry could improve the utility of traditional tests, like the five-times-sit-to-stand test. A study on 40 community-dwelling older adults was conducted. The participants were categorized in fallers or non-fallers based on their falling past history. The conclusion is that the use of body-worn sensors can lead to better predictions and remove subjective conclusions in tests. Also, that it can be used in unsupervised settings. Although it is a study that could benefit from a larger dataset and additional analysis. [DFF+11]

In a similar way, a study analyzed the accelerometer-derived parameters of center of mass (COM) displacement in order to identify older adults at risk of falling. Balance trials were performed in groups of older people already categorized as fallers or non-fallers. During the execution of the tests, the participants had a triaxial accelerometer secured to their lower back. The main conclusion was that the use of accelerometer based estimates of COM displacement may empower the quiet standing falls risk assessments and even enable its use in an unsupervised balance assessment. [DMG+12]

A review of the literature in this field revealed that wearable devices using inertial sensor technology, inexpensive and portable are capable of acting as fall risk assessments that can discriminate between fallers and non-fallers. Further studies must be done to validate the usage of these technologies in home settings with prospective methods in order to plan interventions. One of the reviewed studies, with a sample size of 39 persons, based on prior history of falls, showed that it was possible to distinguish between fallers and non-fallers using accelerometer-derived parameters with 74% accuracy, 80% specificity and 69% sensitivity. [ELD14]

Sensors are showing to be of great help in the assessment of the risk of falling leading to more comprehensive and objective results.

2.1.4 Machine learning on falls

Recent studies started to incorporate machine learning algorithms in their process of understanding how the risk for falling can be predicted having as basis the performance in specific tests.

A model that used Support Vector Machines (SVM) and sensors to differentiate between faller and non-fallers was studied and compared with the Berg Balance Scale. Results show a mean classification accuracy of 71.52% in a study sample of 120 community dwelling older adults. [GMW+12]

In a way to improve the use of sensors a study was conducted which used a computer and console games sensors on the tests. The signal from this sensors was analyzed and a feature vector was computed. Feature selection algorithms (not specified) were also applied in order to restrict the number of features used. The classifiers used were: The Naive Bayes (NB), the locally weighted learning (LWL) classifier, the Adaboost classifier and the Dagging classifier. The sample size was of 37 subjects and the study was based on their prior history of falls. The application of methods resulted in an accuracy of 89.2% with the Dagging classifier. The more accurate classifiers were the Dagging classifier and the Naive Bayes. [LTDRdS14] The study was made on a small sample, with different sensors and different tests from the ones presented in this work and was based on prior history of falls, being the presented work based on future falls.

A study that used a 3d motion capture system, the Vicon, extracted gait features from tests. When the participants reached their comfortable walking speed, the system started to capture data about the motion. Then, it was used machine learning algorithms in order to binary classify participants as fallers (with falls in the past 12 months) or non-fallers (without falls in the past 12 months). The study sample was composed of 35 older adults. It was used the K-Nearest Neighbor (KNN), Naive Bayes, Logistic Regression (LR), Neural Network (NN) and Support Vector Machine. All the methods, except KNN, got accuracies of over 90%. The study requires a larger number of participants and more features should be used. It is also based on prior history of falls and not on future falls. [ZMF+15]

A recent study, reported a statistical method for assessing fall risk using standard clinical fall risk factors, and the combination of the same method with a fall risk assessment algorithm with data from inertial sensors used in the execution of the TUG test. For the standard clinical fall risk factors was developed a logistic regression model that classified the subjects according to their falls history. From the data that included the information on the inertial sensors it was used a regularized discriminant classifier model that outputted a statistical fall risk estimate. This method was validated using prospective falls (two-year follow-up) and prior history of falls. For the combination of the two methods the fall risk estimate is obtained by classifier combination, averaging posterior probabilities calculated from the two. Based on a sample of 292 community dwelling older adults, the study suggests that the combination of the clinical and sensor approach show a classification accuracy of 76%, comparing to 73,6% for sensor only based approach and 68,8% for clinical risk factors only. The study also suggested that heterogeneity between cohorts may be an issue for the generalization of this kind of tools. The validation of the sensor approach in an independent cohort (22 community dwelling elders) showed an accuracy of 72,7%. [GRC16]

The main conclusions from the analysis of these studies are included in table 2.3. The study that achieved highest performance in Accuracy was the study from Lin Zhang that achieved over 90% in Accuracy in all tested methods.

Table 2.3: Comparison of Machine Learning approaches

Study	Population	Sensors	Algorithms applied	Conclusion	Accuracy
Barry R. Green et al.	120	Pressure sensitive platform and a body-worn inertial sensor	SVM	It was compared with the Berg Balance Scale	71.52%
Patricio Loncomilla et al.	37	Kinect, Wii balance board and two Wii motion controllers	NB, LWL, AdaBoostM1, Dagging classifier	The more accurate classifier was the Dagging classifier. Small sample size. Based on prior history of falls.	89.2%
Lin Zhang et al.	35	Vicon (3d motion system)	NB, NN, SVM	Based on prior history of falls. Small sample size.	All methods got accuracies over 90%
Barry R. Green et al.	292	Two wireless inertial sensors	LR, Regualized Discriminant classifier and a combination of the two	The combination showed the best result.	76%

2.2 Applicability of Machine Learning in an Assessment tool

Machine Learning algorithms have started to be incorporated in the analysis of the test results and in the prediction of the risk for falling.

Machine learning can be viewed as the effort to effectively make computers learn something. Normally it is associated with systems that have tasks of diagnosis, prediction, recognition, planning and others related with artificial intelligence and gets its inspiration in fields like probability and statistics, computational complexity, information theory, psychology and neurobiology, control theory, phylosophy and its always evolving. [Nil05] [Mit97]

A Machine Learning system is a system that upon receiving some input information, processes its characteristics and reaches a conclusion. Its said that it learns and learning involves a search through a set of possible hypothesis to find the one that best represents the training examples and prior constraints on knowledge that are presented to it. [Mit97] Machine learning can be broadly divided in two fields based on the type of learning of the systems:

In supervised learning a training set is given to the system. This training set includes vectors of input information and also the desired outputs. The system learns from the information, understanding how the input information can be transformed into the output. The main objective is that when it is presented with new information from which it doesn't know the output, it is capable of producing the correct output. Supervised learning can be broadly divided in two sub-types: Classification and Regression. Classification is the sub-type that deals with discrete outputs, attributing classes to the input information, whilst regression deals with continuous outputs.

In unsupervised learning the system tries to find clusters in the information that it receives on some criterion. In this approach, the system has no prior knowledge of the information that it is receiving neither about its structure and tries to find patterns in the input information.

Semi-supervised learning is a mix of the presented before learning types. The input data is a mixture of information from which is known the desired output and data from which the output is unknown.

The present work is concerned with supervised learning. The information to serve as input to the systems is information on the results from the various tests conducted by the elderly population, similar to the studies described before, and the outcome expected from these systems is the classification of the participants as future fallers (high risk for falls, class 1) or non-fallers (low risk for falls, class 0), a discrete output, being therefore a classification problem.

The next section presents a short description of classification algorithms from the field of supervised learning that can be applied in this scenario.

2.3 Supervised classification algorithms

Before delving deeper in this section it is important to define and clarify some relevant definitions:

- Input data: The group of information that is introduced in the machine learning system. Also mentioned as dataset.
- Observation: An instance from the information that is inputted to the system. In the studies shown before, one observation can be viewed as the information relative to one participant. It can also be known as instance, sample or example.
- Feature: One characteristic of the observations that are in the inputted information. For example, participant (observation) X is 75 years old (the age of the participant as a feature).
- Label: The classification that is real or predicted for the observations. For example, in the previous studies, one participant could be classified as non-faller or faller, this means that

the label of the observation could be "non-faller" or "faller". It is the output of a classifier model.

- Training set: The group of observations and respective labels that are used to train a system.
- Test set: The group of observations and respective predicted labels that are used to test the performance of a system.
- Target function: Function that describes the possible relation between the inputted information and the expected output.
- Hypothesis function: One function that approximates the target function.
- Space of hypothesis: The set of hypothesis that represent possible solutions to the learning process. The goal of the process is to find the one that best approximates the target function.
- Bias: Measures how far the models predictions are from the correct value.
- Variance: Measures how a prediction for a certain observation may vary between different builds of the classification model.

2.3.1 K-Nearest Neighbors

This algorithm tries to classify the information having as basis similar attributions made to the training set (set used for learning). First, the algorithm evaluates if the k (passed parameter) more similar training observations to the observation d (the one that it is trying to classify) are classified as being part of class c and, if the answer is ”yes” to the majority of them, then the decision of also identifying the observation d as being part of class c is taken. If not, the opposite decision is taken. [Seb02]

The k similar observations are chosen based on the distance they have to the observation that the algorithm is trying to classify. This distance is normally the Euclidean distance. If we consider the observation as being described by the vector of features (information on the input set about the observation):

$$(a_1(\chi), a_2(\chi), \ldots a_n(\chi))$$

And assuming $a_r(\chi)$ as being the r feature of the instance X, then, the distance between two instances χ_i and χ_j is given by $d(\chi_i, \chi_j)$, then the distance is calculated as follows:

$$d(\chi_i, \chi_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(\chi_i) - a_r(\chi_j))^2}$$

Using euclidean distance. Other distance metrics can be used. [Kot07]

The value given by the algorithm is just the most common value of the target function $f(\chi_q)$ for the nearest k training examples of χ_q

If k is equal to 1, then we are presented to a special case of the algorithm, called the 1-Nearest Neighbor. In this case it is assigned to the function the value of the training observation nearest to the to be classified observation.

Algorithm:

- Construct a list of training observations. For each observation insert on the list $\left(\chi, f_{\chi}\right)$
- Having an observation χ_q to be classified and $\chi 1 ... \chi k$ being the k observations from the training observations nearest to χq, then the algorithm returns:

$$\hat{f}\left(\chi_q\right) \leftarrow \underset{v \in V}{\arg\max} \sum_{i=1}^{k} \delta\left(v, f\left(\chi_i\right)\right)$$

  - Having $\delta\left(a,b\right) = 1$ if $a = b$ and $\delta\left(a,b\right) = 0$ in the remaining cases.

It is possible to implement it for continuous valued target functions. For this it computes the mean value of the k training examples instead of calculating the most common one. One possible derivation of this algorithm is one that makes the calculation with a weighting factor. This weighting factor can be calculated based on the distance of the neighbors, having a greater weight the ones that are closer, for example.

This algorithm is based on the assumption that the classification of one instance is similar to the one of its closer instances by a distance metric. The distances between observations includes information on the observations attributes. This may lead to problems when dealing with instances with lots of attributes due to increased computational costs. One possible approach to this problem is to give different weights to different attributes when calculating distances or eliminating the least relevant ones from the observation space. There are some methods in order to index the stored training observations, so that identifying the nearest instances can be done in a more efficient way. [Mit97]

A good choice of k must be made in order to get good results. This algorithm doesn't have a principled way for choosing k, except for the use of cross-validation (CV) and similar methods. [Kot07][Mit97]

2.3.2 Naïve Bayes

This algorithm is based on a probabilistic approach to inference.

To start with, we need to explain what is the Bayes theorem, the basis of this algorithm. The Bayes theorem can be translated in the following probability calculation:

$$posterior\ probability = \frac{conditional\ probability \times prior\ probability}{evidence}$$

Basically, what this theorem says is that the probability of an event given evidence can be calculated with the previous formula. [Ras14]

This can be applied to machine learning. Having:

$$P(h/D) = \frac{P(D/h) \times P(h)}{P(D)}$$

- $P(h/D)$ - Posterior probability of having h hypothesis happen given the training data D.
- $P(D/h)$ - Probability of seeing data D in a world where the hypothesis h happens - usually also called likelihood
- $P(h)$ - Prior Probability of holding the hypothesis h - usually also called class prior probability.
- $P(D)$ - Prior probability that the data D will be observed

Providing $P(D/h)$, $P(h)$ and $P(D)$ the $P(h/D)$ can be calculated.
This algorithm assumes independence between all features. The likelihood of an observation being classified in a class can be calculated multiplying the frequency of the attributes values that result in that class on seen data. In the end of the process it is obtained a relative probability for each class to be assigned and the one with greater probability is chosen for the observation in consideration.

2.3.3 Decision Trees

The approach of building a Decision Tree (DT) is known as "divide and conquer". It splits the data into smaller subsets of similar groups. The algorithm begins at the root node (representation of the entire dataset) choosing a feature that is more capable of determining the classification of the observations. The first set of subtrees is created. This subset represents the group of distinct values that the feature may assume. Then, the process of "divide and conquer" continues, always choosing the best feature for the split, until some stopping criterion is reached. [Lan15]

The leaf nodes of the Decision Tree represent the classification of the observations in different classes and the remaining nodes represent a test on some feature of the observations to be classified (this nodes can be seen as "if-then" rules). Each one of the branches of the tree represents a value that the feature may assume. [Mit97][Kot07]

The typical process of classifying an instance goes as follows:

- Starting at the root, a base feature of the observation is tested.
- The value of the feature determines the path followed for the subtrees. Depending on the path followed, another feature is tested.
- New subtree is followed with tests to other features.
- When a leaf node is reached, the observation is classified

Having the Decision Tree example in figure 2.1, trying to classify the first instance:

- The "at1" for the first instance has the value "a1" so, following the branch with the "a1" value, we reach the node "at2".

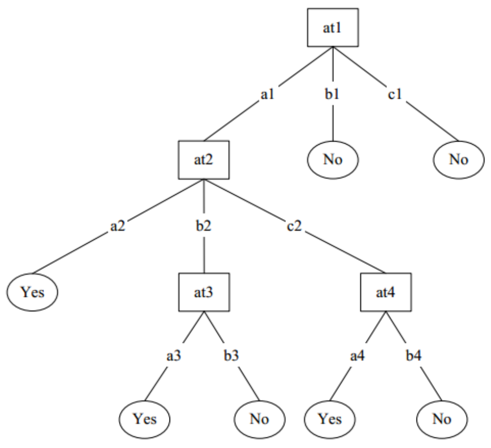


Figure 2.1: Example Decision Tree[Kot07]

- "at2" has the value "a2" so, following the branch with the "a2" value, we reach the classification of the instance to the class "Yes".

There are deviations of the base Decision Tree algorithm, the C4.5 algorithm and ID3, for example.

One way to determine which feature to test at each node is by defining a statistical property that can measure how good the feature is (alone) in determining the classification of the observations. The ID3 algorithm, for example, uses this concept, determining which attribute to use for splitting as it grows the tree. Then, the space of possible trees is searched through an hillclimbing technique, in order to find the simpler acceptable tree. [Mit97]

The process of performing classification is usually developed in two phases. The first one is "Tree Building" where the tree is generated as explained before and the other is "Tree Pruning". "Tree Pruning" enables to prune the leaves that lead to worst results (classification of very few observations). [DDZZ02]

Decision trees have good applicability on problems that have observations described with some fixed set of features. They are also seen as being suitable for application with target functions having discrete value outputs, although they can be expendable to be applicable on target functions with real-valued outputs. They are robust, sustaining the presence of some errors on the features and may even deal with missing features.

2.3.4 Random Forest

Random Forests (RF) can be seen as a collection of Decision Trees that vote for a final classification. The splits of the trees are selected having as basis a randomly selected subset of features, as opposite of what happens in traditional Decision Trees were the split is made on the full feature set. [TG09]

For the tree n that is being generated it is also generated a random vector γ that is independent of the random vectors that were created before. The tree n is grown with the training set and the random vector resulting in a classifier $h(\chi,\gamma)$ where χ is considered an input vector. [Bre01]

The process of creating the Decision Trees is finished. Each one of the trees reaches a classification when processing an observation, just like a traditional Decision Tree. The final classification is achieved counting the votes of the different trees of the forest. The majority of the votes wins (in case of regression it is calculated the average value). [LW02]

In Random Forests there are two parameters to tune:

- t, being the number of trees to grow
- m, number of features to consider in the process of generating the Decision Trees.

Breiman suggested that a showed pratically good value for m is

$$log_2(M+1)$$

being M the total number of features. [Bre01]

Random Forests can achieve low bias and low variance and are fast in making predictions and training, with good applicability for large problems. [TG09]

2.3.5 Support Vector Machines

The Support Vector Machines algorithm has as basis the principle of finding an hyperplane that separates two classes. [XDL+08]

This algorithm was created based on the Vapnik–Chervonenkis Dimension. A set of functions $f(\alpha)$ can show have the Vapnik–Chervonenkis Dimension property. Considering classifying functions that recognize patterns in two classes, having a group of points that can be labeled in all possible ways and that for each of the labellings there is a member of the $f(\alpha)$ that can do it correctly, then we can say that the group of points can be shattered by the set of functions.

In order to understand the simpler application of this algorithm one needes to see the space where the data is located as R^2. In this space there are straight lines, and for each of the lines, all the points that are on one side of it are labeled as being from $class1$ and on the ones that are on the other side are labeled as being from $class2$ ($class1$ and $class2$ are different classes, used as example). This group of lines is considered as being the set of functions $f(\alpha)$.

The region between the two zones where the observations are classified as being of each of the classes is defined as an hyperplane. Hyperplanes separate observations (like the $class1$ and $class2$). Considering that the distance from the plane to the closest $class1$ example is defined as $d(c_1)$ and the distance from the plane to the closest $class2$ example is defined as $d(c_2)$ then we can define the "margin" of this hyperplane as $d(c_1)+d(c_2)$. There can be more than one hyperplane, but what the algorithm will try to do is to find the one with the maximal "margin". This means that the algorithm will search for the hyperplane that is further away from all the training data points, creating the best separation possible. To minimize the error it is needed to obtain the largest margin possible.

The points that are in the hyperplane must satisfy the equation:

$$w \times x + b = 0$$

- w - normal to the hyperplane.
- $|b| / \|w\|$ - perpendicular distance to the origin.
- $\|w\|$ - norm of w.

The training data points that hold the following condition are considered as lying on one of two parallel hyperplanes that make up the solution:

$$y_i (\chi_i \cdot w + b) - 1 \geq 0 \forall i$$

These points compose the called support vectors.

In order to find to solution the algorithm needs to minimize

$$\|w\|^2$$

subject to the previous constraint. The figure 2.2 shows an example of a solution. Support vectors are represented as the extra circles.

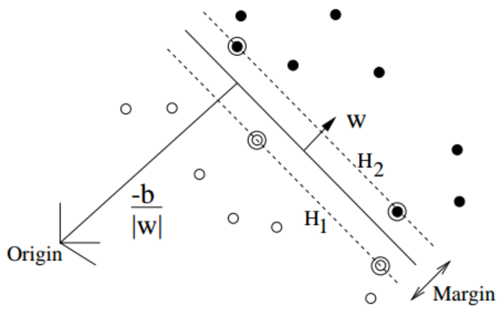


Figure 2.2: Support Vector Machine example, [Bur98]

Having this separating region, the algorithm can then classify observations as being from a certain class if they are on one side of the region or classify them as being from the other class if they are on the opposite side of the region.

Simpler Support Vector Machines, like the exemplified, can only classify data in a binary basis, but this functionality can be expanded making it able to classify multi-class problems. An in depth description of this algorithm can be found in the refereed tutorial. [Bur98]

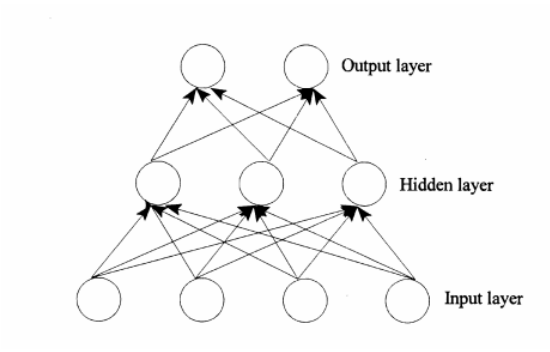


Figure 2.3: Neural Network example [BH00]

2.3.6 Neural Networks

Artificial Neural Networks try to be an abstraction of the biological neural networks of living beings that allow the learning mechanism to occur.

These artificial neural networks are composed of artificial neurons, connected to each other. These connections are weighted. Each of the artificial neurons receives an input from the environment, combines the input through calculations and then passes it through a threshold gate. When the value calculated in the neuron, the "net" input ξ, is over the threshold limit defined it is said that the neuron fires, becoming activated. Then, the output signal is transmitted to another artificial neuron. The net input can be calculated as:

$$y = \begin{Bmatrix} 1, & if \sum_{i=1}^{n} w_i x_i \geq b \\ 0, & if \sum_{i=1}^{n} w_i x_i < b \end{Bmatrix}$$

- n - number of input signals
- x_i - input signal
- w_i - weight
- 1 - on (can be a class if it is a classification problem)
- 0 - off (can be a class if it is a classification problem)

The artificial neural network can be seen as layers of neurons. This layers follow a sequence: Input layer, hidden layers (don't interact with the environment) and the output layer.

The training of the network happens by changing the value of the weights and the thresholds in order to obtain a set of values that correspond to a global minimum. These weights are adjusted in proportion to the error (difference) between the correct output and the neuron solution. There are rules to control how these weights should be adjusted.

