Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección Académica

Case studies in feature extraction and parameter tuning for time-series classification

por

Pedro Sánchez Martínez

como requisito parcial para obtener el grado de

INGENIERO EN TECNOLOGÍA DE SOFTWARE

Mayo 2019

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección Académica

Case studies in feature extraction and parameter tuning for time-series classification

por

Pedro Sánchez Martínez

como requisito parcial para obtener el grado de

INGENIERO EN TECNOLOGÍA DE SOFTWARE

Mayo 2019

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección Académica

Los miembros del Comité de Tesis recomendamos que la Tesis "Case studies in feature extraction and parameter tuning for time-series classification", realizada por el alumno Pedro Sánchez Martínez, con número de matrícula 1590340, sea aceptada para su defensa como requisito parcial para obtener el grado de Ingeniero en Tecnología de Software.

El Comité de Tesis

Dra. Satu Elisa Schaeffer
Asesora

Dr. Romeo Sánchez Nigenda
Revisor

Dra. Sara Elena Garza Villarreal
Revisora

Vo. Bo.

Dr. Arnulfo Treviño Cubero
Subdirección Académica

San Nicolás de los Garza, Nuevo León, mayo 2019

Agradecimientos

Quiero agradecer a la Dra. Satu Elisa Schaeffer que no fue solo mi maestra a lo largo de la carrera, también fue mi tutora y ejemplo a seguir. No solo me hizo mejorar en ambito académico o profesional, sino también en crecimiento como persona en la sociedad para contribuir y ayudar.

A mis padres Gloria Martínez Argumedo y Fidencio Sánchez Perez: gracias por su apoyo y educación que me brindaron para el progreso de mis estudios, siempre dándome apoyo en las desveladas y desmotivaciones para salir adelante sin importar que pase.

Al programa PAICYT de la UANL por la beca dentro del marco del proyecto *Herramientas computacionales para análisis epidemiológico multifactorial* (IT512-15).

Al Dr. Romeo Sánchez Nigenda y a la Dra. Sara Elena Garza Villarreal por brindar su apoyo en la creación de esta investigación como integrantes de comité de revisión, dando su punto de vista y consejos para la mejora del presente trabajo.

Resumen

Pedro Sánchez Martínez.

Candidato para obtener el grado de Ingeniero en Tecnología de Software.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio: Case studies in feature extraction and parameter tuning for time-series classification.

Número de páginas: 52.

Objetivos y método de estudio: En esta tesis se propone una herramienta para seleccionar un modelo de aprendizaje supervisado para clasificar series de tiempo y determinar el ajuste de parámetros que obtiene mejores resultados para una tarea específica, encontrando la mayor puntuación de precisión y el menor error posible.

Las fases que realiza la herramienta son: *preprocesamiento* para dar un formato estructurado a los datos, *entrenamiento* de un conjunto de algoritmos de aprendizaje supervisado, aplicando un proceso de ajuste de parámetros para cada uno con el fin de obtener un valor de precisión adecuada y finalmente *análisis* para reportar, comparar y dar resultados obtenidos.

CONTRIBUCIONES Y CONCLUSIONES: La presente tesis documenta el proceso de entrenamiento de modelos de aprendizaje supervisado: limpieza de datos, procesamiento, clasificación, búsqueda de características, cambios variados en los parámetros de los modelos de clasificación, los pronósticos a través de clasificación y una medición de exactitud y error para pronosticar eventos desde intervalos de tiempo anteriores con la finalidad de tomar desiciones preventivas.

La herramienta propuesta muestra una precisión adecuada para cada caso de estudio; los cambios en los parámetros son reflejados en el desempeño obtenido.

Firma de la asesora: ______________________________
Dra. Satu Elisa Schaeffer

Abstract

Pedro Sánchez Martínez.

Candidate for obtaining the degree of Software Technology Engineer.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of study: Case studies in feature extraction and parameter tuning for time-series classification.

Number of pages: 52.

Objectives and methods: This thesis proposes a tool to select a supervised learning model to classify time series and determine the parameter setting that obtains the best results for a specific task, finding the highest precision score and the least possible error.

The phases that the tool performs are: *preprocessing* to give a structured format to the data, *training* a set of supervised learning algorithms, applying a process of adjustment of parameters for each one in order to obtain an adequate precision value, and *analysis* to report and compare results obtained.

Contributions and conclusions: This thesis presents the training process of supervised learning models: data cleaning, processing, classification, search for characteristics, parameters set of the classification models, forecasts through classification, and a measurement of accuracy and error to forecast events from previous time intervals in order to take preventive decisions.

The proposed tool shows an adequate precision for each case study; the changes in the parameters are reflected in the performance.

Signature of supervisor: ____________________________

Dra. Satu Elisa Schaeffer

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Predicting possible events over time has always been of interest of humanity. Before technology people already made predictions using their own intuition from routines, developing and applying mathematics or statistics to have a scientific basis to support them. Currently, using information technologies, one can quickly find patterns and automate tests.

Information technologies such as computers and programming languages not only help to automate calculations to make them faster, but also help to store large volumes of information and, at the same time, new algorithms are created to store more data. Not only the ability to save large volumes of information has been improved: the hardware has improved multiprocessing allows for increasingly faster calculations that would take weeks or months if done sequentially, but are done in minutes with parallel computing.

There are new infrastructures such as the cloud and distributed systems that provide access to data to multiple users working on the same information simultaneously. Artificial intelligence algorithms that could not be applied before, because of the little progress in hardware, now can be used and explored to define and test hypotheses.

This chapter presents a motivation for the problem studied in the thesis, a hypothesis to examine with applied experiments in different cases, the objectives to be accomplished, and, finally, the structuring of the chapters of the thesis with their corresponding descriptions.

1.1 Motivation

Even with statistical methods, large errors are produced by not finding a feasible precision to predict a time series, because the behavior at the series changes over the time which makes it complicated to identify a pattern. Nowadays, the use of artificial intelligence allows finding a pattern without having to manually search for a relationship, association, or rules for a case.

1.2 Hypothesis

The hypothesis of this work is that it is possible to predict events using artificial intelligence algorithms such as supervised learning for time series.

1.3 Objectives

In this thesis, a supervised learning tool is implemented to classify time series, to predict future events with a feasible precision, and to find which algorithm works best for each case. The specific objectives are:

- Extract features from time series.
- Classify time series depending in the case to predict future events for the model with the best precision in terms of accuracy and error.

- Vary the configuration of the parameters to determine their effect on performance.

1.4 Structure

Chapter 2 presents the fundamental concepts of the problems and algorithms used.

Chapter 3 describes related work, mentioning the important points of each one and their objectives. Also, a descriptive analysis is given between the cited literature and the present work.

Chapter 4 presents the technologies used with their descriptions and criteria for why these were chosen: the techniques for extraction and processing of time series, with a description of each together with snippets of code and a flow chart.

Chapter 5 describes the experiments, visualizing the results and comparing the algorithms.

Finally, Chapter 6 presents the conclusions the results of the experiments in comparison with the hypothesis and describes areas of opportunity for future work.

Chapter 2

Background

This chapter describes the concepts used in this research. In Section 2.1, the concepts of probability and statistics, performed abstractly in the preprocessing of the data and in the learning algorithms, are introduced, Section 2.2 describes the time series; types and examples. Section 2.3 shows the different forms of preprocessing used while Section 2.4 explains how to explore the parameters. And finally, Section 2.5 introduces artificial intelligence, explaining the type of learning, the purpose, and a brief description of learning algorithms.

2.1 Probability and statistics

Probability and statistics [7] are two fields in mathematics that are the fundamental artificial intelligence. They help to measure, estimate, and describe the used data.

2.1.1 Probability

Probability quantifies the possibility that an event happens among several possibilities. It measures the frequency of a possibility that an event happens, usually measured with continuous values $0 \leq p \leq 1$, in percentages from 0% to 100%, or in terms of fractions such as $\frac{1}{4}, \frac{1}{2}$, or 1. For example, the probability of obtaining a two by throwing a die is 0.167 rounded to three decimal places, or 16.67%, or $\frac{1}{6}$.

2.1.2 Statistics

Statistics studies and analyzes data samples to describe or infer a certain phenomenon in a random or conditional way. It is divided into two areas: descriptive and inferential.

Descriptive statistics describe a data set by measurements of characteristics. In a data set X with n elements, index starting at 1, some measures are: the minimum that is the first element of the data ordered in ascending order, the maximum as the largest element, the median is the element in the middle, and the mean that is the average obtained by:

$$\bar{X} = \frac{\sum_{i=1}^{n} X}{n}, \tag{2.1}$$

i.e., the sum of the elements divided by the number of elements. The variance is the measurement of how far the data are from their average:

$$\sigma^2 = \frac{\sum_{i=1}^{n} (X - \bar{X})}{n}. \tag{2.2}$$

Another measure is the standard deviation that measures the dispersion of the data:

$$\sigma = \sqrt{\sigma^2}; \tag{2.3}$$

this type of statistics is used to extract features from a time series, usually when the range of the time series is large.

Inferential statistics is used for estimating, inferring, forecasting, or predicting new points in time that have not been observed in a data set. These new points are usually the answer to a question or hypothesis: numerical estimates used to choose a certain action. In this work, this type of statistics is fundamental as learning algorithms are based on it.

2.2 Time series

A *time series* is a set of data observed in an interval from a start to an end time. The interval between of such measurements varies depending on the case; they can be seconds, minutes, hours, weeks, months, years, seasons. Examples of real life include:

- the amount of profit or loss in a business per week,
- the number of students approved in a semester, and
- the traffic on an avenue in the morning, afternoon or evening.

Some common statistical concepts to characterize a given time series include:

- *Trend* (Figure 2.1) is the way in which a series of time behaves, that is, decreases or increases.
- *Seasonality* (Figure 2.2) shows repetitions in periods in a lapse, an example is the high temperatures in the summer season in the year.
- *Noise* (Figure 2.3) are data points that do not follow a trend or distribution.
- *Movement* or *distribution* (Figure 2.4) shows the behavior, for example: quadratic, exponential.

Finally, time series are classified also in two types of models:

1. Models that represent currents values to predict past values or errors: *Auto Regressive Integrated Moving Average* (ARIMA) models (Figure 2.5).
2. Models that help for a description of the data; these use time indices as horizontal axis such as timestamps, dates or the order of a list. These are *ordinary models* (Figure 2.6).

