TECHNICAL UNIVERSITY OF CRETE

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Predicting the Marketability Potential of Google Play Apps

DIPLOMA THESIS

Angelos Kartakis

Chania, September 2020

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Πρόβλεψη της Δυνατότητας Εμπορικής Αξιοποίησης Εφαρμογών του Google Play

Διπλωματική Εργασία

Άγγελος Καρτάκης

ΕΠΙΤΡΟΠΗ:

Γεώργιος Ν. Γιαννακάκης, Αναπληρωτής Καθηγητής

Μιχαήλ Γ. Λαγουδάκης, Αναπληρωτής Καθηγητής

Αντώνιος Λιάπης, Λέκτορας (Πανεπιστήμιο Μάλτας)

Χανιά, Σεπτέμβριος 2020

Abstract

In recent years there has been a great increase of interest around the field of smart phone application development growing to a $33B market. Applications that meet the various needs of customers are constantly evolving, leading to the development of online application marketplaces, such as Google Play. In this thesis, we examine the key drivers of app user ratings and propose four approaches to predict the ability of applications to be valued highly by their users within the app market, i.e. marketability. To develop these approaches, we leverage the predictive capacity of Machine Learning algorithms by formalizing the marketability prediction problem as a classification problem. In particular, we test and compare six Machine Learning algorithms --- i.e. Random Forests, Decision Trees, Multi-layer Perceptrons, k-Nearest Neighbor, Logistic Regression and Support Vector Machines --- for their ability to predict the ratings of app users based on a set of app features. By evaluating the algorithms against real data from Google play, we achieved up to 86% accuracy on marketability (i.e. user rating) prediction. The proposed solution can be extended to cover other domains, such as commercial capacity forecasting.

Περίληψη

Τα τελευταία χρόνια παρατηρείται μεγάλη αύξηση ενδιαφέροντος γύρω από τον τομέα της ανάπτυξης εφαρμογών. Εφαρμογές χρήσιμες για να καλύπτουν ανάγκες διαφόρων πτυχών αναπτύσσονται συνεχώς, με άμεση συνέπεια την ανάπτυξη ηλεκτρονικών αγορών, όπως το Google Play, αγορά αξίας 33 δισεκατομμυρίων δολαρίων. Στην παρούσα διπλωματική εργασία προτείνουμε προσεγγίσεις για την πρόβλεψη της δυνατότητας εμπορικής αξιοποίησης εφαρμογών που αναπτύσσονται για την προαναφερθείσα πλατφόρμα. Χρησιμοποιούμε το πλαίσιο της μηχανικής μάθησης (machine learning), το οποίο προσφέρει καινοτόμες μεθόδους εκμάθησης για την δημιουργία προβλέψεων. Αξιοποιούνται οι ευρείας χρήσεως αλγόριθμοι Random Forest, Decision Tree, Multi-layer Perceptron, k-Nearest Neighbor, Logistic Regression και Support Vector Machines μετατρέποντας το πρόβλημα της δυνατότητας πρόβλεψης εμπορικής δυναμικότητας σε πρόβλημα κατηγοριοποίησης (classification). Τα αποτελέσματα εξετάζονται και συγκρίνονται εφαρμόζοντας τεχνικές αξιολόγησης ως προς την απόδοση τους, επιτυγχάνοντας ως και 86% ακρίβεια. Θεωρούμε ότι η προτεινόμενη προσέγγιση συμβάλει στον βασικό της στόχο και θα μπορούσε να επεκταθεί στο μέλλον για προσαρμοστικές λύσεις σε περιπτώσεις πρόβλεψης εμπορικής δυναμικότητας, αξιοποιώντας ακόμα και την εναλλακτική προσέγγιση της μάθησης προτιμήσεων (preference learning).

Acknowledgements

First, I would like to thank my advisor Georgios N. Yannakakis for his trust and guidance during the course of this thesis under the challenging circumstances of the pandemic.

My friends who accompanied me to this journey, together we shared some incredible moments over the past 5 years.

Last, but not least, I would like to thank my family for the immense support, unconditional love and constant encouragement.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Smart phones have been widely successful in recent years. A key driver of this success is the ability of the user to install custom applications (apps) from an online marketplace. Indicatively, Google's official market place, i.e. Play store, reached 2.9 million applications by March 2020 counting over 11 billion downloads, with a revenue of $32.8B[1].

Considering this significant size and financial growth of app marketplaces, new challenges have emerged in the current context of mobile apps. For example, to compete in a massive market of apps, app developers must take into consideration novel aspects, such as user ratings and app adoption. The analysis and exploitation of these aspects are crucial for shaping the future application development and roadmap. Specifically, developers or businesses that release applications to the marketplaces view the user rating scores as indication of their work to be marketed, so-called marketability. Thereafter, their main goal is to orchestrate a plan to maximize these scores.

Outdated solutions require the manual analysis of user reviews and ratings, which is a time-consuming and error prone process. Then, the extracted information is used to drive further development of the applications. In our approach, in order to solve the problem of marketability prediction and (in part) automate this process, we are based on Machine Learning (ML) algorithms and their predictive capacity. ML is widely adopted by businesses across industries nowadays, assisting them to solve complex problems efficiently and accurately.

The purpose of this thesis is to reveal the quantitative factors of marketability by building a framework based on ML that efficiently predicts user ratings. In particular, this thesis attempts to answer the following research question:

"Can we efficiently predict the user ratings of applications in Google Play, using merely the information contained within the app features?"

In this thesis we formalize the aforementioned marketability prediction problem as a classification problem. For that purpose, we test and compare six state-of-the-art Machine Learning algorithms, i.e. Random Forest, Decision Tree, Multi-layer Perceptron, k-Nearest Neighbor, Logistic Regression and Support Vector Machines. By evaluating the algorithms against real structured data from Google store,

[1] According to Statista www.statista.com

we achieved up to 86% accuracy on marketability prediction. These significant results reveal that our proposed solution can be used to solve the marketability prediction efficiently and accurately. In the future, we plan to extend our solution to cover other business cases, such as commercial capacity forecasting.

This thesis consists of the following chapters:

- In Chapter 2, we present the background information needed for this thesis. We give an overview of the terms and concepts of marketability & Google Play, while we present Machine Learning framework and state-of-the-art algorithms. Related Work is also discussed in this chapter, providing additional approaches to the problem.
- In Chapter 3, we dive deep into our proposed approach and the implementation.
- In Chapter 4, we evaluate the results of our approach benchmarking multiple algorithms and solutions.
- Lastly, Chapter 5 is the conclusion of this thesis illustrating the limitations & future improvements of our approach.

CHAPTER 2

BACKGROUND

In this chapter we introduce the main techniques and theoretical background related to the Thesis. In particular we discuss topics regarding the platform, data preparation, machine learning and marketability in each of the corresponding sections below.

2.1 Google Play

Google Play, formerly Android Market, is an online platform of services operated and developed by Google. It serves as the official app store for devices running on "Google certified"--meaning they offer support for all official apps--Android operating system, enabling users to search and purchase applications developed with the Android software development kit (SDK) and published through Google.

By 2017, Google Play featured more than 3.5 million Android applications, with developers from over 150 locations distributing apps on Google Play. Regarding the applications that are being marketed developers receive 70% of the application price, while the remaining 30% goes to the distribution partner and operating fees. This platform, gives the opportunity to developers to set up sales, with the original price struck out and a banner underneath informing users when the sale ends. Google Play, also allows developers to release early versions of apps to a selected group of users, as alpha or beta tests. The ability of pre-ordering a number of selected apps to have the items delivered as soon as they become available, is provided as well for the users.

According to *SensorTower*[2], $24.8 billion was spent on Google Play in 2018, which is translated to 27.3% growth compared to 2017, exceeding the 75 billion mark on first-time app installs. Google Play is a platform provisioned to overtake a total revenue of $60 billion by 2023, making it an appealing choice for programmers to make a turn into app development.

Given the popularity of the service and the future projections of its market potential the creation of a framework able to predict the demand on a newly developed app, can be of immense importance for the monetization strategy and marketing campaigns of app developers.

[2] Company that provides market intelligence and analytics for the mobile app economy through their software suite

Fig2-1 Google Play logo as obtained from Wikipedia

2.2 Marketability

Marketability is defined as the competitive position of a brand, product, service with respect to a market (in this case, the Google Play market) and its potential to sell well within this market. Marketability can be represented in various forms. The *cost* and the *price* of a product [1] for example is considered a form of marketability: offering a product at a higher cost compared to the average market cost puts the product at a competitive disadvantage (the unfavorable circumstance that causes the product to underperform in an industry).

Another type of marketability is *Customer Needs* [2], to wit the customers' motivation that the product would fulfill their needs. Identical notions to marketability are those of *Functionality* and *Performance*, *Quality*, and *Distribution* [3]. More specifically, if the product performs better than competitive products on the market, this usually means that the product/service it is reliable, durable, useable and it can reach customers rapidly.. All the types of marketability outlined above, tend to both form and affect the most important type of marketability regarding a product: its *Reputation* and *Recognition*. Reputation and Recognition responds as the ability of a product to establish reputation and brand recognition among other products in a competitive market.

In this thesis we focus on the Reputation and Recognition of applications. In this dissertation, these two aspects of marketability are expressed as self-reported ratings provided by customers of the Google Play service (apps market)..

2.2.1 Correlation of Marketability & App Rating

The huge pace of development in the mobile app market, has attracted more programmers to develop applications. Notwithstanding, while there is a solid portion of developers and companies making extraordinary fortunes over the applications' market, most developers are battling to retain their initial investment. Therefore, understanding the factors leading to high-rated apps is the first step towards successful development and evolution of apps [3],[4].

Google Play Store contains a huge number of Apps. These applications are downloaded and used by millions of clients. Whenever a user browses, or searches for apps on Play store, a list of apps is shown to the user. Generally, a client would prefer to download highly rated applications in light of the fact that highly rated apps reflect clients' satisfaction. So as to increase high ratings, application engineer utilizes various methods and changes other than the application quality itself.

Measuring the success of a software system is difficult as there is neither a universal metric nor a ranking scheme. Due to the close relationship of an app rating regarding the ability of it to be sold or marketed, app rating is chosen as proxy for app success. In this case, the main target is the establishment of a model of effectively predicting application ratings given the application characteristics. Thus, a newly developed app could get sharply increasing ratings or an already developed app could test the efficacy of changes leading to the improvement of its rating, with both cases being associated with the growth of demand, therefore an improvement on apps' [5] marketability.

As stated in [5], the rating-based format represents a player's state with a scalar value or a vector of values. Ratings are seemingly the predominant practice for quantitatively surveying parts of a user's behavior, experience, opinion or emotion. In fact, by far most of user and psychometric studies include rating questionnaires to extract the opinions, preferences and experiences of experiment participant.

2.3 Data Preparation

In order for empirical research to take place, data availability is mandatory. As soon as, data is acquired it requires the process of Data Preparation takes place. Data Preparation consists of Data Discovery (identifying the data), Data Cleaning (or Cleansing), Data Analysis & Data Interpretations.

2.3.1 Data Structure

Data is classified into three main categories based on its structure [6]: structured, semi-structured, unstructured.

Structured data is the least demanding type of data to search and organize [7], since it is generally contained in rows and columns and its components can be mapped into pre-defined fields (i.e. CSV, Excel files)., thereby, making it easier to analyze, search and store. Conversely, unstructured data cannot be contained in the pre-described form of a database and doesn't have a related data model, usually following the structure of a continuous text. The absence of structure makes unstructured data difficult to search, manage and analyze [8]. Semi-Structured Data is a mix between both of categories referred above. It has some defining or consistent attributes yet it does not adjust to a structure as rigid as is expected with a relational database [9].

In summary, structured data is easily organizable and follows a rigid format; unstructured is intricate and often qualitative information is impossible to diminish or be organized and semi-structured data is a fusion of both structured and unstructured data.

2.3.2 Data Cleaning

Data cleaning is a key procedure associated with data analysis, with it being the initial step after data collection. Data cleaning is the principal step of data preparation task that manages rectifying inconsistent data, replacing or removing missing values and smoothing out noisy data [10] following the process seen in Fig.2-2. Missing values are handled in different manners relying upon the prerequisites set, either by overlooking the tuple, or filling in the missing value with the mean estimation of the trait, or utilizing a global constant or some different procedures like decision tree or Bayesian formulae. Noisy data is tackled manually or through various regression or clustering techniques [10][11]. Then after the observation of the common problems it can be automated. There are such a large number of procedures -a number related to the size and structure of the data- engaged with data cleaning, which prepares it for analysis once they are finished.

Data Cleaning is valuable in improving the efficiency -in terms of data quality and accurate depiction of the given information [11] - of the result of data analysis. The procedure of Data Cleaning may include the removal of typographical blunders or filling of data missing fields (regarding structured data). This will be done until the data is reported to meet the data quality criteria [12], which incorporate; validity, accuracy, completeness, consistency, and uniqueness, defined as:

- **Completeness**: The proportion of stored data against the potential of "100% complete.
- **Uniqueness**: It is the inverse of an assessment of the level of duplication.
- **Validity**: Data conformation to the syntax (format, type, range) of its definition.
- **Consistency**: The absence of difference, when comparing two or more representations of a data point against a definition.
- **Accuracy**: The degree to which data correctly describes the "real world" object or event being described

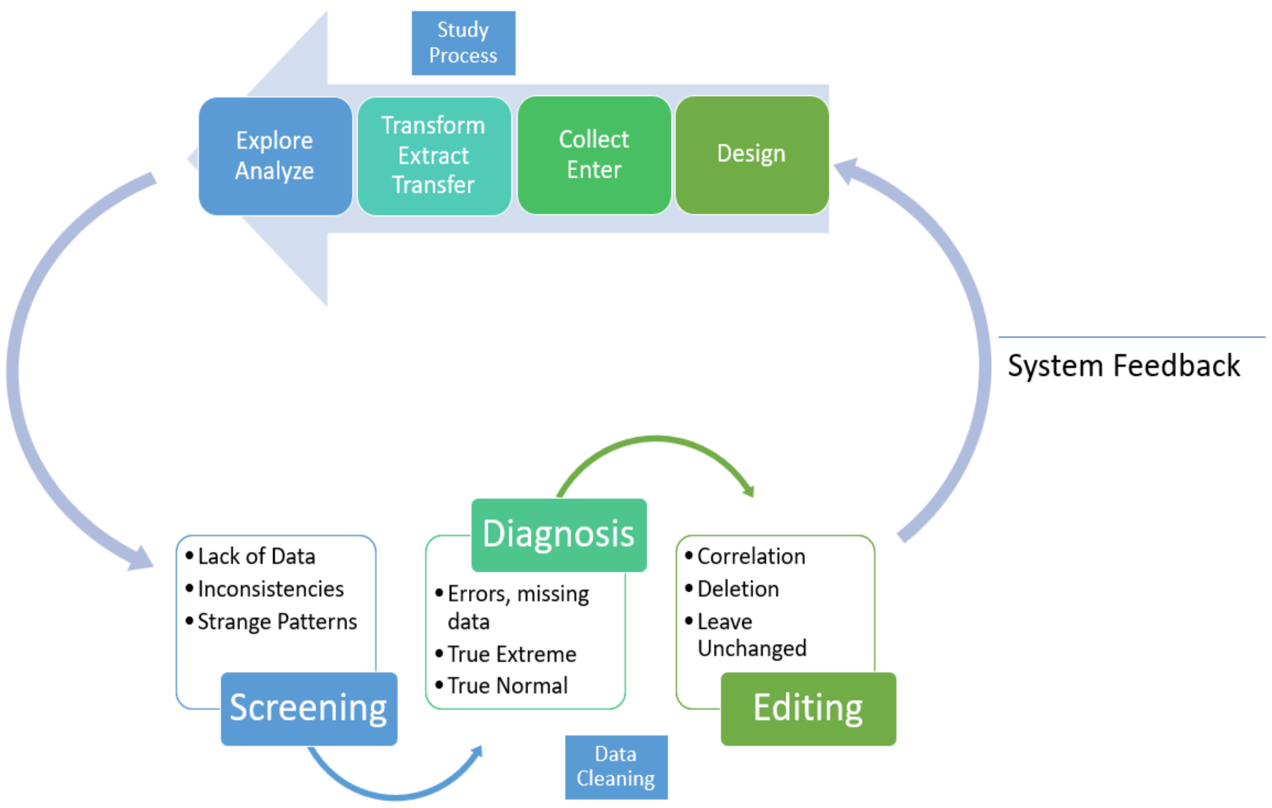


Figure 2-2. A Data-Cleaning Framework adapted from [10]

2.3.3 Data Interpretation and Analysis

Data analysis is, a process that involves examining, and molding collected data for interpretation [11] to:

- discover relevant information
- draw or propose conclusions
- support decision-making to solve a research problem

The Data Analysis process used for evaluating data leveraging expository and logical reasoning to examine each of the data component provided. This type of analysis is only one of the numerous steps that must take place when directing a research experiment. Data from a variety of sources is collected, inspected, and then broke down to shape findings or conclusions. There is an assortment of explicit data analysis methods, like data mining, text analytics, and data visualizations.

Data interpretation alludes to the implementation of procedures via which data is looked into in order to arrive at an informed conclusion. The interpretation of data appoints an importance to the data investigated and determines its signification and implications [13]. This process involves:

- taking the result of Data Analysis
- making inferences on the relations studied
- using the above to conclude.

Therefore, before referring to Data Interpretation, Data Analysis has to be conducted first.

Data Analysis is usually the first step taken towards data interpretation. Interpretation involves attaching meaning and significance to the analysis, explaining descriptive patterns, and looking for relationships and linkages among descriptive dimensions. Moreover, data interpretation and data visualization make the data easy to understand, and encourages people to view the data as it provides a visually appealing summary of the data (i.e. pie charts, histograms, bar charts).