This learning process, that happens iteratively as the network is presented with training examples, implies changes, adaptations of the weights of the connections. The capacity to learn of the neural network depends on the architecture that it reflects and on the algorithm used to train it. [YoS10]

The network can learn through supervised learning, being fed on correct answers or through unsupervised learning, understanding the data, its structure and possible correlation between the examples without any exterior intervention. It is also possible to combine both types of learning. Neural networks can be applied to a lot of fields and problems, being predictive, recognition and classification problems some in which it has shown good results. [BH00]

2.3.7 Convolutional Neural Network

This type of neural networks, Convolutional Neural Networks (CNN), are usually applied to the processing of images, having shown great results in recognition problems. They are similar to the normal neural networks, being composed of nodes and layers, although this networks show different types of layers. The simpler convolutional networks can be viewed as the following sequence of layers:

- Input layer - layer that holds the raw information, normally as convolutional neural networks deal with image processing, this layer holds the pixels from the images to analyze.
- Constitutional layer - the output of the neurons on this layer (that are connected to specific sub regions of the input) compute an output value, multiplying their weights to the small region they are connected.
- Relu layer - layer that applies element wise activation functions.
- Pool layer - performs a down sampling operation.
- Full connected layer - as a normal neural network, the nodes from this layer are connected to all the nodes in the previous layer. In this layer it is computed the final classification.

Convolutional neural networks can be trained using the same methodology of normal neural networks [Duf07]. As this type of neural network has shown impressive results in the image recognition field it is of interest to study its applicability to the problem in analysis. In order to do this, the data that was analyzed needed to be represented as an image.

2.3.8 AdaBoost

A normally made assumption is that the training data is composed of independently and identically distributed observations from a probability distribution $Prob(X,C)$. The main objective of a classification module is to find a classification rule $C(x)$ from training data that allows a new unseen input x to be correctly classified. Considering a Bayes classifier and its error rate $(1 - E_X max_k Prob(C = k|X))$, the expression:

$$C^*(x) = \arg\max_k Prob(C = k|X = x)$$

is used to minimize that rate.

The AdaBoost is an approximation of the Bayes classifier and what it does is an iterative process that combines multiple weak classifiers. The weak classifiers can be, as example, several decision trees. The AdaBoost learning process follows as: A normal process of classification unfolds for a weak classifier, with the observation weights initially set to a specific value. If the prediction with this base classifier originates misclassifications then the weight of each misclassified observation is boosted (increased). A second classifier is trained on the new weights and the process is repeated until a certain number k of weak classifiers is trained. The final classification is determined by the linear combination of the several classifiers.

AdaBoost has shown to be of great applicability in binary classification problems. [ZZRH09]

2.4 Selecting the best features

In this section it is described the problem of having an excess of features and what are some of the methods that can be applied to surpass it.

It is important to make a good selection of the features retaining the ones that can play a determinant role in the learning process and discard the remaining to save computational power and not introduce noise. By reducing the space of features one reduces the size of the hypothesis space and allows the algorithms to be more efficient and faster. It may improve the algorithm performance and/or simplify the representation. [LCiL15] This process of choosing the appropriate features for the problem at hand is called Feature Selection.

2.4.1 Algorithms and Methods for feature selection

There is a set of feature selection algorithms and methods that can be applied in order to reduce the dimensionality of the Machine Learning problem faced. This section serves as a listing and description of some of those methods.

2.4.1.1 Select Percentile

Some methods start by ranking the various features in one defined score and then proceed to the choosing of the more interesting ones by comparing the score they obtained to a fixed threshold. The ones that don't satisfy the comparing condition eliminated.

The features can be ranked according to the value obtained in an f test, that determines if there are differences between two group means [1] calculated between the class label and the feature itself. The selection follows as a selection of a specified percentage of features. It can also be calculated the mutual information between the features and the labels instead of the f-value, following the remaining selection scheme.

The mutual information score can be seen as the amount of knowledge that one feature has of another random feature. Considering one random variable it is possible to say that it includes

[1] http://libguides.library.kent.edu/SPSS/OneWayANOVA

information about another random feature and therefore, seeing one allows to infer about the other. Mathematically speaking, mutual information is the relative entropy that exists between the joint distribution and product distribution of two variables. [CT91] The f-test score value is the representation of the comparison of two variances in this case relating the feature that is being analyzed with the class labels.

2.4.1.2 Select K Best

In the same logical thought, with this method features are ranked according to the same possible scores and the top K features, with the highest K scores, get selected. The ones that are not in this group get rejected.

2.4.1.3 Select False Positive Rate

This method, Select False Positive Rate (Select Fpr), basis its selection by controlling the total amount of false detections through the false positive rate test (number of false positives in relation to the actual number of negatives, as a ratio) and retains the features that show a score below a specified threshold. The base scoring function can be the f-value and the mutual info score as well.

2.4.1.4 Select False Discovery Rate

This method, Select False Discovery Rate (Select Fdr), works in the same way as the previous one, although instead of using the false positive rate it uses the false discovery rate (that is a conceptualization of the rate type 1 errors, the incorrect rejection of a true null hypothesis, known as false positives).

2.4.1.5 Select Family-wise Error Rate

This method, Select Family-wise Error Rate (Select Fwe), works in the same way as the previous ones, although instead of using the false positive rate or the false discovery rate uses the family-wise error rate, that determines the probability of getting a false positive result. The scoring base scorer for this method can also be the f-value and the mutual information score.

2.4.1.6 Recursive Feature Elimination

This method, Recursive Feature Elimination (RFE), also focuses firstly in ranking the features according to the ranking criterion:

$$w_i = (\mu_i(+) - \mu_i(-))/(\sigma_i(+) + \sigma_i(-))$$

In this equation the μ_i represents the mean of the values of the features, the σ_i represents the standard deviation and the + and - represent the classes for the samples i. The value of w can be interpreted basically as being the correlation of the feature with the classes. If the value is highly negative it is more correlated with class (-), if the value is highly positive it shows a greater correlation with class (+). In order to use this, a class predictor that functions with weighted voting of the features. The voting scheme produces a classifier as in:

$$D(x) = w.(x - \mu)$$

The w in this equation represents the weight and was already defined before. The μ represents the mean vector over all training patterns as follows:

$$\mu = (\mu(+) + \mu(-))/2$$

A change on the weight of a feature changes its impact in the cost function. This estimator works by firstly assigning weights to each one of the features with a primary train and then, the features which have the smallest heights get eliminated. The process gets repeated, recursively, until the wanted number of features is achieved. [ABeÇ16]

2.4.1.7 Recursive Feature Elimination with Cross-Validation

The Recursive Feature Elimination with Cross-Validation (RFECV) is the same method as the one presented before but using cross-validation to choose the best number of features.

2.4.1.8 Variance Threshold

This method focus on the removal of features that have low variance (don't change much between all the samples) and so may not introduce richness to the classification process. It acts by eliminating the features that show a variance below a specified threshold value.

2.4.1.9 Principal Component Analysis

The main goal of the Principal Component Analysis (PCA) is to extract the meaningful information from a set of variables that are inter-correlated. It takes information from the initial set of features and creates new ones (by linear combination of the first ones), called Principal Components, that are able to express the same information. The objective is to reduce the size of the set by creating fewer features than the initial ones. Normally, there are processing steps taken on the dataset, approached as a matrix of samples by features, IxJ, the matrix X, prior to the application of PCA. This steps include the centering of its columns, achieving a mean of 0 by column. There are two types of PCA, the covariance PCA, where the features are divided by $\sqrt{I}$, being I the identity matrix, because the X^TX matrix is a covariance matrix. The other type of PCA is the correlation PCA where the features appear with different scales and are standardize, divided each one of them by its norm, because in this case X^TX is a correlation matrix. Considering that the sum of the squared elements of a column represents the column's inertia, the application of PCA requires that:

- The first component found to show maximal variance, thus being the one that extracts most of the inertia from the matrix.
- The following component to be orthogonal to the former, having the largest possible inertia.
- The other components are computed in the same way.

New variables are found for observations and are called factor scores that can be seen as projections of the samples in the principal components. The computations are applied in the singular value decomposition of the matrix X:

$$X = P\Delta Q^T$$

Where $P = IxL$ represents the matrix of the left singular vectors, with L being the rank of the matrix, $Q = JxL$ represents the matrix of the right singular vectors and Δ the diagonal matrix of singular values. [AW10]

2.4.1.10 Feature Agglomeration

This algorithm works by recursively merging clusters by features that minimize a criterion of distance. This criterion can be variance, as an example, and the clusters get merged in a way that they get minimal variance.

2.4.1.11 Manual Feature Selection

The last method can be basically seen as a junction of knowledge of the problem domain and intuition in order to manually select the desired features, creating special subsets.

2.5 Imbalanced Scenario

Most algorithms need the class distributions to be balanced or, at least, to have equal misclassification costs. If the algorithm is not fed with a dataset that presents this kind of characteristics than it may fail to understand and represent the characteristics of the classes. [HG10]

Some important concepts that can be defined before delving deep in this chapter: the class that has more samples in the dataset used for training is referenced from now on as being the majority class; the class that has fewer samples is referenced as being the minority class; by balanced dataset one understands that the dataset is composed of the same amount of samples from both classes (since this is a binary classification problem); by opposition, an imbalanced dataset is one that shows uneven number of samples from the classes. Samples that are borderline are samples that are close to the boundaries between both classes, while noisy samples are samples that, belonging to a certain class, situate themselves far inner the space of the opposite class by having characteristics more common to the samples of that class. Samples that are redundant are far away from the decision boundaries inside the space of their class and safe samples are the ones worth keeping [KM97].

In the domain of classification, an algorithm has always more tendency to predict new samples as belonging to majority class that was used for training the algorithm. This can be explained since the presence of induction rules describing the minority class characteristics are usually fewer and less powerful than the ones from the majority class. [HG10] The imbalanced learning problem can be faced from many perspectives and so, due to its importance, a lot of research has been conducted in recent past. Following is a list of used approaches:

- Sampling approach: This approach intends to balance the distributions by removing samples of the dataset or by creating new samples, either by copy of existing ones or through the generation of new ones. Each approach has its own problems, but this methods have shown good aplicability. [HG10]

- Cost Sensitive approach: While the first methods explore the balancing of distributions by changing the amount of samples from the classes, this methods try to change the learning process by assigning costs to wrong classifications forcing the model to reach better performances.

- Ranking: This methods explore the comparison of samples, predicting which one should be "preferred" instead of the others. This will be further explained later. Ranking surpasses class imbalance since by comparing each observation from one class with all from the opposite the training process gets balanced. [CFCP16]

[HG10]
This work is centered on the two-class imbalanced learning problem.

2.5.1 Sampling methods

The sampling methods can be divided in groups by characteristics.

- Under-sampling methods: Consists in reducing the number of samples from the majority class in order to reach balance.

- Over-sampling methods: Consists in increasing the number of samples from the minority class in order to reach balance.

- Ensemble methods: Consists in creating successive subsets from the initial majority class set and classifiers based on the combination of the subsets with the minority class, inside ensembles.

- Combination of methods: Consists in joining methods in order to reach balance.

2.5.1.1 Cluster Centroids

This algorithm performs under-sampling of the majority class by first discovering the K clusters existent on the class and then substituting those clusters by their centroids, assuming that they are new samples from the majority class, achieving a balanced dataset. It is therefore an under-sampling algorithm.

2.5.1.2 Condensed Nearest Neighbour

This algorithm is based in the Condensed Nearest Neighbour rule. This rule is related to the nearest neighbour rule which consists in classifying a sample as being from the same class as the class of the nearest n examples that are already classified correctly. The condensed nearest neighbour rule shows the same basic approach. A consistent subset is considered as being a subset that, under the nearest neighbor rule, is capable of classifying correctly the remaining set. It can also be minimal if it includes only the minimum necessary number of samples. The algorithm based on this rule finds the subset of majority samples that are capable of attaining such characteristics as described in [Har06]. This way it is possible to find the smaller subset of samples that is capable of describing the characteristics of the majority class, eliminating the remaining. Thus, it is considered as an algorithm of under-sampling.

2.5.1.3 Edited Nearest Neighbor

The Edited Nearest Neighbor (ENN) algorithm starts by creating a set S that is equal to the training set and then, each instance in S gets removed if it does not agree with the majority of its k nearest neighbors in terms of their classification, like the nearest neighbor rule described before. [Wil72] This algorithm reduces the number of samples from the majority class by eliminating the ones that don't agree. Thus, it fits in the under-sampling type of algorithms working as an algorithm to clean data.

2.5.1.4 Repeated Edited Nearest Neighbor

The Repeated Edited Nearest Neighbor (RENN) algorithm can be basically explained as being an extension of the previous one, by applying the same process to the obtained subset in order to obtain a further refined subset. This algorithm also fits the category of the under-sampling algorithms.

2.5.1.5 All-K-Nearest Neighbor

The All-K-Nearest Neighbor (AllKNN) algorithm works by, having a value k and a sample, finding the i nearest neighbours of the sample. If the majority of the nearest neighbours classifies the sample incorrectly then the set gets a flag with value zero. While i is less than k the algorithm goes on, repeating the process, with an increased by one value of i. At the end, the samples that got the flag equal to zero get eliminated [Tom76]. By decreasing the number of samples this algorithm fits in the category of under-sampling algorithms.

2.5.1.6 Instance Hardness Threshold

Hardness stands as a property that represents how likely a sample will be misclassified when the learner is trained on the other samples of the dataset. It works with a base estimator, for example random forests, to allow the calculation of the hardness of the instances. This algorithm has as objetive the reduction of the class overlap and instance hardness and to do so it eliminates the samples that show an instance hardness value greater than a specified threshold [SMGC14]. This algorithm is also considered as being from under-sampling category.

2.5.1.7 Near Miss

This algorithm has three operating possible versions. All of them have as basis the KNN algorithm, already described to find the nearest neighbors of a sample:

- Version 1: Consists on the selection of the samples from the majority class that are closer to some of the samples from the minority class, choosing the ones that show the smaller average distance to three samples from the minority class.
- Version 2: In this version the samples that are close to all the samples from the opposing class are selected. This is done by the calculation of their average distance to the three farthest samples.
- Version 3: In this version a specified number of samples, from the majority class are chosen relative to each one of the samples from the minority class, the ones that are closer, guarantying that all the samples from the minority class are surrounded.

[ZM03]. All versions may be used to reduce the size of the majority class. It is also considered as being an under-sampling algorithm.

2.5.1.8 Neighborhood Cleaning Rule

This algorithm is based on the Neighborhood Cleaning Rule (NCR). What it does, more than reducing the size of the majority class, it concentrates on data cleaning by eliminating noisy samples. Considering that T is the original data, that C is the class of interest and O the remaining data, the algorithm applies in O the Wilson's edited nearest neighbor rule that is capable of removing noisy data by removing the examples which classification is different from the majority classification from its three nearest neighbors. It also cleans C by removing the three nearest neighbors that belong to O and misclassify examples of C. [Lau01] This algorithm is also considered as being from the under-sampling type.

2.5.1.9 Tomek Links

This algorithm works by removing Tomek Links (TL) in order to eliminate borderline samples. A Tomek Link is a concept that can be defined basically as: considering two samples x and y, with different labels (classes), if there is no other sample that is closer to one of the former, x and y, then they are called a TL, which reflects that they are noisy or borderline. By removing these links, this algorithm reduces the size of the majority class, thus being considered an under-sampling type of algorithm that cleans data. [KM97]

2.5.1.10 One Sided Selection

This algorithm is based on the One Sided Selection (OSS) method. This method prunes samples from the majority class in order to obtain a representative subset of the majority class. Being so it is considered as being an under-sampling type of algorithm. The algorithm starts with the creation of a special subset, C, of the training set, S, that includes all the minority samples and only one randomly selected sample of the majority class. Then, re-classification of S occurs using the samples in C applying the nearest neighbour rule considering only the nearest neighbour. The training samples that are misclassified enter the set C. Then and finally, the majority class samples that participate in Tomek links in C are removed. [KM97]

2.5.1.11 Random Under-sampling

This algorithm creates a subset, S, of the majority class, T, so that the probability of choosing one sample from T to S is equal for all the samples. The subset is sized in order to balance the class distributions. By reducing the number of samples from the majority class, this is an under-sampling type of algorithm. [Lau01]

2.5.1.12 Random Over-sampling

This algorithm creates a set, S, from the minority class, T, by picking samples at random from T and repeating them until the desired distribution of classes is achieved. The probability of choosing one sample from T to S is equal for all the samples. By creating new samples through replacement (repeating samples) the number of samples from the minority class increases, thus this is an over-sampling algorithm. [Lau01]

2.5.1.13 Adaptive Synthetic Sampling

The Adaptive Synthetic Sampling (ADASYN) algorithm is based on the Adaptive Synthetic Sampling approach, generating new synthetic data from the minority class. Since it generates new data, it is classified as an over-sampling algorithm.

It starts by calculating the degree of existing class imbalance and the number of synthetic samples that must be generated to balance the dataset. Then, for each sample of the minority class, the k nearest neighbors (euclidean distance) are found and a ratio, defined as $r_i = \Delta_i/K, i = 1, \ldots, m_s$

is calculated, where Δ_i is the number of samples in the k nearest neighbors from the focused sample that belong to the majority class and m_s is the number of samples from the minority class. The ratios are normalized creating a density distribution. This density distribution represents the distribution of weights for different minority class samples that lets the algorithm know how many synthetic samples it must generate for each one of the samples from the minority class. This way the generation goes according to the to level of learning difficulty of the minority samples, forcing the algorithm to require more attention to the hardest to learn samples. Each one of the synthetic samples is generated by randomly choosing one of the k nearest neighbors of the sample in focus from the minority class and then occurs a fusing between the sample chosen from the neighbors and the focused one, through vector calculus with a random number [HBGL08].

2.5.1.14 Synthetic Minority Over-sampling Technique

The Synthetic Minority Over-sampling Technique (SMOTE) algorithm works by creating new synthetic samples for the minority class. It generates samples by joining all or some of the k nearest neighbors of the sample in focus from the minority class. The generation is made as follows: First is calculated the difference between the feature vector of the sample in focus and its nearest neighbor, then the result is multiplied by a random number between 0 and 1 and added to the sample in focus. It is similar to the previous algorithm but it generates an equal number of samples for each one of the samples in the minority class. [CBHK02]

There are other versions of the SMOTE technique that focuses on the samples that are near the border that divides the classes, the borderline samples from the minority class. These techniques are called Borderline SMOTE 1 and Borderline SMOTE 2. There is another technique that approximates the borderline by support vectors that came from the training of an SVM on the training set called SVM-SMOTE.

2.5.1.15 SMOTE and ENN

This algorithm implements over-sampling through the application of SMOTE and then cleans the data using the ENN, in a method denoted by SMOTEENN, all described previously. As this algorithm combines two methods to balance the dataset it fits the type of combination of methods.

2.5.1.16 SMOTE and TL

This algorithm, as the previous one, works by applying over-sampling through SMOTE and then removes Tomek links for data cleaning. This methods were described previously.

2.5.1.17 Easy Ensemble

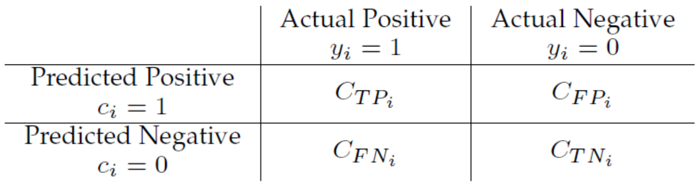
This method performs under-sampling in a special way. The initial training set, T, is divided in the subset of the majority class M, and in the set of the minority class L. The majority class M gets divided, randomly, in subsets M' that have the same size of L. Then, several base classifiers use each one of the M' and the L set to train. Finaly all are combined in an ensemble of classifiers that dictates the final classification [LWZ06]. This method fits in the ensemble methods category.

2.5.2 Cost-sensitive methods

The classification methods aim to minimize the misclassification of the samples. This assumes that misclassification errors have all the same costs, which, in most real world applications, is not true. Taking this work as an example, misclassifying one participant as a possible faller when he won't fall carries costs on treatment but doesn't put the participant in danger. Misclassifying one participant as not going to fall can be more dangerous since there wouldn't be proper intervention and the safety of the participant could be threatened. So, in terms of safety, one can conclude that misclassifying a participant as not going to fall when he would fall is worse than classifying a participant as going to fall when he wouldn't. Thus, it is possible to approach this problem as a cost-sensitive problem.

2.5.2.1 Cost-sensitive Random Forests

This method can be basically described as an ensemble of Cost-sensitive Decision Trees (CDST) trained on random sub-samples of the training set. The final classification can be obtained with majority voting, selecting the decision that has the highest number of votes from the base cost-sensitive decision trees. Each one of the cost-sensitive decision trees uses costs during the training and pruning phases of the tree growth.



	Actual Positive $y_i = 1$	Actual Negative $y_i = 0$
Predicted Positive $c_i = 1$	C_{TP_i}	C_{FP_i}
Predicted Negative $c_i = 0$	C_{FN_i}	C_{TN_i}

Figure 2.4: Classification cost matrix, as in [BAO15]

In figure 2.4 is presented a cost matrix that shows the cost of predictions versus real classifications. Considering the sample-dependent cost statistic as:

$$Cost(f(x_i^*)) = y_i(c_i C_{TP_i} + (1-c_i)C_{FN_i}) + (1-y_i)(c_i C_{FP_i} + (1-c_i)C_{TN_i})$$

, where f is a classifier that predicts a sample i as being from class c_i, y_i as the true label of the sample i and the remaing as costs from the matrix in 2.4.

The cost calculation is made for every node of a CDST and used as a splitting criteria. It is also calculated the gain that is obtained with each split as the decrease of the total cost of the algorithm. To measure impurity it is calculated the total cost considering that all samples from a leaf are classified as positive and, in other calculation, as negative, choosing the minimal of the two. Then, it is calculated the gain obtained with the splitting rules.