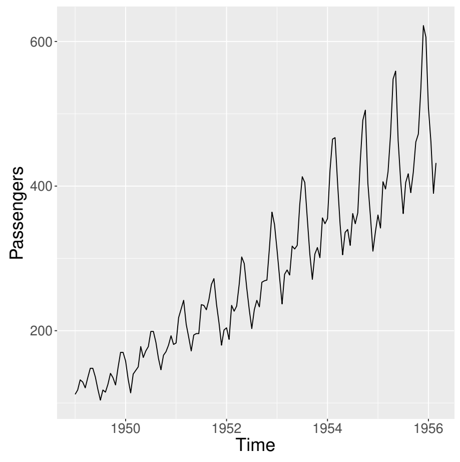


Figure 2.1: *Example of an increasing trend*

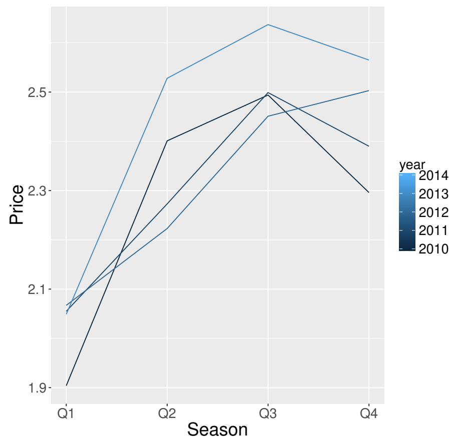


Figure 2.2: *Example of seasonality for years*

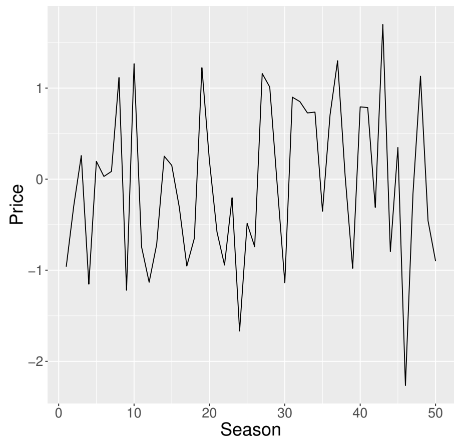


Figure 2.3: *Example of noise, with no trend and no seasonality*

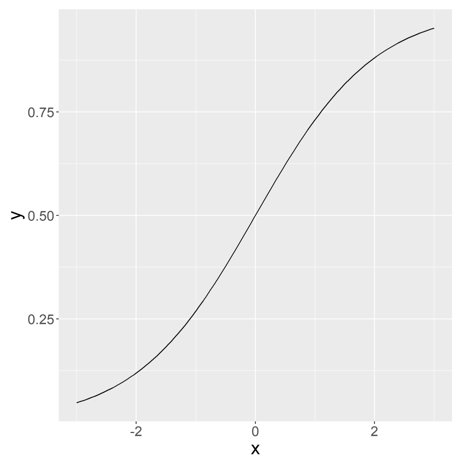


Figure 2.4: *Example of a logistic function*

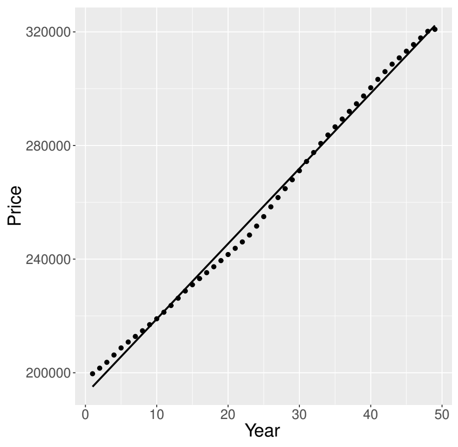


Figure 2.5: *An ARIMA example, predicting values*

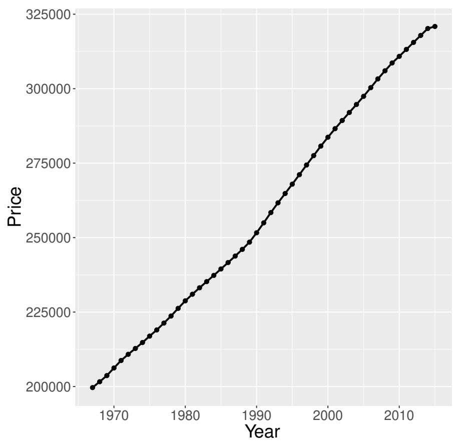


Figure 2.6: *Ordinary example, date as index*

2.3 Preprocessing

Preprocessing is a stage where the data is cleaned, normalized, and filled. Features are extracted and the relevant ones are selected.

Data cleaning is the process where the missing values are cleaned. These values are missing either because a user did not enter data, there was a failure in the moment of entering it, or they are not contemplated. The missing data are cleaned by filling them with an average or an extrapolation according to how the information is distributed in an instance. There are also cases where an entire instance contains missing data; these instances are ignored.

Transformation of data is the process where important features [Grandell, 1998] are obtained from a transformation formula. Two examples used in this thesis are:

- ***Discrete Fourier Transform (DTF)*** [Bloomfield, 2004] Fourier analysis describes a function as a sum of components and recovers them. When the function and its transform are discretely separated, it is called a discrete Fourier transform. In other words, it separates the inputs into components that create discrete frequencies. The output is called a spectrum or transform, and it is in the frequency domain. It is defined by a set N of complex numbers $X_1, \ldots, X_N$ is transformed by:

$$X_k = \sum_{n=1}^{N} X_n e^{-\frac{2\pi i}{N} kn}, \tag{2.4}$$

  where i is an imaginary unit and $e^{\frac{2\pi i}{N}}$ is the root of unit.

- ***Discrete Wavelet Transform (DWT)*** [Shensa, 1992] removes noise and compresses signals and images. These are wavelet transformations in a discrete way: oscillations with an amplitude that starts from zero, grows, and decreases to zero. Its advantage over Fourier transforms is the location in real space.

Normalization is the process of scaling a set of data so that a learning algorithm improves in its processing times. These algorithms tend to have poor behavior in results and time if the input values are not scaled. In this case, the standardization used based on the standard deviation for each feature:

$$X' = \frac{X - \bar{X}}{\sigma}. \tag{2.5}$$

Finally, it is common to *extract* important features to reduce the size of each instance by applying dimensions reduction algorithms. These contain information of the originals. An example is to obtain statistics such as minimum, maximum, average, standard deviation, or variance. On the other hand, there are unsupervised learning algorithms such as principal component analysis (PCA), a dimension reduction algorithm using the decomposition of singular values. It is based on the decomposition of eigenvalues of a covariance matrix, which is calculated by:

$$cov(\mathbf{X}) = \frac{\mathbf{X}^{\intercal}\mathbf{X}}{n-1}, \tag{2.6}$$

$$cov(\mathbf{X})p_a = \lambda_a p_a, \tag{2.7}$$

$$\sum_{a=1}^{m} \lambda_a = 1. \tag{2.8}$$

The eigenvalues are vectors that when multiplied by the operator give a scalar multiple of themselves:

$$\mathbf{A}v = c\mathbf{v}, \tag{2.9}$$

$$v \neq 0, c \in \mathbb{R}, \tag{2.10}$$

The projections of X in p_a measure captured variance on these vectors and t_a represents scores that contain information on how the information is related then have the property of being orthogonal that is a generalization of perpendicularity.

2.4 Artificial intelligence

The objective of *artificial intelligence* (AI) [Russell and Norvig, 2003] is to give a computer the ability to learn from conditions for a certain task to save time, resources, or discover behaviors.

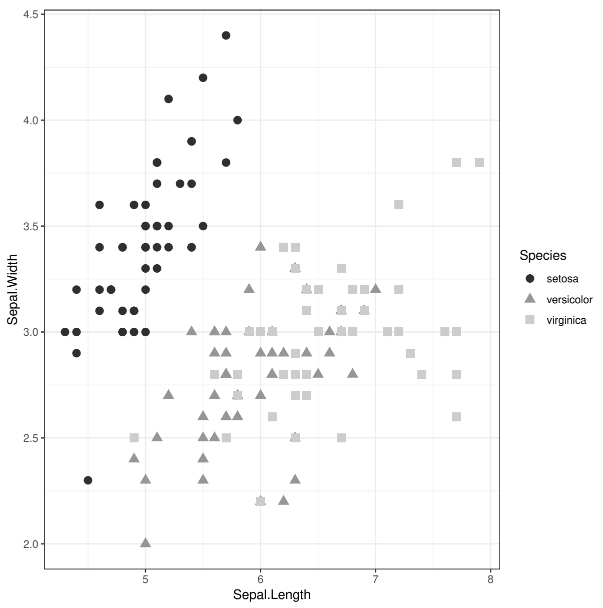


Figure 2.7: *Example of classifying data points using the iris dataset [8]*

Machine learning (ML) [Mitchell, 1997] gives machines the ability to learn without being granted a function or rules. It detects patterns in data. The machine learning algorithms are divided are supervised, unsupervised.

Supervised learning [Cunningham et al., 2008] focuses on learning to find an output which learns because a target is provided previously, this type of learning is divided into classification and regression algorithms.

Classification is the process of joining instances with similar characteristics (Figure 2.7). *Regression* is the process of predicting new values a sequence of data; this is learned from the input and output data provided to find a function that fits (Figure 2.8).

Unsupervised learning aims to group data into n groups that have similar features; this means the instances do not have a label which explains in which group it belongs. It is also possible to find similarities in sequences of tasks in order to find repetitive behavior. Another of its applications are to remove elements out of the ordinary that does not follow a trend to that of others.

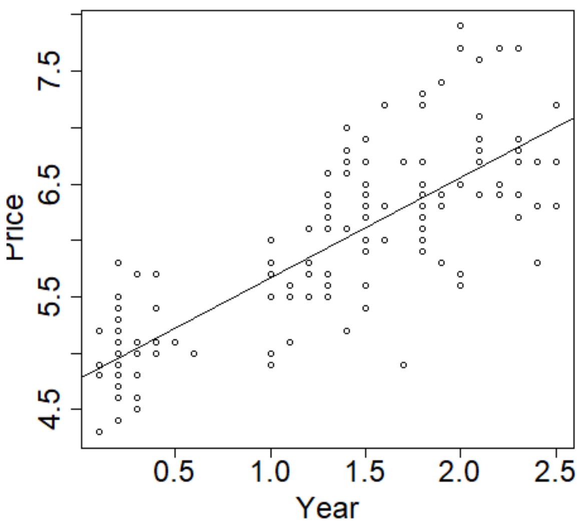


Figure 2.8: *Example of predicting new data points in linear regression using the iris dataset [8]*

Semi-supervised learning [Chapelle et al., 2010] uses both tagged and untagged data, the amount of tagged data is usually a small fraction of the data set in order to noticeably improve learning accuracy and reduce the time costs of tagging instances.