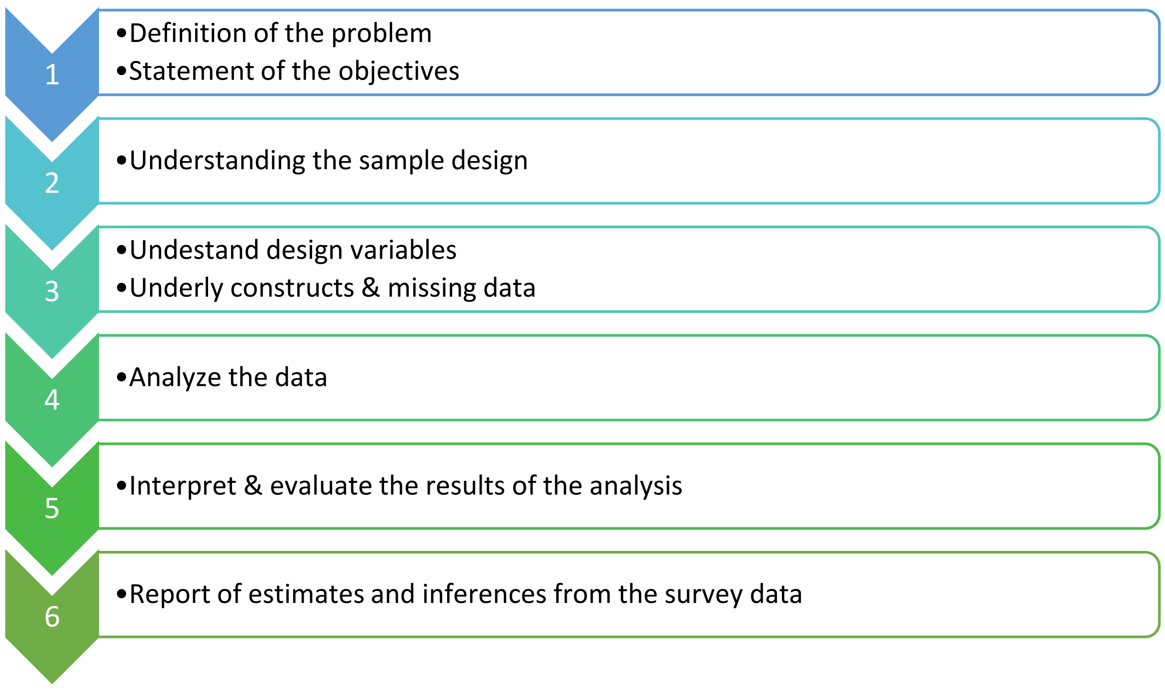

1
•Definition of the problem
•Statement of the objectives
2
•Understanding the sample design
3
•Undestand design variables
•Underly constructs & missing data
4
•Analyze the data
5
•Interpret & evaluate the results of the analysis
6
•Report of estimates and inferences from the survey data

Figure 2-3 Steps in applied survey data analysis.[13]

2.4 Machine Learning

People are not only generators but also consumers of data in need of products and services specialized for them. As a result, their interests have to be predicted in order to provide them with personalized services and offerings. Machine Learning is the natural approach to follow in this thesis, since we have data to analyze, learn and make predictions based on it (i.e. app ratings).

More specifically, Machine learning is not just a database problem; it is the subfield of AI connected with software that learns from experience [14]. In particular, machine learning methods are already the best methods available for developing particular types of software, in applications where [14][15]:

- The application is too complex for people to manually design the algorithm; like perception and pattern recognition tasks, such as speech recognition and computer vision.
- The application requires that the software customizes to its operational environment after it is fielded; machine learning, in such an application, provides the mechanism for adaptation.

In general, Machine Learning is a natural intersection of Computer Science and Statistics [14][15], and more specifically:

- Statistics are utilized by Machine Learning in constructing mathematical models, tackling the core task of making inference from a sample.
- Computer Science serves a twofold role:
    - First, in training, we need effective algorithms on solving the optimization issues, as well as storing and processing the massive amount of data we for the most part have.
    - Second, as soon as the training is over, its representation and algorithmic solution for inference needs to be proficient as well. In some sort of applications, the efficiency of the learning, its space and time complexity, may be as significant as its ability to accurately predict.

2.4.1 Supervised Learning

Machine Learning algorithms can be classified into three main types; supervised learning, unsupervised learning, and reinforcement learning as shown in Fig.2-4. The types of machine learning algorithms differ in the mathematical approach, type of input/output data, and type of problem that they are intended to solve. In this thesis the approach adopted for predicting user ratings is Supervised Learning because of both the acquired data and problem structure, as described below.

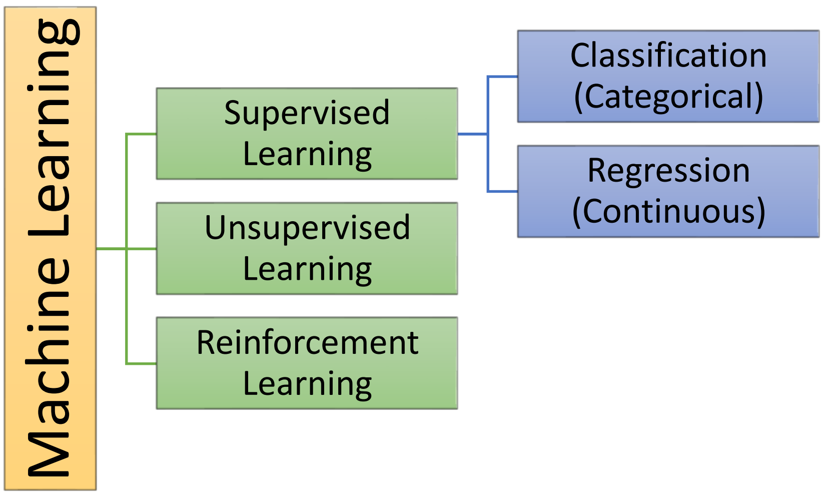


Figure 2-4 Machine Learning algorithm taxonomy

In machine learning every instance of a particular dataset is represented by a set of features, which are individual measurable properties or characteristics of the phenomenon being observed. The nature of these features could be continuous, categorical or binary. If instances are given with known labels (i.e. the corresponding correct outputs) then the learning scheme is known as *supervised learning*, while in unsupervised learning approach the instances are unlabeled [16]. In our particular problem, the input samples correspond to the list of features, while the target outputs correspond to the annotations of the rating -in this case- for each of the input samples that we are interested to learn to predict [5]. Therefore, supervised learning algorithms build a mathematical model of a set of data that contain both the inputs and the desired outputs. In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor [14], as the learner receives a set of labeled examples as training data and makes predictions for all unseen points [17].

In order to solve a given problem of supervised learning, there is a set of steps that have to take place as follows (Fig.2-5 illustrates the process):

- the type of training examples must be determined: the user should decide what kind of data is to be used as a training set
- gather a training set: the training set needs to be representative of the real-world use of the function
- determine the input feature representation of the learned function: the accuracy of the learned function depends strongly on how the input object is represented
- determine the structure of the learned function and corresponding learning algorithm
- complete the design: run the learning algorithm on the gathered training set
- evaluate the accuracy of the learned function: the performance of the resulting function should be measured on a test set that is separate from the training set.

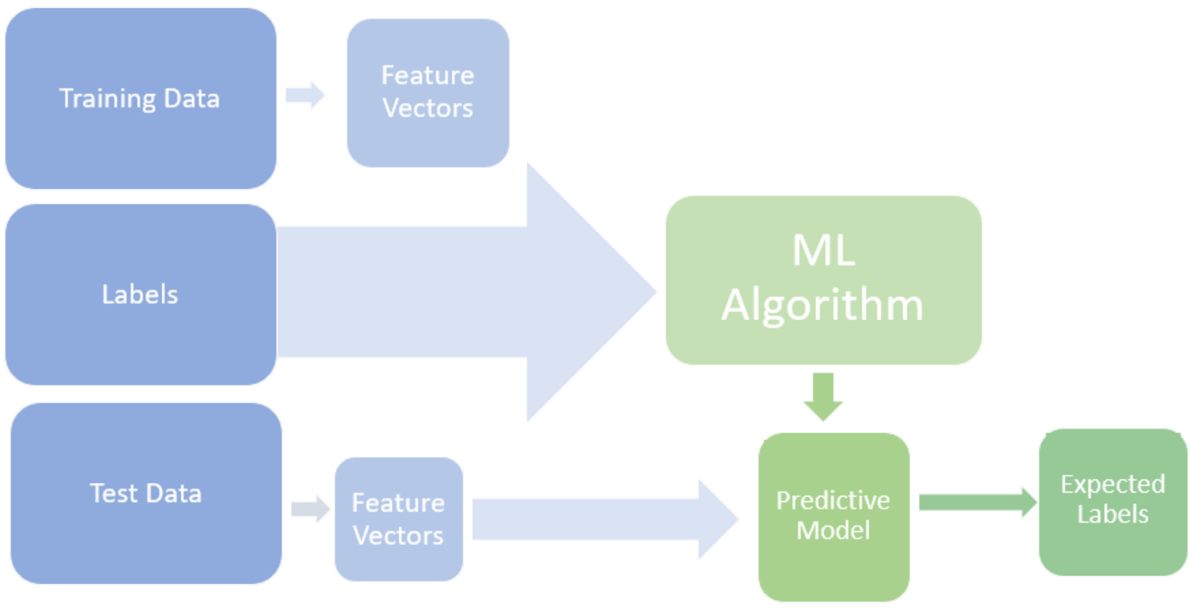


Figure 2-5 Supervised Machine Learning model [17]

2.4.2 Classification

Classification is a supervised learning paradigm which learns to categorize a set of data into classes. It can be performed on both structured or unstructured data with the process starting by predicting the class (class is a Y collection of outcomes occurring under certain circumstances (e.g. the High_Rating_Class contains ratings higher than 4.3) of given data points [14]. The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables. The main goal is to identify which class/category the new data will fall into. The procedure followed, while approaching a classification problem is shown below in Fig.2-6.

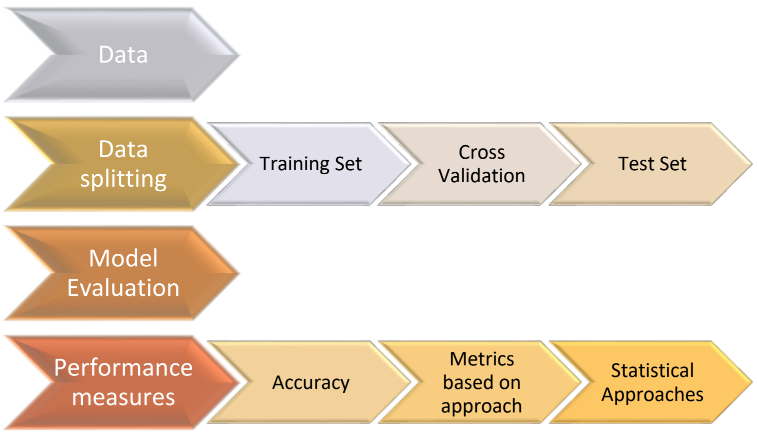


Figure 2-6 Classification Framework

Classification is the key approach used in this thesis by transforming the numeric depiction of ratings into a nominal scale of separate classes. For instance, ratings of applications can be transformed from numerical values that lie between 1-5, into “High” and “Low” classes. Classes are easy to analyze and their use dispenses part of the inter-personal biases introduced with ratings [5].

Classification Algorithms

In this section we outline the various classification methods used in this thesis. These include decision trees, random forests, k-NN, logistic regression, support vector machines, and multilayered perceptrons.

- **Decision Tree:** A decision tree or a classification tree (see an example in Fig. 2-7) follows a tree structure in which each internal node is labeled one of the input features. Each node that is labeled with an input feature, as described above, is conjoint with either a subordinate node labeled with a different input feature or with the target feature, depending on each possible outcome. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into either a specific class, or into a particular probability distribution [19]. A decision tree predicts the label of an instance, by moving from a root node of a tree towards a leaf. At each node of the path –from root to leaf--, the successor child is selected based on a splitting of the input space. Usually, the splitting relates on one of the features of the instance. Seemingly important to note, that every leaf contains a specific label [20].

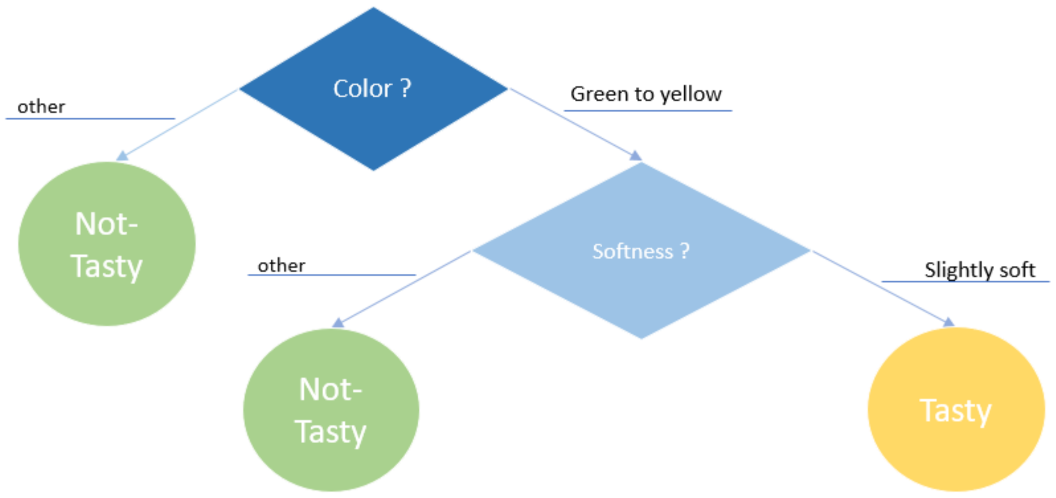


Figure 2-7 Example of a Decision Tree deciding if a fruit is tasty [20].

- **Random Forest:** A Random Forest is an ensemble learning method, which means that it utilizes multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [21]. The idea behind the Random Forest is the training of many decision trees, each on a random subset of training set or a random subset of the input features. After a large number of trees is generated, they vote for the most popular class, aiming at overall accuracy increase [14] as represented in Fig.2-8.

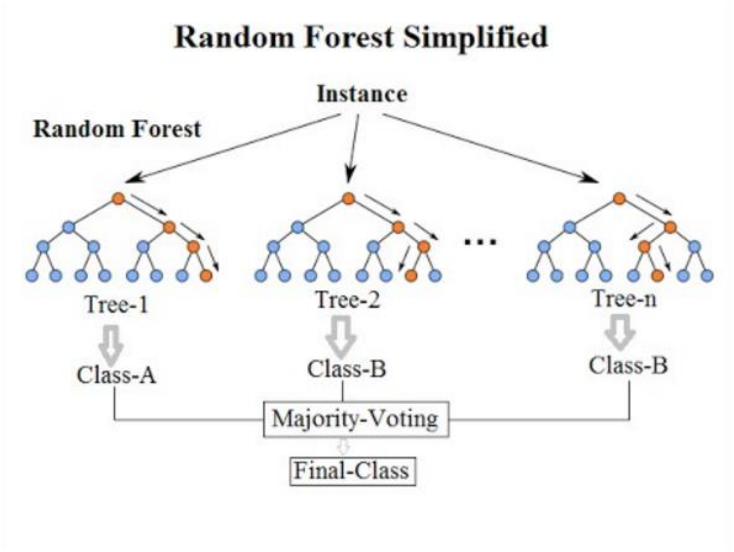


Figure 2-8 Random Forest framework-acquired from Wikipedia

- **K-Nearest Neighbor (k-NN):** k-NN is one of the non-parametric classification algorithms as the algorithm is not based solely on parametrized families of probability distributions (i.e. the mean and the variance). Its operations instead are based on the nearest neighbor (NN) rule, that identifies the category of unknown data point on the basis of its nearest neighbor whose class is already known. In this case of k-nearest neighbor (k-NN), the nearest neighbor is calculated on the basis of value of k, that specifies how many nearest neighbors need to be considered to define a class of a sample data point (see Fig.2-9) [22]. The value of k is normally determined using a validation set or using cross-validation [23]. In binary (two class) classification problems, it is often helpful to choose k to be an odd number as this eliminates tied votes.

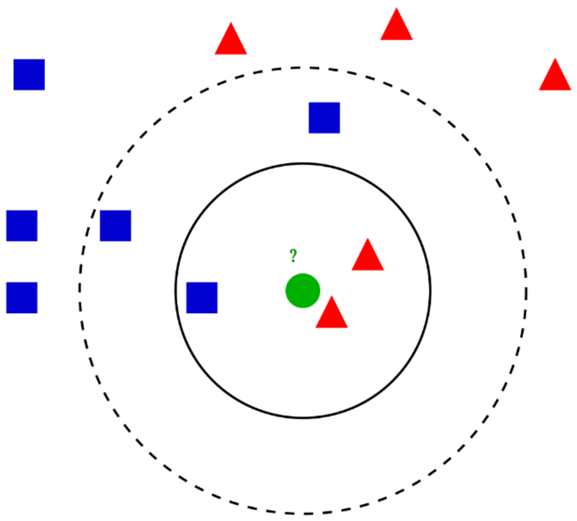


Figure 2-9Example acquired from Wikipedia of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

Nearest neighbor classifiers are instance-based or lazy learners in that they store all of the training samples and do not build a classifier until a new (unlabeled) sample needs to be classified [24].

- **Logistic Regression:** Logistic Regression is used to model the probability of a certain class or a binary event such as pass/fail, win/lose, alive/dead or healthy/sick. The method can be extended to model multiple classes of events as well.. Logistic regression can show which of the various factors being assessed has the strongest association with an outcome and provides a measure of the magnitude of the potential influence [25]. The core of Logistic Regression is the logistic function (or sigmoid curve):

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}},$$

where X_0 is the x value of the sigmoid's midpoint, L is the curve's maximum value and $\mathbf{k}$ is the logistic growth rate or steepness of the curve. The logistic function is an S-shaped curve that can take any real-valued number and maps it into a value between 0 and 1, but never exactly at those limits. By analyzing the equation $-k(x - x_0)$ is the actual numerical value that we want to transform. As shown at the example bellow, is a plot of the numbers between -6 and 6 transformed into the range of 0 and 1 using the logistic function (Fig. 2-10).