The tree gets fully grown and pruned following the pruning criteria: $PC_c = Cost(f(S)) - Cost(f^*(S))$, considering that f^* is the tree without the pruned node. [BAO15]

2.5.3 Ranking methods

Ranking bases its approach by comparisons between samples to predict if the sample x_i is "preferred" to the sample x_j, denoted by $x_i \succ x_j$. There are different types of ranking: Pointwise: in which the samples are trained individually and a score function determines their relevance; Listwise: the training loss function has as basis all documents and scores; and Pairwise: in which each sample is compared to all of the remaining and, if one is preferred, a scoring ranker is trained, denoted by f, being a pairwise scoring ranker if $x_i \succ x_j$ implies $f(x_i) > f(x_j)$, or pairwise non-scoring ranker if f is the one that decides which of the samples is preferred.

The pairwise rankers are trained assuring that, for a binary classification, having two samples (x_i, x_j) and the class labels (y_i, y_j), a trasformation f is applied so that $x_i \succ x_j$ if $P(y_i = 1) > P(y_i = 1)$ and $x_i \prec x_j$, in the opposite scenario. By convention, 1 is considered as being the minority class.

A recent study, [CFCP16], focused on pairwise rankers and achieved interesting results. The study focused on three pairwise scoring rankers:

- RankSVM:
  - Pre-processing: In this step, the dataset X is transformed to a new one X' such that for every pair (i, j), the new dataset will include $x'_{ij} = x_i - x_j$, considering that $y_i \neq y_j$, with $y'_{ij} = y_i$.
  - Training: A linear SVM is trained where the decision rule $w \cdot (x_i - x_j) > 0$ is viewed as the scoring function that determines $s(x_i) > s(x_j)$. The loss function is the hinge loss.
- RankBoost:
  - Pre-processing: In this step the dataset becomes X' in which, for all combination (i, j) and symmetric, $X'_{ij} = x_i, y'_{ij}$, such that y'_{ij} represents the preference relation between the samples, being $y'_{ij} = y_i$, omitting all the relation $y_i = y_j$.
  - Training: Performs training on several base estimators, one at each iteration t, using a distribution of weights for each pair, considering $D_{ij} = D_{ji}$. It is similar to AdaBoost and the loss function is the exponential loss.
- RankNet:
  - Pre-processing: In this step the dataset becomes X' in which, for all combination (i, j) and symmetric, $X'_{ij} = x_i, y'_{ij}$, where y'_{ij} represents the preference relation between the

samples. This ranker, estimates y'_{ij} by: $y'_{ij} = 0, \quad if y_i < y_j$ or $y'_{ij} = 1, \quad if y_i > y_j$ or $y'_{ij} = 0.5, \quad if y_i = y_j$.

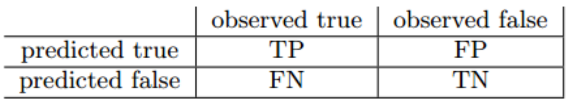
– Training: Uses a neural network to estimate y'_{ij} explained before. The loss function is the logistics loss.

These methods show training times superior to those of ordinary classifiers due to the pre-processing that occurs in data. The rankers produce a ranking score that is used to make the prediction of classes using a threshold T. The authors maximized the f1-score, appropriate for class imbalanced scenarios. After scoring and ordering by score the training data through $s_i = f(x_i)$ the midpoints are selected, expressed as $s'_i = \frac{s_i + s_i + 1}{2}$. The midpoints are possible thresholds T. The final T is chosen as being $T = \arg\max_{s'_i} F_1(s'_i)$.

This approach was studied due to the interesting results obtained in the study [CFCP16], that showed that these methods could perform better than their counterparts from the literature. The implementation of these methods were kindly made available by the authors of the study.

2.6 Evaluating the performance of a classification model

To evaluate the performance of a model developed for a binary classification problem it is needed to evaluate the correctness of the classification it produces. This correctness can be evaluated by calculating the number of correctly classified observations (called true positives (TP)), calculate the number of correctly classified observation that don't belong to the class in focues (called true negatives (TN)), the number of observations that were incorrectly classified has being from the class in focus (called false positives (FP)) or that were misclassified as not belonging to the class in focus (false negatives (FN)). This leads to the definition of the confusion matrix, a matrix that represents this information as in figure 2.5.



	observed true	observed false
predicted true	TP	FP
predicted false	FN	TN

Figure 2.5: Confusion Matrix, [BHG12]

- TP - true positive - observation that was classified as being part of the high risk for falls class when it really is part of the high risk for falls class, as example.
- FP - false positive - observation that was classified as being part of the high risk for falls class when actually it is not part of the high risk for falls class.
- TN - true negative - observation that was classified as not being part of the high risk for falls class when actually is not part of the high risk for falls class.
- FN - false negative - observation that was classified as not being part of the high risk for falls class when actually it is part of high risk for falls class.

Some of the well-known measures that are good for evaluating binary-class problems (like the one presented in this work) are the receiver operating characteristic area under the curve (ROC AUC), since it measures the ability of the classifier to avoid false classification, the F_1-score, that translates the relation between the data labeled as positive and the positives given by the classifier, accuracy (ACC), that represents the overall effectiveness of the classifier, precision (PRE), that translates the agreement between the data labels and the positive labels given by the classifier, and recall (REC), that represents how effectively the classifier can identify negative observations. [GFLH09][SL09].

The performance measures can be calculated as follows:

- Accuracy - Overall effectiveness - $\frac{TP+TN}{TP+FN+FP+TN}$
- Precision - percentage that was classified as being from *class*1 and actually are - $\frac{TP}{TP+FP}$
- Recall (Sensitivity) - percentage of *class*1 observations that were classified as being so - $\frac{TP}{TP+FN}$
- F_1-score - combination of precision and recall - $2\frac{precision.recall}{precision+recall}$
- ROC AUC - capacity to avoid false classification - $\frac{1}{2}\left(\frac{TP}{TP+FN}+\frac{TN}{TN+FP}\right)$

These performance measures can be calculated for the models developed with the proposed algorithms. In the final phase of the work the algorithms can be compared, having as basis the performance measures obtained on their evaluation. Conclusions can be assessed on which one of them has greater capacity to make the classification.

2.7 Tools for machine learning

Following is presented a list of tools that can be used to build machine learning models.

2.7.1 Weka

The study [LTDRdS14] used the Weka[2], an open source data mining toolkit. It is targeted for Java, making available machine learning algorithms for regression, classification or clustering. It also has available methods to process and present the data.

2.7.2 Rapidminer

The Rapidminer platform is another known tool used for machine learning [3]. The main advantage of this tool is that it has an easy to use graphical interface that allows the interation and building of Machine Learning models in a drag-and-drop mechanism.

[2]http://www.cs.waikato.ac.nz/ml/weka/
[3]https://rapidminer.com/

2.7.3 R

The R language[4] is largely used for statistical computing and is also suitable for approaching a Machine Learning problem, allied to machine learning libraries, like the caret library[5], that incorporates machine learning algorithms and methods that can be used to build predictive models.

2.7.4 Python

Python is another programming language that can be used for Machine Learning if allied to external libraries. One of its well-known Machine Learning libraries is the scikit-learn library[6] that incorporates the easiness and flexibility of Python with hundreds of algorithms and methods that cover most of the fields from Machine Learning. It is doted with an impressive documentation[7] that, more than presenting what the methods of the library do, also explains in detail the referenced concepts of Machine Learning.

The work was developed using Python, in combination with the scikit-learn library, that makes available the base methods, combined with the imbalanced-learn library[8] that makes available the methods to balance the dataset, the costcla library[9] that makes available methods of classification based on costs, the keras library[10] that allows the implementation of neural networks and convolutional neural networks (suitable to run on CPU or GPU), and for the ranking methods, as already described, the methods developed by the authors of the study were kindly made available for use in this work.

All of the libraries used followed the scikit-learn library structure and implementation of methods, which eased their application and combination. The flexibility and easiness of combination of the methods of these libraries were of extreme importance due to the large number of combinations tested.

[4]`https://www.r-project.org/`
[5]`http://topepo.github.io/caret/index.html`
[6]`http://scikit-learn.org/stable/index.html`
[7]`http://scikit-learn.org/stable/user_guide.html`
[8]`http://contrib.scikit-learn.org/imbalanced-learn/api.html`
[9]`http://albahnsen.com/CostSensitiveClassification/`
[10]`https://keras.io/`

Chapter 3

The Dataset

The main focus of this chapter is the description of the dataset that was used for building the Machine Learning classification models.

At first, it will be described how and what it represents, and later on some of its main characteristics, related issues and how the dataset was preprocessed before the phase of building the Machine Learning models.

3.1 Samples

In order to understand if and how it is possible to distinguish between the persons that probably will fall in the future from the ones who won't, the dataset used to train the models must include information capable of describing characteristics of the two groups.

Fraunhofer Portugal AICOS in partnership with ESTeSC - Coimbra Health School, collected data from 467 persons, in various environments: from people in the community, in day-care centers and at nursing homes. All of them were voluntary participants with an informed consent. The collected information included personal data, health related data, information on previous number of falls and information collected on the execution of physical tests, described later, with the help of sensors. [SMT+16]

3.2 Classification

From the initial collection of participants two groups were created. One group is characterized as being the group of participants that, in a period of six months after the initial collection of information, hadn't fallen (being so in low risk for falls, may be mentioned as class 0) and the other group is characterized as being the group of participants that had fallen in a period of six months after the initial collection of information (being so in high risk for falls, may be mentioned as class 1).

In order to have this kind of information, Fraunhofer and ESTeSC underwent a follow-up process, in which the participants were accompanied, asked if they had fallen in the follow-up time period. After six months of the initial collection of information it was known the falling status of 292 participants. These 292 participants are acceptable for the Machine Learning models, since the remaining 175 (from the initial 467) are unusable for the learning process (not having a known, real, classification, needed in supervised learning models).

The dataset constructed, with information of the participants, must have diverse and rich enough information so it would be possible to understand if a person's representation is closer to the representations from the group with high risk for falls or from the group with low risk for falls. This is what the Machine Learning models tried to learn. The objective was that later, after a successful learning phase, the models were capable of, through the input of new information of unseen, participants, predict if they would probably fall in a near future.

3.3 Sample characteristics

The dataset, as mentioned before, is composed of information of 292 participants and the following analysis focuses on these participants.

One participant is described by its personal information, being: participant's age, gender, height, weight, number of intaking medications (not being discriminated), number of health conditions (not being discriminated), usage of walking aids and metrics evaluated in physical tests (explained later).

The mean age of the participants is 71,67 years old, being the youngest participant 50 years old and the oldest 95 years old. The majority of the participants (75%) are younger than 81 years old and the most frequent age is 71 (15 participants).

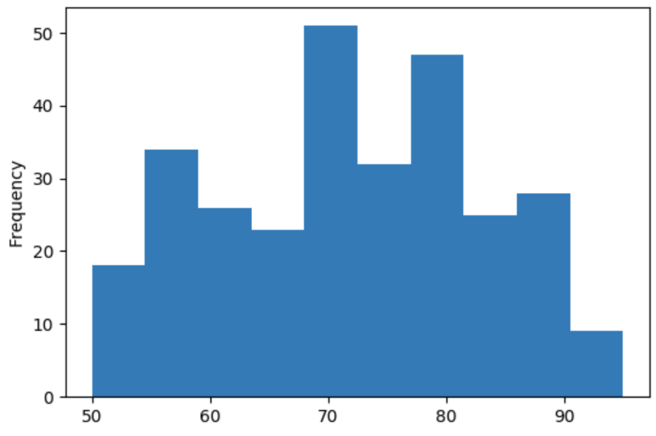


Figure 3.1: Age histogram

From the total number of participants, the majority, 195 (66,8%), are women and 97 (33,2%) are men.

The mean height of the participants is 159,83cm high, being the tallest participant 178cm and the shortest 129cm. The median height is 160cm which is also the most common height, characteristic of 37 participants. The mean weight of the participants is 72,18kg, being the heaviest 110kg and the lightest 39kg. The most common weight among the participants is 70,0kg (22 participants) and the median is 72,0kg.

The presence of health conditions may have impact on one's risk for falling. For this dataset, 149 participants (51,0%) don't have any health condition, 92 (31,5%) have one health condition, 40 (13,7%) have two health conditions and 11 (3,8%) have three. None of these health conditions is discriminated in the dataset, being only the total number of health conditions present. The use of a walking aid is also relevant for the risk for falling since needing to use one may indicate an abnormal gait or disability, for example. From the total number of participants, 241 (82,5%) don't need a walking aid, while 51 (17,5%) do.

3.3.1 Physical tests evaluated

More than the personal information described in the previous section, the dataset includes results and statistics extracted from the sensors used on the execution of the tests. Some of the tests were instrumented with an inertial sensor developed and assembled at Fraunhofer AICOS. For example, the TUG test was conducted wearing the sensor at the lower back. The information was obtained using its 3-axial accelerometer and 3-axis gyroscope.

Other tests were conducted using a pressure platform, the PhysioSensing platform (Sensing Future Technilogies, Lda). The raw data from the sensors was already processed at the time of execution of this work. [SMT+16]

Following is a description of the tests executed and the information that is stored in the database for each one of them.

3.3.1.1 The Timed Up and Go

This test consists in measuring the time one participant needs to get up from a chair, walk three meters in the fastest speed possible, turn around and walk back to sit again in the chair. The time is measured in seconds and studies show that taking more than ten seconds to conclude the test may indicate that the executer has a high risk for falling. [WLC05] [BFA+11] [MSS+16]

A set of examples of values of features extracted with the help of the sensors from this test, represented on the database, is shown in table 3.1.

For this test, 79 features are stored for each participant. One of those features is the time the participant needed to complete the test, the usual measure that is used to evaluate a person on this test.

Table 3.1: Example values of features from the TUG test stored in the database for two different participants.

...	tug_stand_mean	tug_stand_median	tug_stand_max	tug_stand_min	...
...	9.30	9.24	13.39	8.28	...
...	9.60	9.73	10.69	8.47	...

3.3.1.2 30 Seconds Sit-to-Stand test, STS

In the 30 Seconds Sit-to-Stand (STS) test the participant is asked to execute as many as possible full stand up positions beginning from a sited position. The number of repetitions made is counted on a 30 seconds period.

Some of the conditions for the execution of this test indicate that the participant must be sited in the middle of the chair with arms crossed at the chest and with its feet placed on the floor, hip-width apart from each other. The usual score for this test is the number of STS cycles, which is the sum of sit-to-stand and stand-to-sit cycles. [MSS+16] [CBKH12]

In table 3.2 is presented an example set of values of features that were extracted with the help of sensors and are shown as they are stored in the database used.

Table 3.2: Example values of features from the 30 Seconds Sit-to-Stand test stored in the database for two different participants

...	sts_rms	sts_stdDev	sts_medianDev	sts_iqr	...
...	9.33	1.08	1.08	0.46	...
...	9.28	0.61	0.61	0.43	...

For this test, 22 features are stored for each one of the participants. These group of features includes also the number of transitions that the participant was able to complete, the usual score measure for this test.

3.3.1.3 10 meter walk test

In this test it is measured the performance of one participant while he goes through a 10 meter walk. This is done at the fastest pace possible and the participant may take 5 meters to accelerate and 5 meters to decelerate. [PFK13] [MSS+16]

Some values of features are presented in table 3.3 as they are stored in the database to serve as example of what is stored for each participant.

Table 3.3: Example values of features from the 10 meter walk test stored in the database for two different participants

...	walk10m_maxAVG	walk10m_peakHeight	walk10m_avgPeakHeight	walk10m_meanCrossCount	...
...	15.17	15.00	9.10	131	...
...	13.45	10.14	6.91	133	...

For this test, 23 features are stored in the database for each participant. One of the stored features for this test is the time one takes to complete the test. [PFK13]

3.3.1.4 Step test

This test measures the performance of one participant in executing steps. To execute a step is to place one foot onto a step and then back on the floor, as fast as possible, counting the number of correct repetitions for 15 seconds. A score of less than 10 steps in 15 seconds implies high risk for falls. [MSS+16]

Some values of features are shown in table 3.4 as an example of what is stored in the database for this test for each one of the participants.

Table 3.4: Example values of features from the Step test stored in the database for two different participants

...	stepTest_fft_max_amp	stepTest_energy	stepTest_entropy	stepTest_skewness	...
...	0.05	89.66	9.53	0.46	...
...	0.03	89.28	9.53	- 0.38	...

For this test 20 values of features are stored in the database for each one of the participants. In this set of features is also included the number of steps that the participant was able to execute in the 15 seconds period.

3.3.1.5 4 Stage Balance test "modified"

In the execution of this test the participants were asked to maintain four foot positions of balance (side by side stance, semi-tandem stance, tandem stance and unipedal stance) for 10 seconds without support, neither being able to change their position. From these four positions, the first three were executed once with eyes open and then with eyes closed. So, for this test there were 7 different scenarios of test results (side by side stance with eyes open, side by side stance with eyes closed, semi-tandem stance with eyes open, semi-tandem stance with eyes closed, tandem stance with eyes open, tandem stance with eyes closed, unipedal stance with eyes open).

The participants were instructed to stand quietly on a pressure platform in order to take some plantar pressure distributions measurements from their performance. [SMT+16] [TOG+14]

Some values of features are presented in table 3.5 as example of what is stored in the database for each one of the participants.

Table 3.5: Example values of features from the 4 Stage Balance test stored in the database for two different participants

...	ex7_rmsX	ex7_rmsY	ex7_maxOscillationX	ex7_minOscillationX	...
...	14.11	25.18	1.46	-1.34	...
...	31.28	27.66	0.36	-2.03	...

For this test, 119 feature values are stored for each one of the participants, that corresponds to a set of features extracted from the center of pressure for each one of the seven tests. The last stage reached by the participant is also stored as one of the features.

3.3.1.6 X-Reach test

With this test it is possible to measure the capacity and stability of a participant by requiring him to reach as far as possible from one central position: to the front left, the front right, back left and back right. [New01]

As done before, some examples of values for some features of this test are presented in table 3.6. The numeration of the 'ex' corresponds to different directions taken in the test execution.

Table 3.6: Example values of features from the Reach test stored in the database for two different participants

...	**x_test_ex1_areaEllipse**	**x_test_ex2_maxOscillation**	**x_test_ex2_minOscillation**	**x_test_ex2_meanCOPx**	...
...	0.16	0.22	0.0	16.97	...
...	0.13	0.24	0.0	19.60	...

For this test 68 features are stored for each participant.

3.3.1.7 Grip Strength test

This test is used to measure the strength that the participant has and uses to grab objects. [RDM+11]

It is only stored one feature for this test for each participant that represents the strength of the participant in kg. An example of what is stored in the database is presented in table 3.7.

Table 3.7: Example values of the feature from the Grip Strength test stored in the database for two different participants

...	**gripStrength**	...
...	14.0	...
...	18.0	...

This was the last test from the test suite. From the description of the various tests feature values that are included in the database it is possible to conclude that there is a total of 340 features for each one of the participants. Its relevant to say that the number of features per participant surpasses the total number of eligible participants (292 participants). It is also observable that the different features have different scales and that some may even assume negative values.

3.4 Importance of features

Different features have different impacts in the classification, some may have a greater importance due to their contribution in distinguishing between the possible set of classes. For example (completely supposed example): It is probably intuitive to think that the fact that one person needs a walking aid to walk may be more relevant to the definition of their risk of falling than the fact that their height is 164cm.

The different values that one feature may assume may make the decision boundaries of the classification clearer, being so more important to the learning process than other features. This importance is measurable. In an effort to understand the importance of the different features present in this dataset an algorithm was applied, the Extra Trees Classifier implemented in the scikit-learn [1] library, following the methodology proposed on [GEW06]. A forest of 250 trees was grown and the importance of the different features averaged. In figure 3.2 are showed, as example, the importance of the 25 most important features of the dataset, as determined by the algorithm.

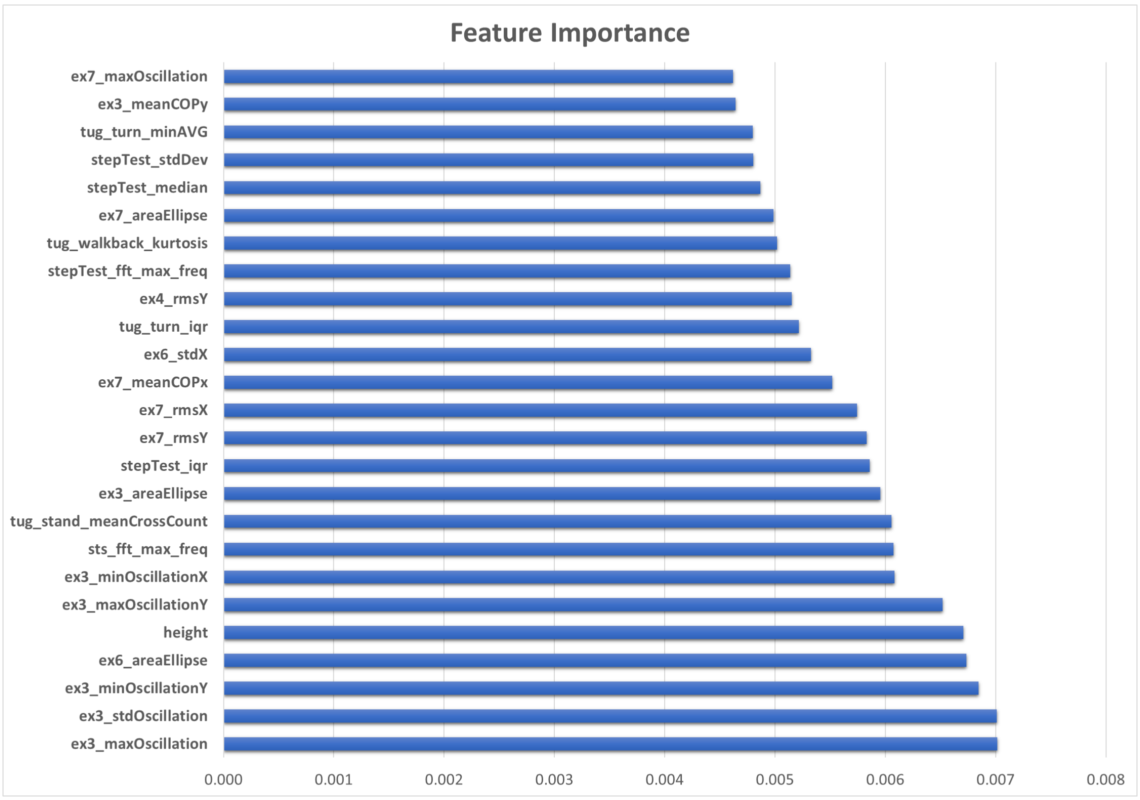


Figure 3.2: Graph with the importance value of the 25 most important features

As shown in the graph, the height feature plays a more important role than the use of an walking aid that doesn't even appear in the top 25 most important features as suggested by the application of the algorithm.