This thesis focuses partially on semi-supervised learning, given that the outputs are obtained from certain rules depending in a criteria in Chapter 4.

Logistic Regression [Harrell, 2001] is a classification algorithm used to assign instances of a data set to a discrete set of classes. It transforms its output using a logistic sigmoid function to calculate a probability value that will be assigned to two or more classes. The sigmoid function is an activation function. that assigns probabilistic values to real values between 0 to 1:

$$S(z) = \frac{1}{1 + e^{x}}, \tag{2.11}$$

where $S(z)$ is a probability between 0 and 1, z is an activation function and e is a base of natural log. New values are predicted with:

$$y = wx + b, \tag{2.12}$$

where y is the result, w is the weight, and b the bias. Aiming to reduce the loss function is calculated by:

$$L(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(1)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]; \tag{2.13}$$

if $y = 0$, the first side cancels out, and if $y = 1$ the second side cancels out, so there are two separate loss functions for each solution:

$$L(h_\theta(x), y) = -\log(h_\theta(x)) \tag{2.14}$$

if $y = 1$, and

$$L(h_\theta(x), y) = -\log(1 - h_\theta(x)) \tag{2.15}$$

if $y = 0$. Once the probabilities are obtained, 1 or 0 is assigned based on the following conditions:

$$p \geq 0.5, y = 1, \tag{2.16}$$

$$p \leq 0.5, y = 0. \tag{2.17}$$

This algorithm uses gradient descent used to iteratively minimize the loss function. The loss function seeks to find the weight and bias values, that is, from the derivative of the sigmoid function:

$$s'(z) = s(z)(1 - s(z)). \tag{2.18}$$

Multilayer Perceptron (MLP) [Kruse et al., 2016] is an algorithm that learns an approximate non-linear function for any classification or regression. It is different from to the logistic regression because here there are hidden abilities which allow learning non-linear models. To classify, a logistic function is used to obtain values between 0 and 1. If there are more than two classes, the output layer is a vector of n classes, and instead of passing a logistic function

$$g(x) = \frac{1}{(1 + e^{-z})}, \tag{2.19}$$

the softmax function given by

$$s(z)_i = \frac{\exp(z_i)}{\sum_{i=1}^{k}\exp(z_i)}, \tag{2.20}$$

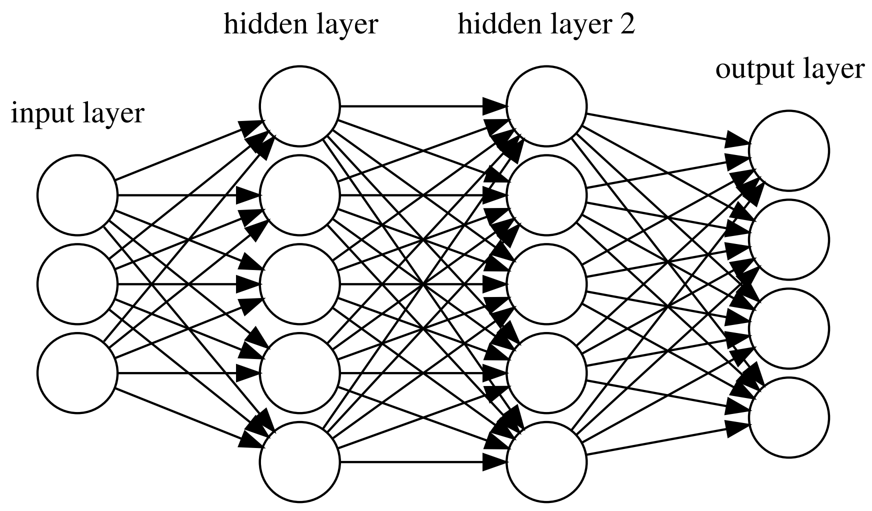


Figure 2.9: *An example of a multilayer perceptron*

is used, where z_i is the i class and k is the number of classes. Having a vector that contains the results in probabilities for each one, the element with the highest probability is the output. Although the algorithm is very similar to logistic regression, other changes are:

- in each layer, the sum of the weights is calculated, after that it goes to an activation function;
- the loss is calculated in the output layer;
- in the hidden layers, its respective error is calculated to then update its values for each iteration;
- the accumulated error is calculated as measurement metric of the whole model.

Support Vector Machine (SVM) [Cortes and Vapnik, 1995] creates hyperplanes in a finite set of dimensions, which makes it possible to classify using these hyperplanes depending on the distance closest to the points of each instance. The way SVM separates the data points is by kernel functions, splitting the dimensions in the space. Common kernel functions are:

- linear using (x, x'),
- polynomial $(\gamma(x, x') + r)^d$, and
- sigmoid $(\tanh(\gamma(x, x') + r))$.

Random Forest [Breiman, 2001] is a heuristic learning algorithm based on taking multiple decision trees with different characteristics such as number of nodes or depth of leaves. It allows a greater exploration in obtaining ideal parameters to obtain a good performance. These sets of trees search for a different solution independently, which allows parallelizing for greater exploration with a feasible time. The most common parameters to explore are the quantities of trees to be explored and the number of characteristics.

Chapter 3

Literature review

This chapter presents a summary of literature related to this thesis. This set of works presents different ways of processing the structuring and processing of a data set. Also, each of these presents different criteria depending on the problem to be solved. They also propose different ways of training a learning model.

Geurts [2001] proposes using classifiers such as Decision Trees, Naive Bayes and Boosting algorithms to find patterns in time series which are not recognizable to the single view or do not follow a pattern. This work solves problems from different sources to make a comparison between these algorithms in terms of accuracy.

Marti et al. [2016] propose using statistical methods to correlate financial time series windows, the objective of this research is to find feasible window sizes and to do an unsupervised training to group financial sources that have similar behaviors. It aims to find distributions from these correlations that adapt to the greatest number of sources.

Xiong and Yeung [2002] propose to create a combination of coefficients by grouping different models and sources to find parameters that generally work for any problem. The number of groups is calculated by minimizing the distances of each instance with a group with similar characteristics.

Ratanamahatana and Keogh [2004] focus on finding better solutions using Dynamic Time Warping (DTW) instead of the Euclidean distance for classification or clustering tasks. These two techniques are used to measure accuracy in the results when grouping instances with the same characteristics.

Liao [2005] provides a summary of algorithms for unsupervised learning, discussing selection and extraction of parameters by similar characteristics with greater impact in improving accuracy or by reducing parameters for large data set, and explains results of grouping algorithms, using different systems to minimize distances between the centroids of each group with their instances.

Wei and Keogh [2006] focus on classifying instances in large data sets where classes or categories were not provided. This is semi-supervised learning: general rules are set for time series with expected results or use grouping learning algorithms to group instances without category with those that have better similar characteristics with instances with a category already planned.

The related works apply different methods of feature extraction and data processing. We determined for each work the presence or absence of techniques and phases relevant to our proposed tool:

- **Transformation**: whether the data is transformed or scaled.
- **Features extraction**: whether feature extraction is applied.
- **Statistical clustering**: whether statistical clustering takes place.
- **Supervised learning**: whether pre-labeled training inputs are used.
- **Unsupervised learning**: whether learning is done without pre-labeled training data.
- **Semi-supervised learning**: whether only a small fraction of the training data is labeled.

- **Hyperparameter**: whether parameter adjustment is carried out.
- **Pipeline**: whether multiple algorithms are compared.
- **Multi-sources**: whether different types of time series (such as financial, medical, weather) are considered.

Table 3.1 shows a comparison between the cited works; most use a specific algorithm for a unique dataset. Also, most use machine learning methodologies (supervised and unsupervised), using a fixed parameter configuration for each one. Only two explore multiple algorithms. When it comes to supervised learning methods, Geurts [2001] as well as Ratanamahatana and Keogh [2004] work with multiples algorithms and metrics to identify the best one for a particular problem, whereas Xiong and Yeung [2002] implement unsupervised learning algorithm to cluster and finding the best number of classes. Wei and Keogh [2006] label the data using semi-supervised learning while Liao [2005] applies both supervised and unsupervised learning algorithms.

The advantage of this work is the use of a search of the best parameters with a set of ranges and those that better adjust to the focused problem, the minimization of characteristics using preprocessing of time series to scale them and then using their statistics which provides improvement in time performance, and finally finding the best learning model for each type of case and one in general that can provide a feasible accuracy.

Table 3.1: *Comparison of related literature.*

Work/Feature	Transformation	Feature extraction	Statistical learning	Supervised learning	Unsupervised learning	Semi-supervised learning	Hyperparamter	Pipeline	Multi-source
Geurts [2001]	✓	✓	✗	✓	✗	✗	✗	✓	✓
Xiong and Yeung [2002]	✓	✓	✗	✗	✓	✗	✓	✗	✓
Ratanamahatana and Keogh [2004]	✓	✓	✗	✓	✗	✗	✓	✓	✓
Liao [2005]	✓	✗	✗	✓	✓	✓	✗	✓	✗
Wei and Keogh [2006]	✓	✗	✗	✗	✗	✓	✓	✓	✗
Marti et al. [2016]	✓	✗	✓	✗	✗	✗	✓	✓	✗
Present work	✓	✓	✗	✓	✗	✗	✓	✓	✓
✓ Implemented — ✗ Not implemented									

Chapter 4

Proposed solution

This chapter describes the proposed solution for the problem of time-series classification: the methodology, the tools (programming languages, open-source libraries, and additional software), the input data used and their formats, and finally the implementation. The code, data, and plots of the present project are found in the Github repository: `https://github.com/pedrosanzmtz/TimeSeriesClustering`.

4.1 Methodology

This section explains the software development process, the procedures, techniques, tools, and development phases with the purpose of providing an outline of the solution proposed in this research.

4.1.1 Tools

The tools used in this research are:

- ***Python 3.4.5*** `https://www.python.org/` is an interpreted high-level programming language for general-purpose programming with a set of open-sources libraries specially for statistics, cleaning, manipulate data, and data mining.