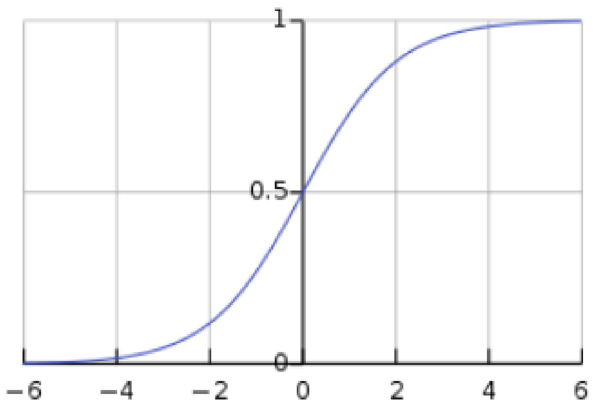


Figure 2-10 Equation of Logistic Function interpretation for L=1,k=1,x_0=0, example acquired from Wikipedia

Logistic regression uses an equation as the representation Input values (x) are combined linearly using weights or coefficient values to predict an output value (y). An example of logistic regression equation is shown, below [26]:

$$y = \frac{e^{(b_0+b_1 \cdot x)}}{1 + e^{(b_0+b_1 \cdot x)}}$$

Where y is the predicted output, $b_0$is the bias or intercept term and $b_1$is the coefficient for the single input value (x). Each column in the input data has an associated b coefficient (a constant real value) that must be learned from the training data. The coefficients (b values) are calculated using maximum-likelihood estimation. Maximum-likelihood estimation is a method of estimating the parameters of a probability distribution by maximizing a likelihood function, so that under the assumed statistical model the observed data is most probable; a common learning algorithm used by a variety of machine learning algorithms.

- **Support Vector Machines (SVMs):** An SVM is an algorithm set to find an optimal hyperplane in multidimensional space (the number of dimensions is depended on the number of input features), distinctly classifying the data points. By referring to an optimal hyperplane, we consider a hyperplane that maximizes the margin (i.e. the maximum distance between the data points of the different classes), as shown in Fig.2-11. The defining data points, that affect the position and orientation of the hyperplane are called Support Vectors. The support vectors are defined as the boundaries of each class and include the data points that are the closest to the hyperplane. Support Vector Machines are of significant matter, as they can use both linear and non-linear data in order to make predictions [5][27][28]

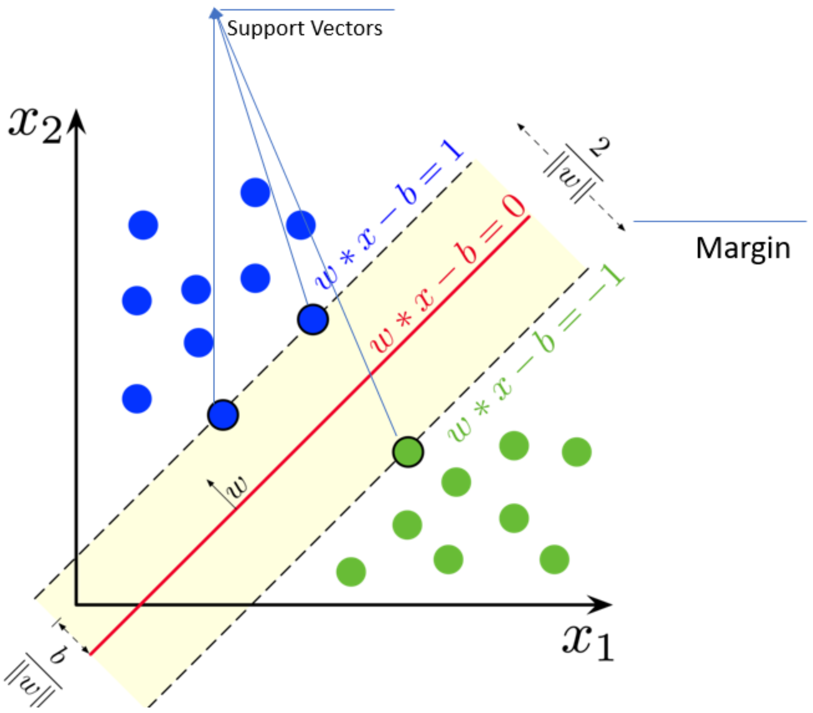


Figure 2-11 SVM model acquired from Wikipedia and edited

- **Multi-layer Perceptron (MLP):** MLP is an Artificial Neural Network defined by a function $f: R^m \rightarrow R^o$, where m is the size of input vector and o is the size of output vector (i.e. the result of f). Given a set of features $X = x_1, x_2, \dots, x_n$ and a target y an MLP can learn a non-linear function approximator for either classification or regression. On this type of neural network, multiple hidden layers might exist between the input and output layer increasing the computational power.

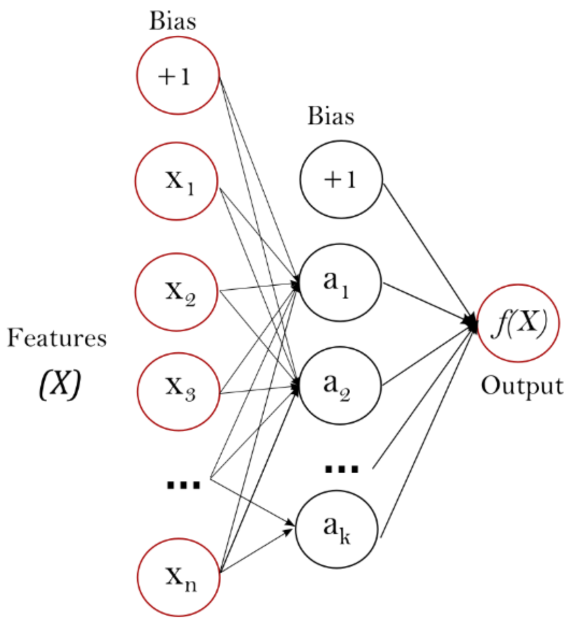


Figure 2-12 One hidden layer MLP with scalar output acquired from sklearn documentation

The leftmost layer, known as the input layer, consists of a set of neurons $\{x_i | x_1, x_2, \ldots, x_n\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + \cdots + w_nx_n$ followed by a non-linear activation function $g(\cdot) = R \quad \rightarrow R$ - In many applications the units of these networks apply a sigmoid function as an activation function. The output layer receives the values from the last hidden layer and transforms them into output values [29].

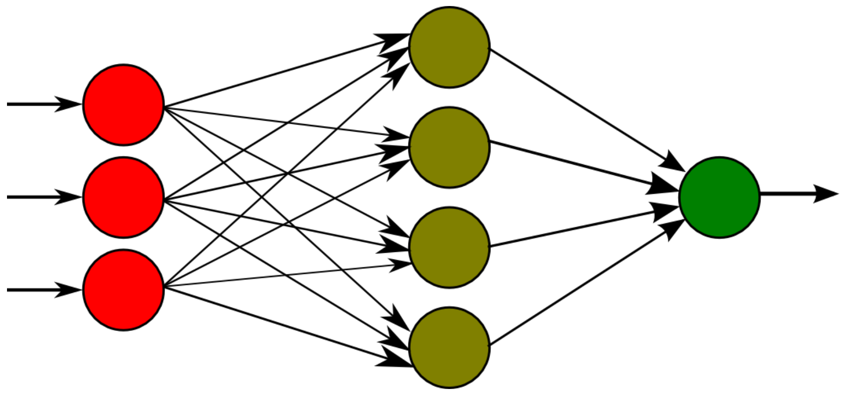


Figure 2-13 Architecture of MLP neural network with one hidden layer acquired from Wikipedia

Once configured, the neural networks have to be trained on the given dataset, a procedure that can be achieved by a variety of methods. One state-of-the-art learning technique that is constantly utilized by MLPs is called back-propagation. In this particular instance, the output we retrieve from the network is compared to the expected output, thus calculating an error-function. The error is propagated back through the network, updating the weights based respectively on their contribution to the error. This process takes place until the MLP is either trained through every instance of the training data or as long as the error calculation stabilizes around a small value [30].

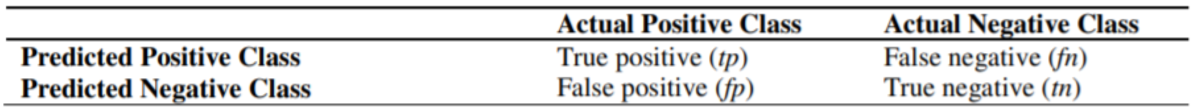
Classification Algorithm Evaluation

To evaluate the results of the classification algorithms, we used the following performance measures and techniques [31-34]:

- Cross-Validation (CV): Training an algorithm and evaluating its statistical performance on the same data yields an overoptimistic result. CV was raised to fix this issue, starting from the remark that testing the output of the algorithm on new data would yield a good estimate of its performance [31]. Thus, CV is a very commonly employed technique used to evaluate classifier performance. Schemes such as k-fold cross-validation are commonly used in classification [32]. The general procedure of k-fold CV is as follows:
    - Shuffle the dataset randomly.

  - Split the dataset into k folds
  - For each unique fold:
    - Label a fold as a hold out or test data set
    - Label the remaining fold as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Retain the evaluation score and discard the model
  - Summarize the performance of the model using the sample of model evaluation scores (e.g. average cross-validation performance)

- **Confusion Matrix:** A confusion matrix is a specific table layout that allows the visualization of the performance of a classification algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). In a confusion matrix, True Positive (tp) and True Negative (tn) values denote the number of positive and negative instances that are correctly classified. Meanwhile, False Positive (fp) and False Negative (fn) values denote the number of misclassified negative and positive instances, respectively, as shown in Table 2-1.



	Actual Positive Class	**Actual Negative Class**
Predicted Positive Class	True positive (*tp*)	False negative (*fn*)
Predicted Negative Class	False positive (*fp*)	True negative (*tn*)

Table 2-1 form of a Binary Classification Confusion Matrix [33].

- **Accuracy:** Accuracy metric measures the ratio of correct predictions over the total number of instances evaluated, calculated as:

$$\frac{tp + tn}{tp + fp + tn + fn}$$

- **Precision:** Precision measures the positive patterns that are correctly predicted from the total predicted patterns in a positive class, described by the following equation:

$$\frac{tp}{tp + fp}$$

- **Recall:** Recall measures the fraction of positive patterns that are correctly classified, using the equation bellow:

$$\frac{tp}{tp + tn}$$

- **F1-score:** This metric represents the harmonic mean between recall and precision values:

$$\frac{2 * Precision * Recall}{Precision + Recall}$$

Once evaluation metrics and techniques are established, Hyperparameter Tuning is a critical step for achieving higher classification performances in a given task. Every machine learning algorithm has a set of hyperparameters that need to be optimized for classification performance. Hyperparameter tuning can be applied in a variety of ways; the two fundamental methods that we follow in this thesis is Grid Search and Random Search [35]:

- **Grid Search:** During Grid Search the user specifies a finite set of values for each hyperparameter, and grid search evaluates the Cartesian product of these sets. This method suffers from the curse of dimensionality since the required number of function evaluations grows exponentially with the dimensionality of the configuration space. An additional problem of grid search is that increasing the resolution of discretization substantially increases the required number of function evaluations.
- **Random Search:** random search samples configurations at random until a certain budget for the search is exhausted. This tuning method works better than grid search when some hyperparameters are much more important than others. Random search is a useful baseline because it makes no assumptions on the machine learning algorithm being optimized, and, given enough resources, it is expected to achieve performance arbitrarily close to near optimal performance values.

2.5 Related Work

This section presents a number of relevant key studies within the domain of rating prediction. Firstly, approaches of rating predicting frameworks based on sentiment analysis of reviews are discussed. Secondly, a proposed research of feature analysis that shows the ways rating is correlated to them, is presented. Lastly, a rating predicting model using user specific information is analyzed.

2.5.1 Sentiment analysis of reviews

In [36], the authors focus on the sentiment analysis of annotated reviews (specifically in German Language) to train a model capable of predicting ratings. The structure of the information/data used is referred bellow:

- **all.data**: The list of all the written reviews, the application's name and the given rating.
- **all.txt:** The list of reviews that were used in the annotation process.

- **all.csv:** The list of all annotated subjective phrases and aspects, an ID for each subjective phrase and the polarity.
- **all.rel:** A list of all annotated relations between the subjective phrases and their aspects.

The prediction framework of that study is shown in Fig.2-14 below.

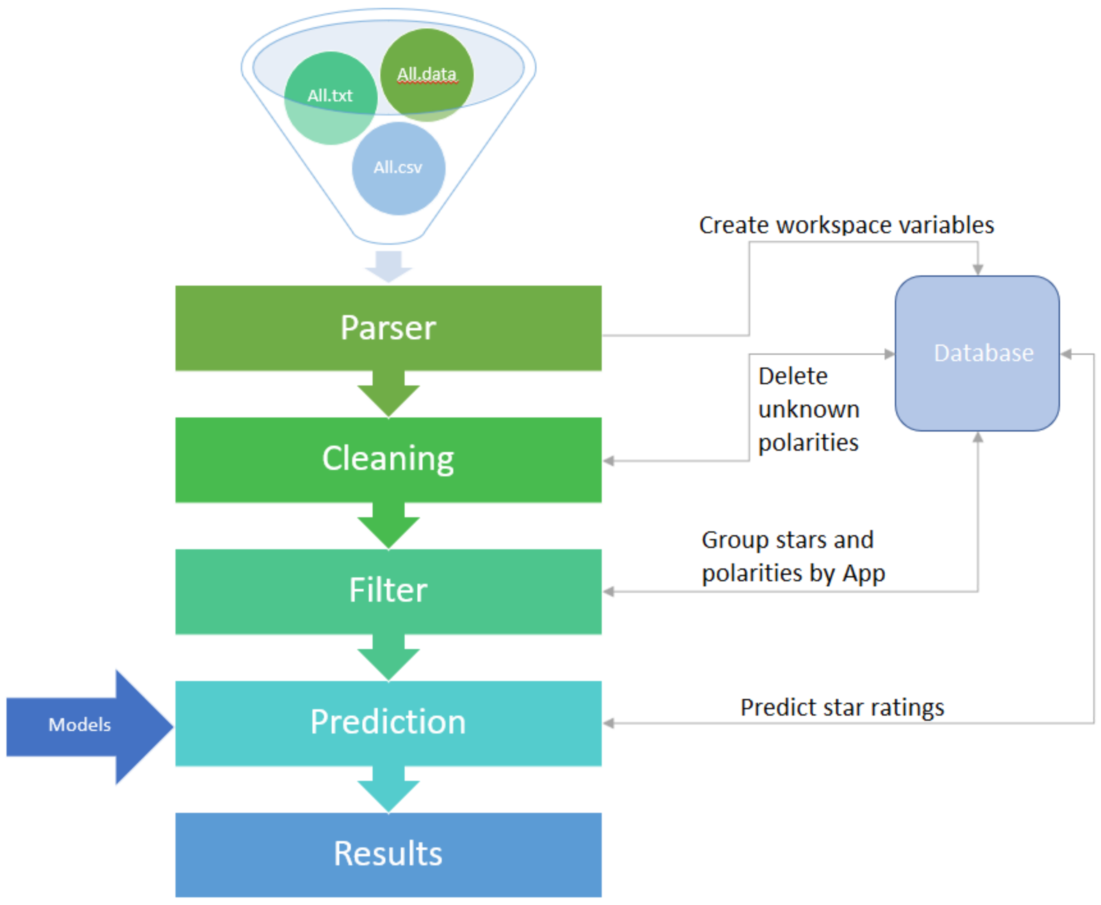
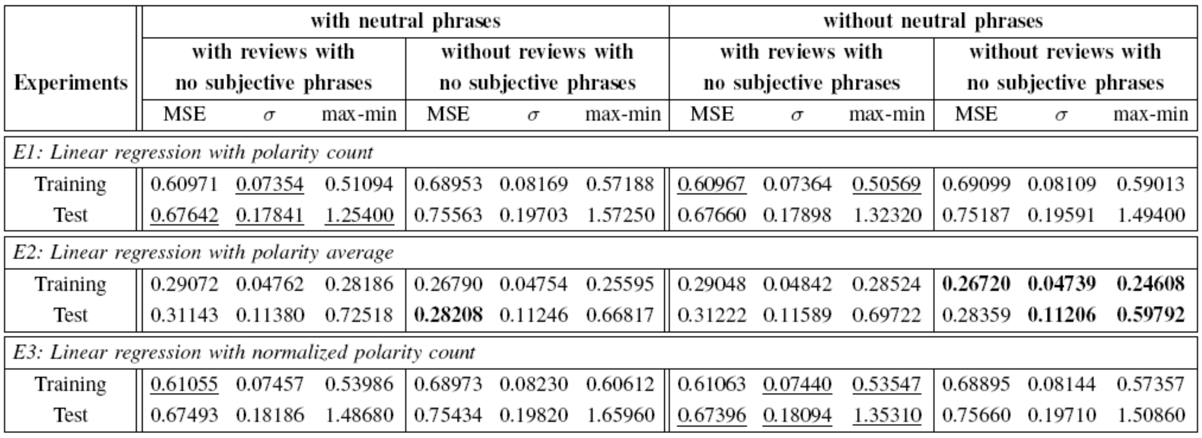


Figure 2-14. Prediction Process adapted from [36].

As the information is gathered, the subjective phrases of the sentences (information of the Reviews) were annotated and their polarity (i.e. Positive, Negative, Neutral meaning) was determined, creating an annotated collection of data filtered based on the predefined polarities. Afterwards, the annotated reviews were assigned to the corresponding ratings giving a clearer statistical view on how the analyzed reviews correspond to ratings. Thus, the predictor of user ratings was determined.