3.4.1 Statistical differences in groups

There should exist some difference between the samples of the two classes in order for the Machine Learning models to learn the classification mechanism with this dataset. Therefore, a sta-

[1] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

tistical analysis was conducted to check if statistical significant difference exists. As presented in [Rux06], the Welch's t-test can be used to, having as basis samples from two groups, check if their central tendencies are different. Assuming an alpha level of 0.05, if the *p*-value calculated with the Welch's t-test for a feature is lower than alpha (0.05) it is possible to reject the null hypothesis that the value of a certain feature has equal means for both populations. There were 250 samples from the class 0 (did not fall in the six months following the evaluation) and 42 samples from the class 1 (have fallen in the six months after the evaluation). The two groups show unequal sample sizes and variances for the feature values which led to the application of the Welch's t-test in alternative to the Student's t-test. The process of analysis went as follows:

- Division of samples from the database in two groups (manual division by classes)
- Statistical analysis of the values for each feature for the two groups
- Application of the Welch's t-test
- Understanding which of the features show statistical differences between the two groups of samples.

An example of the analysis on the feature ex1_meanCOPx and on the feature ex2_maxOscillationX are presented in table 3.8.

Table 3.8: Example of the statistical analysis made on the values of the feature ex1_meanCOPx and ex2_maxOcillationX for the class 0 and for the class 1

Feature	**Group of Class 0**	**Group of Class 1**	**t-value**	**p-value**	**Conclusion**
ex1_meanCOPx	Mean: 19.196	Mean: 18.870	0.966	0.338	p-value > 0.05 Differences are not statistically significant
	Standard Deviation: 2.074	Standard Deviation: 1.950			
	Minimum: 11.43	Minimum: 13.985			
	5-th percentile: 15.913	5-th percentile: 15.513			
	Q1: 18.051	Q1: 17.814			
	Median: 19.287	Median: 19.032			
	Q3: 20.391	Q3: 20.175			
	95-th percentile: 22.564	95-th percentile: 21.599			
	Maximum: 26.051	Maximum: 22.203			
ex2_maxOscillationX	Mean: 0.332	Mean: 0.259	2.117	0.035	p-value < 0.05 Differences are statistically significant
	Standard Deviation: 0.427	Standard Deviation: 0.133			
	Minimum: -1	Minimum: 0.06			
	5-th percentile: 0.07	5-th percentile: 0.07			
	Q1: 0.147	Q1: 0.172			
	Median: 0.235	Median: 0.25			
	Q3: 0.41	Q3: 0.33			
	95-th percentile: 0.845	95-th percentile: 0.477			
	Maximum: 4.82	Maximum: 0.7			

From a space of 340 features, 125 (37%) showed statistically significant differences between the two groups, the group of people that fallen and the ones that didn't.

3.4.2 Relevant Issues

When building Machine Learning models one must follow successive tasks. The first task is understanding the dataset that will be fed into the learning process, understanding the representation of its feature values and the problems that it may carry. Next follows a task of pre-processing the dataset, where the problems revealed in the first step are overcomed and the dataset is prepared for the construction of the Machine Learning models.

The issues that the dataset carries may make it impossible for the model to complete the learning process or make the classification boundaries unclear. This is common when dealing with real world data such as the one presented in this work. This section serves to present the faced issues relative to the dataset used and how they were surpassed.

3.4.2.1 Missing Values

Some datasets may not contain all the information for each one of the samples that it describes. This may arise for many reasons:

- Non-existent information
- Inability to collect the information
- Errors in information collection that led to missing values
- Other causes

The inexistence of information may induce bias in the learning process or even prevent it from occurring at all depending on the adopted model.

For the presented dataset in particular, there are some values that are absent since the participant was unable to conclude the test. For example, if the participant was unable to complete the TUG test, then the feature that represents the time taken by the participant to conclude the test is empty in the database. There are some ways to deal with missing values, and, as in the literature, there is no single approach that appears to rule over the others [JMGL+10]. Some approaches could be:

- Removing the features or the samples that show missing values:
  - As mentioned before, there are 340 feature values for each one of the 292 participants. This means that there are 99 280 (292*340) feature values stored in the database. From these, there are 10 216 values missing. These missing values are relative to

many features and occur for various samples. Eliminating completely the features that show missing values for some of the samples can have severe impact in the learning process since 332 from 340 features would be eliminated for all the samples.

One sample that has only one missing value for a specific feature may not affect greatly the learning process, but one sample that shows a large portion of missing features, for example, appears as having missing values for 75% of the features, may be inconclusive for the learning process or may induce the introduction of bias if not treated the right way. Because of this, every sample that showed more than 50% of missing values of features (170 missing values) was removed from the presented dataset by being considered inconclusive. This reflected the removal of 12 samples from the dataset. All of the removed samples were from the class that didn't fall in the follow-up period (class 0).

- Replacing the missing values by a meaningful value:
  - Missing values are valuable in this scenario since they reflect the lack of capacity of the participant to conclude the test and this may reflect deficient functional mechanisms and an increased tendency to fall, although all the samples that were removed due to having too much missing values didn't correspond to the class that did fall.
    One missing value can be replaced by a specific value outside the range of the feature, for example by -1, but, since all the features have different ranges in this scenario, some being negative, it could lead to the introduction of wrong information.
    One possible solution is substituting by the worst possible performance on the test (since missing indicates inability to conclude the test).

Table 3.9: Relation between scores obtained in the tests and the risk for falling

Test	**Score**	**Risk for falling**
Timed Up and Go	+ Time	+ Risk
10m walk	+ Time	+ Risk
30 Seconds Sit to Stand	- Repetitions	+ Risk
4 Stage Balance	- Positions	+ Risk
Step Test	- Steps	+ Risk
Grip Strength	- kg	+ Risk
X-Reach Test	- cm	+ Risk

Table 3.9 shows the relation between the magnitude of a test result with the risk for falling. It should be interpreted for the 30 Seconds Sit to Stand test, as example: "If the participant has done few repetitions of the exercise then it probably has a greater risk for falling. The smaller the number of repetitions, the greater the risk". Some of these scores are represented in the database as features for each one of the samples and some of these scores are not present for some samples, being considered missing values.

The missing values for these scores are substituted by the worst possible result, for example, if a participant has as missing value the score on the 30 Seconds Sit to Stand test then it gets substituted by the worst possible result, 0 repetitions. The missing values for other features that represent metrics extracted from the sensors for which the worst possible result is not known got substituted by the mean value of the feature for all the samples in the dataset. This is done in order to introduce the minimum possible bias in the learning process.

3.4.2.2 Outlier observations

There is a set of normal patterns that is present in groups of data. A dataset used in Machine Learning models show these normal patterns but some samples may be composed of characteristics that step outside this normal pattern zone.

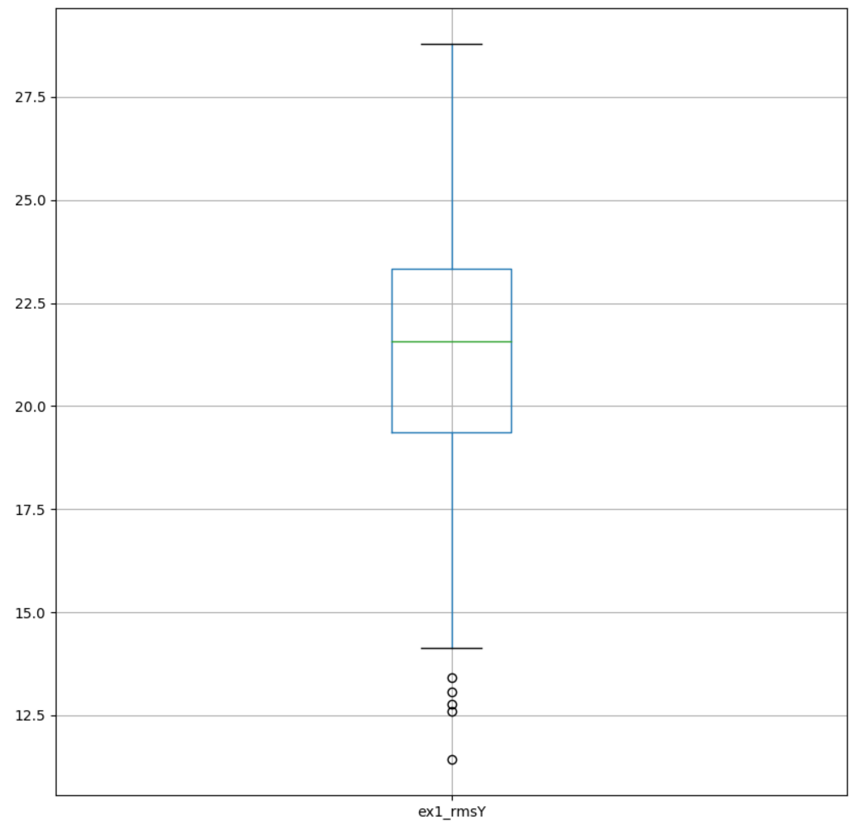


Figure 3.3: Boxplot of the values assumed by all the samples on the feature rmsY from one of the 4 Stage Balance test positions

In figure 3.3 is an example of outlier values for the feature ex1_rmsY that some samples assume in this dataset. They are represented by circles in the boxplot. The green line represents the median and the outer lines represent the distribution in quartiles. If one sample is characterized by many outlier values for its features then it may be considered as being an outlier.

This group of outlier samples in a dataset may prejudice the learning process. A method was applied in order to remove these outliers from the dataset, the Isolation Forest algorithm. This algorithm, as described in [LTZ08], is capable of understanding that the anomalies are fewer instances and that their values differ greatly from the considered normal instances, being isolated from them. The method shows advantages such as:

- It doesn't utilize distance or density measures and so doesn't have major computational costs
- Showing linear time complexity and low memory consumption
- Being capable of handling high-dimensional problems which is the scenario of this problem

The model takes two arguments, the number of trees in the ensemble and the size of the sub-sampling. The algorithm builds an ensemble of binary trees, being the structure of each one of them defined through recursive partitioning. In each partitioning step, consisting on the selection of a random attribute and a random p value for that attribute, the samples get divided in two groups (one that shows a p value greater than the chosen and another group that shows a p value smaller than the chosen one). From this division the tree grows on new branches.

The process of growth is affected by a stop condition, for example an height limit for the tree. If a sample is in a node near the root it means that it is more deviated from the other samples, in other words, it is more isolated and was easier to separate.

This process of defining outliers goes as finding the samples that are closer to the root nodes averaging the ensemble of trees. [LTZ08]

A forest of 100 Isolation Trees was grown in order to find the outliers of the presented dataset.

In figure 3.4 is presented, as example, a tree from the ensemble of trees obtained with this method. It was randomly chosen the feature tug_time_to_stand as the first attribute at the random split value of 0.6235. The samples that assumed a value smaller than the chosen one got on the left child node of the root node (the process continued in this branch) and only one sample got a superior value. The one that got a superior value stayed in the condition of the right child node of the root node. It is possible to conclude that it got isolated. If this sample appears in many trees of the ensemble, so close to the root node, it may be chosen as an outlier sample.

The application of this method led to the discovery of 19 outlier samples that were removed from the dataset used for training. From these 19 outliers, 15 corresponded to samples from the class 0 (didn't fall in the follow-up period) and 4 to samples from the class 1 (fallen in the follow-up period).

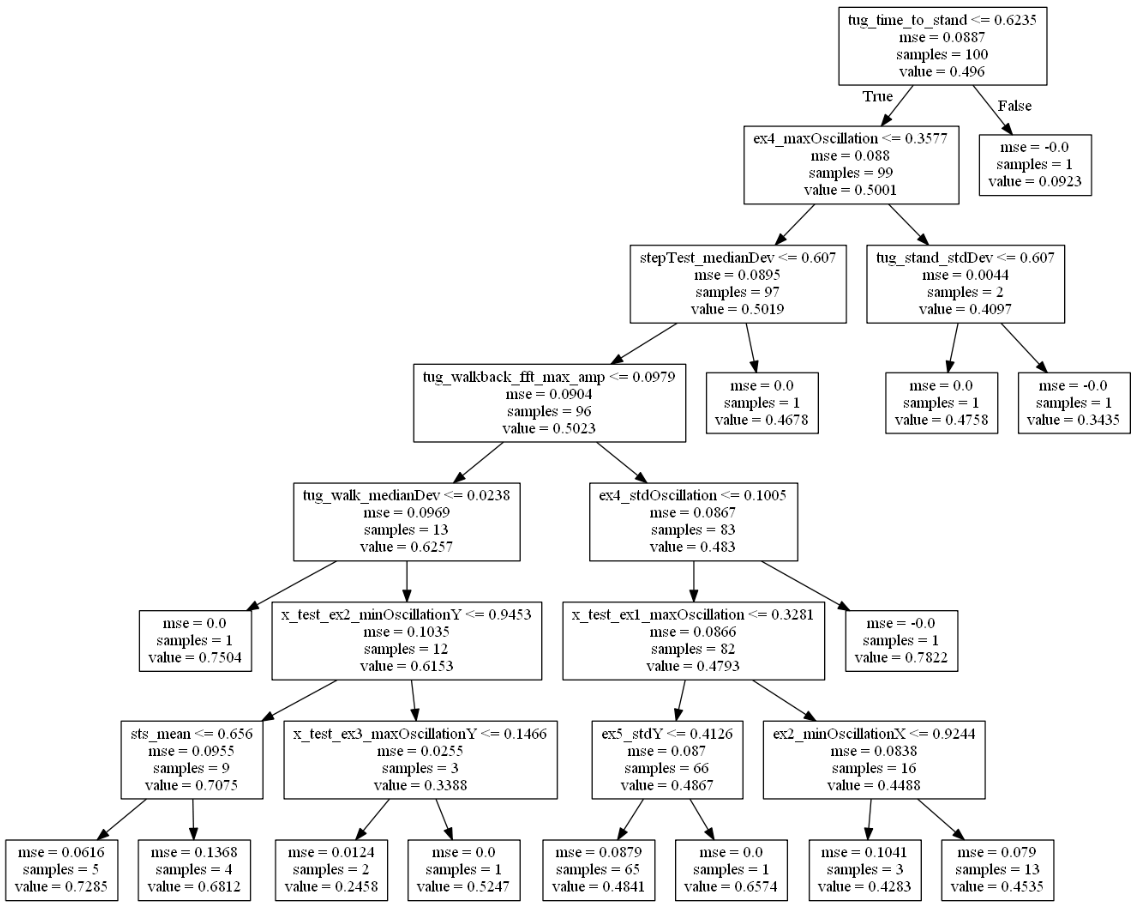


Figure 3.4: Example of a tree present in the ensemble

3.4.2.3 Excess of features and correlation

One of the problems that datasets may present is the excessive number of features that they gather. Some of the features may be redundant or even irrelevant and prejudice the learning process.

While one algorithm may work fine in low dimensional input it may be unable to generalize when the input gets high-dimensional. In this scenario specifically there are 340 features for each one of the 292 participants. The number of features is greater than the number of samples themselves. An excessive number of features may not introduce new relevant information that enrich the ability of differentiating the classes and may actually introduce values of noise in the learning process and prevent it from occur correctly [Dom12].

There are some methods to choose the best features for the problem at hands, as already described, many have been applied to the dataset, as described later.

There is a large amount of features that are correlated with each other. While this fact may not prejudice the learning process, a feature that is highly correlated with another probably does not add new valuable information that is already expressed by the former. Being so, they can be removed in order to save computational effort and resources.

It was used the pandas profiling[2] library that uses as basis the method from the pandas library[3] to calculate correlation between features. This method calculates the Pearson's correlation coefficient [4], that reveals the existence of statistical linear correlation between every two features of the dataset. The value of linear correlation may vary from -1 to 1, where 1 represents total linear correlation and -1 represents negative linear correlation. The method was used in order to select and eliminate the features that showed more than 0.9 in Pearson's correlation (highly correlated).

From the whole dataset with 340 features, 134 features (39%) show a Pearson correlation with other features above the threshold of 0.9. From the highly correlated features, 50 were from the 4 Stage Balance test (42% of the features from this test), nine were from the Step Test (45% of the features from this test), nine were from the 30 Seconds Sit-to-Stand test (41% of features from this test), 36 were from the Timed Up and Go test (46% of features from this test), 14 were from the 10 Meter Walk test (61% of features from this test) and 16 were from the X-Reach test (24% of features from this test).

3.4.2.4 Imbalanced Scenario

One last problem faced when working with the presented dataset was the fact that it is imbalanced. By imbalanced it should be understood that the dataset has more samples from one class (the majority class) relatively to the second class (minority class).

Most of the algorithms expect even class distributions. If they have to handle imbalanced data sets they may fail to understand the distributive characteristics of the data and so reach unfavorable results. This field research is a hot topic due to its major importance which reflects the big number of publications made in the last years [HG10]. The dataset used is highly imbalanced, presenting, after the removal of outliers and samples that had excessive missing values: 223 samples from class 0 (low risk for falls) and 38 samples from class 1 (high risk for falls). The approach to this issue is further described in a following chapter.

[2]https://pypi.python.org/pypi/pandas-profiling
[3]http://pandas.pydata.org
[4]http://libguides.library.kent.edu/SPSS/PearsonCorr

Chapter 4

Approaching the problem

As mentioned earlier, one of the objectives of this work was to analyze the applicability of different Machine Learning classifiers to the problem at hand, understanding which one could lead to better results by comparison of their scores on performance. Therefore, the work went as by trying the most combinations of methods possible with later comparison of their performance. For that, the tools, as described in 2.7, were used. Each test made followed the general workflow present in 4.1.

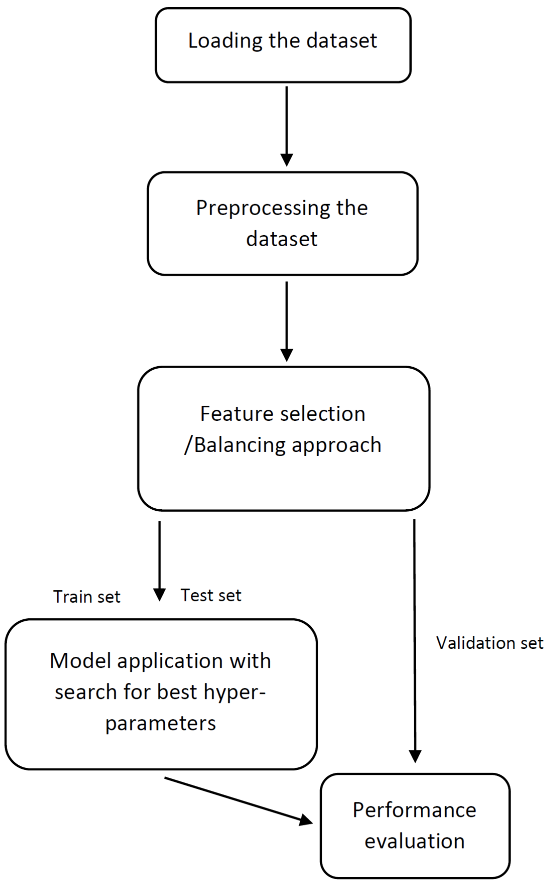

Loading the dataset
Preprocessing the dataset
Feature selection /Balancing approach
Train set
Test set
Validation set
Model application with search for best hyper-parameters
Performance evaluation

Figure 4.1: General workflow adopted

This chapter goes by first describing which tests were conducted, then describing each step of the workflow presented in 4.1 used in each test, ending by explaining specific characteristics of the application of each algorithm.

4.1 Process step-by-step

Trying to maximize the number of tests and combination of methods with the algorithms, several tests were made for each one of the applied algorithms: Decision Trees, Random Forests, K-Nearest Neighbors, Support Vector Machines, Naive Bayes, AdaBoost, Neural Networks, Convolutional Neural Networks, Cost Sensitive Random Forests, RankSVM, RankBoost, RankNet and Easy Ensemble. All of these algorithms were already described.

For each one of the algorithms the following tests were conducted to compare their performance:

- Test with no methods to balance the dataset or to choose features. One test.
- Tests with all methods described in the imbalanced section 2.5 with no other method. Seventeen tests in total for each algorithm.
- Tests with all feature selection methods described earlier in 2.4.1 with no other method. Twenty-five tests in total for each algorithm.
- Tests with the combination of the best performing method for balancing the dataset with all feature selection methods to check if performance is improved. Twenty-five tests in total for each algorithm.
- Test the combination of the best performing method for balancing the dataset with all sets of features manually selected (as described in 4.1.3). Twenty-six tests in total for each algorithm.