- ***Numpy 1.11.3*** `http://www.numpy.org/` is a library for the Python programming language, providing support for large, multi-dimensional arrays and matrices, as well as high-level mathematical functions to operate on such structures.
- ***Scipy 0.18.1*** `https://www.scipy.org/` is an open-source Python library used for scientific computing and technical computing.
- ***Scikit-learn 0.18.1*** `https://scikit-learn.org/` is a machine-learning and data-mining module for Python that is distributed under the BSD license.
- ***PyWavelets 1.0.1*** `https://pywavelets.readthedocs.io/` provides an open-source implementation of the wavelet transform for Python.

4.1.2 Data

All of our input data is stored in Comma-Separated Values (CSV) files, where each row represents a single time-series instance from an initial time to a final time; non-available null data are indicated by a special character string. The metadata of each dataset is in Table 4.1.

The three data sets of our case studies are:

- ***Purchases*** that includes client transactions in terms of the number of purchased and consumed unitary services per month; the purpose is to identify ahead of time which clients will be lost, i.e., cease making purchases. This is a binary classification because the output is lost or retaines, the features used here are the number of purchases for client from January 2014 to March 2017 with a total number of 1909 records. In this case study the range of dates was reduced by three months at the beginning and at the end because most of the customers started at this date and others where just starting at the end of this dataset.

- ***Grades*** that includes grades obtained by students in distinct activities throughout a semester; the purpose is identify ahead of time which students will obtain a passing grade. This is a multilabel classification problem because the class is the opportunity where the student gets to obtain a passing grade or not at all, this is: first or second opportunity. The data contains seven columns where five are the scores in the homeworks in order from one to five, the sixth column is the score in the mid term exam and the last column is the class to predict: 1 for passing at the first opportunity, 2 for passing at the second opportunity, and 3 for not passing. This class depends on the final score on the student with the final project and the final exam, which are not included for training. The total number of records is 244 which is the number of students from semesters from January 2016 to December 2017.

- ***Diseases*** includes data from the web page: `http://www.epidemiologia.salud.gob.mx/anuario/html/anuarios.html` which reports epidemiological diseases per week for men and women for the 52 states of México. This case study is a binary classification for diseases transmitted via respiratory and via midge picket. From this web page, these data were extracted from the year 2008 to the year 2013. The diseases are: Zika, Shikungunya, Hemorrhagic Dengue, and Classic Dengue: These four diseases are transmitted by means of a mosquito through a picket, these diseases were classified as *midge*. The diseases classified as *respiratory* include Influenza virus, Acute Respiratory Influenza virus, and Pneumonias: these diseases were extracted from 2014 to 2017 for the states of Nuevo León, Tamulipas, Coahuila, and the federal total. Because in the data respiratory diseases does not include all states and is not separated as gender, these are not included as features. The total number of records is 4900, which includes the number of cases reported for 13 weeks as columns, these 13 weeks are a season of the year, given that each case behaves

Table 4.1: *Metadata of each dataset.*

Case	Instances	Attributes	Classes
Purchases	1909	162	2
Grades	244	7	3
Diseases	4900	13	2

differently in each season.

4.1.3 Phases

In this section it is discussed how the data is processed before it is classified using the machine learning algorithms to get a feasible performance in time and accuracy.

4.1.3.1 Pre-processing

Data pre-processing [Hand, 2007] is a data-mining technique to transform raw input data into a format understandable by latter processing phases. Real-world data is usually incomplete, possibly inconsistent, and may also lack certain behaviors or trends; additionally it is quite likely to contain errors. The purpose of pre-processing is to minimize the impact of such issues in the output. Figure 4.1 shows a time serie before preprocessing and Figure 4.2 after; the differences are the discretited and scaled values.

- ***Fill missing values***, consisting in the transformation of missing values [Gómez et al., 1992] with a consistent option
  - fill with a zero,
  - fill with the mean,
  - fill with the median, or

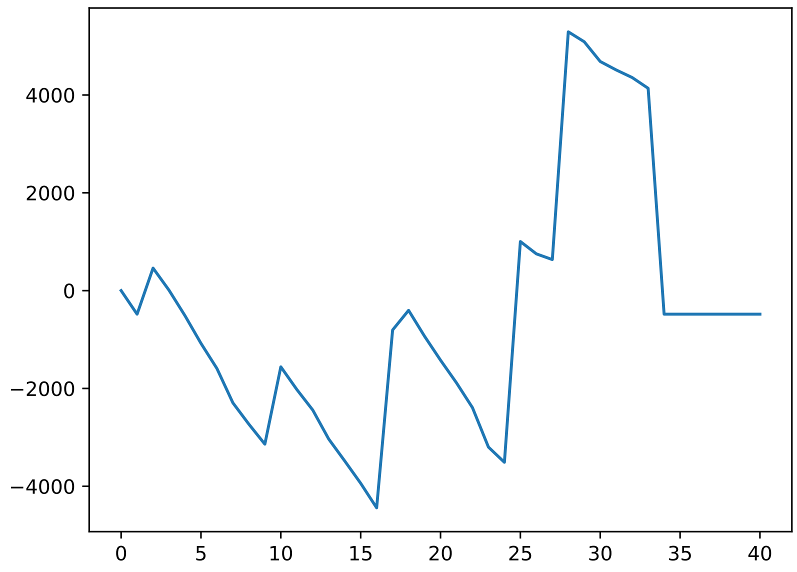


Figure 4.1: *Example of time serie before preprocessing*

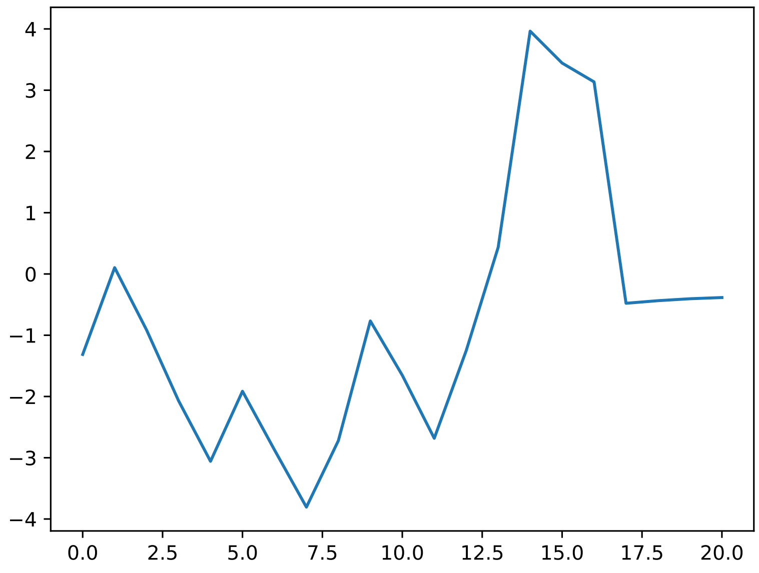


Figure 4.2: *Example of time serie after preprocessing*

  - fill with the most frequent value.

- ***Scaling***, meaning the standardization Boeva and Tsiporkova [2010] of data (a rather common requirement for machine-learning algorithm that can behave poorly if any individual feature does not approximately follow a standard normal distribution (that is, Gaussian with zero mean and unit variance).

4.1.3.2 Feature extraction

Feature extraction [Guyon and Elisseeff, 2006] is the transformation of data into a set of characteristics. The purpose is to extract important characteristics and minimize the the number of them.

To simplify the components obtained in feature extraction, basic statistics are used descriptors of their distributions: minimum, maximum, variance, mean, skewness, and kurtosis.

Another step is to remove one slice of time, represented by a unit of time or period such as semester, season, day, week, or month. The purpose of removing time periods is for exploring the accuracy depending in the size of attributes; this provides knowledge to choose which period of time use to predict an event.

4.2 Implementation

This subsection explains the most relevant snippets of Python code: importing libraries, reading and pre-processing the input, and classifying the time series.

- The code snippet 4.3 shows the modules used in the project, these imports are in the pipelinegridesearch.py script which create pipelines and contains the logic for the learning part.

- The code snippet 4.4 shows the structure of each model, this is by creating a dictionary (just a hashmap) which contains the object for each model, the random state which is used to reproduce the same results each running time and the standard scaler object which makes de scaling in automatic.

- The code snippet 4.5 returns a dictionary search objects which contain the parameters to iterate over each model.

- The code snippet 4.6 generates pipelines with the parameters and structure for each model.

- The code snippet 4.7 retrieves the best results found for each model, and the best parameters used for those results.

- The code snippet 4.8 calls the function to init the pipeline and gridsearch dictionaries, then starts training the models and saves the results in csv files.

- The code snippet 4.9 replaces NA values for the mean in each instance, then transforms the data using DWT, then finally returns the summary of each record.

- The code snippet 4.10 contains the main function for the purchases case where gets the data from the CSV file, then calls the function *get_data()* which removes the first 3 months and the last 3 months, this is to ensure that almost all the clients started at the same date and also finished. The function also split the features columns X and the class y and return them.

- The code snippet 4.11 contains the main function which initializes the inputs to save the data, calls the *get_data()* function thats splits the columns into features X and the columns y classes, the y value has three values to specify which opportunity the student passed the subject. The main function iterates over from the second homework until the last homework.

- The code snippet 4.12 contains the main function for the diseases case that initializes the variables used and calls the *get_data()* function which removes

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import Imputer
from sklearn import svm
from sklearn.metrics import mean_squared_error
from pprint import pprint
import numpy as np
import pandas as pd
from time import time
from scipy import stats
import pywt
```

Figure 4.3: *Code snippet for importing libraries*

```
def get_pipes(random_state):
  pipes = dict()
  pipes["lr"] = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(random_state=random_state))])

  pipes["rf"] = Pipeline([('scl', StandardScaler()),
                        ('clf', RandomForestClassifier(random_state=random_state))])

  pipes["svm"] = Pipeline([('scl', StandardScaler()),
                        ('clf', svm.SVC(random_state=random_state))])

  pipes["mlp"] = Pipeline([('scl', StandardScaler()),
                        ('clf', MLPClassifier(random_state=random_state))])
  return pipes
```

Figure 4.4: *Code snippet for creating pipelines*

records with all values are NA or 0 and return the X and y values. The main function iterates over six weeks, this because a disease can take at least a month to present symptoms.