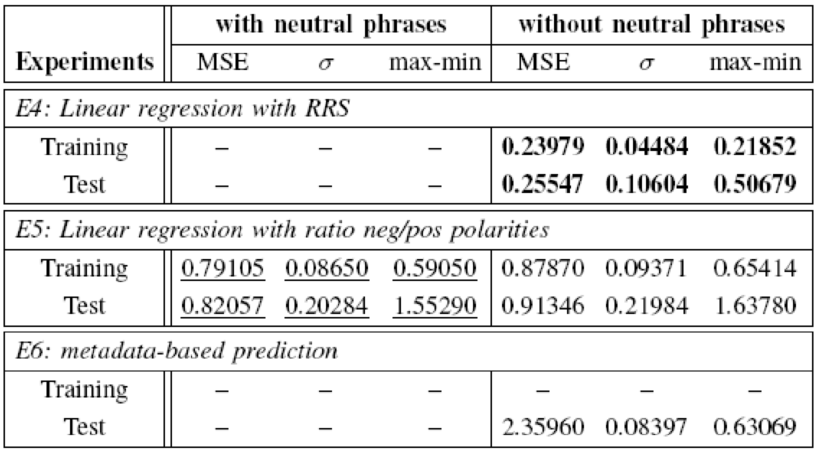
Regarding the prediction, multi-variate regressors [36] were used, taking into account the numbers of each positive, negative and neutral polarities (referred to as features). A cost function was resolved, in search of optimal parameters (tuning the predictor parameters) as well as normalization was applied at the values of the features, due to the different amount of reviews between the apps (i.e. some apps have many more reviews than others).

Two different sets of experiments were conducted In that study. The first set of experiments is used to assess the significance of sentiment by applying different regression models, as introduced above. The second set of experiments makes use of other predictors (average based). For each experiment the Monte Carlo Cross Validation [37] is used as a performance measure. The existence of the polarity of phrase-level sentiment is mandatory for textually-derived rating prediction to take place as seen from the results of Tables.2-2 and 2-3.



Experiments	with neutral phrases						without neutral phrases					
	with reviews with no subjective phrases			without reviews with no subjective phrases			with reviews with no subjective phrases			without reviews with no subjective phrases		
	MSE	σ	max-min	MSE	σ	max-min	MSE	σ	max-min	MSE	σ	max-min
E1: Linear regression with polarity count												
Training	0.60971	0.07354	0.51094	0.68953	0.08169	0.57188	0.60967	0.07364	0.50569	0.69099	0.08109	0.59013
Test	0.67642	0.17841	1.25400	0.75563	0.19703	1.57250	0.67660	0.17898	1.32320	0.75187	0.19591	1.49400
E2: Linear regression with polarity average												
Training	0.29072	0.04762	0.28186	0.26790	0.04754	0.25595	0.29048	0.04842	0.28524	**0.26720**	**0.04739**	**0.24608**
Test	0.31143	0.11380	0.72518	**0.28208**	0.11246	0.66817	0.31222	0.11589	0.69722	0.28359	**0.11206**	**0.59792**
E3: Linear regression with normalized polarity count												
Training	0.61055	0.07457	0.53986	0.68973	0.08230	0.60612	0.61063	0.07440	0.53547	0.68895	0.08144	0.57357
Test	0.67493	0.18186	1.48680	0.75434	0.19820	1.65960	0.67396	0.18094	1.35310	0.75660	0.19710	1.50860

Table 2-2 Mobile apps rating prediction: Importance of sentiment in the reviews [36].



Experiments	with neutral phrases			without neutral phrases		
	MSE	σ	max-min	MSE	σ	max-min
E4: Linear regression with RRS						
Training	–	–	–	**0.23979**	**0.04484**	**0.21852**
Test	–	–	–	**0.25547**	**0.10604**	**0.50679**
E5: Linear regression with ratio neg/pos polarities						
Training	0.79105	0.08650	0.59050	0.87870	0.09371	0.65414
Test	0.82057	0.20284	1.55290	0.91346	0.21984	1.63780
E6: metadata-based prediction						
Training	–	–	–	–	–	–
Test	–	–	–	2.35960	0.08397	0.63069

Table 2-3 Mobile apps rating prediction: Other (average-based) predictors [36].

Similarly, in [38], the authors stem out of the observation that application ratings might differ from given reviews. Proposing a unified rating system composed by combining the rating given by a user and a numeric polarity as it is extracted of the reviews only to generate the final rating, their proposition was based on a combination of sentiment analysis with an optimized probabilistic approach.

At the beginning of the process, some words were collected for each category as baseline words from sentiment-lexicons (SentiWordNet, MPQA and General Inquirer). Then candidate expressions were generated by splitting reviews into sentences and extracting the ways the words depended to each other. Due to the fact that a review might contain multiple expressions some reviews considered inconsistent by not being identified as positive or negative. Thus, the authors ultimately created a network of inconsistent and a network of consistent expressions---including a set of weighted edges that denotes the respective relations between the candidate expressions---based on propositional Logic/Propositional inference.

In the second set of experiments of that study a polarity probability was determined using both the positive and the negative sensitivity of the expressions, leading to a function. The resulting function was used by L-BFGS-B [39] algorithm to generate the polarities. Following the procedure Positive and Negative probabilities are assigned, in order to determine the sentiment of the user reviews. Finally, the polarity generated is converted to the rating scale (0 to 5), with the final rating of the application being equal to the mean value of both the scaled polarity and numeric app rating. Below an example is provided using this technique considering the rating and the review of a unanimous user for the app *Viber*.

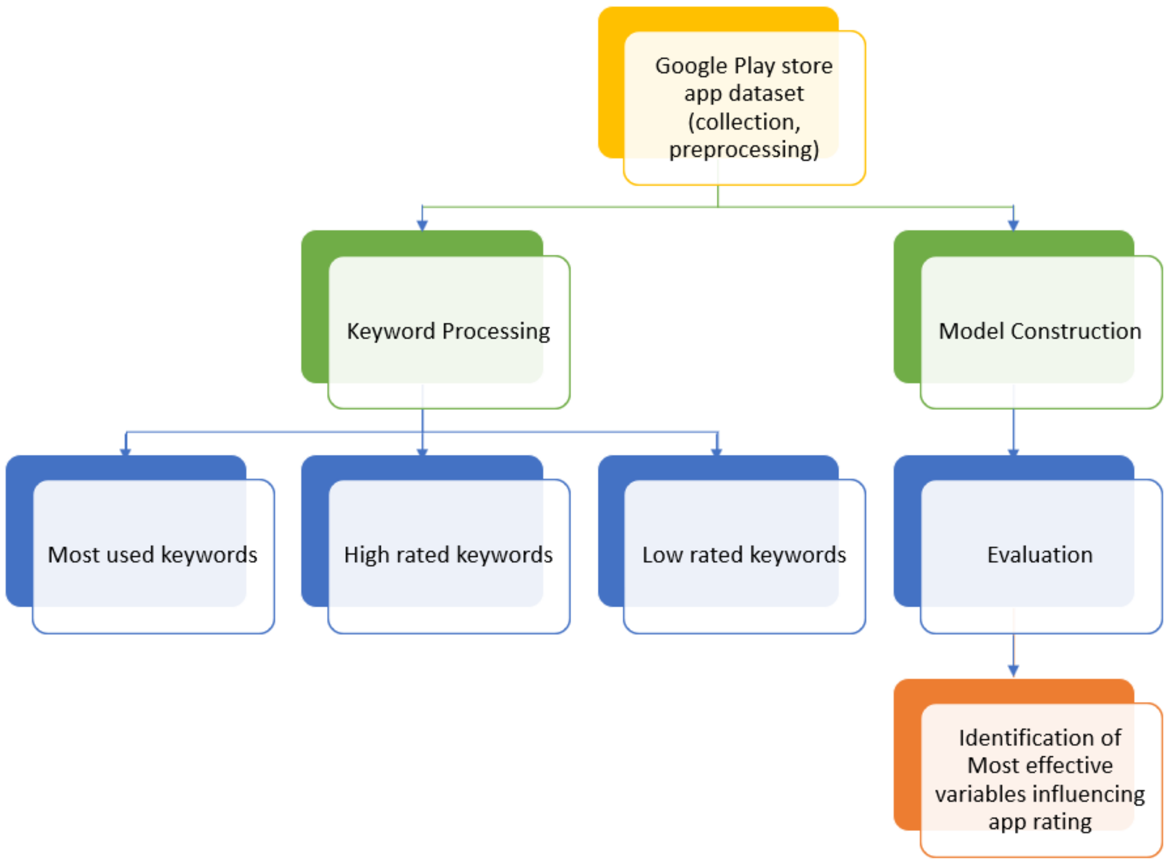
- **The given Rating:** 4
- **The given Review**:
  “Pathetic, the new update of the emoticons is stupid. I hate the emoticon panda icon opening his face that’s really weird, replace that with normal icon.”

A difference is observed between the starred rating and the review written by the user. Following the procedure described above the probabilistic outcome of the review was 0.135, and the normalized rating is 0.8. The average of those two numeric values results to 0.4675 which is a rating below average. Thus, the combined rating limits the level of distinction between the two unique evaluations posted by the user.

To conclude, both approaches are quite similar on using sentiment analysis and mathematical models to predict ratings. The main difference between those approaches is the fact that the second one proposes a new objective rating model based on reviews sentiment analysis, compared to the first one that makes rating predictions solely based on basic sentiment analysis using regression models.

2.5.2 Feature Analysis of User Ratings

In [40], the authors use different linear and non-linear regression models (Random Forest, Linear Regression, Support Vector Regression) for identifying the importance of app variables (i.e. app titles, number of words in title, digits in title). An additional focus of the study is the extraction of significant "keywords" (i.e. words from an app title) helpful on achieving higher app ratings. This study identifies the importance variable for app rating as a regression method. The framework of the proposed model is shown in Fig.2-15.



Google Play store app dataset (collection, preprocessing)
Keyword Processing
Model Construction
Most used keywords
High rated keywords
Low rated keywords
Evaluation
Identification of Most effective variables influencing app rating

Figure 2-15. Research Model adapted from [40].

The model is separated into two sections. In the first section a variety of variables are identified and then tested across different regression and correlation models. In the second section, keywords from app titles are processed and various rankings are calculated based on ratings and frequency.

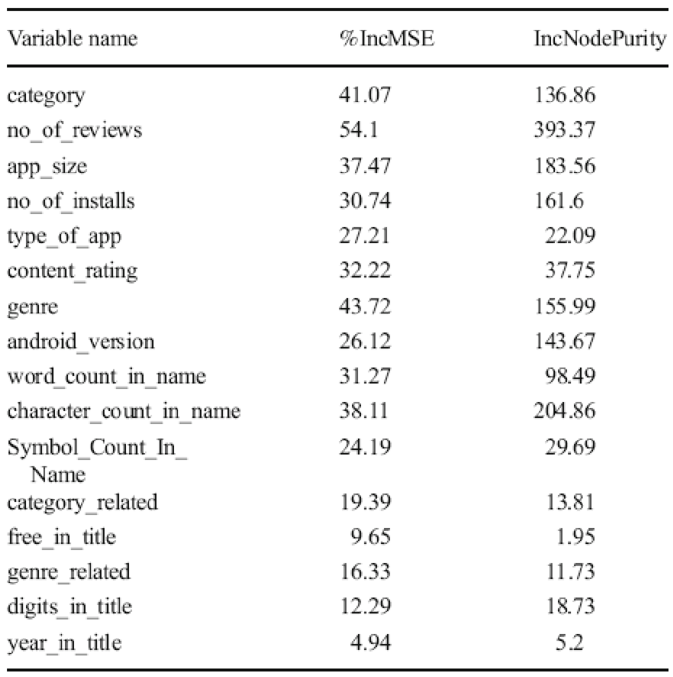
Bellow we provide a brief description of the regression models used in that study:

- **Random Forest**: Random Forest Regression model was used to determine the significance of all the variables and their level of affection. This instance was evaluated afterwards using Mean Squared Error.
- **Support Vector Machine (SVM)**: SVM Regression model was utilized to predict the significance of all numeric variables.
- **Linear Regression:** Linear Regression model was also leveraged to exploit the variable importance of different variables with ratings, using only numeric values.

- **Pearson Correlation**: a Pearson Correlation model was applied for computing the correlation between ratings and binary variables.

Following up are the evaluation techniques used, depending on each model.

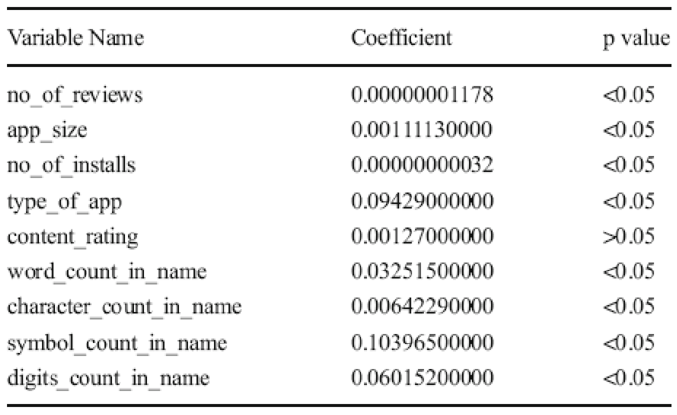
- **%IncMSE:** The higher MSE shows that the variable is more significant while the lower number shows that variable is less significant.
- **RMSE (Root Mean Square Error):** calculates the difference of the predicted values and the estimator
- **MAE (Mean Absolute Error)**: is the average distance between each point and the identity line.
- **p value**: p value determines the importance of the results during hypothesis test.



Variable name	%IncMSE	IncNodePurity
category	41.07	136.86
no_of_reviews	54.1	393.37
app_size	37.47	183.56
no_of_installs	30.74	161.6
type_of_app	27.21	22.09
content_rating	32.22	37.75
genre	43.72	155.99
android_version	26.12	143.67
word_count_in_name	31.27	98.49
character_count_in_name	38.11	204.86
Symbol_Count_In_ Name	24.19	29.69
category_related	19.39	13.81
free_in_title	9.65	1.95
genre_related	16.33	11.73
digits_in_title	12.29	18.73
year_in_title	4.94	5.2

Table 2-4 Random Forest Regression Results [40]

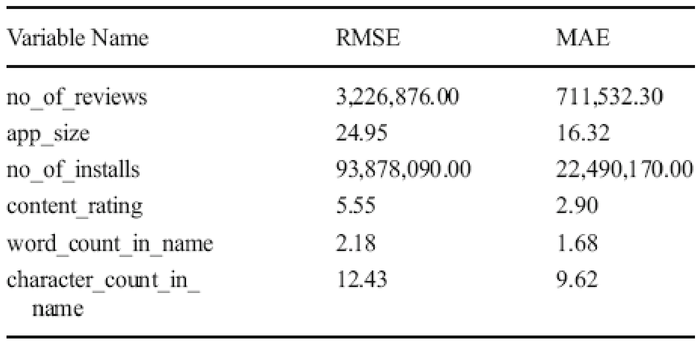
The results of the study (see Table.2-4) show that features such as *no_of_reviews*, *genre*, *character_count_in_name*, and *app_size* are predictive factors that influence the most and appear to highly impact the user ratings of the applications.



Variable Name	Coefficient	p value
no_of_reviews	0.00000001178	<0.05
app_size	0.00111130000	<0.05
no_of_installs	0.00000000032	<0.05
type_of_app	0.09429000000	<0.05
content_rating	0.00127000000	>0.05
word_count_in_name	0.03251500000	<0.05
character_count_in_name	0.00642290000	<0.05
symbol_count_in_name	0.10396500000	<0.05
digits_count_in_name	0.06015200000	<0.05

Table 2-5 Linear Regression Results [40]

Based on the linear regression model (see Table 2-5), *symbol_count_in_name* and *type_of_app* are also significant variables compared to the others. Although, the importance of these two variables is relatively low (3-9%), which is still higher than the coefficient values of the other variables (<2%). Therefore, the correlation between apps, rating, and other variables is not linear.