In figure 4.2 is presented the type of tests that were described in a graphical way. For each one of the tests, the steps described next were executed.

4.1.1 Step 1: Loading the dataset

Prior to the application of the classification algorithms, scripts that allowed data loading/parsing to an adequate structure from the database (csv files) to create the dataset were implemented and executed. The dataset was then loaded to a python dataframe structure with the help of the pandas library[1].

[1] http://pandas.pydata.org/

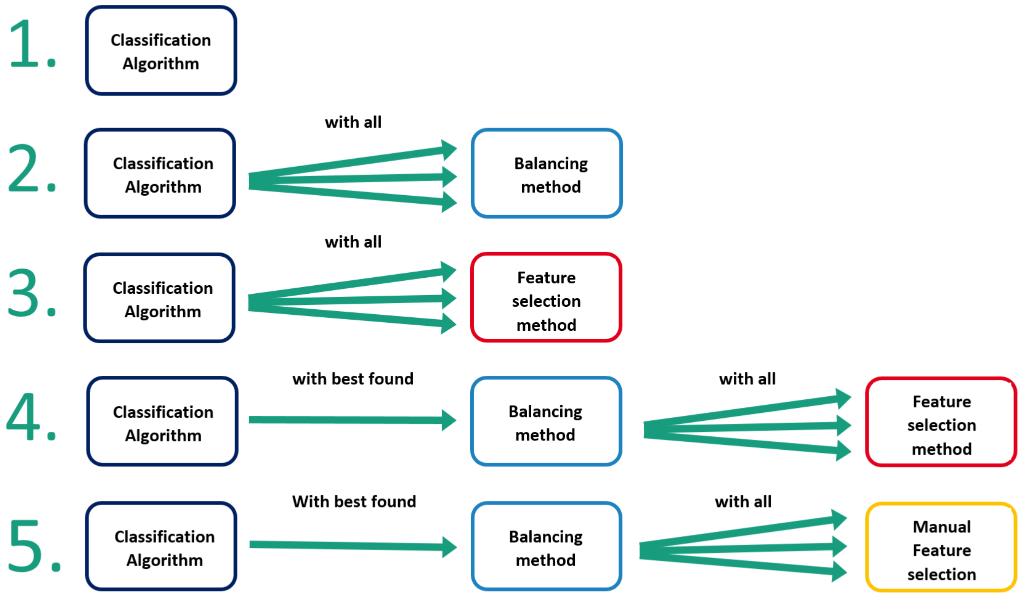


Figure 4.2: Type of tests executed.

4.1.2 Step 2: Preprocessing the dataset

After loading the dataset, it needed to be pre-processed in order to be accepted by the classification model. This step can be further divided in smaller sub-steps:

- Removing samples that present an excess of missing values and replacing the remaining missing ones: As already described in 3.4.2.1, this leads to the removal of 12 samples from the class that didn't fall, the missing values of the conventional scores which meaning is known got substituted by the worst possible result and the remaining missing values for metrics of the sensors which scales are unknown got substituted by the mean value of the dataset, to avoid bias.

- Removing highly correlated features: As already described in 3.4.2.3, applying this procedure to the whole dataset leads to the removal of 134 features as described earlier.

- Normalizing the dataset: It consists on making all feature values assume the same scale without losing information for the learning process. This favors the classifier work. This step is done with a scikit-learn method, as described in the documentation[2]. The method was set to normalize the dataset by making all features assume a norm equal to one as by the l2-norm definition. The l2-norm of a vector is calculated by squaring all the elements of the vector, summing the squared values and then calculating the square root of the sum.

These steps sum up the conducted pre-processing phase.

[2]http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html

4.1.3 Step 3: Feature selection / Balancing approach

At this point manual feature selection is applied (if the test includes manual feature selection) to determine, manually, which features go through the learning process. If other traditional feature selection methods were to be applied in a specific test, then the manual feature selection process would not occur. The execution of manual feature selection at this point emerged from having the possibility of understanding the impact of different physical tests (as described in 3.3.1) in the classification process, understanding which are more important and which are redundant. One of the manually selected sets includes only the features that showed to be statistically different (as described in 3.4.1) and other only the conventional scores of the tests that are traditionally used to classify possible fallers. The different sets that could be chosen by manual feature selection are described in table 4.1.

Before delving deeper on this step one must understand the concept of pipeline used by the scikit-learn library and used in these tests. As described in the documentation[3], a pipeline allows the definition of a sequence of steps, with a final estimator, a classifier for example. More than that, it allows the definition of different sets of parameters to be applied to the intermediate steps of the pipeline.

If the test at hands used a feature selection method then it was applied in this step as a starting point. It is independently applied in order to be applied to the validation set V, later described.

After the feature selection process occurs (either manual or with traditional methods), two steps occur:

- Division in sub-sets: The complete dataset is divided in Train and Test set, T, with 70% of the database entries randomly selected, and Validation set V, remaining 30% of database entries. This usually happens in order to save part of the cases that have not been used in the learning process of the classifier, in set V, to evaluate the performance of the model in new cases. The Train and Test set T will be divided for training and testing the model in a later step. To do this, a method from scikit-learn was used, as described in the documentation[4], that randomly divides arrays in train and test sets. The random seed is fixed in order to produce the same sets for all tests, to improve the performance comparison.

- Removal of outlier samples: As already described in 3.4.2.2, 19 outlier samples were found on the training set T and were removed. From these 19 outliers, 15 corresponded to samples from the class 0 (didn't fall in the follow-up period) and 4 to samples from the class 1 (fallen in the follow-up period). The set V is not searched if it includes outliers since it represents real-world information, that may include outliers.

After reducing the number of features and diving the main set, if a balancing method is used, it is inserted in a pipeline as an intermediate step.

[3]http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
[4]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Table 4.1: Obtained sets by manual feature selection

Set	**Features included**	**Set**	**Features Included**
1	Personal information as described in 3.3	15	Combination of set 1 and 8 (Personal information + Reach test)
2	Features of the STS test as described in 3.3.1.2	16	Features of all tests and personal information. Complete dataset
3	Features of the TUG test as described in 3.3.1.1	17	All but Reach test
4	Features of the 10MW test as described in 3.3.1.3	18	All but 4 Stage Balance test
5	Features of the Step test as described in 3.3.1.4	19	All but Grip Strength test
6	Features of the Grip Strength test as described in 3.3.1.7	20	All but Step test
7	Features of the 4 Stage Balance test as described in 3.3.1.5	21	All but 10MW test
8	Features of the X-Reach test as described in 3.3.1.6	22	All but TUG test
9	Combination of set 1 and 2 (Personal information + STS test)	23	All but STS test
10	Combination of set 1 and 3 (Personal information + TUG test)	24	All but personal information
11	Combination of set 1 and 4 (Personal information + 10MW test)	25	Features that showed to be statistically different as described in 3.4.1
12	Combination of set 1 and 5 (Personal information + Step test)	26	Traditional test scores
13	Combination of set 1 and 6 (Personal information + Grip Strength test)	27	Traditional test scores and personal information
14	Combination of set 1 and 7 (Personal information + 4 Stage Balance test)		

For example, in a test that uses the feature selection method Variance Threshold and, as balancing method, Random Under-Sampling, then the Variance Threshold is applied and the pipeline is initiated with an instance of the balancing method preceded by the step name. The imbalanced-learn library has its implementation of the pipeline allowing the introduction of balancing methods in a pipeline in scikit-learn style[5].

[5]http://contrib.scikit-learn.org/imbalanced-learn/generated/imblearn.pipeline.Pipeline.html

4.1.4 Step 4: Model application with hyper-parameters search

At this point the algorithm in testing is added to the pipeline as the final estimator. Adding to the previous example a random forest classifier leads to the pipeline:

```
estimators = [("random_undersampling", RandomUnderSampler())
        ,("rf",RandomForestClassifier())]
pipe = Pipeline(estimators)
```

Another interesting and valuable method available in the scikit-learn is the gridsearch method[6], that allows the choosing of the best hyper-parameters (the parameters used in the methods) through cross-validation to train the model. It also has the capability of receiving a pipeline and search for the best combination of parameters for all intermediate steps. It chooses the combination of parameters that produces the best performance of the model, according to a passed metric scoring function. This method tests all possible combinations of parameters and may take an excessive amount of time to complete the training process.

Another method, the RandomizedSearchCV[7], does the same operation, but instead of searching through all possible combination of parameters, it searches only for k combinations of parameters, reducing training times drastically and maintaining quality in the choosing process, as shown in the documentation[8]. The RandomizedSearchCV performs stratified cross-validation, which works by the same principle as cross-validation but divides the set maintaining the percentage of samples in each class (the set is already balanced when RandomizedSearchCV is applied so the sub-sets retain 50% of samples from each class).

Earlier was mentioned that the T set would be further divided in train and test sets. This happens with the application of the RandomizedSearchCV and its stratified cross-validation approach. To feed RandomizedSearchCV with possible parameters for the methods application, a grid with possible parameters must be created. After the definition of the possible set of parameters to be searched, the method is set to maximize the receiver operating characteristic area under the curve, ROC AUC, scoring metric. This metric is used because it has showed to be a good performance evaluator for binary-classification problems [SL09]. Then, the parameters that maximize the scoring metric are chosen and the model is finally trained. The parameters for the balancing methods are also defined and searched.

```
params_grid = dict(rf__n_estimators=[5, 10, 50, 150],
                   rf__criterion=['gini', 'entropy'],
                   rf__max_features=[None, 'auto', 'sqrt', 'log2'],
                   rf__max_depth=[None],
```

[6]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[7]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[8]http://scikit-learn.org/stable/auto_examples/model_selection/randomized_search.html

```
                 rf__min_samples_split=[2, 5, 10],
                 rf__min_samples_leaf=[1, 5],
                 rf__max_leaf_nodes=[None],
                 rf__min_impurity_split=[0.00000005, 0.0000001, 0.0000002,
                     0.0000003],
                 rf__bootstrap=[True],
                 rf__oob_score=[False],
                 rf__random_state=[None],
                 rf__warm_start=[False],
                 rf__class_weight=['balanced', 'balanced_subsample', None],
                 rf__n_jobs=[-1])

random_search=RandomizedSearchCV(estimator=pipe,param_distributions=params_grid,
    n_iter=2000,n_jobs=-1, cv=5,scoring='roc_auc')
random_search.fit(datafra.X_train, datafra.y_train)
```

Here is a basic example of a possible training of a Random Forest, after the application of the variance threshold method and random under-sampling, searching through 2000 combinations of parameters (defined in params_grid) using stratified cross validation with a k of five to choose the combination that maximizes the ROC AUC scoring metric with the RandomizedSearchCV method.

4.1.5 Step 5: Performance evaluation

After ending the training process the model is evaluated, making predictions on the validation set V that was put aside earlier in the process. By comparing the predicted classifications made by the model with the real classifications of the samples in the validation set V some performance metrics are calculated and stored in a file. The metrics stored are: accuracy, f1-score, precision, recall, ROC AUC and the confusion matrix that represents the number of TP, TN, FP and FN. It is also stored the best combination of parameters obtained by RandomizedSearchCV when suitable.

The validation set is composed of 93 samples, being 13 from the class 1 (high risk for falls) and 80 from the class 0 (low risk for falls). The validation set is unbalanced so the scores for precision, recall and f1 were made as weighted calculations, taking in account the different number of samples from each class.

4.1.6 Test characteristics by algorithm

In this subsection, parameters and specific deviations from the standard process defined previously are explained for the application of each one of the algorithms. The possible set of parameters passed to RandomizedSearchCV for each of the non-deviating algorithms is presented in A (do not forget that parameters for the balancing methods were also searched at the same time): for Decision Trees in A.0.1, for Random Forests in A.0.2, for K-Nearest Neighbors in A.0.3, for Support Vector Machines in A.0.4, for Naive Bayes in A.0.5, for AdaBoost in A.0.6 and for RankSVM in A.0.7.

4.1.6.1 Cost Sensitive Random Forests

The training of Cost Sensitive Random Forests is different because it receives as input a cost matrix to adapt the learning process, as described in 2.5.2.1.

Assuming that it is worse to have a participant that will fall being classified as won't fall than the opposite scenario, it is possible to conclude that the cost of getting False Negatives is bigger than the cost of getting False Positives. With this in mind, two versions of the cost matrix were created:

- One with relatively low costs for FP and FN as in 4.2.

Table 4.2: Cost matrix for version one of the test

	Actual Positive	**Actual Negative**
Predicted Positive	$C\_TP = 0$	$C\_FP = 1$
Predicted Negative	$C\_FN = 3$	$C\_TN = 0$

- Another with high costs for FP and FN as in 4.3.

Table 4.3: Cost matrix for version two of the test

	Actual Positive	**Actual Negative**
Predicted Positive	$C\_TP = 0$	$C\_FP = 25$
Predicted Negative	$C\_FN = 90$	$C\_TN = 0$

The possible set of parameters fed to the RandomizedSearchCV were:

```python
params_rf = dict(rf__n_estimators=[25],
                     rf__combination=['majority_voting'],
                     rf__max_features=[0.8],
                     rf__pruned=[True],
                     rf__n_jobs=[-1])
```

The training of this algorithm took a lot of time. The tests included the combination of all feature selection and balancing methods with the Cost Sensitive Random Forests.

4.1.6.2 RankNet

This version was not applicable in a pipeline that would also include the balancing methods so it was not inserted in one. Two versions of this algorithm were applied, one that included 10 hidden nodes, and other that included 50 hidden nodes to test different variations of internal node density. The tests included the combination of feature selection and balancing methods with the RankNet.

4.1.6.3 RankBoost

The same way, this algorithm was not included in a pipeline with the balancing method. It was tested a version that included 100 base estimators (Decision Trees). The tests included the combination of feature selection and balancing methods with RankBoost.

4.1.6.4 Neural Networks

As for the implementation of the Neural Network algorithm, it was used the keras library[9], and built a KerasClassifier of type Sequential that allowed the insertion of the neural network in the pipeline, which could be used for hiper-parameters search. The hyper-parameters used for the possible combinations were thought to allow flexibility, permitting that the hyper-parameters search would have influence in the architecture of the network itself. Therefore, the parameters for the network hyper-parameter search were the following:

```
size = [X_train.shape[1]]

params_grid = dict(nn__batch_size=[5,10,50,100]
                   ,nn__epochs=[50, 100]
                   ,nn__optimizer=['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam',
                       'Adamax', 'Nadam']
                   ,nn__init_mode=['uniform', 'lecun_uniform', 'normal', 'zero', '
                      glorot_normal', 'glorot_uniform','he_normal', 'he_uniform']
                   ,nn__activation1=['softmax', 'softplus', 'softsign', 'relu', '
                      tanh', 'sigmoid', 'hard_sigmoid', 'linear']
                   ,nn__activation2=['softmax', 'softplus', 'softsign', 'relu', '
                      tanh', 'sigmoid', 'hard_sigmoid','linear']
                   ,nn__activation3=['softmax', 'softplus', 'softsign', 'relu', '
                      tanh', 'sigmoid', 'hard_sigmoid','linear']
                   ,nn__dropout_rate=[0.1, 0.3, 0.5, 0.7, 0.9]
                   ,nn__weight_constraint=[1, 3, 5]
                   ,nn__size=size
                   ,nn__nr_layer=[1, 2, 3]
                   ,nn__nodes_hidden=[50, 25]
```

These parameters would affect the creation of the network through the following process of creating a keras sequential model[10]:

```
#Function to create the network architecture
def create_model_mlp(nr_layer=3, nodes_hidden=50, optimizer='adam', init_mode='
    uniform', activation1='relu',activation2='relu', activation3='relu',
    dropout_rate=0.0, weight_constraint=0, size=93):
    # It's a keras sequential model
```

[9]urlhttps://keras.io/scikit-learn-api/

[10]https://keras.io/getting-started/sequential-model-guide/

```
model = Sequential()
# Creating the inital layer with parameters from the hyper-parameter search
model.add(Dense(62,activation=activation1,kernel_initializer=init_mode,
    input_shape=(size,),kernel_constraint=maxnorm(weight_constraint)))
model.add(Dropout(dropout_rate))
# Adding the internal layers, being the number of layers also defined in the
    hyper-search
for i in range(1, nr_layer):
    model.add(Dense(nodes_hidden,activation=activation2,kernel_initializer=
        init_mode))
    model.add(Dropout(dropout_rate))
# Adding the final layer that ends the binary classification
model.add(Dense(1,activation=activation3,kernel_initializer=init_mode))
model.compile(optimizer=optimizer,loss='binary_crossentropy',metrics=['accuracy
    '])
```

The tests also included the combination of feature selection and balancing methods with the network.

4.1.6.5 Convolutional Neural Networks

It is needed a large amount of samples to train a Convolutional Neural Network from scratch. One possible approach to surpass this problem is through the method of Transfer Learning that consists in retraining a network that was already trained for other related problem. If adopted properly, it could diminish the amount of time taken to train the model while achieving good results.

The keras library[11] allows such application of the method[12]. It distributes some complex architectures of networks and allows to have them pre-trained, setting their weights as if they were trained on the ImageNet[13] database (a database with hundreds of samples, used to train CNN's in image classification). In this work the archictecture InceptionV3 was adopted since it is computational effective when compared with denser models while using less parameters [SVI+15].

This problem is not based in images, but it is possible to transform the available dataset into images, transforming each sample in a different image where different tonalities represent different characteristics. Each sample is scaled to show values between zero and one, and then reshaped in order to assume the representation of a matrix where a colormap[14], from the matplotlib library[15], with the attribute "gist_earth" is applied. The final result for a sample is presented in 4.3 as an example.

In the first step of the Transfer Learning process, to re-train the model, only the top layers from the InceptionV3 are retrained, adapting to the new images that it received. The convolutional layers were frozen during the process. The bottom convolutional layers of the model got retrained

[11] https://keras.io/
[12] https://keras.io/applications/
[13] http://www.image-net.org/
[14] https://matplotlib.org/api/colors_api.html
[15] https://matplotlib.org/

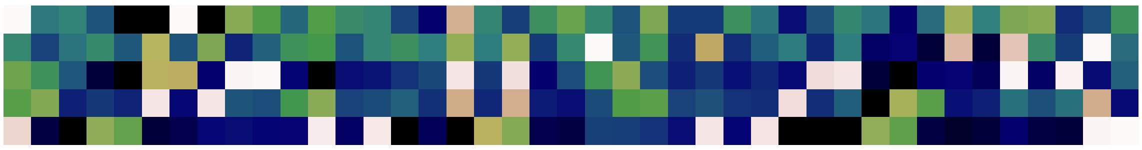


Figure 4.3: Transformation from one sample to image for the convolutional neural network application.

in the same way (freezing the top 249 layers). The final layer was also adapted to have a binary output. This way the model is retrained but maintaining the weights associated with the internal convolutional layers, trained with the ImageNet database.

4.1.6.6 Easy Ensemble

The Easy Ensemble implementation from the imbalanced-learn library[16] didn't allow its combination with the defined process pipeline. Therefore, a method, based on [LWZ06] was developed. After choosing the number of sub-sets to use, denoted as *n*, the method executes the following steps (after applying the methods of feature selection and balancing):

- For *n*:
  - A base classifier, that can be chosen through a parameter, is created. A decision tree for example.
  - An instance of Random Under-sampling to under-sample the training set, is created.
  - Both are inserted in a pipeline.
- The pipelines are used for the construction of a VotingClassifier[17]. This method implies a voting scheme for its estimators, allowing the group to decide the classification.
- The VottingClassifier gets trained with the training set.
  - Each sub-classifier gets fitted on a random under-sampled sub-set from the main training set.
- The final classification comes from the majority voting rule, that determines that the final classification is chosen because it was the most voted by the sub-classifiers.

[16]`http://contrib.scikit-learn.org/imbalanced-learn/generated/imblearn.ensemble.EasyEnsemble.html`
[17]`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html`

Chapter 5

Test Results

In this chapter will be presented the most relevant test results for each one of the algorithms applied. The results will be presented through tables that show performance metrics scores.

The performance of the developed models was measured by measuring the capacity of prediction of the model through the validation set, the set that was unseen by the model in the training process, representing new real world information. Each table, for each one of the algorithms, includes five rows. Each row represents the best result of a set of tests, as described in 4.1, being:

- Row **1**: The performance of the model with no feature selection or balancing method applied.
- Row **2**: The combination that showed best performance from the experiments that combined the classification algorithm with all balancing methods with no feature selection.
- Row **3**: The combination that showed best performance from the experiments that combined the classification algorithm with the feature selection methods with no balancing method applied.
- Row **4**: The combination that showed best performance from the experiments that combined the classification algorithm with the best performing balancing method (previously determined) and all feature selection models, checking if the feature selection enhances the combination performance.
- Row **5**: The combination that showed best performance from the experiments that combined the classification algorithm with the best performing balancing method (previously determined) and all manually selected set of features as described in 4.1.