```
def get_grids():
  grids = dict()
  grids["lr"] = [{'clf__penalty': ['l2'],
                  'clf__C': [0.3, 0.5, 1],
                  'clf__solver': ['lbfgs', 'newton-cg']}]

  grids["rf"] = [{'clf__criterion': ['gini', 'entropy'],
                  'clf__n_estimators': [10, 20, 30]}]

  grids["svm"] = [{'clf__kernel': ['sigmoid', 'rbf'],
                   'clf__C': [0.3, 0.5, 1]}]

  grids["mlp"] = [{'clf__activation': ['logistic', 'relu'],
                   'clf__solver': ['sgd', 'adam'],
                   'clf__hidden_layer_sizes': [(10, ), (10, 5)]}]
  return grids
```

Figure 4.5: *Code snippet for gridsearch dictionaries*

```
def generate_pipeline(cv, jobs=-1, random_state=42):
  # Construct some pipelines
  pipes = get_pipes(random_state)
  grids = get_grids()

  # Construct grid searches
  gs_lr = GridSearchCV(estimator=pipes["lr"],
                       param_grid=grids["lr"],
                       scoring='accuracy',
                       cv=cv)

  gs_rf = GridSearchCV(estimator=pipes["rf"],
                       param_grid=grids["rf"],
                       scoring='accuracy',
                       cv=cv,
                       n_jobs=jobs)

  gs_svm = GridSearchCV(estimator=pipes["svm"],
                        param_grid=grids["svm"],
                        scoring='accuracy',
                        cv=cv,
                        n_jobs=jobs)

  gs_mlp = GridSearchCV(estimator=pipes["mlp"],
                        param_grid=grids["mlp"],
                        scoring='accuracy',
                        cv=cv)

  # List of pipelines for ease of iteration
  grids = [gs_lr, gs_rf, gs_svm, gs_mlp]
  # Dictionary of pipelines and classifier types for ease of reference
  grid_dict = {0: 'LR',
               1: 'RF',
               2: 'SVM',s
               3: 'MLP'}
  return (grids, grid_dict)
```

Figure 4.6: *Code snippet to generate pipeline*

```
def get_results(means, stds, gs, gs_out):
  for mean, std, params in zip(means, stds, gs.cv_results_['params']):
      print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
      for p in params:
          gs_out.write(str(params[p]) + ',')
      gs_out.write(str(mean) + '\n')
  gs_out.close()
```

Figure 4.7: *Code snippet to get the best results for each model*

```python
def run_pipeline(X, y, cv, out, sub, name):
  # Fit the grid search objects
  (grids, grid_dict) = generate_pipeline(cv, jobs=-1,
                                         random_state=42)
  X_train, X_test, y_train, y_test = train_test_split(X, y,
                                         test_size=0.33, random_state=42)
  y_test = y_test.values
  print('Performing model optimizations...')
  best_acc = 0.0
  best_clf = 0
  best_gs = ''
  for idx, gs in enumerate(grids):
      initial_time = time()
      gs_out_name = grid_dict[idx] + '_' + name + '.csv'
      gs_out = open(gs_out_name, 'w')
      gs_out_count_name = "count_" + gs_out_name
      gs_out_count = open(gs_out_count_name, "w")
      print('\nEstimator: %s' % grid_dict[idx])
      # Fit grid search
      toi = time()
      gs.fit(X_train, y_train)
      tof = time() - toi
      # Best params
      print('Params Test:')
      means = gs.cv_results_['mean_test_score']
      stds = gs.cv_results_['std_test_score']

      get_results(means, stds, gs, gs_out)

      print('Best params: %s' % gs.best_params_)
      # Best training data accuracy
      print('Best training accuracy: %.3f' % gs.best_score_)
      # Predict on test data with best params
      y_pred = gs.predict(X_test)

      count_results(gs_out_count, y_pred, y_test)

      # Test data accuracy of model with best params
      acc = '%.2f' % accuracy_score(y_test, y_pred)
      mse = '%.3f' % mean_squared_error(y_test, y_pred)
      total = str(y_test.shape[0])
      acc_n = str(int(y_test.shape[0] * float(acc)))
      total_time = '%.2f' % (time() - initial_time)
      print('Test set accuracy score for best params:', acc)
      print('Test set mse score for best params:', mse)
      print('Total time:', total_time)
      # Track best (highest test accuracy) model
      out.write(grid_dict[idx] + ',' + sub + ',' + acc + ','
      + mse + ',' + total_time + '\n')
```

Figure 4.8: *Code snippet to execute pipeline*

```python
def preprocess(X, na_values):
  imp = Imputer(missing_values=na_values, strategy='mean', axis=1)
  imp.fit(X)
  X = imp.transform(X)
  X, _ = pywt.dwt(X, 'db1')
  X_stats = stats.describe(X, axis=1)
  X = pd.DataFrame({'min': X_stats.minmax[0],
                    'max': X_stats.minmax[1],
                    'kurtosis': X_stats.kurtosis,
                    'skwness': X_stats.skewness,
                    'variance': X_stats.variance,
                    'mean': X_stats.mean})
  return X
```

Figure 4.9: *Code snippet for pre-processing*

```
import pandas as pd
import numpy as np
from pipeline_gridsearch import *

def get_data():
  df = pd.read_csv('inventariosActivos.csv', index_col=0, na_values=['NaN', 'NA'])
  # Remove client column
  df = df.drop(['client'], axis=1)

  X = df.iloc[:, :-1]
  y = df.iloc[:, -1]

  # Remove first 3 months and last 3 months
  X = X.iloc[:, 3:-3]

  X = X.dropna(how='all', axis=0)
  return X, y

def main():
  X, y = get_data()

  na_index = X.isnull().all(axis=1)

  out = open('inventories_performance.csv', 'w')
  out.write('model,slice,acc,mse,time\n')

  n_months = 6
  name = 'inventories'
  na_values = np.nan

  for i in range(1, n_months + 1):
      sub = str(i) + 'M'
      sub_X = X.iloc[:, :-i]
      sub_X = sub_X.dropna(how='all', axis=0)
      indx = sub_X.index
      sub_y = y[indx]
      sub_X = preprocess(sub_X, na_values)
      run_pipeline(sub_X, sub_y, 10, out, sub, name)

  out.close()
```

Figure 4.10: *Code snippet for the purchases case study*

```python
from sklearn import preprocessing
import pandas as pd
import numpy as np
from pipeline_gridsearch import *

def get_data():
  df = pd.read_csv('grades.csv')
  X = df.iloc[:, :-1]
  y = df.iloc[:, -1]
  return X, y

def main():
  X, y = get_data()
  out = open('grades_performance.csv', 'w')
  out.write('model,slice,acc,mse,time\n')

  name = 'grades'
  na_values = 'NaN'

  for i in range(2, 6):
      cols = ['c1', 'mc']
      sub = str(i) + "H"
      for j in range(2, i+1):
          cols.append('c' + str(j))
      sub_X = preprocess(X[cols], na_values)
      run_pipeline(sub_X, y, 10, out, sub, name)
  out.close()
```

Figure 4.11: *Code snippet for the grades case study*

```python
from sklearn import preprocessing
import pandas as pd
import numpy as np
from pipeline_gridsearch import *

def get_data():
    df = pd.read_csv("diseases.csv")
    X = df.iloc[:, :-1]
    X_true = X.sum(axis=1) > 0
    X = X[X_true]
    le = preprocessing.LabelEncoder()
    y = pd.Series(le.fit(df['class']).transform(df['class']))
    y = y[X_true]
    return X, y