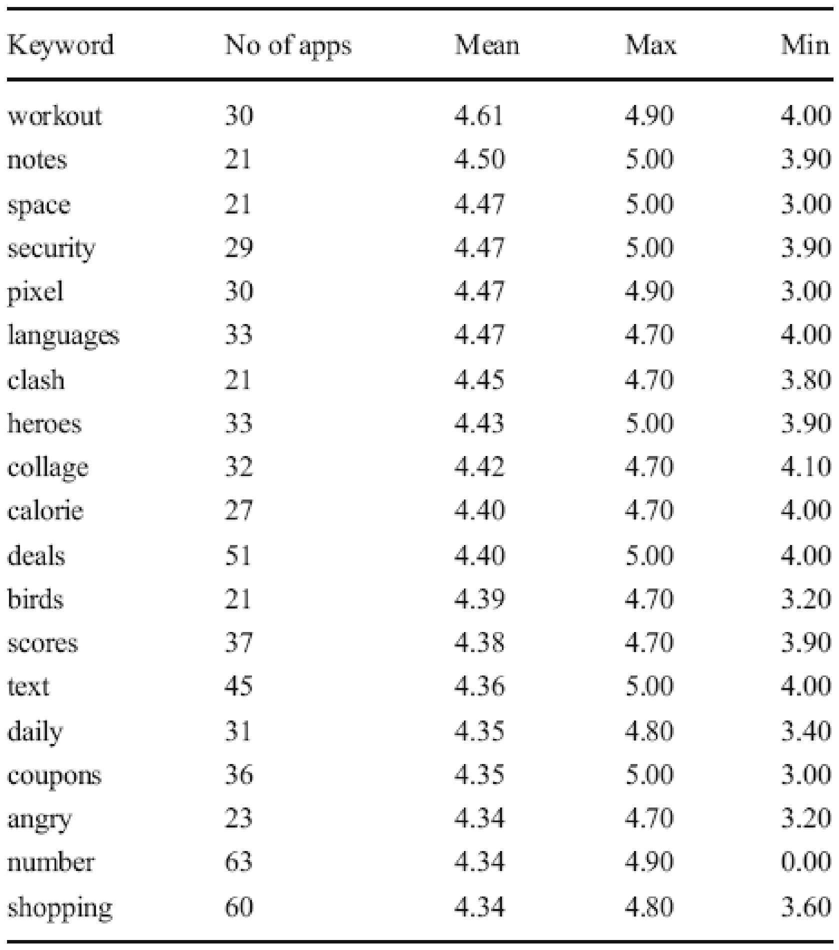


Variable Name	RMSE	MAE
no_of_reviews	3,226,876.00	711,532.30
app_size	24.95	16.32
no_of_installs	93,878,090.00	22,490,170.00
content_rating	5.55	2.90
word_count_in_name	2.18	1.68
character_count_in_name	12.43	9.62

Table 2-6 SVR results [40]

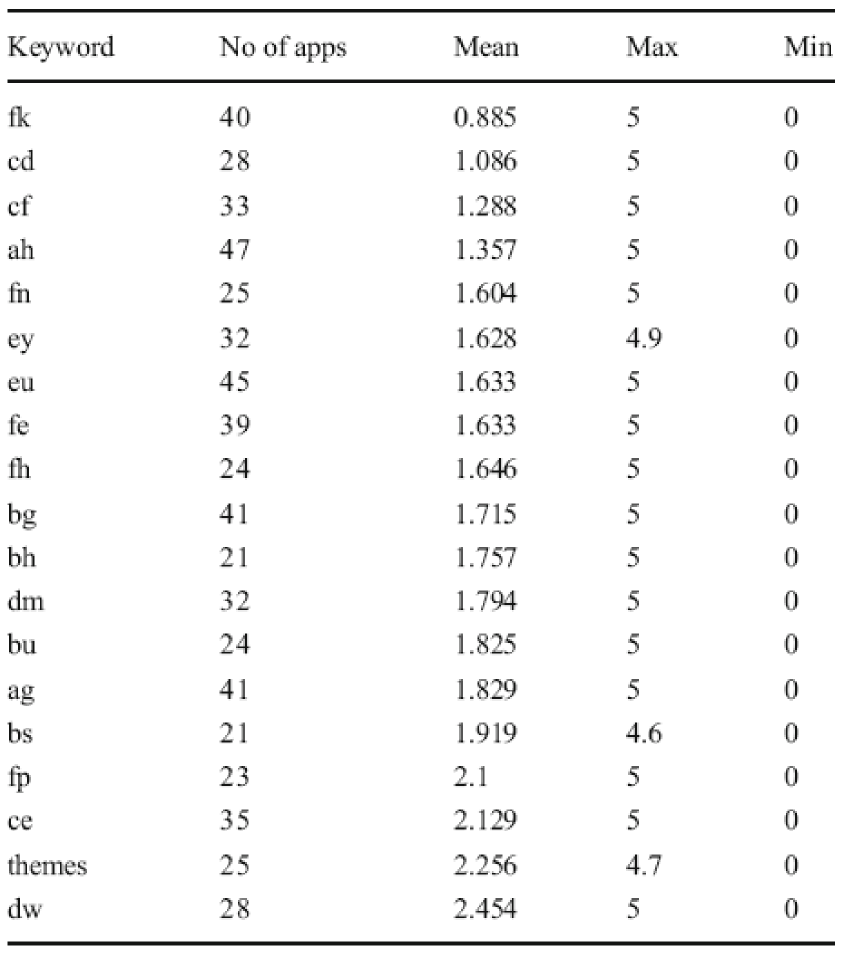
Similarly, as the SVR results indicate (Table 2-6), *word_count_in_name* and *content_rating* are significant variables that influence user ratings.

As for the second part of that study---i.e. keyword analysis---the following tables present the core results of the study. :



Keyword	No of apps	Mean	Max	Min
workout	30	4.61	4.90	4.00
notes	21	4.50	5.00	3.90
space	21	4.47	5.00	3.00
security	29	4.47	5.00	3.90
pixel	30	4.47	4.90	3.00
languages	33	4.47	4.70	4.00
clash	21	4.45	4.70	3.80
heroes	33	4.43	5.00	3.90
collage	32	4.42	4.70	4.10
calorie	27	4.40	4.70	4.00
deals	51	4.40	5.00	4.00
birds	21	4.39	4.70	3.20
scores	37	4.38	4.70	3.90
text	45	4.36	5.00	4.00
daily	31	4.35	4.80	3.40
coupons	36	4.35	5.00	3.00
angry	23	4.34	4.70	3.20
number	63	4.34	4.90	0.00
shopping	60	4.34	4.80	3.60

Table 2-7 Keywords with high rating [40]



Keyword	No of apps	Mean	Max	Min
fk	40	0.885	5	0
cd	28	1.086	5	0
cf	33	1.288	5	0
ah	47	1.357	5	0
fn	25	1.604	5	0
ey	32	1.628	4.9	0
eu	45	1.633	5	0
fe	39	1.633	5	0
fh	24	1.646	5	0
bg	41	1.715	5	0
bh	21	1.757	5	0
dm	32	1.794	5	0
bu	24	1.825	5	0
ag	41	1.829	5	0
bs	21	1.919	4.6	0
fp	23	2.1	5	0
ce	35	2.129	5	0
themes	25	2.256	4.7	0
dw	28	2.454	5	0

Table 2-8 keywords with Low Rating [40]

The results of variable analysis show that there are some factors that might impact the rating of apps. As app ratings have a small and finite scale, minor changes regarding the rating can contribute into achieving an improved outcome, thus higher number of downloads and visibility. Thereafter, even the factors of lower importance are considered important to the domain. The keywords analysis results (Tables 2-7 and 2-8) show that there are some words that appear mostly on higher rated applications, while on the other hand there are some keywords appearing mainly to low rated applications.

2.5.3 User specific rating prediction

In [41], the authors treat every unique downloaded application as a feature that describes the user's personal preference. First of all, "User-Specific Interest" is defined as the representation of applications with increased level of relevance for the target user, used to express user's distinctive interests. The discussed interface, is built upon the idea of predicting applications' ratings based on user-specific information, a framework operating as seen in Fig. 2-16.

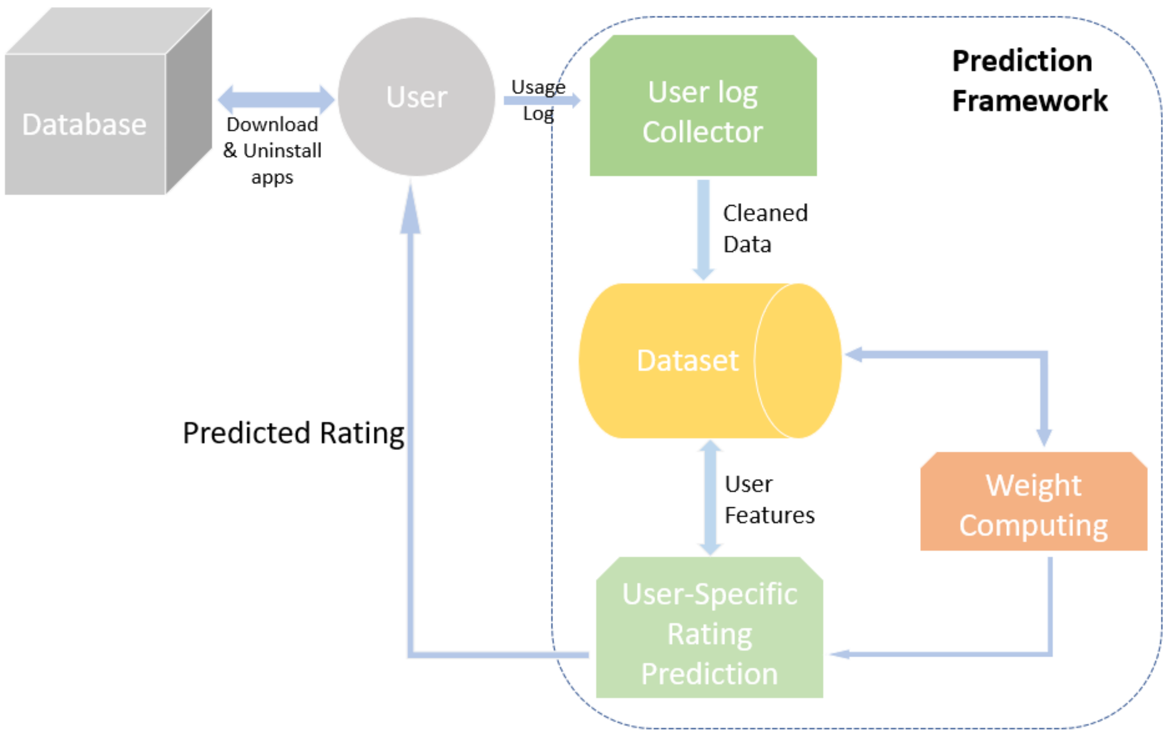
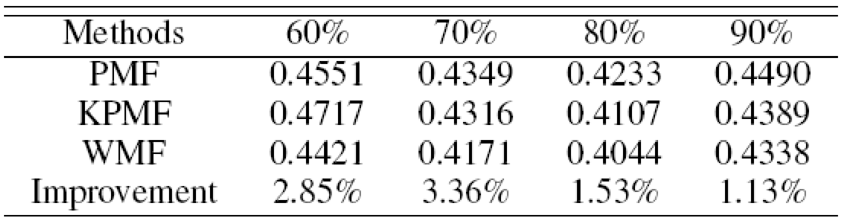


Figure 2-16. Weight-based rating prediction framework [41].

Based on Fig.2-16 information related to each user's installs-uninstalls log is retrieved from application marketplaces, and forwarded to a prediction server. Then, the ratings for these apps are predicted by breaking down usage logs, and user-specific apps. The authors used the Weight-Based Matrix Factorization (WMF) model, in which the weight of applications was computed based on Term Frequency Inverse Document Frequency (TF-IDF) algorithm [42].A measurement metric called TMAE [43] similar to mean absolute error (MAE), was used to measure the difference between the estimated values and the observed values, utilizing top-k selection process.

The proposal of this research was compared to the existing methods of Probabilistic Matrix Factorization (PMF) [44] and Kernelized Probabilistic Matrix Factorization (KPMF) [45]. During the evaluation process, all rating prediction methods are tested on 4 different training sets that are subdivisions of the starting dataset in the following matter 60%, 70%, 80% and 90%, respectively. In particular, a dataset deviation of 60% implies that 60% of the known rating serves as an input of the matrix and is utilized in order to predict missing rating values. The known rating entries occupy 3% of the total rating matrix. All the experiments were conducted with k = 10. The following results (see Table 2-9), shows that WMF method improves the results of the rating predictions as it performs better than the other two methods on every experiment.



Methods	60%	70%	80%	90%
PMF	0.4551	0.4349	0.4233	0.4490
KPMF	0.4717	0.4316	0.4107	0.4389
WMF	0.4421	0.4171	0.4044	0.4338
Improvement	2.85%	3.36%	1.53%	1.13%

Table 2-9 MAE Accuracy comparison on Rating prediction [41].

2.5.4 Related work compared to our approach

Approach 2.5.1: The research described in this sub-section of related work, focuses mainly on analyzing textual reviews and predicting ratings, using only this feature as a guideline. On the other hand, on our approach we do not make use of textual data, contrariwise we take advantage of structured data that includes information about application features (e.g. number of reviews, size of app). Furthermore, our approach includes a classification treatment of the problem which is more effective compared to regression, on experience modeling problems.

Approach 2.5.2: This sub-section's approach makes use of just nominal features included in applications' textual characteristics; and utilizes regression models to make a variety of predictions. An approach that differs both on the features used, but also on the machine learning model as we proceed to solve the problem using classification.

Approach 2.5.3: In this case as well, the feature information was different. The users' log was used as feature providing a user-specific approach compared to our solution that uses openly accessible data.

Summary

In this Chapter , we introduced fundamental definitions that are utilized in the thesis, such as Marketability. We gave a brief description of data structures, therefore introducing the way structured data is cleansed and interpreted. Also, we provided an extensive analysis of Machine Learning, with a main focus on paradigms that are used in our approach i.e. Classification, Classification Algorithms, Evaluation Metrics. Furthermore, related work to the thesis was quoted in order to state the problem and make an introductory to our approach.

CHAPTER 3

OUR APPROACH

Section 2.5 showed that the frameworks for resolving application rating prediction, treat the problem as either regression, i.e. predicting exact numerical value of ratings, or by analyzing the factors that affect rating. Our approach, instead, formulates the task as a classification problem, since our main goal is to investigate applications based on their marketability level. Furthermore, as stated by Yannakakis & Togelius [5][46], on a common problem of creating player models using ratings, the use of classes eliminates part of the inter-personal biases introduced with ratings. As player modeling is similar to customer modeling (both essentially being subclasses of user modeling), classification is considered to be effective since *rating* is the feature we focus at.

Thereafter, we survey the ability to predict the predefined rating "hot-zone" (an interval of ratings that is correlated to marketable apps i.e. The class "High") of an application based on its features, answering the core research question about the ability of an application to be effectively marketed on Google Play.

3.1 Overview

To solve the classification problem at hand, we view the problem as in three different ways: 1) as a Binary Classification problem, 2) as a Binary Classification problem with a varying threshold between classes and 3) as a Multiclass Classification problem. In binary classification, the data of Section 3.2 has been labelled as two classes "Low" and "High" based on the mean count of ratings. The second solution is binary as well, although applying a threshold that cuts off values with a purpose to observe if better training is attained, thus better learning. In the last framework, we label the data in three rating classes, i.e. "Low", "Mid" and "High". All three frameworks utilize the same classifiers, cross-validation method, hyperparameter tuning technique and evaluation approach.

In addition to our formulation of the rating prediction problem, our approach consists of the following stages:

- Data Preprocessing & Data Analysis.
- Machine Learning phase.
- Evaluation of solutions and models.

3.2 Data Preprocessing & Data Analysis

To test our approaches we use a dataset from Kaggle, which includes details of the applications on Google Play. In the dataset used there are 13 features including the application name, the category each app belongs in, the overall user rating of each app, the number of reviews for each app, the size of each app, the number of user downloads for each app, the type of each app (Paid, Free), the price of each app (null in case app is free), the age group each application is targeted at, the secondary genres an app belongs to (apart from its main category i.e. An application might belong to "Games" category and on "Family" category as well, with the second one referring to secondary genres), the date of the last update of each app on the platform, the current version of each app, and the minimum required Android version for each app in order to operate. The dataset contains information for **10,841** applications by year 2018, split into two data type categories float64 (1) and object (12) as shown in Table. 3-1.

Column - Feature (data type) - non null values

- 0. App (object) - 10,841 non-null values
- 1. Category (object) - 10,841 non-null values
- 2. Rating (float64) - 9,367 non-null values
- 3. Reviews (object) - 10,841 non-null values
- 4. Size (object) - 10,841 non-null values
- 5. Installs (object) - 10,841 non-null values
- 6. Type (object) - 10,840 non-null values
- 7. Price (object) - 10,841 non-null values
- 8. Content Rating (object) - 10,840 non-null values
- 9. Genres (object) - 10,841 non-null values
- 10. Last Updated (object) - 10,841 non-null values
- 11. Current Ver (object) - 10,833 non-null values
- 12. Android Ver (object) - 10,838 non-null values

Table 3-1. Features of the Dataset

At a first glance, *rating* is the feature with the fewer values compared to the other features. A high level statistical analysis on the *rating* of the apps, provided us with the information displayed in Table 3-2.:

mean = 4.19	std = 0.53
min =1.0	max = 19.0

Table 3-2 Description of Rating column (data type float64)

The ratings of the applications in the Google Play store range from 1 to 5, in that case a maximum value of 19.0 is considered an outlier. When the rating is equal to 19.0, the remaining features of this entry are filled with inconsistent and incomplete data. The first value of this record is missing (App name). All the other values are respectively propagated backward starting from "Category" towards the "Current Ver", and the last column which is "Android Ver" is null. This specific row is removed from the dataset.

A core challenges in our analysis , is the missing values of ratings , which sum up to 1.474 values (I.e. 13.5% of all rating values). These rows cannot be used for further Machine Learning processing. However, to avoid removing this data and lose information (as our dataset has limited size), we decided to fill the empty values using the median of the remaining ratings, resulting to the rating statistics given in Table 3-3:

mean = 4.20	std = 0.48
min =1.0	max = 5.0

Table 3-3 Description of Rating column after removing row 10,472 (data type float64)

Type, Content Rating, Current Ver. and Android Ver. (6,8,11,12 respectively) features contain 13 empty values in total (less than 0,2% of the rows), considering that it will not affect the dataset.

Therefore, we choose to completely remove them from the dataset. The rating distribution after removing and modifying the null values is shown in Fig.3-1.

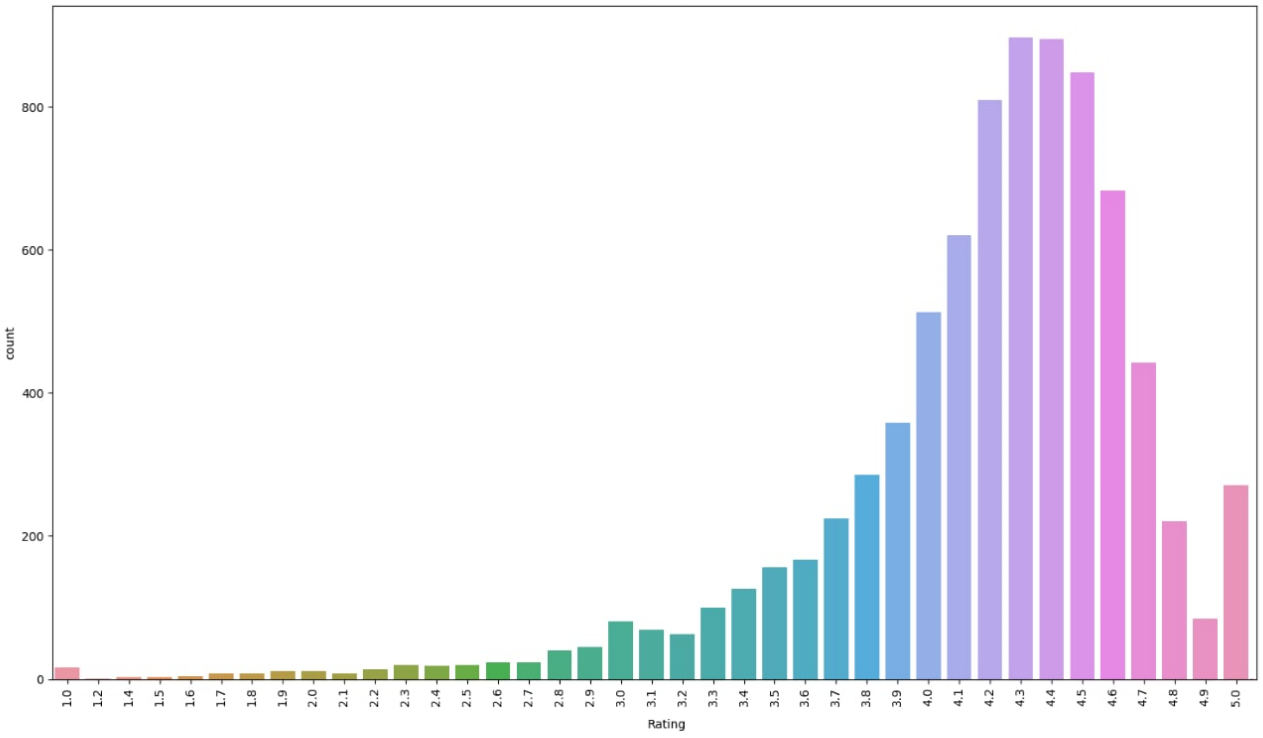


Figure 3-1 Distribution of rating after modifying NaN values.