5.1 Best combinations of methods

The tables that show the best combinations for each one of the classification algorithms, based on their ROC AUC scoring metric, are:

Table 5.1: Best combinations for the Decision Tree classification algorithm

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	Decision Tree	**0,83**	0,69	**0,83**	0,5	**0,75**
2	Decision Tree + Condensed Nearest Neighbor	0,75	0,31	0,62	**0,7**	0,41
3	Decision Tree + Variance Threshold	0,56	**0,83**	0,56	0,65	0,62
4	Decision Tree + Select Kbest (75 features) + Condensed Nearest Neighbor	0,68	0,82	0,68	0,65	0,72
5	Decision Tree + Condensed Nearest Neighbor + Manually selected set 9	0,62	0,82	0,62	0,65	0,67

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.2: Best combinations for the Random Forest classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	Random Forest	0,75	0,73	0,75	0,44	0,74
2	Random Forest + NearMiss	0,61	**0,84**	0,61	0,68	0,67
3	Random Forest + Select Percentile (40%)	**0,86**	0,82	**0,86**	0,56	**0,83**
4	Random Forest + Select Kbest (50) + NearMiss	0,66	0,82	0,66	0,64	0,71
5	Random Forest + NearMiss + Manually selected set 23	0,43	0,7	0,43	**0,78**	0,51

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.3: Best combinations for the K-Nearest Neighbors classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	K-Nearest Neighbors	**0,86**	0,74	**0,86**	0,5	**0,8**
2	K-Nearest Neighbors + Random Under-sampling	0,58	0,82	0,58	0,63	0,64
3	K-Nearest Neighbors + RFECV (with Decision Tree)	0,75	0,79	0,75	0,57	0,77
4	K-Nearest Neighbors + Random Under-sampling + Select Kbest (75 features)	0,61	0,82	0,61	0,65	0,67
5	K-Nearest Neighbors + Random Under-sampling + Manually selected set 27	0,61	**0,91**	0,61	**0,73**	0,69

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.4: Best combinations for the Support Vector Machine classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	Support Vector Machines	**0,86**	0,74	**0,86**	0,5	**0,8**
2	Support Vector Machines + Instance Hardness Threshold	0,61	0,81	0,61	0,61	0,67
3	Support Vector Machines + Select Kbest (50 features)	0,7	0,81	0,7	0,63	0,74
4	Support Vector Machines + Select Percentile (40%) + Instance Hardness Threshold	0,73	0,82	0,73	**0,65**	0,76
5	Support Vector Machines + Instance Hardness Threshold + Manually selected set 27	0,44	**0,89**	0,44	0,64	0,53

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.5: Best combinations for the Naive Bayes classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	Naive Bayes	0,65	0,75	0,65	0,47	0,69
2	Naive Bayes + Instance Hardness Threshold	0,46	0,81	0,46	0,59	0,53
3	Naive Bayes + Select Kbest (75 features)	**0,7**	0,83	**0,7**	0,66	**0,74**
4	Naive Bayes + Select Percentile (80% features) + Instance Hardness Threshold	0,56	**0,85**	0,56	**0,68**	0,62
5	Naive Bayes + Instance Hardness Threshold + Manually selected set 22	0,59	0,79	0,59	0,63	0,64

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.6: Best combinations for the AdaBoost classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	AdaBoost (Decision Tree)	0,66	0,72	0,66	0,41	0,69
2	AdaBoost (Decision Tree) + Instance Hardness Threshold	0,63	0,83	0,63	0,66	0,69
3	AdaBoost (Decision Tree) + RFECV (Decision Tree)	**0,82**	0,81	**0,82**	0,6	**0,81**
4	AdaBoost (Decision Tree) + Variance Threshold + Instance Hardness Threshold	0,6	0,84	0,6	**0,67**	0,66
5	AdaBoost (Decision Tree) + Instance Hardness Threshold + Manually selected set 1	0,53	**0,88**	0,53	0,64	0,62

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.7: Best combinations for the Neural Networks classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	Neural Network	**0,87**	0,76	**0,87**	0,5	0,81
2	Neural Network + Random Over-sampler	0,83	0,84	0,83	0,66	**0,83**
3	Neural Network + All feature selection methods	**0,87**	0,76	**0,87**	0,5	0,81
4	Neural Network + Select Kbest (25 features) + Random Over-sampler	0,74	0,85	0,74	**0,71**	0,78
5	Neural Network + Random Over-sampler + Manually selected set 19	0,41	**0,86**	0,41	0,63	0,47

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.8: Best combinations for the RankSVM classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	RankSVM	0,71	0,75	0,71	0,48	0,73
2	RankSVM + Instance Hardness Threshold	0,59	0,8	0,59	0,6	0,65
3	RankSVM + Select Fpr	**0,74**	0,74	**0,74**	0,64	**0,74**
4	RankSVM + Select Fdr + Instance Hardness Threshold	0,65	0,8	0,65	0,6	0,7
5	RankSVM + Instance Hardness Threshold + Manually selected set 26	0,65	**0,89**	0,65	**0,71**	0,72

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.9: Best combinations for the RankBoost classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	RankBoost	0,84	0,78	0,84	0,52	0,8
2	RankBoost + Repeated Edited Nearest Neighbors	0,76	0,8	0,76	0,61	0,78
3	RankBoost + Select Kbest (75 features)	**0,86**	**0,83**	**0,86**	0,6	**0,84**
4	RankBoost + PCA + Repeated Edited Nearest Neighbors	0,76	0,8	0,76	0,6	0,78
5	RankBoost + Repeated Edited Nearest Neighbors + Manually selected set 14	0,69	0,77	0,69	**0,63**	0,72

1: Best performance, no balancing nor feature selection methods.

2: Best performance, only balancing methods.

3: Best performance, only feature selection methods.

4: Best performance, balancing method (from row 1) and feature selection methods.

5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.10: Best combinations for the RankNet (with 10 internal nodes) classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	RankNet	0,47	0,71	0,47	0,4	0,55
2	RankNet + Edited Nearest Neighbors	0,34	0,84	0,34	0,59	0,38
3	RankNet + RFE (Decision Tree)	**0,55**	0,83	**0,55**	**0,64**	**0,61**
4	RankNet + RFECV (Decision Tree) Edited Nearest Neighbors	0,42	0,86	0,42	0,63	0,47
5	RankNet + Edited Nearest Neighbors + Manually selected set 27	0,51	**0,88**	0,51	0,63	0,6

1: Best performance, no balancing nor feature selection methods.

2: Best performance, only balancing methods.

3: Best performance, only feature selection methods.

4: Best performance, balancing method (from row 1) and feature selection methods.

5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.11: Best combinations for the RankNet (with 50 internal nodes) classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	RankNet	0,23	0,79	0,23	0,52	0,21
2	RankNet + Instance Hardness Threshold	**0,65**	0,77	**0,65**	0,54	**0,69**
3	RankNet + Select Kbest (75 features)	0,57	0,8	0,57	**0,64**	0,59
4	RankNet + Select Fwe + Instance Hardness Threshold	0,45	**0,83**	0,45	0,62	0,51
5	RankNet + Instance Hardness Threshold + Manually selected set 12	0,47	**0,83**	0,47	0,63	0,53

1: Best performance, no balancing nor feature selection methods.

2: Best performance, only balancing methods.

3: Best performance, only feature selection methods.

4: Best performance, balancing method (from row 1) and feature selection methods.

5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.12: Best combinations for the Cost Sensitive Random Forests (with cost matrix number one) classification algorithm.

Row	Methods	Accuracy	Precision	Recall	ROC AUC	F1
1	Cost Sensitive RF	0.86	0.74	0.86	0.5	0.8
2	Cost Sensitive RF + Random Under-Sampler	0.55	0.83	0.55	0.64	0.66
3	Cost Sensitive RF + Select Percentile (40%)	**0.87**	**0.89**	**0.87**	0.54	**0.88**
4	Cost Sensitive RF + Select Kbest (75 features) + Random Under-Sampler	0.63	0.86	0.62	**0.72**	0.72
5	Cost Sensitive RF + Random Under-Sampler + Manually selected set 17	0.68	0.85	0.68	0.69	0.75

1: Best performance, no balancing nor feature selection methods.
2: Best performance, only balancing methods.
3: Best performance, only feature selection methods.
4: Best performance, balancing method (from row 1) and feature selection methods.
5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.13: Best combinations for the Easy Ensemble classification algorithm.

Row	Methods	Accuracy	Precision	Recall	ROC AUC	F1
1	Easy Ensemble	0,65	0,8	0,65	0,6	0,7
3	Easy Ensemble + Select Kbest (100 features)	0,69	**0,84**	0,69	0,69	0,73
5	Easy Ensemble + Manually selected set 5	**0,7**	**0,84**	0,7	**0,71**	**0,74**

1: Best performance, no balancing nor feature selection methods.
3: Best performance, no balancing, only feature selection methods.
5: Best performance, no balancing, only manually selected sets.

Table 5.14: Best combinations for the Cost Sensitive Random Forests (with cost matrix number two) classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
1	Cost Sensitive RF	**0.86**	0.74	**0.86**	0.5	**0.8**
2	Cost Sensitive RF + Random Under-Sampler	0.54	0.83	0.54	0.63	0.65
3	Cost Sensitive RF + Feature Aglomeration	0.82	0.76	0.82	0.51	0.79
4	Cost Sensitive RF + Random Under-Sampler + RFECV(Decision Tree)	0.66	**0.85**	0.66	**0.7**	0.74
5	Cost Sensitive RF + Random Under-Sampler + Manually Selected set 23	0.56	0.83	0.56	0.65	0.74

1: Best performance, no balancing nor feature selection methods.

2: Best performance, only balancing methods.

3: Best performance, only feature selection methods.

4: Best performance, balancing method (from row 1) and feature selection methods.

5: Best performance, balancing method (from row 1) and manually selected sets.

Table 5.15: Best combination for the Convolutional Neural Networks classification algorithm.

Row	**Methods**	**Accuracy**	**Precision**	**Recall**	**ROC AUC**	**F1**
All	Convolutional Neural Networks	0,87	0,76	0,87	0,5	0,81

5.2 Global comparison of the best results

As seen in table 5.1, from all the combinations of methods tested from the Decision Tree experiments, as explained in 4.1, the combination of Decision Tree and the Condensed Nearest Neighbor algorithm was the one that achieved a higher scoring in AUC.

For all combinations of methods tested from the Random Forest experiments, the combination that showed best performance score in AUC was the combination of Random Forest with the Near Miss algorithm and the manually selected set 23 (all features but the STS test features), as can be seen in table 5.2.

Considering the combinations made for the experiments with K-Nearest Neighbor, the combination that showed best AUC scoring was the combination between K-Nearest Neighbor with the Random under-sampling algorithm and the manually selected set 27 (traditional test scores and personal information), as shown in table 5.3

For the Support Vector Machines experiments, the combination that achieved the best AUC score was the combination between a Support Vector Machine with the Select percentile (set to choose 40% of features) algorithm and the Instance Hardness Threshold for balancing, as can be seen in table 5.4

The combination that showed best AUC scoring from the Naive Bayes experiments was the combination of Naive Bayes with the Select Percentile (set to choose 80% of features) and the Instance Hardness Threshold, as can be seen in table 5.5.

The AdaBoost set of experiments led to the conclusion that the combination of AdaBoost (Decision Tree) with the Variance Threshold and the Instance Hardness Threshold algorithm was the best AUC scoring, as can be seen in table 5.6

For the Neural Networks set of experiments, the combination that led to the best AUC scoring model was the combination of a Neural Network with the Select Kbest (set to choose 25 features) and the Random Over-sampling algorithm, as can be seen in table 5.7

The set of experiments of the RankSVM type showed that the combination that led to best AUC scoring results was the combination between the RankSVM algorithm with the Instance Hardness Threshold algorithm and the manually selected set 26 (traditional test scores), as can be seen in table 5.8.

For the RankBoost set of experiments, the combination that led to the best AUC scoring was the combination of the RankBoost algorithm with the Repeated Edited Nearest Neighbor algorithm and the manually selected set 14 (combination of the personal information of the participant with its performance on the 4 stage balance test), as can be seen in table 5.9.

The set of experiments for the RankNet (with 10 internal nodes) type showed that the combinations of methods that led to the best AUC scoring was the combination between the RankNet algorithm with the RFE (Decision Tree) algorithm, as can be seen in table 5.10.

The combination of methods that showed the best AUC scoring for the set of experiments of the RankNet (with 50 internal nodes) type was the combination of the RankNet algorithm with the Select Kbest (set to choose 75 features) algorithm, as can be seen in table 5.11.

The combination of methods that showed the best AUC scoring for the set of experiments of the Cost-Sensitive Random Forest (with cost-matrix one) type was the combination of the Cost-Sensitive Random Forest (with cost-matrix one) with the Select Kbest (set to choose 75 features) feature selection method and the Random Under-Sampler for balancing method, as can be seen in table 5.12.

The combination of methods that showed the best AUC scoring for the set of experiments of the Cost-Sensitive Random Forest (with cost-matrix two) type was the combination of the Cost-Sensitive Random Forest (with cost-matrix two) with the manually selected set 23 with the Random Under-Sampler balancing method, as can be seen in table 5.14.

For the Easy Ensemble implementation and set of experiments, the combination that led to the best AUC scoring was the combination of the Easy Ensemble algorithm with the manually selected set 5 (features of the Step Test), as can be seen in table 5.13.

Table 5.16: Comparison between the best results by algorithm type.

Algorithm with balancing and feature selection methods	**Accuracy**	**Precision**	**Recall**	**AUC**	**F1**
Decision Tree + Condensed Nearest Neighbor	**0.75**	0.31	0.62	0.7	0.41
Random Forest + Near Miss + Manually selected set 23	0.43	0.7	0.43	**0.78**	0.51
K-Nearest Neighbors + Random Under-Sampling + Manually selected set 27	0.61	**0.91**	0.61	0.73	0.69
Support Vector Machine + Select Percentile (40%) + Instance Hardness Threshold	0.73	0.82	0.73	0.65	0.76
Naive Bayes + Select Percentile (80% features) + Instance Hardness Threshold	0.56	0.85	0.56	0.68	0.62
AdaBoost (Decision Tree) + Variance Threshold + Instance Hardness Threshold	0.6	0.84	0.6	0.67	0.66
Neural Network + Select Kbest (25 features) + Random Over-Sampler	0.74	0.85	**0.74**	0.71	**0.78**
Cost Sensitive RF (matrix one) + Select Kbest (75 features) + Random Under-Sampler	0.63	0.86	0.62	0.72	0.72
Cost Sensitive RF (matrix two) + Random Under-Sampler + RFECV(Decision Tree)	0.66	0.85	0.66	0.7	0.74
RankSVM + Instance Hardness Threshold + Manually selected set 26	0.65	0.89	0.65	0.71	0.72
RankBoost + Repeated Edited Nearest Neighbors + Manually selected set 14	0.69	0.77	0.69	0.63	0.72
RankNet (10 internal nodes) + RFE (Decision Tree)	0.55	0.83	0.55	0.64	0.61
RankNet (50 internal nodes) + Select Kbest (75 features)	0.57	0.8	0.57	0.64	0.59
Easy Ensemble + Manually selected set 5	0.7	0.84	0.7	0.71	0.74

Table 5.16 shows the comparison between the best results obtained by algorithm type. The algorithm that showed best AUC score (combined with feature selection and balancing methods) was the Random Forest algorithm, with a score of 0.78. The K-Nearest Neighbor (combined with feature selection and balancing methods) was the second highest scorer. The RankBoost algorithm (combined with balancing and feature selection methods) showed the lowest AUC score, of 0.63.

5.2.1 Comparison of balancing methods

In the tables A.1 and A.2 presented in appendix A.1, the performance of the several balancing methods for the different types of experiments made (by type of classification algorithm) are presented according to their AUC score obtained with the validation set. There are two tables just to improve reading quality.

The balancing method that showed best AUC score was the Condensed Nearest Neighbor, with an AUC of 0.7, in the Decision Trees experiments. The one that showed worst performance was the SMOTEENN method with an AUC score of 0.55.

5.2.2 Comparison of feature selection methods

In the tables A.3 and A.4 presented in appendix A.1, the performance of the several feature selection methods for the different types of experiments made (by type of classification algorithm) are presented according to their AUC score obtained with the validation set. There are two tables just to improve reading quality.

The method that showed the best AUC score was the Select Kbest (mutual info, of 75 features), with an AUC score of 0.72. The Select Kbest (mutual info, of 100 features) method was the second highest with an AUC score of 0.69. The lowest score was showed by the RFECV (Decision Tree) and the PCA methods, with a score of 0.6.

5.2.3 Comparison of manually selected features

In the tables 5.17 and 5.18, the performance of the several manual feature selection methods, for the different types of experiments made (by type of classification algorithm), are presented according to their AUC score obtained with the validation set. There are two tables just to improve reading quality.

Table 5.17: Comparison of manual feature selection methods, part one.

AUC comparison	**DT**	**RF**	**KNN**	**SVM**	**NB**	**AdaBoost**	**NN**	**Easy Ensemble**
Set 1	0.53	0.61	0.61	0.5	0.59	**0.64**	0.5	0.6
Set 2	0.58	0.59	0.55	0.5	0.55	0.58	0.5	0.48
Set 3	0.49	0.58	0.49	0.5	0.44	0.62	0.54	0.59
Set 4	0.42	0.58	0.56	0.6	0.58	0.45	0.53	0.44
Set 5	0.58	0.48	0.56	0.5	0.51	0.56	0.54	**0.71**
Set 6	0.5	0.57	0.45	0.5	0.53	0.53	0.5	0.37
Set 7	0.57	0.57	0.46	0.54	0.56	0.53	0.5	0.45
Set 8	0.51	0.5	0.52	0.5	0.47	0.54	0.54	0.53
Set 9	**0.65**	0.53	0.63	0.54	0.57	0.6	0.53	0.64
Set 10	0.59	0.5	0.71	0.49	0.5	0.61	0.5	0.68
Set 11	0.44	0.58	0.64	0.54	0.57	0.59	0.56	0.6
Set 12	0.48	0.59	0.58	0.5	0.45	0.56	0.61	0.51
Set 13	0.39	0.59	0.39	0.5	0.61	0.59	0.5	0.4
Set 14	0.51	0.57	0.53	0.42	0.58	0.53	0.5	0.55
Set 15	0.51	0.45	0.51	0.5	0.53	0.44	0.6	0.55
Set 16	0.64	0.61	0.5	0.5	0.55	0.51	0.5	0.6
Set 17	0.52	0.5	0.55	0.47	0.52	0.54	0.47	0.53
Set 18	0.5	0.58	0.47	0.44	0.52	0.4	0.62	0.57
Set 19	0.6	0.49	0.56	0.57	0.6	0.56	**0.63**	0.68
Set 20	0.53	0.54	0.59	0.55	0.6	0.51	0.5	0.55
Set 21	0.46	0.48	0.56	0.61	0.51	0.53	0.5	0.5
Set 22	0.5	0.57	0.44	0.52	**0.63**	0.41	0.5	0.38
Set 23	0.5	**0.78**	0.44	0.4	0.51	0.6	0.5	0.61
Set 24	0.38	0.36	0.51	0.53	0.55	0.56	0.53	0.51
Set 25	0.6	0.59	0.49	0.47	0.56	0.52	0.5	0.54
Set 26	0.46	0.58	0.39	0.62	0.44	0.55	0.5	0.58
Set 27	0.61	0.69	**0.73**	**0.64**	0.61	0.51	0.5	0.65

Table 5.18: Comparison of manual feature selection methods, part two.

AUC comparison	CNN	CSRF (matrix one)	CSRF (matrix two)	Rank SVM	Rank Boost	Rank Net(10)	Rank Net(50)	Best AUC
Set 1	0.5	0.61	0.54	0.52	0.59	0.51	0.47	0.64
Set 2	0.5	0.49	0.64	0.5	0.6	0.58	0.53	0.64
Set 3	0.5	0.6	0.62	0.42	0.47	0.45	0.49	0.62
Set 4	0.5	0.57	0.55	0.61	0.46	0.55	0.45	0.61
Set 5	0.5	0.48	0.57	0.52	0.45	0.54	0.47	0.71
Set 6	0.5	0.46	0.38	0.53	0.43	0.51	0.45	0.57
Set 7	0.5	0.47	0.46	0.59	0.46	0.53	0.47	0.59
Set 8	0.5	0.6	0.54	0.53	0.46	0.48	0.52	0.6
Set 9	0.5	0.51	0.62	0.61	0.53	0.6	0.57	0.65
Set 10	0.5	0.65	0.57	0.56	0.39	0.56	0.47	0.71
Set 11	0.5	0.42	0.45	0.59	0.49	0.58	0.52	0.64
Set 12	0.5	0.5	0.58	0.59	0.55	0.56	**0.63**	0.63
Set 13	0.5	0.46	0.47	0.58	0.54	0.53	0.47	0.61
Set 14	0.5	0.46	0.38	0.53	**0.63**	0.51	0.46	0.63
Set 15	0.5	0.48	0.61	0.48	0.41	0.5	0.48	0.61
Set 16	0.5	0.56	0.51	0.53	0.6	0.61	0.5	0.64
Set 17	0.5	**0.69**	0.58	0.55	0.49	0.49	0.44	0.69
Set 18	0.5	0.62	0.58	0.54	0.46	0.45	0.44	0.62
Set 19	0.5	0.55	0.6	0.6	0.56	0.42	0.48	0.68
Set 20	0.5	0.62	0.52	0.54	0.42	0.53	0.54	0.62
Set 21	0.5	0.55	0.59	0.55	0.45	0.37	0.51	0.61
Set 22	0.5	0.46	0.56	0.66	0.53	0.46	0.5	0.66
Set 23	0.5	0.61	**0.65**	0.52	0.62	0.57	0.49	0.78
Set 24	0.5	0.48	0.55	0.61	0.53	0.47	0.5	0.61
Set 25	0.5	0.61	0.58	0.54	0.57	0.53	0.47	0.61
Set 26	0.5	0.59	0.51	**0.71**	0.58	0.56	0.58	0.71
Set 27	0.5	0.64	0.6	0.53	0.52	**0.63**	0.52	0.73

From the tables 5.17 and 5.18, it is perceivable that the set that showed best AUC result was the set 23 (all features but the STS test) on the Random Forests experiments. The set that showed worst performance was the set 6 (features from the Grip Strength test alone), showing a best score of 0.57. Focusing on the sets that were built to understand which differences would make if one the several tests were excluded (set 17, 18, 19, 20, 21, 22, 23, 24 as described in table 4.1), it is possible to perceive that the set that got an higher AUC score was the set 23 (as mentioned before), and the one that showed worst AUC score was the set 24 (features from all tests but the personal information) and the set 21 (features from all tests but the 10MW test), both showing an AUC score of 0.61. The set 27 (traditional test scores and personal information) showed the second highest AUC score.