def main():
    X, y = get_data()
    out = open('diseases_performance.csv', 'w')
    out.write('model,slice,acc,mse,time\n')
    n_weeks = 6
    name = 'diseases'
    na_values = np.nan
    for i in range(1, n_weeks + 1):
        sub = str(i) + 'W'
        sub_X = X.iloc[:, :-i]
        sub_X = sub_X.dropna(how='all', axis=0)
        indx = sub_X.index
        sub_y = y[indx]
        print("subX shape:", sub_X.shape)
        print("suby shape:", y.shape)
        sub_X = preprocess(sub_X, na_values)
        run_pipeline(sub_X, sub_y, 10, out, sub, name)
```

Figure 4.12: *Code snippet for diseases case*

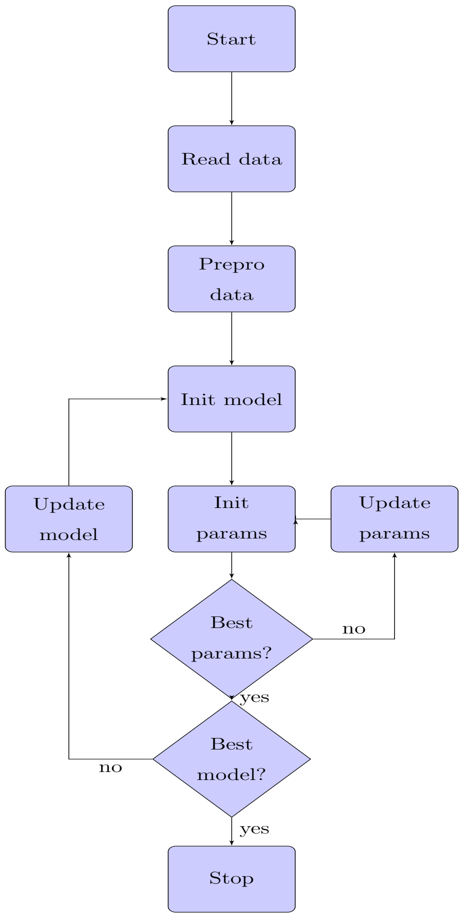


Figure 4.13: *Flow chart of tool; each block is a subroutine that can be used to train multiples classifiers with different data sources.*

Chapter 5

Experiments

The purpose of computational experiments is to assess the performance of a proposed solution and the impact of any potential configurations on that performance. Experimental design [Montgomery, 2006] aims at an experimental setup that produces objective and valid conclusions. It is also important to ensure that the results of the experiments will permit the validation or rejection of the hypothesis. For the problem at hand, the next experiment is necessary: a classification test to verify whether an algorithm obtains the correct class to each instance of input data and compute the accuracy and MSE of the method.

As the distribution of the input data depends on its origin and the amount of data in each case study varies, a different method and/or a different parameter configuration may be the best of option for one than for the other.

To evaluate the accuracy and MSE of each classifier in each of the three case studies, we prepared statistical reports and visualizations for each.

The performance of each algorithms depends on the parameter configuration; the values of the parameters were chosen with the methods provided by the scikit-learn package so that the results were feasible using the mean of a cross validation of ten iterations.

For *logistic regression*, two parameters were tuned: C which is the inverse of regularization strength (smaller values specify stronger regularization) and the choice of *solver*, i.e., the algorithm implementation to be used in solving the the optimization problem incorporated in the method. Table 5.4a displays the accuracy for the three case studies — the results are quite similar for each set, although higher values of C result in slightly better accuracy. We underline the best value of accuracy for each case study. Table 5.5a displays the MSE, in this case, using the value 1 for C and using any of the solvers, it gets the best results in MSE for the three cases.

The results for SVM are given in Table 5.4b; SVM has no use for a solver, but instead has several options for kernel functions — for our case studies, the sigmoid kernel is the best for the grades data whereas the "rbf" kernel works best for the purchases data set; C is still the regulation strength. Table 5.5b displays MSE for SVM where it seems that using the values 0.5 or 1.0 for C and the solver "sigmoid" gets the best results.

For the random forest, the results are shown in Table 5.4c: the parameters to vary are the criterion and the number of estimators. The entropy criterion requires an elevated number of estimators to increase its accuracy and is better for the three case studies. Table 5.5c displays the MSE where is getting better results using minimun number of estimators, even though the elevated number of estimators also works.

Table 5.4d shows the results for the multi-layer perceptron model in terms of the combination of solvers, activation functions, and the number of nodes in each of the two layers. Stochastic gradient descent does not help, whereas the "adam" optimizer yields the best results using the logistic activation function for both case studies. Table 5.5d displays low errors using minimun size of neuros, even though the accuracies are not better with these results.

We now further experiment with the best parameter configuration of each model for each case study to determine the effect of the slice length (that is, the

Table 5.1: *Confusion matrices for grades case.*

(a) *Confusion matrix for logistic regression model.*

actual / predicted	1st attempt	2nd attempt	Failed
1 opportunity	36	8	6
2 opportunity	0	2	0
Not passing	1	2	26

(b) *Confusion matrix for SVM model.*

actual / predicted	1st attempt	2nd attempt	Failed
1 opportunity	34	4	6
2 opportunity	2	4	3
Not passing	3	2	25

(c) *Confusion matrix for random forest model.*

actual / predicted	1st attempt	2nd attempt	Failed
1 opportunity	36	8	6
2 opportunity	0	2	0
Not passing	1	2	26

(d) *Confusion matrix for multi-layer perceptron model.*

actual / predicted	1st attempt	2nd attempt	Failed
1 opportunity	34	10	6
2 opportunity	0	2	0
Not passing	2	3	24

Table 5.2: *Confusion matrices for purchases case.*

(a) *Confusion matrix for logistic regression model.*

predicted \ actual	Lost	Not lost
Lost	208	76
Not lost	216	98

(b) *Confusion matrix for SVM model.*

predicted \ actual	Lost	Not lost
Lost	181	44
Not lost	248	125

(c) *Confusion matrix for random forest model.*

predicted \ actual	Lost	Not lost
Lost	225	80
Not lost	212	81

(d) *Confusion matrix for multi-layer perceptron model.*

predicted \ actual	Lost	Not lost
Lost	209	71
Not lost	221	97

Table 5.3: *Confusion matrices for diseases case.*

(a) *Confusion matrix for logistic regression model.*

actual / predicted	Midge	Respiratory
Midge	951	67
Respiratory	42	0

(b) *Confusion matrix for SVM model.*

actual / predicted	Midge	Respiratory
Midge	951	67
Respiratory	42	0

(c) *Confusion matrix for random forest model.*

actual / predicted	Midge	Respiratory
Midge	956	33
Respiratory	76	5

(d) *Confusion matrix for multi-layer perceptron model.*

actual / predicted	Midge	Respiratory
Midge	951	67
Respiratory	42	0

Table 5.4: *Accuracies of the methods under different parameter configurations.*

(a) *Accuracy for logistic regression model*

C	Solver	Grades	Purchases	Diseases
0.3	lbfgs	0.773	<u>0.686</u>	0.928
0.3	newton-cg	0.773	<u>0.686</u>	0.928
0.5	lbfgs	0.773	0.685	0.934
0.5	newton-cg	0.773	0.685	0.934
1.0	lbfgs	<u>0.779</u>	0.685	<u>0.938</u>
1-0	newton-cg	<u>0.779</u>	0.685	<u>0.938</u>

(b) *Accuracy for SVM model*

C	Kernel	Grades	Purchases	Diseases
0.3	sigmoid	0.773	0.662	0.915
0.3	rbf	0.773	0.681	0.931
0.5	sigmoid	<u>0.779</u>	0.661	0.909
0.5	rbf	0.773	0.680	0.936
1.0	sigmoid	<u>0.779</u>	0.655	0.909
1.0	rbf	0.773	<u>0.685</u>	<u>0.940</u>

(c) *Accuracy for random forest model*

Criterion	Est. #	Grades	Purchases	Diseases
gini	10	0.766	0.679	0.961
gini	20	<u>0.779</u>	<u>0.688</u>	0.963
gini	30	<u>0.779</u>	<u>0.688</u>	0.961
entropy	10	0.766	0.675	<u>0.964</u>
entropy	20	0.766	0.682	0.963
entropy	30	<u>0.779</u>	<u>0.688</u>	<u>0.964</u>

(d) *Accuracy for multi-layer perceptron model*

L_1	L_2	Activation	Solver	Grades	Purchases	Diseases
10	0	logistic	sgd	0.466	0.641	0.905
10	0	logistic	adam	0.742	0.693	0.915
10	5	logistic	sgd	0.533	0.535	0.905
10	5	logistic	adam	0.754	0.692	0.905
10	0	relu	sgd	0.668	0.679	0.915
10	0	relu	adam	0.656	0.693	0.938
10	5	relu	sgd	0.748	0.665	0.915
10	5	relu	adam	<u>0.766</u>	<u>0.706</u>	<u>0.941</u>

Table 5.5: *MSE of the methods under different parameter configurations.*

(a) *MSE for logistic regression model*

C	Solver	Grades	Purchases	Diseases
0.3	lbfgs	0.088	0.065	0.11