Similar to the rating feature, other features are cleaned and normalized. Analyzing the dataset, we decided to focus the data cleaning process on columns that contain values with unwanted characters as described:

- **Size:** data entries containing either the character 'k' or the symbol 'M', referring to 'kilobytes' and 'megabytes' respectively. The characters were removed and the value was converted to the corresponding float.
- **Installs, Price, Last Updated, Current Ver, Android Ver:** Punctuation and special symbols are removed (i.e. +, {, }, [,]) and the values is casted to floats.

The remaining features (App, Category, Type, Content Rating, Genres, Reviews) did not contain any character or symbols that required removal. Therefore, label encoding is performed on these features to map each value to a number. Thus, values that ties or are close to each other losing such information after encoding.

After the data cleaning and preliminary analysis, we investigated the correlations between the features.

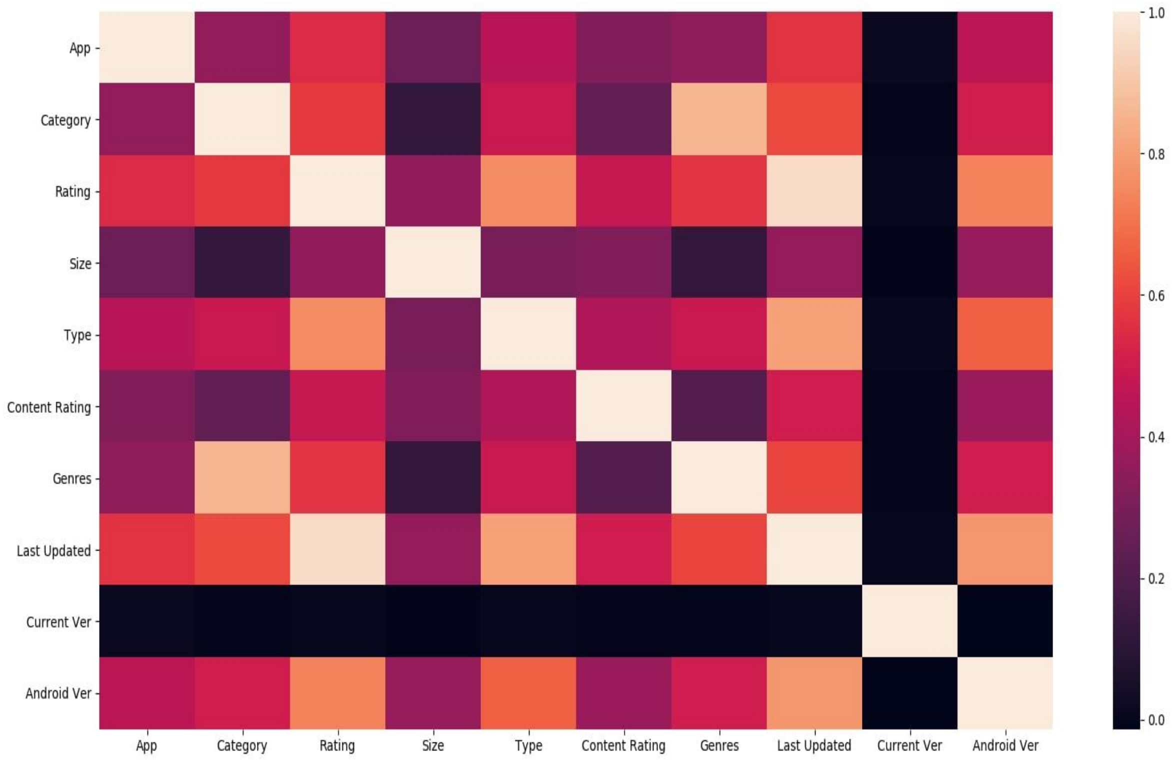


Figure 3-2 Feature correlation matrix

For this purpose, we used the Pearson's correlation coefficient. Pearson's correlation is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample. Normalization of the covariance between the two variables gives an interpretable score, that describes the level of correlation. Based on Fig.3-2, the correlation among "Rating" and the other features is not strong (Mostly around 0.6), besides "Last Updated", meaning, that features are dependent upon each other. The strong correlation between "Last Updated" and "Rating" features (>90%) is of significant importance on predicting "Rating". Note that Fig.3-2 illustrates that the features have only positive or null correlations, therefore negative correlations do not exist. This implies that the features change either in the same direction or there is no relationship between them. In general, most of the correlations between the features are positive and range from 35% to 70%, not indicating any strong relations, although can be helpful in matters of pattern recognition. Exceptional case is "Current Ver" feature that does not correlate at all with any of the other features (Pearson's correlation = 0.0)

3.3 Proposed Machine Learning Models

3.3.1 Classifying Rating

As described on previous sections, the proposed approach implements classification models. The first step is to classify the *Ratings*. The splitting criterion used to classify the ratings was the mean value (Table 3-2 & Figure 3-1). With this approach a balanced dataset is assumed containing almost the same number of objects per class. For the binary classification, we assigned values that exceeded the rating of 4.2, in class ‘High’ (5523) and those lower or equal to 4.2 in class ‘Low’ (5307).

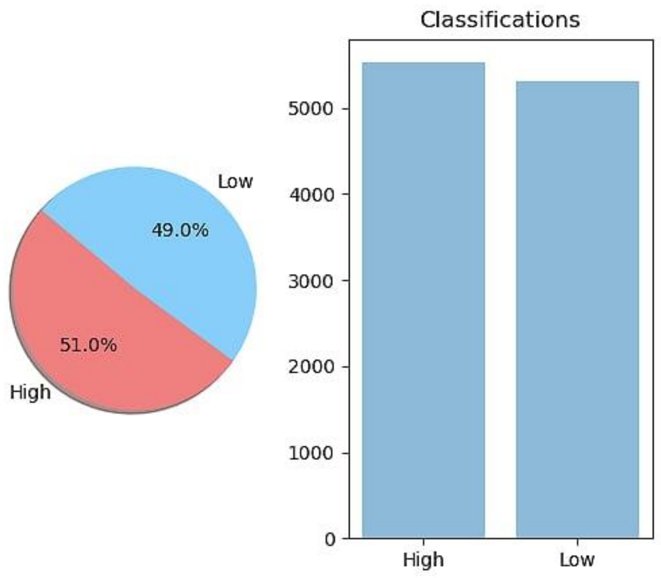


Figure 3-3 Rating Binary classification.

On the other hand, to create the binary classification paradigm with a varying threshold we have set 4.2 as ‘m’ and we dropped from the dataset the data with rating values that belong in the intervals [m-x, m], [m, m+y], where ‘x’ and ‘y’ are variables defining the range of values to be removed respectively on left and right side of mean (threshold = y+x) as in Fig.3-4. The condition we tried to satisfy, was the threshold to cut off the values in a way, that one class would not strongly outnumber the other (e.g. 80%-20%). Nevertheless, the threshold was assigned on the first case as threshold=0.5 and on the second case as threshold=1.0, as those two thresholds would both satisfy the condition described above and would be different enough for this move to be meaningful. That way, the first model would not include the values that belong at [4.0,4.4] and the second model would not include the values that belong at [3.5,4.4]. Therefore, we performed a mutual approach on assigning the classes as the one used above..

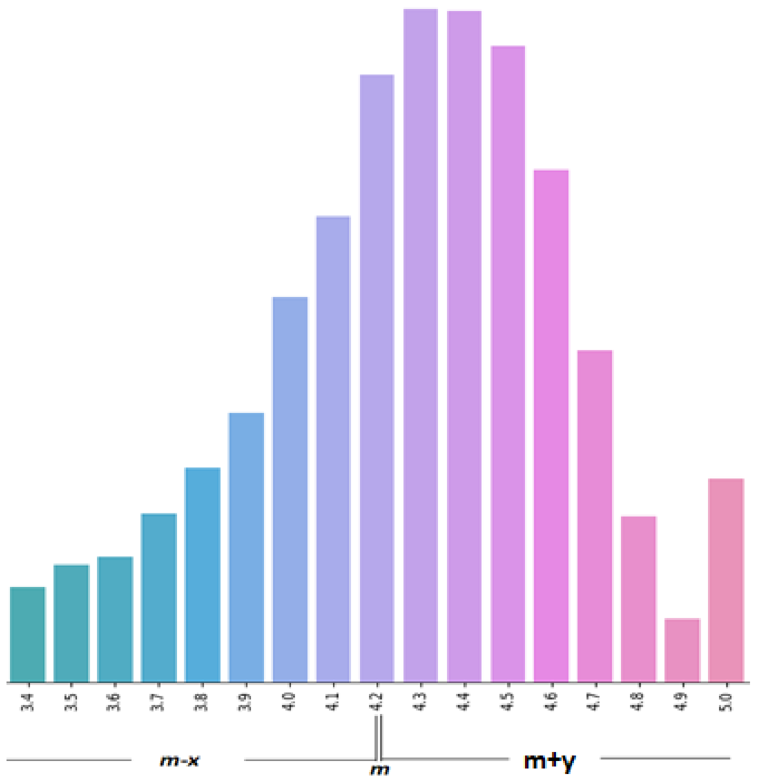


Figure 3-4 Threshold creation using the zoomed Fig.3-1.

The outcome is presented, bellow:

- Threshold = 0.5, “High” = 2459, “Low” = 3496

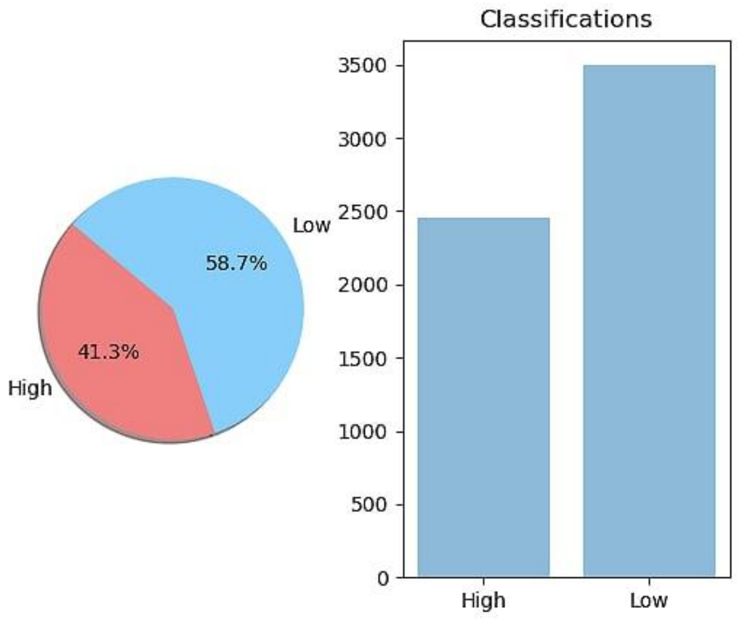


Figure 3-5 Rating Binary Classification with Threshold=0.5

- Threshold = 1.0, “High” = 2459 – “Low” = 2382

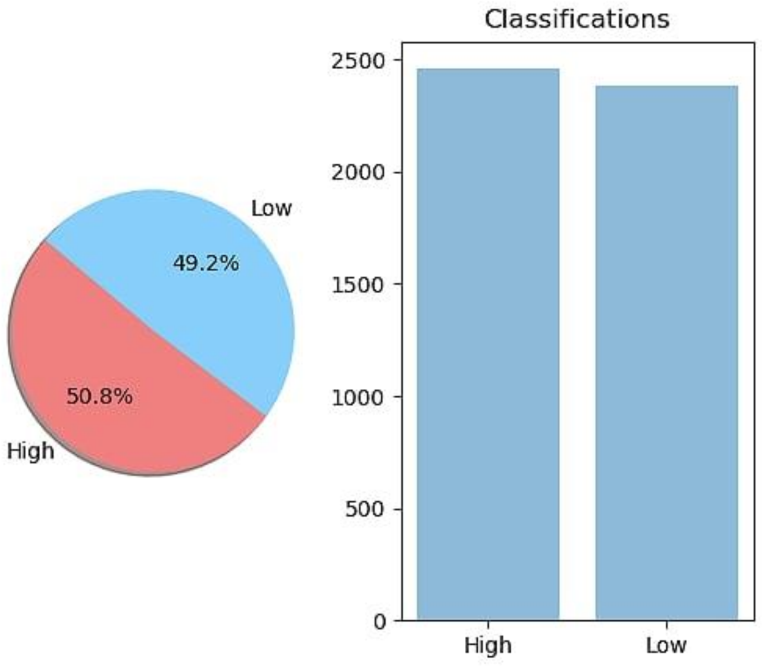


Figure 3-6 Rating Binary Classification with Threshold=1.0

The last approach was the multiclass classification paradigm, creating three classes “Low” (3162), “Mid” (4375), “High” (3293) using the same approach, of keeping the sizes of the classes as similar as possible.

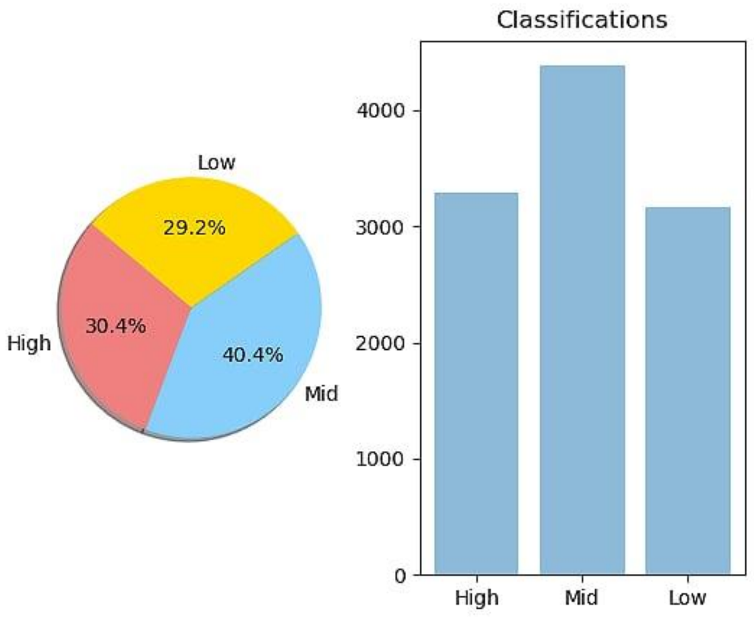


Figure 3-7 Rating Multi-class Classification.

The baseline for each solution was calculated, as follows:

$$baseline = \frac{number\ elements\ of\ the\ bigger\ class}{number\ of\ total\ elements\ in\ classes}$$

3.3.2 Machine Learning Adaptation

Once the labelling of the data is completed, the next step is to run and test our Machine Learning algorithms. In our case, we use a variety of popular algorithms-classifiers to predict ratings: those are Random Forest, Decision Trees, Multi-layer Perceptron, K-Nearest Neighbors, Logistic Regression and Support Vector Machines. The goal of this step is to find the best classifier for our problem by benchmarking the aforementioned algorithms based on the accuracy metrics as described in section 2.4.2.

The techniques are divided into two categories:

- Non-deterministic: The results vary in terms of consistency on each execution, due to the non-deterministic nature of some classifiers. Random Forest, Decision Trees & Multi-layer Perceptron are the classifiers on this set.
- Deterministic: The results are consistent in every single execution of the algorithm. . The algorithms of k-Nearest Neighbor, Logistic Regression and Support Vector Machines belong in this category.

We are handling the randomness of the first set by executing each algorithm ten consecutive times and keeping track of each iteration's results. Note that, firstly, we performed hyperparameter tuning (section 2.4.2) to identify the optimal parameters that maximize the accuracy of each classifier.

As we use six different classifiers that need to be tuned, we leverage advantage of Randomized Hyperparameter tuning, which utilizes Stratified k-Fold Cross-Validation to evaluate the performance of the tested parameters (section 2.4.2).

Then, the tuning method is as follows:

- sampled 20 different sets of parameters; from a predefined parameters' matrix
- tested them
- evaluated them
- and returned the sample that acquired the highest accuracy

We preferred this method rather than exhaustive search (i.e. trying every possible combination of parameters) because it is less time consuming, but in practice it also yields promising results on selecting hyperparameters.

The visualization of the process described above is depicted in Fig 3-8.