Chapter 6

Discussion, conclusions and future work

In this section will be reviewed the main objectives of this dissertation, their state of accomplishment and will be made some suggestions on possible future work that can be done.

6.1 Discussion

From the several results presented, it is possible to understand the existence of several models that reveal high accuracy but low AUC. This is due to the fact that the validation set was not balanced, thus including more samples from the class of participants that didn't fall in the six-months follow-up period compared to the number of samples from the class of participants that have fallen during the follow-up period. This is also one of the reasons why AUC was chosen to compare performances since it is a better evaluator of the real predicting capacity of a binary model.

The differences in the cost matrices used with the Cost Sensitive Random Forests appear to not cause great differences in the learning mechanism for this scenario, since they showed similar highest scoring values.

The set of experiments from the Convolutional Neural Networks type showed that the models were unable to learn, therefore all combinations presented the same performance result, as showed in table 5.15. The model was trained by transfer learning, a technique that allows a pre-trained model to re-set some of its weights, adapting to new classes, while maintaining other weights on its structure, to ease training times and computational resources. The replacing classes that were introduced to the model were classes that were defined by groups of images, as explained in 4.1.6.5. This images are different from the ones which were used to pre-train the model, the images from the ImageNet database. With this database the model was trained to classify images that represented physical objects or beings, which is not the case here. This may justify the lack of adaptability to the new presented classes. The results on Convolutional Neural Networks were omitted from the table that compared the all best combinations, table 5.16.

6.1.1 Which is the best performing classification model?

The results presented in table 5.16, and in the graph A.1, show the performance of the combinations from the several sets of experiments (by classification algorithm type) that achieved highest scores in AUC. The combination that showed the best performance from all combinations tested was the combination of Random Forest with the Near Miss algorithm and the manually selected set 23 (all but the STS features), with an AUC score of 0.78. This may indicate that this algorithm deserves further focus and study for usability in this scenario and that not utilizing the features from the STS test may not prejudice the predicting capacity of the models.

Neural Networks appear as having high recall and F_1 scores while maintaining good accuracy, precision and AUC scores, appearing to be one of the better balanced combinations, so as the Easy Ensemble combination. The score for the Easy Ensemble experiment was obtained using only features from the Step Test which may reflect its importance.

6.1.2 Should any physical test be removed?

The best result in AUC may indicate that not using the STS test may not prejudice the predicting capacity of the models as referenced before. Also mentioned before was the possible importance of the Step Test in the predicting capacities of the models. Further studies should be conducted in order to reach clearer conclusions.

The low result showed by the Grip Strength manually selected set of features, set 6, may reflect that it does not influence the learning process, although it is a test that results in only one feature, which may not be enough for the learning process to occur when applied by itself.

K-Nearest Neighbors showed the second highest general AUC score, 0.73, in the manually selected set 27 (the traditional test scores and personal information). This may reflect the benefit that occur through the combination of the several scores (as in the set 27) instead of using just an isolated traditional score for the classification.

The removal of highly correlated features, as performed in this work, did not prejudiced the learning process which achieve relatively good results. The highly correlated features show the following quantities (when considering the whole dataset) by test: 50 are from the 4 Stage Balance test (42% of the features from this test), nine are from the Step Test (45% of the features from this test), nine are from the 30 Seconds Sit-to-Stand test (41% of features from this test), 36 are from the Timed Up and Go test (46% of features from this test), 14 are from the 10 Meter Walk test (61% of features from this test) and 16 are from the X-Reach test (24% of features from this test). Further studies on the contribution of each one of them and their possible elimination should be made.

Considering the comparison of the application of the feature selection methods solely, as presented in tables A.3 and A.4, the Select Kbest (mutual info, on 75 features) produced the highest AUC score when applied by itself, of 0.72, although, the feature selection method that led to the highest general AUC score was the manually selected set 23, as mentioned before.

The set of features that showed statistical differences at the beginning of the work appeared as producing an in-between normal average performance in AUC.

6.1.3 Which was the best balancing method?

Considering the comparison of balancing methods solely, with no feature selection, as presented in tables A.1 and A.1, it is perceivable that the balancing method Instance Hardness Threshold appears as being the method that produced the best AUC scoring more times than all the remaining methods. Considering 14 sets of experiments it appears to be the best balancing method when applied by itself in five of the sets (36%). The balancing method that led to the highest AUC score, when applied by itself, was the Condensed Nearest Neighbor, with an AUC score of 0.7, higher than the one presented by the Instance Hardness Threshold. The Near Miss algorithm showed the second highest AUC score and was the method included in the Random Forest combination that led to the general highest AUC score of 0.78.

6.1.4 Is the use of sensors and Machine Learning profitable for classification?

The best result obtained in AUC, 0.78, was obtained with the manually selected set 23, that incorporated features from all tests but the STS test. This included the features extracted from sensors and personal information. The second highest result was obtained with the manually selected set 27, that incorporated features that represented only traditional test scores and personal information. This may indicate that the use of instrumented tests, in combination with Machine Learning models may be best at discriminating groups in high risk for falls from groups in low risk for falls. This result also suggests that combining several traditional scores, instead of using just one can be beneficial.

Considering the state of the art in the use of Machine Learning approaches to predict the risk of falling, as presented in table 2.3, the population from each one of the conducted studies is different. The most similar study was the one conducted with a sample size of 292 participants, evaluating the TUG test with two inertial sensors. This study achieved a predictive accuracy of 76% with its best method, based on prior history of falls. Another study, with a sample of 120 participants, that also used a pressure platform and an inertial sensor, achieved a mean accuracy score of 71.52%, on the application of SVM, based on prior history of falls. The accuracy score of the best model in AUC scoring, in this work, focusing on the set of experiments that used SVM, was of 73%. Another study based on a smaller sample size of 37 participants, and based in prior history of falls, showed the best performance on accuracy, with scores of 90%.

Although the tests evaluated, the type of study (prospective in our case) and the sensors used were different, the results obtained in the current study are comparable to the ones presented on the literature since the maximal score obtained, in AUC, was of 78%. Searching for the best models according to accuracy can be misleading, not representative of the true predictability of the model. Some of the models obtained showed higher accuracy than the ones in the literature, but, as mentioned before, this scores do not represent the real predictive capacity of the models developed, being the AUC score more appropriate in this scenario.

6.2 Conclusion

Falls are affecting the life of a large number of persons, mainly the elderly population. They carry lots of consequences, from physical and psychological to monetary.

Achieving good prevention of falls instead of post-intervention has been a goal in research and a great effort has been made to develop a tool that is capable of making the distinction between future fallers and non-fallers.

Presented to a binary-classification problem, a set of possible classification algorithms was chosen that spanned from the various fields of Machine Learning. A methodology that consisted on several tests that evaluated the performance of the combination of the algorithms with feature selection and balancing methods was developed and the tests conducted, to search the widest range of possibilities for later implementation in the screening tool. The dataset was studied and analyzed statistically in order to be properly pre-processed. The biggest problem faced in this dissertation was the class imbalance problem, that reflected the difference in number of observations between the problem classes, which implied a broader research on methods to surpass it and successive application. Tests that allowed the comparison of the different physical tests in the collection of participant information were also developed.

The developed work achieved the primary objectives planned. Random Forest was the best performing algorithm obtaining an AUC score of 0.78 (combined with the Near Miss balancing algorithm and the manually selected set of features 23, that included all features from the tests but the STS test).

The traditional scores appear to have good predicting capacity, as mentioned on the literature, while the highly correlated features may not be useful for the learning process and could be eliminated as done in this work.

The Step Test appears to have importance in the learning process and the Grip Strength test appears as not when evaluated alone, although further studies should be developed in order to reach more informed conclusions. The Select Kbest method for feature selection was one of the best performing methods from all.

The results indicate that the use of instrumented tests combined with Machine Learning may show better predicting capacity than the traditional test scores of each test individually, although when combining traditional scores using appropriate machine learning techniques, they also show relatively high predicting capacities. The results obtained in the current study are comparable to the ones reported in the literature, having a best AUC score of 0.78, based on a prospective study, whilst the highest scoring known study from the literature showed an accuracy of 90% on a sample size of 37 participants, based on prior history of falls.

6.3 Future work

The work done can be improved in many aspects.

The objective of making as much tests as possible led to the superficial study of the application of some classification algorithms. For example, the application of transfer learning with Convolutional Neural Networks showed to be ineffective in this scenario. The development of a proper convolutional architecture, specific for this scenario, can be studied.

The good balance obtained with the application of Neural Networks could be further explored; access to more information and a tuned architecture could lead to better results.

Results of classification based on prior history of falls could be deeper compared to these obtained with a prospective history of falls, an important question to the literature.

Other cost matrices could be studied and applied in cost sensitive tests and more cost sensitive algorithms can be explored. Finally, the study could be also remade using the F_1 scoring metric for building the models, which is also a good evaluating metric for a binary classification problem.

One of the limitations of this study was that the follow-up period was of only six-months, further studies could be made with results from a follow-up period of twelve-months as most prospective studies do.

References

[ABeÇ16] Kemal Akyol, Şafak Bayir, Baha Şen, and Hasan B Çakmak. Detection of Hard Exudates in Retinal Fundus Images based on Important Features Obtained from Local Image Descriptors. 4(3):59–66, 2016.

[ACH$^+$11] Yuri Agrawal, John P Carey, Howard J Hoffman, Daniel A Sklare, and Michael C Schubert. The modified Romberg Balance Test: normative data in U.S. adults. *Otology & neurotology : official publication of the American Otological Society, American Neurotology Society [and] European Academy of Otology and Neurotology*, 32(8):1309–11, 2011.

[Ann16] Hendrich Ann. Fall Risk Assessment for Older Adults: The Hendrich II Fall Risk Model. *Best Practices in Nursing Care to Older Adults*, (8), 2016.

[APH13] Anne Felicia Ambrose, Geet Paul, and Jeffrey M. Hausdorff. Risk factors for falls among older adults: A review of the literature. *Maturitas*, 75(1):51–61, 2013.

[AW10] Hervé Abdi and Lynne J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[BAO15] Alejandro Correa Bahnsen, Djamila Aouada, and Bj??rn Ottersten. Example-dependent cost-sensitive decision trees. *Expert Systems with Applications*, 42(19):6609–6619, 2015.

[BFA$^+$11] Olivier Beauchet, B Fantino, Gilles Allali, S W Muir, M Montero-Odasso, and C Annweiler. Timed Up and Go test and risk of falls in older adults: a systematic review. *The journal of nutrition, health & aging*, 15(10):933–938, 2011.

[BH00] I. A. Basheer and M. Hajmeer. Artificial neural networks: Fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000.

[BHG12] David Bowes, Tracy Hall, and David Gray. Comparing the performance of fault prediction models which report multiple performance measures. *Proceedings of the 8th International Conference on Predictive Models in Software Engineering - PROMISE '12*, pages 109–118, 2012.

[Bre01] Leo Breiman. Random Forests. pages 1–33, 2001.

[Bur98] C.J.C. J Christopher J CJC Christopher J C Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[CBHK02] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[CBKH12] Kang Hee Cho, Soo Kyung Bok, Young Jae Kim, and Seon Lyul Hwang. Effect of lower limb strength on falls and balance of the elderly. *Annals of Rehabilitation Medicine*, 36(3):386–393, 2012.

[CFCP16] Ricardo Cruz, Kelwin Fernandes, Jaime S. Cardoso, and Joaquim F. Pinto Costa. Tackling class imbalance with ranking. *Proceedings of the International Joint Conference on Neural Networks*, 2016-October:2182–2187, 2016.

[CPP$^+$15] Luca Cattelani, Pierpaolo Palumbo, Luca Palmerini, Stefania Bandinelli, Clemens Becker, Federico Chesani, and Lorenzo Chiari. FRAT-up, a Web-based fall-risk assessment tool for elderly people living in the community. *Journal of medical Internet research*, 17(2):e41, feb 2015.

[CT91] Thomas M Cover and Joy a Thomas. *Chapter 2 Entropy , Relative Entropy and Mutual Information*, volume 19. 1991.

[DDZZ02] Wenliang Du, Wenliang Du, Zhijun Zhan, and Zhijun Zhan. Building decision tree classifier on private data. *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*, pages 1–8, 2002.

[DFF$^+$11] Emer P. Doheny, Chie Wei Fan, Timothy Foran, Barry R. Greene, Clodagh Cunningham, and Rose Anne Kenny. An instrumented sit-to-stand test used to examine differences between older fallers and non-fallers. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pages 3063–3066, 2011.

[DMG$^+$12] Emer P. Doheny, Denise McGrath, Barry R. Greene, Lorcan Walsh, David McKeown, Clodagh Cunningham, Lisa Crosby, Rose Anne Kenny, and Brian Caulfield. Displacement of centre of mass during quiet standing assessed using accelerometry in older fallers and non-fallers. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pages 3300–3303, 2012.

[Dom12] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78, 2012.

[Duf07] Stefan Duffner. *Face Image Analysis With Convolutional Neural Networks*. PhD thesis, 2007.

[ELD14] Andreas Ejupi, Stephen R. Lord, and Kim Delbaere. New methods for fall risk prediction. *Current Opinion in Clinical Nutrition and Metabolic Care*, 17(5):407–411, 2014.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006.

[GFLH09] S. García, Alberto Fernández, Julian Luengo, and F. Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability. *Soft Computing*, 13(10):959–977, 2009.

[GMW$^+$12] Barry R Greene, Denise Mcgrath, Lorcan Walsh, Emer P Doheny, David Mckeown, Chiara Garattini, Clodagh Cunningham, Lisa Crosby, Brian Caulfield, and Rose a Kenny. Quantitative falls risk estimation through multi-sensor assessment of standing balance. *Physiol. Meas*, 33:2049–2063, 2012.

[GRC16] Barry R. Greene, Stephen J. Redmond, and Brian Caulfield. Fall risk assessment through automatic combination of clinical fall risk factors and body-worn sensor data. *IEEE Journal of Biomedical and Health Informatics*, 21(3):1–1, 2016.

[Har06] P Hart. The Condensed Nearest Neighbor Rule (Corresp.). *IEEE Trans. Inf. Theor.*, 14(3):515–516, 2006.

[HBGL08] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *Proceedings of the International Joint Conference on Neural Networks*, (3):1322–1328, 2008.

[HG10] Haibo HE and Edwardo a. Garcia. Learning from Imbalanced Data Sets. *IEEE Transactions on knowledge and data engineering*, 21(9):1263—-1264, 2010.

[HYF$^+$08] Yuji Higashi, Kenichi Yamakoshi, Toshiro Fujimoto, Masaki Sekine, and Toshiyo Tamura. Quantitative evaluation of movement using the timed up-and-go test. *IEEE Engineering in Medicine and Biology Magazine*, 27(4):38–46, 2008.

[JMGL$^+$10] José M. Jerez, Ignacio Molina, Pedro J. García-Laencina, Emilio Alba, Nuria Ribelles, Miguel Martín, and Leonardo Franco. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial Intelligence in Medicine*, 50(2):105–115, 2010.

[KM97] Miroslav Kubat and Stan Matwin. Addressing the Curse of Imbalanced Training Sets: One Sided Selection. *Icml*, 97:179–186, 1997.

[Kot07] Sotiris B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[Lan15] Brett Lantz. Divide and Conquer – Classification Using Decision Trees and Rules. *Machine Learning With R*, page 452, 2015.

[Lau01] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. *Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine*, pages 63–66, 2001.

[LCiL15] Clare M. Lewandowski, New Co-investigator, and Clare M. Lewandowski. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. *The effects of brief mindfulness intervention on acute pain experience: An examination of individual difference*, 1:1689–1699, 2015.

[LMT03] Stephen R Lord, Hylton B Menz, and Anne Tiedemann. A physiological profile approach to falls risk assessment and prevention. *Physical therapy*, 83(3):237–52, 2003.

[LOJNG03] Lillemor Lundin-Olsson, Jane Jensen, Lars Nyberg, and Yngve Gustafson. Predicting falls in residential care by a risk assessment tool, staff judgement, and history of falls. *Aging clinical and experimental research*, 15(1):51–9, 2003.

[LRW+11] Ying Liu, Stephen J. Redmond, Ning Wang, Fernando Blumenkron, Michael R. Narayanan, and Nigel H. Lovell. Spectral analysis of accelerometry signals from a directed-routine for falls-risk estimation. *IEEE Transactions on Biomedical Engineering*, 58(8):2308–2315, 2011.

[LTDRdS14] Patricio Loncomilla, Claudio Tapia, Omar Daud, and J Ruiz-del Solar. A Novel Methodology for Assessing the Fall Risk Using Low-Cost and Off-the-Shelf Devices. *IEEE Transactions on Human-Machine Systems*, 44(3):406–415, 2014.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi Hua Zhou. Isolation forest. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 413–422, 2008.

[LW02] a Liaw and M Wiener. Classification and Regression by randomForest. *R news*, 2(December):18–22, 2002.

[LWZ06] Xu Ying Liu, Jianxin Wu, and Zhi Hua Zhou. Exploratory under-sampling for class-imbalance learning. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 965–969, 2006.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[MOPO03] Mary A Murphy, Sharon L Olson, Elizabeth J Protas, and Averell R Overby. Screening for Falls in Community-Dwelling Elderly. *Aging*, 1994:66–80, 2003.

[MSH96] Suzanne MacAvoy, Teresa Skinner, and Maria Hines. Fall Risk Assessment Tool. *Applied Nursing Research*, 9(ii):213–218, 1996.

[MSS+16] Anabela Martins, Joana Silva, António Santos, João Madureira, Carlos Alcobia, Luís Ferreira, Pedro Mendes, Cláudia Tonelo, Catarina Silva, Daniela Baltazar, and Inês Sousa. Case-Based Study of Metrics Derived from Instrumented Fall Risk Assessment Tests. *Sciprints*, (August), 2016.

[New01] R. A. Newton. Validity of the Multi-Directional Reach Test: A Practical Measure for Limits of Stability in Older Adults. *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences*, 56(4):M248–M252, 2001.

[Nil05] Nils J. Nilsson. Introduction To Machine Learning an Early Draft of a Proposed Text. pages 1–188, 2005.

[NRS+10] Michael R. Narayanan, Stephen J. Redmond, Maria Elena Scalzi, Stephen R. Lord, Branko G. Celler, and Nigel H. Lovell. Longitudinal falls-risk estimation using triaxial accelerometry. *IEEE Transactions on Biomedical Engineering*, 57(3):534–541, 2010.

[OBS+97] D Oliver, M Britton, P Seed, Martin FC, and Hopper AH. Development and evaluation of evidence based risk assessment tool (STRATIFY) to predict which elderly inpatients will fall: case-control and cohort studies. *BMJ: British Medical Journal (International Edition)*, 315(7115):1049–1053 5p, 1997.

[PFK13] Denise M. Peters, Stacy L. Fritz, and Debra E. Krotish. Assessing the Reliability and Validity of a Shorter Walk Test Compared With the 10-Meter Walk Test for Measurements of Gait Speed in Healthy, Older Adults. *Journal of Geriatric Physical Therapy*, 36(1):24–30, 2013.

[Ras14] Sebastian Raschka. Naive Bayes and Text Classification I - Introduction and Theory. *arXiv preprint arXiv:1410.5329*, page 20, 2014.

[RDM$^+$11] Helen C. Roberts, Hayley J. Denison, Helen J. Martin, Harnish P. Patel, Holly Syddall, Cyrus Cooper, and Avan Aihie Sayer. A review of the measurement of grip strength in clinical and epidemiological studies: Towards a standardised approach. *Age and Ageing*, 40(4):423–429, 2011.

[RLOK$^+$03] Erik Rosendahl, Lillemor Lundin-Olsson, Kristina Kallin, Yngve Gustafson, and Lars Nyberg. Prediction of falls among older people in residential care facilities by Downton index. *Aging Clin Exp Res*, 15(2):142–7, 2003.

[Rux06] Graeme D. Ruxton. The unequal variance t-test is an underused alternative to Student's t-test and the Mann-Whitney U test. *Behavioral Ecology*, 17(4):688–690, 2006.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[SL09] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45(4):427–437, 2009.

[SMGC14] Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.

[SMT$^+$16] Joana Silva, João Madureira, Cláudia Tonelo, Daniela Baltazar, Catarina Silva, Anabela Martins, Carlos Alcobia, and Inês Sousa. Comparing Machine Learning Approaches for Fall Risk Assessment. 2016.

[SVI$^+$15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. 2015.

[SVSC07] Vicky Scott, Kristine Votova, Andria Scanlan, and Jacqueline Close. Multifactorial and functional mobility assessment tools for fall risk among older adults in community, home-support, long-term and acute care settings. *Age and Ageing*, 36(2):130–139, 2007.

[TG09] Pucktada Treeratpituk and C. Lee Giles. Disambiguating authors in academic publications using random forests. *Proceedings of the 2009 joint international conference on Digital libraries - JCDL '09*, pages 39–48, 2009.