0.3	newton-cg	0.088	0.065	0.11
0.5	lbfgs	0.088	0.065	0.12
0.5	newton-cg	0.088	0.065	0.12
1.0	lbfgs	0.060	0.076	0.10
1-0	newton-cg	0.060	0.076	0.10

(b) *MSE for SVM model*

C	Kernel	Grades	Purchases	Diseases
0.3	sigmoid	0.088	0.043	0.10
0.3	rbf	0.088	0.052	0.14
0.5	sigmoid	0.076	0.042	0.19
0.5	rbf	0.088	0.056	0.12
1.0	sigmoid	0.076	0.054	0.19
1.0	rbf	0.088	0.054	0.11

(c) *MSE for random forest model*

Criterion	Est. #	Grades	Purchases	Diseases
gini	10	0.128	0.059	0.17
gini	20	0.140	0.058	0.19
gini	30	0.128	0.048	0.18
entropy	10	0.134	0.040	0.14
entropy	20	0.134	0.048	0.17
entropy	30	0.140	0.031	0.19

(d) *MSE for multi-layer perceptron model*

L_1	L_2	Activation	Solver	Grades	Purchases	Diseases
10	0	logistic	sgd	0.033	0.032	0.004
10	0	logistic	adam	0.114	0.067	0.10
10	5	logistic	sgd	0.033	0.003	0.004
10	5	logistic	adam	0.125	0.060	0.004
10	0	relu	sgd	0.066	0.057	0.10
10	0	relu	adam	0.099	0.059	0.009
10	5	relu	sgd	0.117	0.061	0.10
10	5	relu	adam	0.101	0.057	0.13

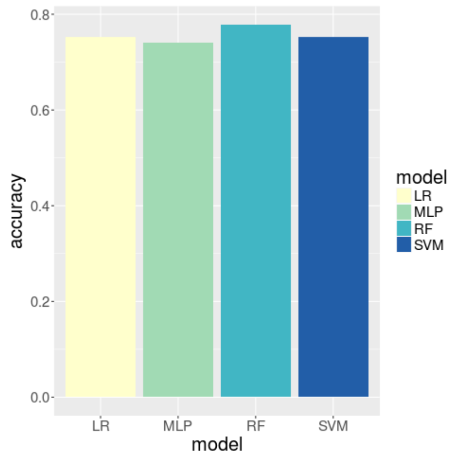
Table 5.6: *Performance of the classification methods for the grades case study using homework slices of various lengths*

Model	Slice	Accuracy	MSE	Correct	Incorrect
LR	2	0.753	0.506	60	11
RF	2	0.778	0.444	63	8
SVM	2	0.753	0.543	60	11
MLP	2	0.741	0.593	60	11
LR	3	0.778	0.407	63	8
RF	3	0.778	0.407	63	8
SVM	3	0.765	0.494	61	10
MLP	3	0.802	0.496	64	7
LR	4	0.790	0.469	63	8
RF	4	0.802	0.383	64	7
SVM	4	0.790	0.469	63	8
MLP	4	0.765	0.494	61	10
LR	5	0.790	0.469	63	8
RF	5	0.778	0.407	63	8
SVM	5	0.790	0.469	63	8
MLP	5	0.765	0.494	61	10

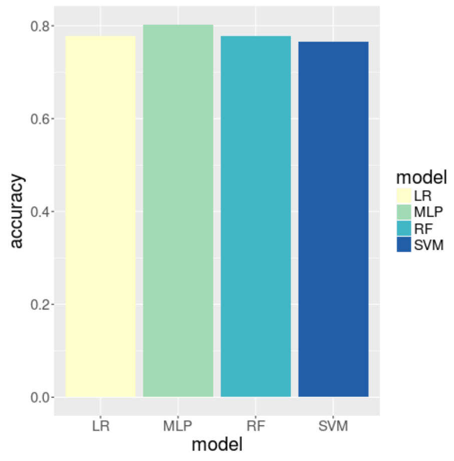
time series is cut to a smaller sequence).

For the grades case, the highest accuracy is 0.802, obtained by a multi-layer perceptron using three homeworks and the exam (Figure 5.1b) and by a random forest using four homeworks and the exam (Figure 5.1c), the former being better as it requires less data. Even the worst models have an acceptable precision as there is not much variation; with just two homeworks and one exam, it is possible to forecast whether or not a student passes the class with an accuracy of nearly three quarters. The best predictions counting true/false values are obtained by the linear regression model Table 5.1a and the random forest model Table 5.1c.

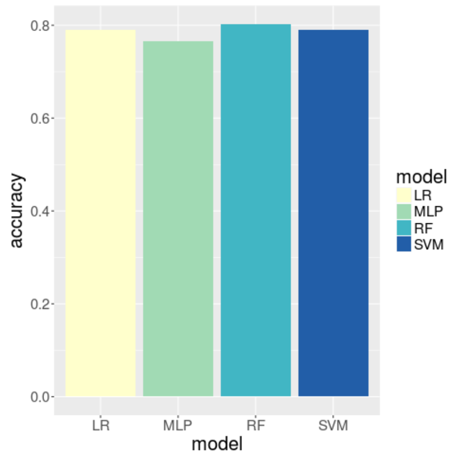
For the purchases case study, the random forest model outperforms the others regardless of the slice length. The lowest accuracy is that of the SVM with a four-month slice. The results are shown in Figure 5.2. The best predictions counting



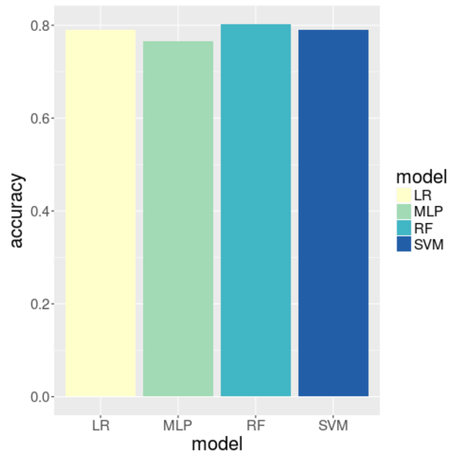
(a) *Two homeworks, middle exam*



(b) *Three homeworks, middle exam*



(c) *Four homeworks, middle exam*

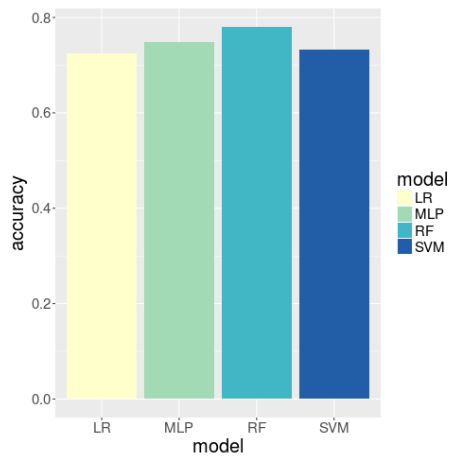


(d) *Five homeworks, middle exam*

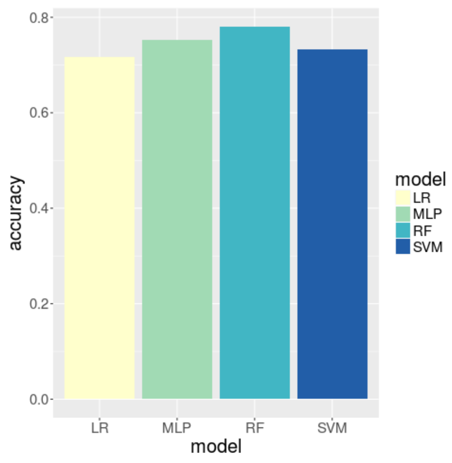
Figure 5.1: *Performance of the classification methods for the grades case study using slices of various lengths*

true/false values are obtanined by the random forest model Table 5.2c to predict loss.

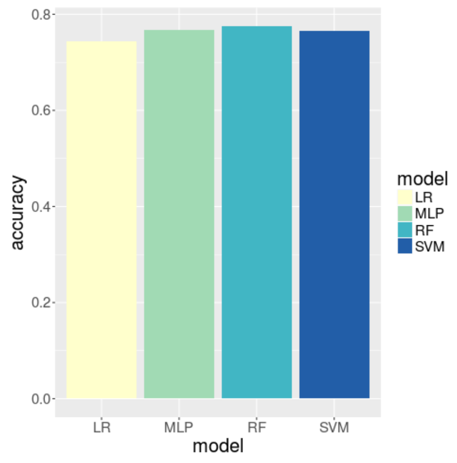
For the diseases case study, the random forest model outperforms the others regardless of the slice length. The lowest accuracy is 0.94 that is repeated by linear regression and support vector machine models in all the length of slices. The results are shown in Figure 5.3. The best predictions counting true/false values are obtanined by all the models in Table 5.3 to predict diseases obtained by midge, but to predict respiratory diseases, the random forest model Table 5.3c which is able to predict for this class. The problem in this study case is that a high accuracy is obtained because of the high volume of the migde class.



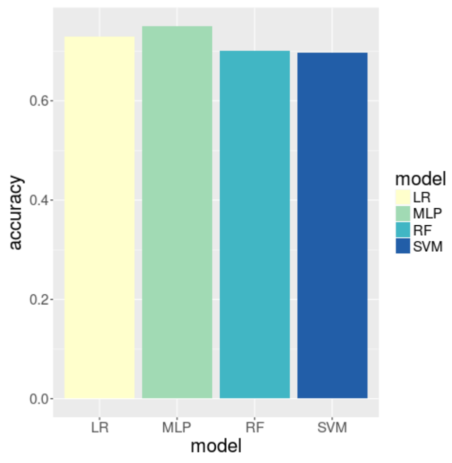
(a) *One month*



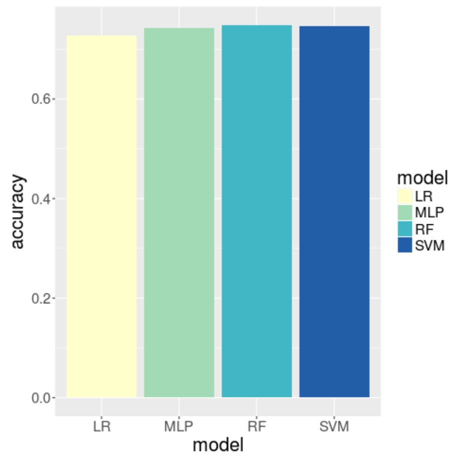
(b) *Two months*



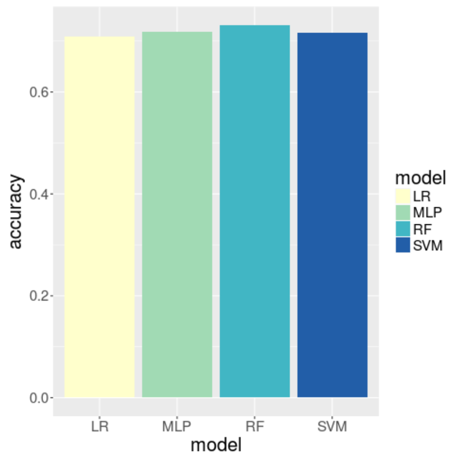
(c) *Three months*



(d) *Four months*



(e) *Five months*

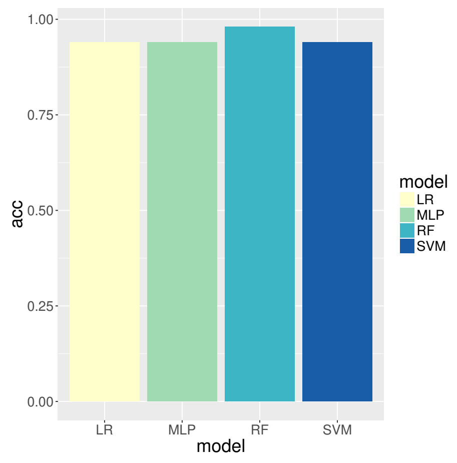


(f) *Six months*

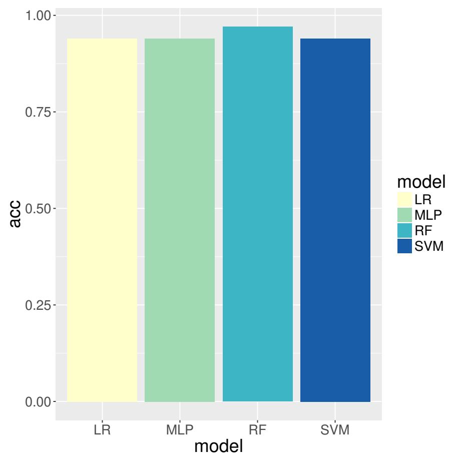
Figure 5.2: *Performance of the classification methods for the purchases case study in terms of how many months ahead the forecast is made*

Table 5.7: *Performance of the classification methods for the purchases case study in terms of how many months ahead the forecast is made*

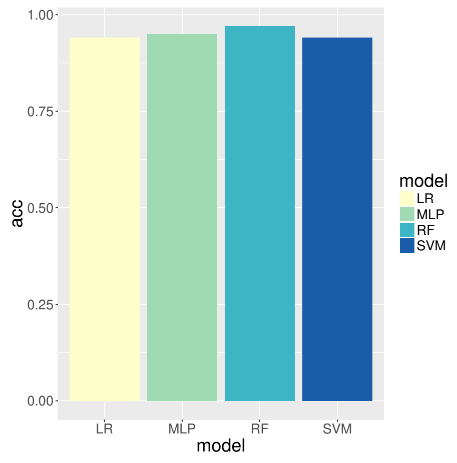
Model	Slice	Accuracy	MSE	Correct	Incorrect
LR	1	0.725	0.275	442	168
RF	1	0.780	0.220	475	135
SVM	1	0.733	0.267	447	163
MLP	1	0.748	0.252	456	154
LR	2	0.717	0.283	435	173
RF	2	0.780	0.220	474	134
SVM	2	0.732	0.268	445	163
MLP	2	0.752	0.248	457	151
LR	3	0.744	0.256	450	155
RF	3	0.775	0.225	468	137
SVM	3	0.765	0.235	462	143
MLP	3	0.767	0.235	464	141
LR	4	0.730	0.270	438	163
RF	4	0.700	0.300	420	181
SVM	4	0.696	0.304	418	183
MLP	4	0.750	0.250	450	151
LR	5	0.727	0.273	436	164
RF	5	0.748	0.252	448	152
SVM	5	0.747	0.253	448	152
MLP	5	0.743	0.257	445	155
LR	6	0.709	0.291	423	175
RF	6	0.731	0.269	437	161
SVM	6	0.717	0.283	428	170
MLP	6	0.719	0.281	429	169



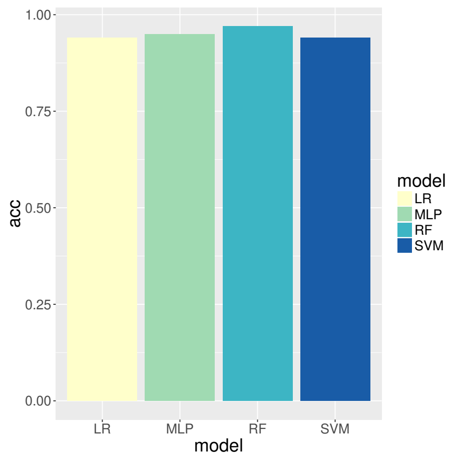
(a) *One week*



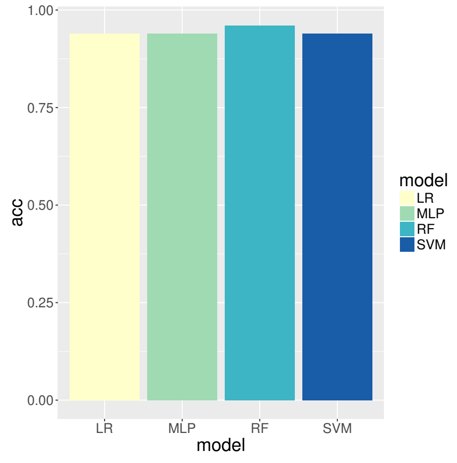
(b) *Two weeks*



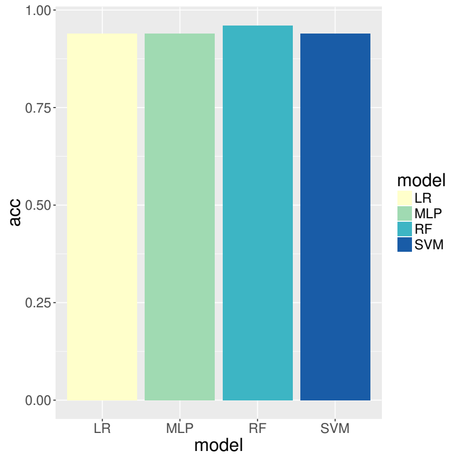
(c) *Three weeks*



(d) *Four weeks*



(e) *Five weeks*



(f) *Six weeks*

Figure 5.3: *Performance of the classification methods for the diseases case study in terms of how many weeks ahead the forecast is made*

Table 5.8: *Performance of the classification methods for the diseases case study in terms of how many weeks ahead the forecast is made*

Model	Slice	Accuracy	MSE	Correct	Incorrect
LR	1W	0.94	0.062	4606	294
RF	1W	0.98	0.024	4802	98
SVM	1W	0.94	0.062	4606	294
MLP	1W	0.94	0.057	4606	294
LR	2W	0.94	0.062	4606	294
RF	2W	0.97	0.028	4753	147
SVM	2W	0.94	0.061	4606	294
MLP	2W	0.94	0.056	4606	294
LR	3W	0.94	0.062	4606	294
RF	3W	0.97	0.033	4753	147
SVM	3W	0.94	0.061	4606	294
MLP	3W	0.95	0.054	4655	245
LR	4W	0.94	0.062	4606	294
RF	4W	0.97	0.032	4753	147
SVM	4W	0.94	0.063	4606	294
MLP	4W	0.95	0.053	4655	245
LR	5W	0.94	0.063	4606	294
RF	5W	0.96	0.039	4704	196
SVM	5W	0.94	0.063	4606	294
MLP	5W	0.94	0.061	4606	294
LR	6W	0.94	0.063	4606	294
RF	6W	0.96	0.036	4704	196
SVM	6W	0.94	0.063	4606	294
MLP	6W	0.94	0.063	4606	294

Chapter 6

Conclusions

The problem addressed in this research work is the determining the minimum amount of information sufficient to adequately predict future values of a time series through the use of classifiers based on supervised learning, through parameters adjustment and limiting the input length in order to achieve a feasible accuracy with the least amount of input data.

This thesis documents the complete process of training a supervised learning model, through the phases of cleaning the input data, processing and classifying the time series as well as searching for characteristics, varying the values of the parameters of the classification models, computing forecasts based on varying-length inputs, concluding with the systematic measurement of performance in order determine with how little data can the final values of the time series be forecast with a feasible accuracy.

A pre-processing structure for the time series was implemented, cleaning out null data, scaling the data so that all the characteristics are within the same range (as this may speed up supervised learning models), examining potential time windows to detect a certain event of interest.