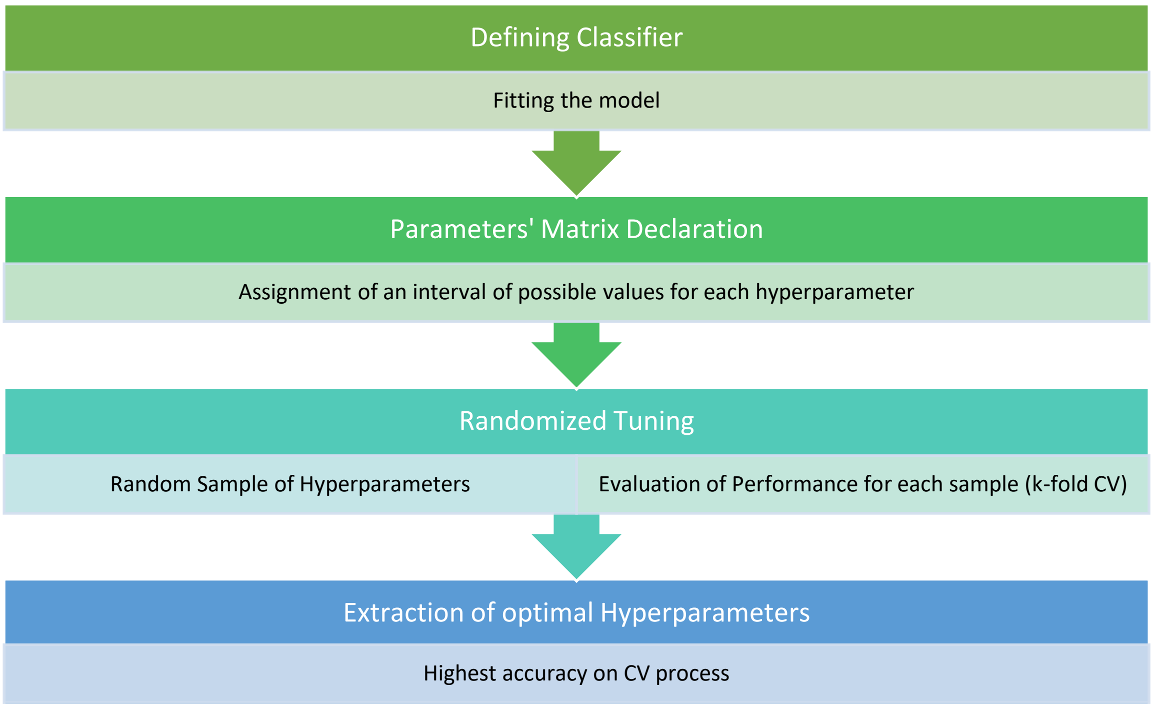

Defining Classifier
Fitting the model
Parameters' Matrix Declaration
Assignment of an interval of possible values for each hyperparameter
Randomized Tuning
Random Sample of Hyperparameters
Evaluation of Performance for each sample (k-fold CV)
Extraction of optimal Hyperparameters
Highest accuracy on CV process

Figure 3-8 Hyperparameter tuning process.

Table 3-5 shows the configuration of the hyperparameters after the tuning, while Table 3-4 outlines the hyperparameters used per algorithm.

Random Forest

- **criterion:** The function to measure the quality of a split.
- **max_depth:** The maximum depth of the tree
- **n_estimators:** Number of trees in the forest
- **min_samples_split:** The minimum number of samples required to split an internal node
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node
- **max_features:** The number of features to consider when looking for the best split (the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features.

Decision Tree

- **max_depth:** The maximum depth of the tree
- **max_features:** The number of features to consider when looking for the best split
- **min_samples_split:** The minimum number of samples required to split an internal node
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node
- **splitter:** The strategy used to choose the split at each node
- **criterion:** The function to measure the quality of a split.

Multi-layer Perceptron

- **hidden_layer_sizes:** number of neurons of each hidden layer.
- **solver:** The solver for weight optimization.
- **learning_rate:** Learning rate schedule for weight updates.
- **activation:** Activation function for the hidden layer.
- **alpha:** L2 penalty (regularization term) parameter.

Logistic Regression

- **penalty:** Used to specify the norm used in the penalization
- **solver:** Algorithm to use in the optimization problem.
- **C:** Inverse of regularization strength

k-Nearest Neighbor

- **n_neighbors:** Number of neighbors to get
- **metric: T**he distance metric to use for the tree
- **weights:** Weight function used in prediction

Support Vector Machines

- **C:** Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.
- **kernel:** Specifies the kernel type to be used in the algorithm
- **gamma:** Kernel coefficient

Table 3-4 Description of all the parameters we tuned on every algorithm

Tuned Hyperparameters:

Binary Classification

- **Random Forest:** (criterion= 'gini', max_depth = 6 , max_features = 3 , min_samples_leaf = 2 , min_samples_split = 12 ,n_estimators =1000)
- **Decision Tree:** (max_features='auto', max_depth=6, min_samples_split=11, splitter='best', min_samples_leaf=2, criterion='gini', splitter='best')
- **Multi-layer Perceptron:** (solver = 'lbfgs' , learning_rate= 'adaptive' , hidden_layer_sizes = (8,) ,activation='tanh',alpha=0.01)
- **k-Nearest Neigbor:** (n_neighbors=25, metric = 'manhattan', weights='uniform')
- **Logistic Regression:** (penalty ='l1', solver = 'liblinear', C =0.02811768697974228)
- **Support Vector Machines:** (C = 1000, kernel= 'rbf', gamma = 0.001)

Binary Classification (threshold = 0.5)

- **Random Forest:** (criterion='gini' , max_depth = 9 , max_features = 'auto' , min_samples_leaf = 5 , min_samples_split = 12 ,n_estimators = 1000)
- **Decision Tree:** (max_features='auto', max_depth=9, min_samples_split=5, splitter='random', min_samples_leaf=3, criterion='entropy', splitter='best')
- **Multi-layer Perceptron:** (solver = 'lbfgs' , learning_rate= 'constant' , hidden_layer_sizes = (8,),activation='tanh',alpha=0.1)
- **k-Nearest Neigbor:** (n_neighbors = 25, metric='euclidean', weights='uniform')
- **Logistic Regression:** (penalty ='l2', solver='liblinear', C=109.85411419875572)
- **Support Vector Machines:** (C=1000, kernel='rbf', gamma=0.0001)

Binary Classification(threshold = 1.0)

- **Random Forest:** (criterion= 'gini' , max_depth = 12 , max_features = 'auto' , min_samples_leaf = 4 , min_samples_split = 10 ,n_estimators = 300)
- **Decision Tree:** (max_features='auto',max_depth=12,min_samples_split=3,splitter='random',min_samples_leaf=4,criterion='gini', splitter='best')
- **Multi-layer Perceptron:** (solver = 'lbfgs' , learning_rate= 'adaptive' , hidden_layer_sizes = (8,), activation='tanh',alpha=0.01)
- **k-Nearest Neigbor:** (n_neighbors = 29, metric='euclidean', weights='uniform')
- **Logistic Regression:** (penalty='l1', solver='liblinear', C=719.6856730011514)
- **Support Vector Machines:** (C = 1000, kernel='rbf', gamma = 0.0001)

Multiclass Classification

- **Random Forest:** (criterion= 'gini' , max_depth = 6 , max_features = 'auto' , min_samples_leaf = 3 , min_samples_split = 10 ,n_estimators =300)
- **Decision Tree:** (max_features=3 ,max_depth=6,min_samples_split=11,splitter='best',min_samples_leaf=3,criterion='gini', splitter='best')
- **Multi-layer Perceptron:** (solver = 'adam' , learning_rate= 'adaptive' , hidden_layer_sizes = (8,2),activation='tanh',alpha=0.0001)
- **k-Nearest Neigbor:** (n_neighbors = 27 , metric= 'manhattan', weights='uniform')
- **Logistic Regression:** (penalty ='l1', solver='liblinear' , C=0.2682695795279725)
- **Support Vector Machines:** (C=1000, kernel='rbf', gamma=0.0001)

Table 3-5 Tuned Hyperparameters

A few key aspects observed during the parameter tuning process and according to the resulted parameters (according to Table 3-5) are as follows:

- Whenever the Tree classifiers' (Random Forest, Decision Tree) "max_depth" parameter was set at values higher than the number of features, the models would overfit.
- Logistic Regressions' parameter "C", appears to be the parameter that affects the tuning the most.
- K-Nearest Neighbors uses a significant population of neighbors (more than 20) to perform best.
- Support Vector Machines, appear to provide optimal results in this dataset when using RBF-kernels.

- Multi-layer Perceptron appears to perform best with the following architectures: one hidden layer of 8 neurons in binary classification cases and two hidden layers of 8 and 2 neurons respectively in multiclass classification case (the number of neurons should not exceed the number of features).

3.4 Performance Evaluation

Using the tuned hyperparameters, we created a Machine Learning framework for general evaluation purposes, in which every case is using the parameters as described in Table 3-5 in order to acquire the Confusion Matrices, Classification Reports, CV train & test accuracy diagrams and t-Tests between, the classifiers results. The process is described in the Fig. 3-9 as follows:

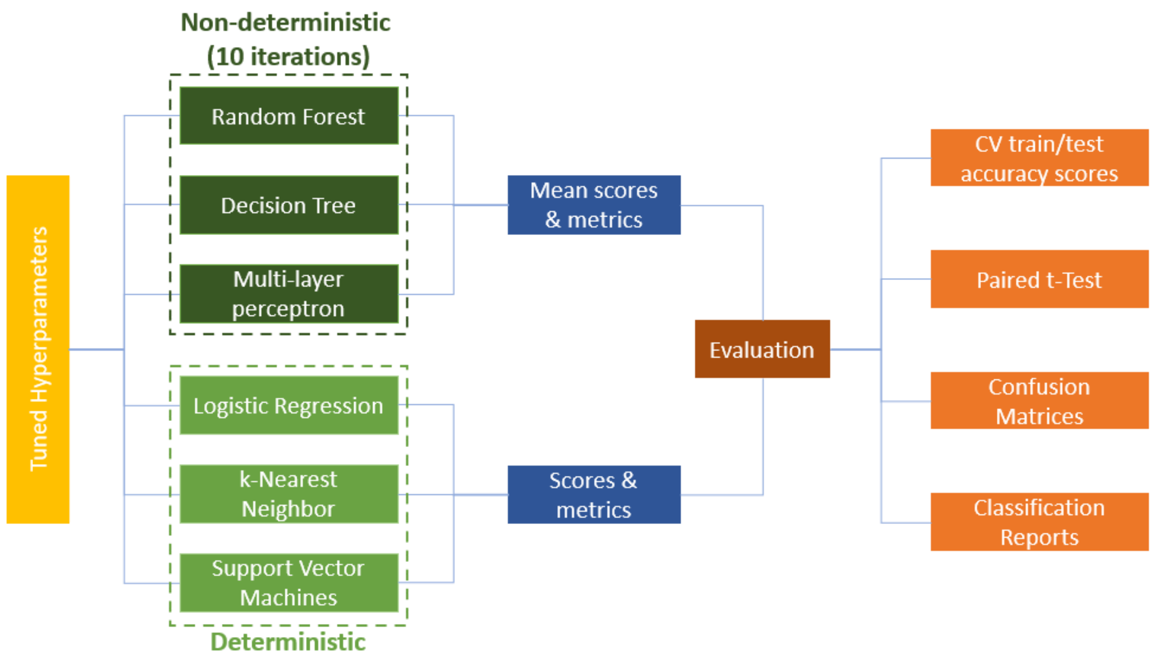


Figure 3-9 Evaluation Framework

Based on this framework, the tuned Hyperparameters are parsed towards the two different categories of classifiers. The first category of classifiers would be executed ten times each and the mean scores and metrics for each algorithm would be extracted. On the other hand, the scores and metrics of the deterministic algorithms would be extracted just by executing each algorithm one time, since the result would not be different. The final step of the process was the evaluation of the following measures::

- The Cross-Validation accuracy of both the training & testing process for each algorithm would be plotted on the same graph (see Results chapter for an extensive analysis). The graph is used to give a general performance overview of all algorithms. The Confusion Matrices showing the actual and predicted number of points for each algorithm.

- The Classification Report for each case, that includes metrics calculated out of the Confusion Matrices such as Recall, Precision, and f1-score.
- A statistical comparison of the test accuracy results via paired t-tests as depicted in the template of Table 3-6 (all algorithms are tested for significance in pairs).

	Random Forest	k-Nearest Neighbor	Logistic Regression	Support Vector Machines	Decision Tree
k-Nearest Neighbor	**RF** vs **kNN**				
Logistic Regression	**RF** vs **LR**	**kNN** vs **LR**			
Support Vector Machines	**RF** vs **SVM**	**kNN** vs **SVM**	**LR** vs **SVM**		
Decision Tree	**RF** vs **DT**	**kNN** vs **DT**	**LR** vs **DT**	**SVM** vs **DT**	
Multi-layer Perceptron	**RF** vs **MLP**	**kNN** vs **MLP**	**LR** vs **MLP**	**SVM** vs **MLP**	**DT** vs **MLP**

Table 3-6 t-Test represented application

Each cell of Table 3-6 contains two values: a t-statistic and a p value. If t-statistic<0 then the row classifier is considered better and if the t-statistic>0 then the column classifier is considered better. The p-value is used to assess whether the result of t-statistic is random or whether the difference between two performance entries is significant (in this thesis we use a significance level of 5% i.e. $p_{baseline} = 0.05$).

CHAPTER 4

RESULTS

In this Chapter we present the key results of our experiments in the following fashion:

- Binary Classification Results
    - Confusion matrices and Classification reports for each model used (Fig.4-1 to Fig 4-12).
    - Cross validation train & test accuracy results.
    - T-test performed amongst models.
    - Overview of solution
- Binary Classification Results (threshold = 0.5)
    - Confusion matrices and Classification reports for each model used (Fig.4-14 to 4-25).
    - Cross validation train & test accuracy results.
    - T-test performed amongst models.
    - Overview of solution
- Binary Classification Results (threshold = 1.0)
    - Confusion matrices and Classification reports for each model used (Fig.4-27 to 4-38).
    - Cross validation train & test accuracy results.
    - T-test performed amongst models.
    - Overview of solution
- Multiclass Classification Results
    - Confusion matrices and Classification reports for each model used (Fig.4-40 to 4-51).
    - Cross validation train & test accuracy results.
    - T-test performed amongst models.
    - Overview of solution
- Overview of solutions

Binary Classification Results

In the subsection bellow, Figures 4-1 to 4-12 represent the confusion matrices and classification reports for each classification model used (Binary classification threshold = 0.0)