[TOG$^+$14] Julia C. Thomas, Charles Odonkor, Laura Griffith, Nicole Holt, Sanja Percac-Lima, Suzanne Leveille, Pensheng Ni, Nancy K. Latham, Alan M. Jette, and Jonathan F. Bean. Reconceptualizing balance: Attributes associated with balance performance. *Experimental Gerontology*, 57:218–223, 2014.

[Tom76] Ivan Tomek. An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(6):448–452, 1976.

[VHLW09] Joe Verghese, Roee Holtzer, Richard B. Lipton, and Cuiling Wang. Quantitative gait markers and incident fall risk in older adults. *Journals of Gerontology - Series A Biological Sciences and Medical Sciences*, 64(8):896–901, 2009.

[Wil72] Dennis L. Wilson. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man and Cybernetics*, 2(3):408–421, 1972.

[WLC05] Julie C. Whitney, Stephen R. Lord, and Jacqueline C T Close. Streamlining assessment and intervention in a falls clinic using the Timed Up and Go Test and Physiological Profile Assessments. *Age and Ageing*, 34(6):567–571, 2005.

[XDL$^+$08] Yanshan Xiao, Feiqi Deng, Bo Liu, Shouqiang Liu, Dan Luo, and Guohua Liang. A learning process using svms for multi-agents decision classification. *Proceedings - 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT Workshops 2008*, (4):583–586, 2008.

[YoS10] Xiuxin (Department of Electrical Yang and Computer Engineering University of Saskatchewan). *A Wearable Real-Time System for Physical Activity Recognition and Fall Detection*. PhD thesis, 2010.

[ZM03] Jianping Zhang and Inderjeet Mani. kNN Approach to Unbalanced Data Distributions: A Case Study involving Information Extraction. *Workshop on Learning from Imbalanced Datasets II ICML Washington DC 2003*, pages 42–48, 2003.

[ZMF$^+$15] Lin Zhang, Ou Ma, Jennifer M Fabre, Robert H Wood, Stephanie U Garcia, Kayla M Ivey, Evan D Mccann, and A Test Participants. Classification of Older Adults with / without a Fall History using Machine Learning Methods. *IEEE Engineering in Medicine and Biology Society*, pages 6760–6763, 2015.

[ZZRH09] Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics and Its Interface*, 2:349–360, 2009.

Appendix A

Parameters for methods used in RandomizedSearchCV

In this chapter will be presented the parameters used on the RandomizedSearchCV application for the algorithms that followed the proposed set of steps, considered as being non-deviating.

To start it is important to describe the logspace function from the numpy library[1] that creates an array with a specified number of floats from a possible range. The smallest number of the range is defined by the value of base 10 to the power of start and the end value defined by the value of base 10 to the power of stop. As an example:

```
np.logspace(start=float(-2), stop=float(1), num=25)
```

May originate:

```
[  0.01         0.01333521   0.01778279   0.02371374   0.03162278
   0.04216965   0.05623413   0.07498942   0.1          0.13335214
   0.17782794   0.23713737   0.31622777   0.4216965    0.56234133
   0.74989421   1.           1.33352143   1.77827941   2.37137371
   3.16227766   4.21696503   5.62341325   7.49894209  10.        ]
```

This method was used to define possible ranges of parameters for the algorithm.

The scales were not chosen randomly: each individual parameter was tested individually with large scales and the sub-scales that benefited the performance of the model were chosen.

A.0.1 Decision Trees

In the training of Decision Trees[2] the parameters to include in the possible combinations were:

[1] https://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.logspace.html
[2] http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```

log = np.logspace(start=float(-8), stop=float(-6), num=25)

params_grid = dict(dt__splitter=['best', 'random'],
                   dt__criterion=['gini', 'entropy'],
                   dt__max_features=[None, 'auto', 'sqrt', 'log2'],
                   dt__max_depth=[None],
                   dt__min_samples_split=[2, 5, 10],
                   dt__min_samples_leaf=[1, 5, 10],
                   dt__max_leaf_nodes=[None],
                   dt__min_impurity_split=log,
                   dt__presort=[True, False],
                   dt__random_state=[None],
                   dt__class_weight=['balanced', None])
```

All possible combinations were searched.

A.0.2 Random Forests

As for the Random Forests[3] algorithm, the parameters to include in the possible combinations were:

```

log = np.logspace(start=float(-8), stop=float(-6), num=25)

params_grid = dict(rf__n_estimators=[5, 10, 50, 150],
                     rf__criterion=['gini', 'entropy'],
                     rf__max_features=[None, 'auto', 'sqrt', 'log2'],
                     rf__max_depth=[None],
                     rf__min_samples_split=[2, 5, 10],
                     rf__min_samples_leaf=[1, 5],
                     rf__max_leaf_nodes=[None],
                     rf__min_impurity_split=log,
                     rf__bootstrap=[True],
                     rf__oob_score=[False],
                     rf__random_state=[None],
                     rf__warm_start=[False],
                     rf__class_weight=['balanced', 'balanced_subsample', None],
                     rf__n_jobs=[-1])
```

For this algorithm 2000 combinations were searched.

[3]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

A.0.3 K-Nearest Neighbors

For the K-Nearest Neighbors[4] algorithm, the parameters to include in the possible combinations were:

```
params_grid = dict(knn__n_neighbors=[1, 5, 10],
                   knn__weights=['uniform', 'distance'],
                   knn__algorithm=['auto', 'kd_tree', 'brute'],
                   knn__leaf_size=[2, 5, 10],
                   knn__metric=['manhattan', 'chebyshev'],
                   knn__p=[1, 2, 3],
                   knn__metric_params=[None],
                   knn__n_jobs=[-1])
```

For this algorithm 300 combinations were searched.

A.0.4 Support Vector Machines

For the SVM[5] algorithm, the parameters to include in the possible combinations were:

```
c = np.logspace(start=float(1), stop=float(2), num=25)
coef = np.logspace(start=float(-2), stop=float(1), num=25)
tol = np.logspace(start=float(-4), stop=float(-1), num=25)
log = np.logspace(start=float(-4), stop=float(-1), num=25)
gamma = list()
for el in log:
  gamma.insert(len(gamma), el)
    gamma.insert(len(gamma), 'auto')

params_grid = dict(svm_svc__C=c,
                   svm_svc__kernel=['linear', 'poly', 'rbf', 'sigmoid'],
                   svm_svc__degree=[1, 2, 3],
                   svm_svc__gamma=gamma,
                   svm_svc__coef0=coef,
                   svm_svc__tol=tol,
                   svm_svc__class_weight=['balanced', None],
                   svm_svc__probability=[True, False],
                   svm_svc__shrinking=[True, False],
                   svm_svc__max_iter=[-1],
                   svm_svc__decision_function_shape=['ovo', 'ovr', None],
                   svm_svc__random_state=[None])
```

For this algorithm 1000 combinations were searched.

[4] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[5] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

A.0.5 Naive Bayes

For the Naive Bayes[6] algorithm, the parameters to include in the possible combinations were:

```
priors = [[0.1, 0.9], [0.2, 0.8], [0.3, 0.7], [0.4, 0.6], [0.5, 0.5], [0.6, 0.4],
    [0.7, 0.3], [0.8, 0.2], [0.9, 0.1]]

params_grid = dict(nb__priors=priors)
```

For this algorithm all combinations were searched.

A.0.6 AdaBoost

For the AdaBoost[7] algorithm, the parameters to include in the possible combinations were:

```
priors = [[0.1, 0.9], [0.2, 0.8], [0.3, 0.7], [0.4, 0.6], [0.5, 0.5], [0.6, 0.4],
    [0.7, 0.3], [0.8, 0.2], [0.9, 0.1]]
log = np.logspace(start=float(1), stop=float(2), num=25)
params_grid = dict(adaboost__base_estimator=[DecisionTreeClassifier()],
                   adaboost__n_estimators=[50,100,150],
                   adaboost__learning_rate=log,
                   adaboost__algorithm=['SAMME', 'SAMME.R'],
                   adaboost__random_state=[None])
```

For this algorithm all combinations were searched.

A.0.7 RankSVM

The RankSVM algorithm was applied with the default parameters that assume the model as being a LinearSVC as in the documentation[8]:

```
model = LinearSVC(fit_intercept=False, penalty='l1', tol=1e-3,
                                                    dual=False)
```

For this algorithm all combinations were searched.

[6]http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[7]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[8]http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

Parameters for methods used in RandomizedSearchCV

A.1 Test results

Table A.1: Comparison of balancing methods, part one.

AUC comparison	DT	RR	KNN	SVM	NB	AdaBoost	NN
Cluster Centroids	0.53	0.53	0.48	0.53	0.53	0.58	0.5
Condensed Nearest Neighbor	**0.7**	0.5	0.59	0.48	0.46	0.64	0.5
Edited Nearest Neighbor	0.53	0.61	0.51	0.58	0.49	0.5	0.5
Repeated Edited Nearest Neighbors	0.54	0.48	0.56	0.49	0.56	0.47	0.5
AllKNN	0.48	0.53	0.49	0.53	0.57	0.49	0.5
Instance Hardness Threshold	0.49	0.61	0.57	**0.61**	**0.59**	**0.66**	0.5
Near Miss	0.45	**0.68**	0.55	0.47	0.43	0.56	0.62
Neighbourhood Cleaning Rule	0.51	0.51	0.47	0.47	0.52	0.57	0.5
One Sided Selection	0.47	0.52	0.53	0.5	0.54	0.57	0.5
Random Under Sampler	0.39	0.5	**0.63**	0.56	0.48	0.6	0.5
Tomek Links	0.48	0.5	0.5	0.56	0.53	0.46	0.5
ADASYN	0.39	0.6	0.49	0.48	0.48	0.5	0.5
Random Over Sampler	0.58	0.46	0.54	0.5	0.43	0.48	**0.66**
SMOTE	0.56	0.48	0.57	0.5	0.53	0.48	0.5
SMOTEENN	0.48	0.47	0.49	0.55	0.5	0.54	0.5
SMOTE Tomek	0.5	0.55	0.5	0.59	0.48	0.48	0.5

Table A.2: Comparison of balancing methods, part two.

AUC comparison	CNN	CSRF (matrix one)	CSRF (matrix two)	Rank SVM	Rank Boost	Rank Net(10)	Rank Net(50)	Best AUC
Cluster Centroids	0.5	0.55	0.61	0.59	0.51	0.46	0.53	0.61
Condensed Nearest Neighbor	0.5	0.62	0.62	0.42	0.54	0.57	0.52	0.7
Edited Nearest Neighbor	0.5	0.51	0.5	0.44	0.51	**0.59**	0.47	0.61
Repeated Edited Nearest Neighbors	0.5	0.55	0.56	0.48	**0.61**	0.52	0.54	0.61
AllKNN	0.5	0.5	0.5	0.52	0.47	0.43	0.53	0.57
Instance Hardness Threshold	0.5	0.46	0.49	**0.6**	0.54	0.5	**0.54**	0.66
Near Miss	0.5	0.56	0.58	0.49	0.56	0.45	0.53	0.68
Neighborhood Cleaning Rule	0.5	0.48	0.48	0.46	0.46	0.44	0.5	0.57
One Sided Selection	0.5	0.49	0.49	0.48	0.49	0.49	0.5	0.57
Random Under Sampler	0.5	**0.64**	**0.63**	0.52	0.51	0.49	0.47	0.64
Tomek Links	0.5	0.49	0.62	0.52	0.46	0.53	0.52	0.62
ADASYN	0.5	0.5	0.49	0.48	0.48	0.53	0.45	0.6
Random Over Sampler	0.5	0.53	0.59	0.48	0.49	0.43	0.51	0.66
SMOTE	0.5	0.51	0.53	0.5	0.48	0.45	0.5	0.57
SMOTEENN	0.5	0.55	0.5	0.49	0.48	0.53	0.47	0.55
SMOTE Tomek	0.5	0.48	0.5	0.46	0.41	0.56	0.53	0.59

Table A.3: Comparison of feature selection methods, part one.

AUC comparison	DT	RF	KNN	SVM	NB	AdaBoost	NN	Easy Ensemble
Select Percentile (f_classif) 20	0.56	0.54	0.54	0.5	0.62	0.54	0.5	0.56
Select Percentile (f_classif) 40	0.5	0.54	0.5	0.62	0.65	0.51	0.5	0.53
Select Percentile (f_classif) 60	0.5	0.5	0.52	0.55	0.56	0.48	0.5	0.58
Select Percentile (f_classif) 80	0.46	0.5	0.5	0.5	0.64	0.5	0.5	0.63
Select Kbest (f_classif) 25	0.42	0.51	0.5	0.5	0.61	0.44	0.5	0.61
Select Kbest (f_classif) 50	0.45	0.5	0.54	**0.63**	0.65	0.54	0.5	0.62
Select Kbest (f_classif) 75	0.53	0.5	0.5	0.59	**0.66**	0.51	0.5	0.63
Select Kbest (f_classif) 100	0.58	0.53	0.53	0.5	0.62	0.49	0.5	0.6
Select Fpr (f_classif)	0.42	0.5	0.49	0.5	0.6	0.59	0.5	0.62
Select Fdr (f_classif)	0.59	0.49	0.5	0.57	0.65	0.58	0.5	0.66
Select Fwe (f_classif)	0.62	0.49	0.5	0.5	0.6	0.56	0.5	0.55
RFE (Decision Tree)	0.5	0.51	0.49	0.5	0.46	0.49	0.5	0.65
RFECV (Decision Tree)	0.5	0.51	**0.57**	0.5	0.57	**0.6**	0.5	0.53
Variance Threshold	**0.65**	0.53	0.48	0.5	0.55	0.55	0.5	0.58
Select Percentile (mutual_info) 20	0.47	0.55	0.5	0.5	0.57	0.42	0.5	0.66
Select Percentile (mutual_info) 40	0.51	**0.56**	0.54	0.5	0.55	0.48	0.5	0.68
Select Percentile (mutual_info) 60	0.42	0.48	0.5	0.5	0.61	0.48	0.5	0.59
Select Percentile (mutual_info) 80	0.5	0.53	0.54	0.5	0.47	0.5	0.5	0.55
Select Kbest (mutual_info) 25	0.47	0.56	0.53	0.5	0.58	0.5	0.5	0.62
Select Kbest (mutual_info) 50	0.48	0.48	0.53	0.57	0.6	0.58	0.5	0.6
Select Kbest (mutual_info) 75	0.49	0.51	0.53	0.5	0.49	0.47	0.5	0.54
Select Kbest (mutual_info) 100	0.44	0.51	0.5	0.5	0.66	0.46	0.5	**0.69**
PCA	0.5	0.53	0.49	0.57	0.53	0.48	0.5	0.51
Feature Agglomeration	0.54	0.5	0.48	0.43	0.64	0.52	0.5	0.52

Table A.4: Comparison of feature selection methods, part two.

AUC comparison	**CNN**	**CSRF (matrix one)**	**CSRF (matrix two)**	**Rank SVM**	**Rank Boost**	**Rank Net(10)**	**Rank Net(50)**	**Best AUC**
Select Percentile (f_classif) 20	0.5	0.53	0.49	0.57	0.48	0.51	0.47	0.62
Select Percentile (f_classif) 40	0.5	0.59	0.5	0.58	0.47	0.53	0.44	0.65
Select Percentile (f_classif) 60	0.5	0.66	0.49	0.51	0.48	0.47	0.45	0.66
Select Percentile (f_classif) 80	0.5	0.61	0.5	0.54	0.48	0.62	0.51	0.64
Select Kbest (f_classif) 25	0.5	0.62	0.5	0.48	0.51	0.55	0.53	0.62
Select Kbest (f_classif) 50	0.5	0.63	0.49	0.51	0.48	0.51	0.48	0.65
Select Kbest (f_classif) 75	0.5	0.67	0.5	0.62	0.44	0.53	**0.64**	0.67
Select Kbest (f_classif) 100	0.5	0.65	0.49	0.52	0.45	0.48	0.56	0.65
Select Fpr (f_classif)	0.5	0.6	0.5	**0.64**	0.51	0.49	0.51	0.65
Select Fdr (f_classif)	0.5	0.61	0.5	0.57	0.48	0.49	0.51	0.66
Select Fwe (f_classif)	0.5	0.67	0.48	0.57	0.51	0.55	0.53	0.67
RFE (Decision Tree)	0.5	0.57	0.5	0.57	0.53	**0.64**	0.5	0.65
RFECV (Decision Tree)	0.5	0.49	0.49	0.58	0.45	0.52	0.53	0.6
Variance Threshold	0.5	0.6	0.5	0.57	0.5	0.45	0.53	0.65
Select Percentile (mutual_info) 20	0.5	0.62	0.49	0.59	0.51	0.62	0.54	0.66
Select Percentile (mutual_info) 40	0.5	0.62	0.5	0.55	0.47	0.52	0.54	0.68
Select Percentile (mutual_info) 60	0.5	0.63	0.5	0.54	0.52	0.58	0.56	0.63
Select Percentile (mutual_info) 80	0.5	0.64	0.5	0.51	0.48	0.54	0.54	0.64
Select Kbest (mutual_info) 25	0.5	0.66	0.49	0.61	0.49	0.47	0.42	0.66
Select Kbest (mutual_info) 50	0.5	0.65	0.49	0.49	0.55	0.51	0.43	0.65
Select Kbest (mutual_info) 75	0.5	**0.72**	0.5	0.48	**0.6**	0.51	0.48	0.72
Select Kbest (mutual_info) 100	0.5	0.63	0.49	0.52	0.53	0.4	0.47	0.69
PCA	0.5	0.6	0.49	0.54	0.47	0.52	0.55	0.6
Feature Agglomeration	0.5	0.51	**0.51**	0.5	0.52	0.53	0.41	0.64

A.2 Performance comparison

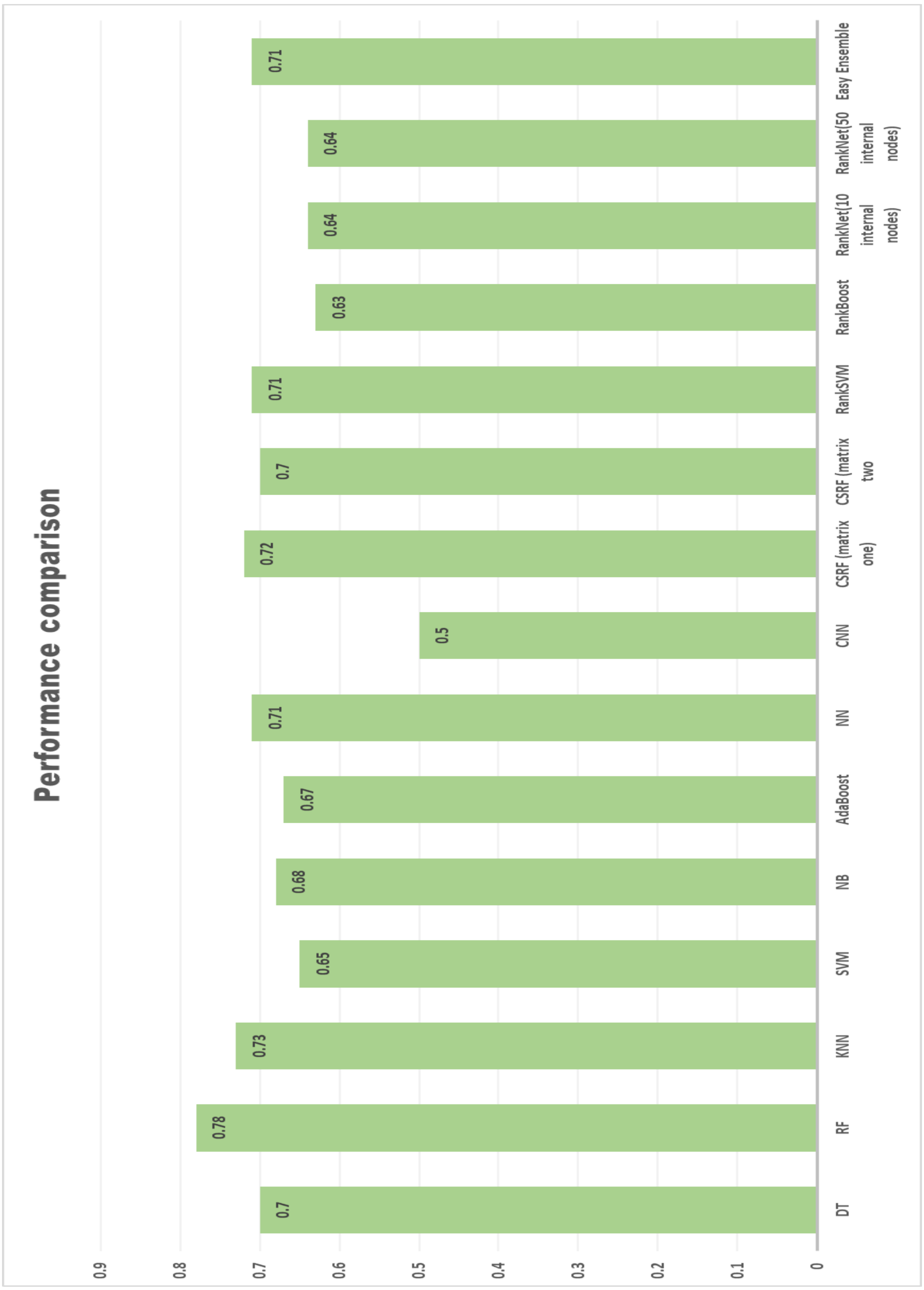


Figure A.1: Comparison of the best performing combinations by set of experiments.