To obtain a good classification performance, different parametrized models examined in terms of metrics such as cross validation; an average of all the results was obtained in order to compare the results of the various models.

The development for the majority of the tasks was carried out in the Python programming language (pre-processing, scaling, classification, and obtaining the validation metrics), whereas for the data visualization the R programming language was used with the *ggplot* package.

6.1 Discussion

The software shows feasible results in terms of accuracy and the least amount of error, depending on the situation, as in the Random Forest model that obtains the best results in most of the results but its MSE results are not good in comparison with the muti-layer perceptron. It is convenient to find a balance between both metrics and use other metrics as Recall, Precision, or Support.

Although the *scikit-learn* library helps us with multiprocessing, it would still be convenient to keep the execution times in problems with large instances to take that in mind when deploying. Because the number of instances impacts in certain models and also in memory space that is another fact not taken, even though many services in the cloud have limited resources.

The software at the moment needs few resources to run because models that use low computational performance are used, due to the number of instances used, the results may vary in performance and in time that was not taken in mind, the exploration of parameters were moderate for each type of model, although more could be added. Some models have better performance depending on the number of instances, the type of classification: binary or multiclassification as in the case of grades.

6.2 Contributions

The software created in this thesis achieves a feasible accuracy in predicting a certain event of interest in a data series, given the parameter configuration for the model and a series of preprocessing steps. The parameter configurations do have a clear effect on the model performance.

6.3 Future work

Further exploration of the parameter space (that is, a more fine-grained set of values for each factor) woulddocke be useful as well as the use of additional training models, as the present work is limited by the scope of the *scikit-learn* package. Also methods beyond the scope of supervised learning (unsupervised and reinforcement leraning in particular) are of potential interest, although often computationally more expensive and explore other metrics to get a better approach on how to select the right model.

The amount of window slicing is limited to a short fragment, there are two options: ask a person who has experience on a particular case to limit what periods to take, also the window breaks were only of length one, but it would be convenient to explore different jumps between windows. The second option would be to exploring multiple windows and jumps.

Also modern variants of neural networks (i.e., deep learning), especially for recurrent neural networks are of interest but beyond the present scope of the *scikit-learn* package, for which other options could be considered in future expansions of the present work. Another future direction is additional case studies and larger set of longer time series so as not to over-adjustthe models to the training data.

Bibliography

P. Bloomfield. *Fourier Analysis of Time Series: An Introduction*. Wiley Series in Probability and Statistics. Wiley, Raleigh, North Carolina, 2004. ISBN 9780471653998.

V. Boeva and E. Tsiporkova. *A Multi-purpose Time Series Data Standardization Method*, pages 445–460. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13428-9. doi: 10.1007/978-3-642-13428-9_22.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001. doi: 10.1023/A:1010933404324.

O. Chapelle, B. Schlkopf, and A. Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010. ISBN 0262514125, 9780262514125.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 1573-0565. doi: 10.1007/BF00994018.

P. Cunningham, M. Cord, and S.J. Delany. *Supervised Learning*, pages 21–49. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-75171-7. doi: 10.1007/978-3-540-75171-7_2.

M.H. DeGroot and M.J. Schervish. *Probability and Statistics*. Addison-Wesley, Boston, USA, 2012. ISBN 9780321500465.

D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

P. Geurts. Pattern extraction for time series classification. In L. De Raedt and A. Siebes, editors, *Principles of Data Mining and Knowledge Discovery*, pages 115–127, Riverside, California, 2001. Springer Berlin Heidelberg.

V. Gómez, A. Maravall, and D. Peña. Computing missing values in time series. In Y. Dodge and J. Whittaker, editors, *Computational Statistics*, pages 283–296, Berlin, Heidelberg, Germany, 1992. Physica-Verlag HD.

J. Grandell. *Time series analysis.* Lecture notes, KTH Stockholm, 1998.

I. Guyon and A. Elisseeff. *An Introduction to Feature Extraction*, pages 1–25. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-35488-8. doi: 10.1007/978-3-540-35488-8_1.

D.J. Hand. Principles of data mining. *Drug Safety*, 30(7):621–622, Jul 2007. ISSN 1179-1942. doi: 10.2165/00002018-200730070-00010.

F.E. Harrell. *Ordinal Logistic Regression*, pages 331–343. Springer, New York, NY, 2001. ISBN 978-1-4757-3462-1.

R. Kruse, C. Borgelt, C. Braune, S. Mostaghim, and M. Steinbrecher. *Multilayer Perceptrons*, pages 47–92. Springer London, London, 2016. ISBN 978-1-4471-7296-3. doi: 10.1007/978-1-4471-7296-3_5.

T.W. Liao. Clustering of time series data — a survey. *Pattern Recognition*, 38(11): 1857–1874, November 2005. doi: 10.1016/j.patcog.2005.01.025.

G. Marti, S. Andler, F. Nielsen, and P. Donnat. Clustering financial time series: How long is enough? In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2583–2589. AAAI Press, 2016.

T.M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1997. ISBN 9780070428072.

D.C. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons, Inc., Cary, North Carolina, 2006. ISBN 0470088109.

C.A. Ratanamahatana and E. Keogh. *Making Time-series Classification More Accurate Using Learned Constraints*. SDM, 2004.

S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Englewood, New Jersey, 2 edition, 2003. ISBN 0137903952.

M. J. Shensa. The discrete wavelet transform: wedding the a trous and mallat algorithms. *IEEE Transactions on Signal Processing*, 40(10):2464–2482, October 1992. doi: 10.1109/78.157290.

L. Wei and E. Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 748–753, New York, NY, USA, 2006. ACM. doi: 10.1145/1150402.1150498.

Y. Xiong and D.-Y. Yeung. Mixtures of arma models for model-based time series clustering. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings*, pages 717–720, 2002. doi: 10.1109/ICDM.2002.1184037.

Resumen autobiográfico

Pedro Sánchez Martínez

Candidato para obtener el grado de
Ingeniero en Tecnología de Software

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Tesis:
Case studies in feature extraction and parameter tuning for time-series classification

Nací el 12 de Septiembre de 1995 en el municipio de San Pedro Garza García, Nuevo León. Único hijo de Gloria Martínez Argumedo y Fidencio Sánchez Martínez. Hice mis estudios nivel medio superior en la Preparatoria No. 23 Unidad Santa Catarina de la UANL durante el periodo 2011–2013. Inicié mis estudios nivel superior en el año 2013 en la Facultad de Ingeniería Mecánica y Eléctrica de la UANL en la carrera de Ingeniero en Tecnología de Software.