Random Forest

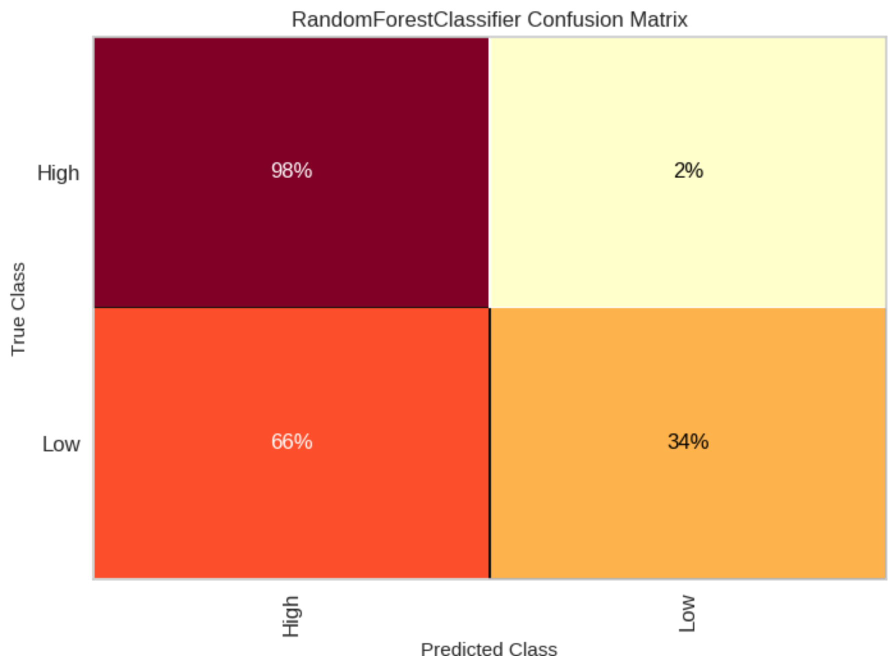


Figure 4-1 RF confusion matrix

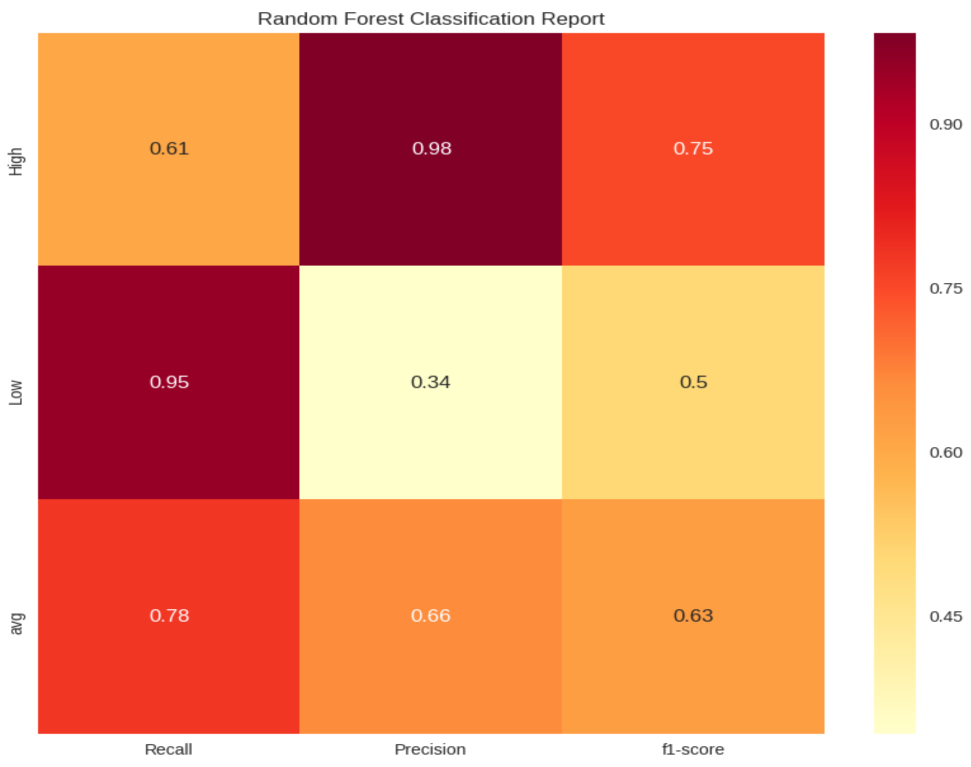


Figure 4-2 RF classification report

K-Nearest Neighbor

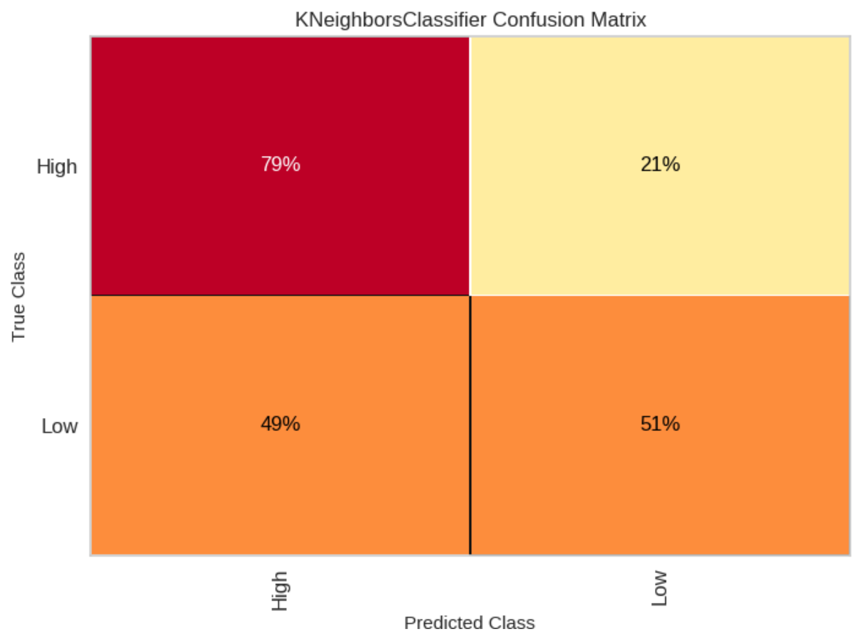


Figure 4-3 kNN confusion matrix

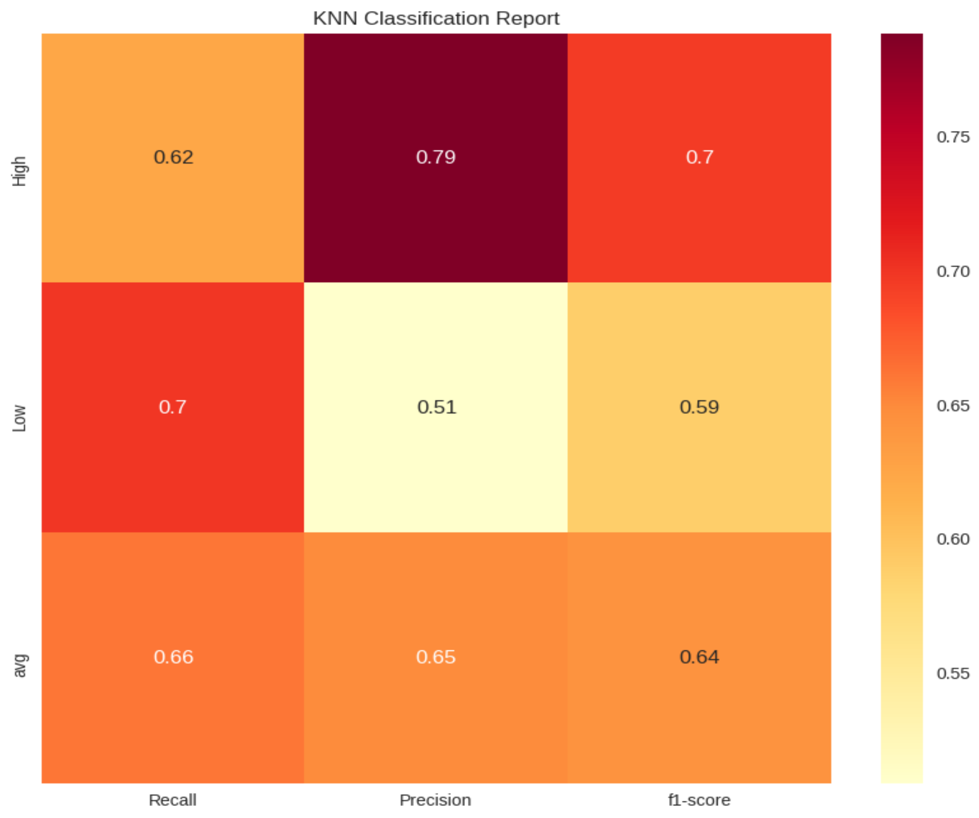


Figure 4-4 kNN confusion matrix

Logistic Regression

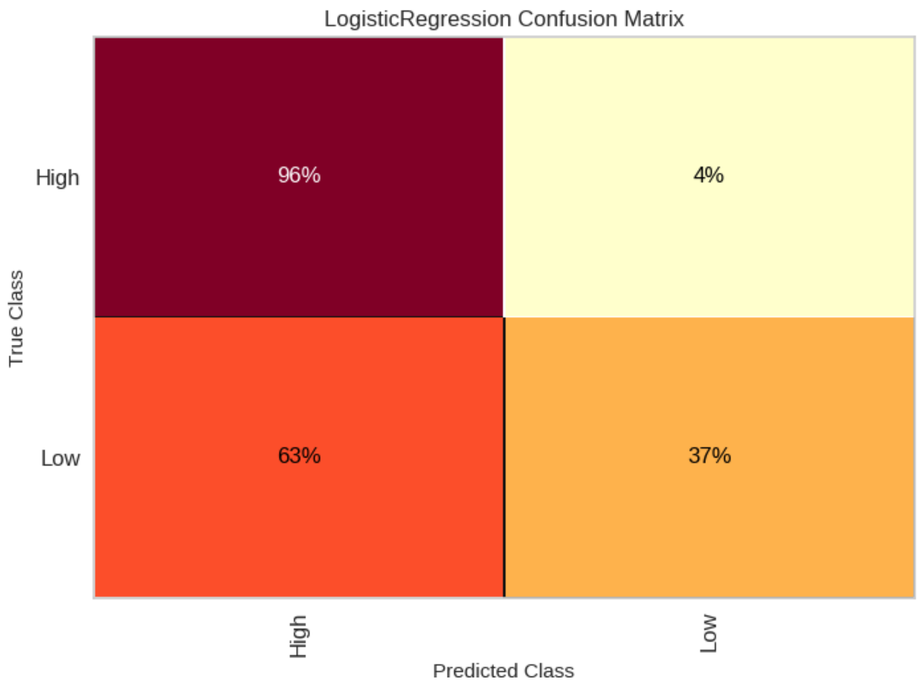


Figure 4-5 LR confusion matrix

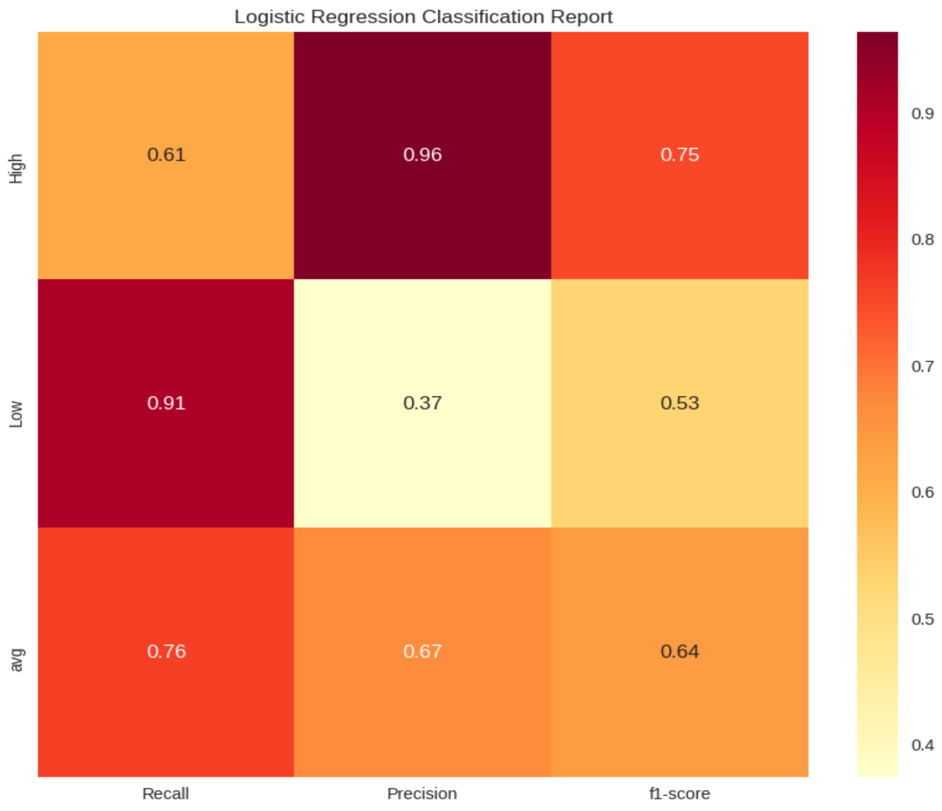


Figure 4-6 LR classification report

Support Vector Machine

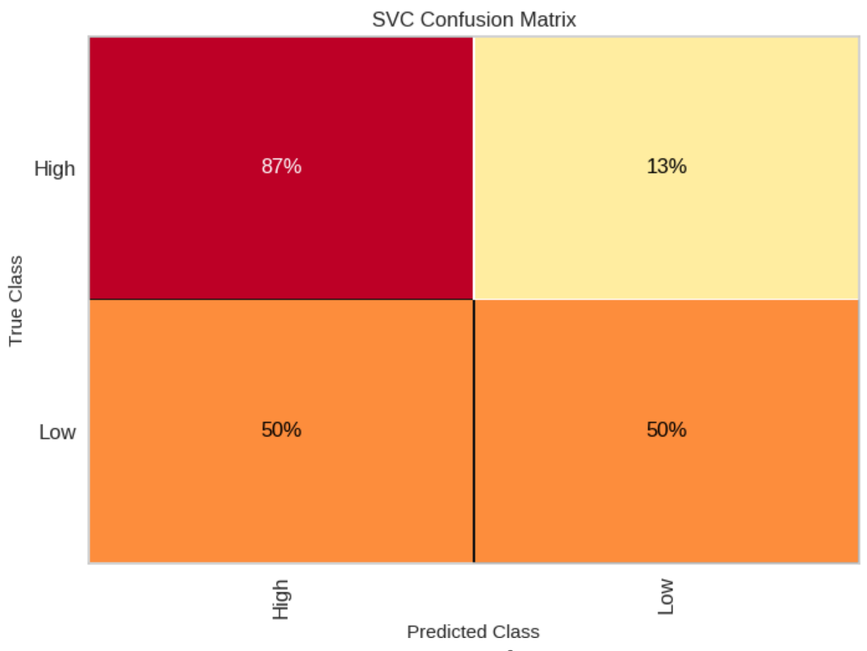


Figure 4-7 SVM confusion matrix

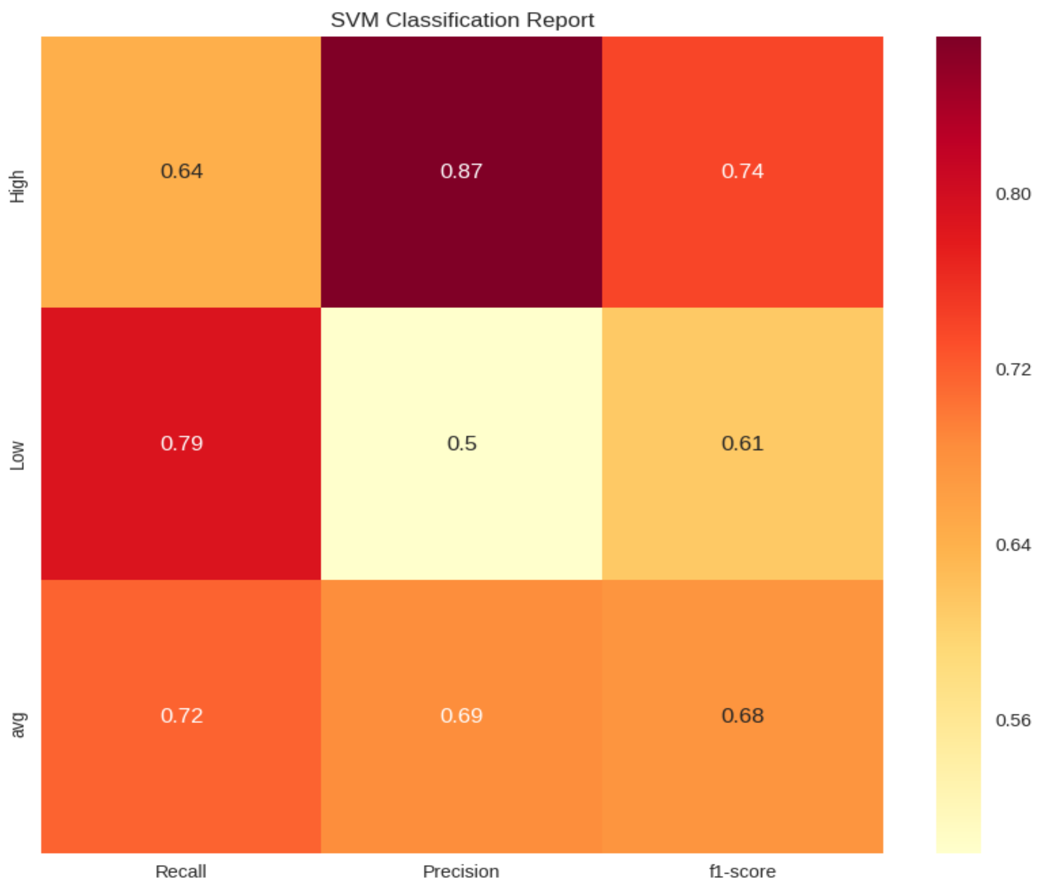


Figure 4-8 SVM classification report

Decision Tree

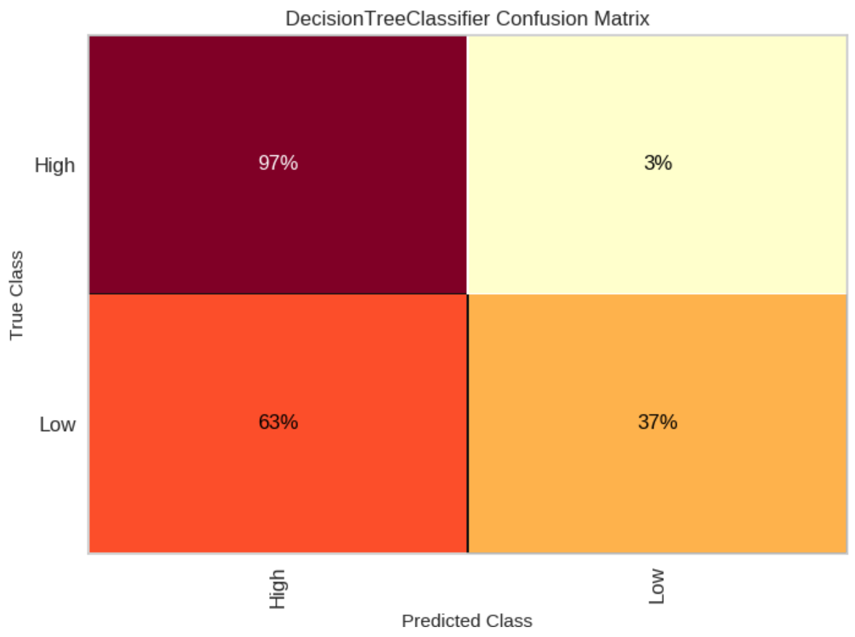


Figure 4-9 DT confusion matrix

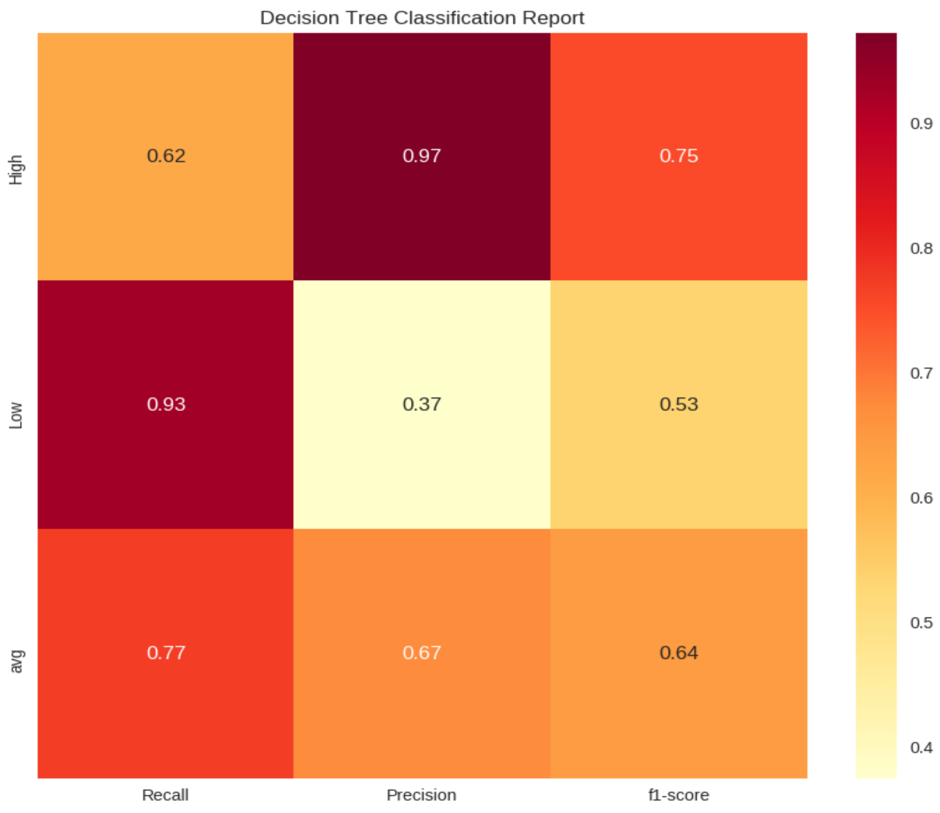


Figure 4-10 DT classification report

Multi-layer Perceptron (MLP)

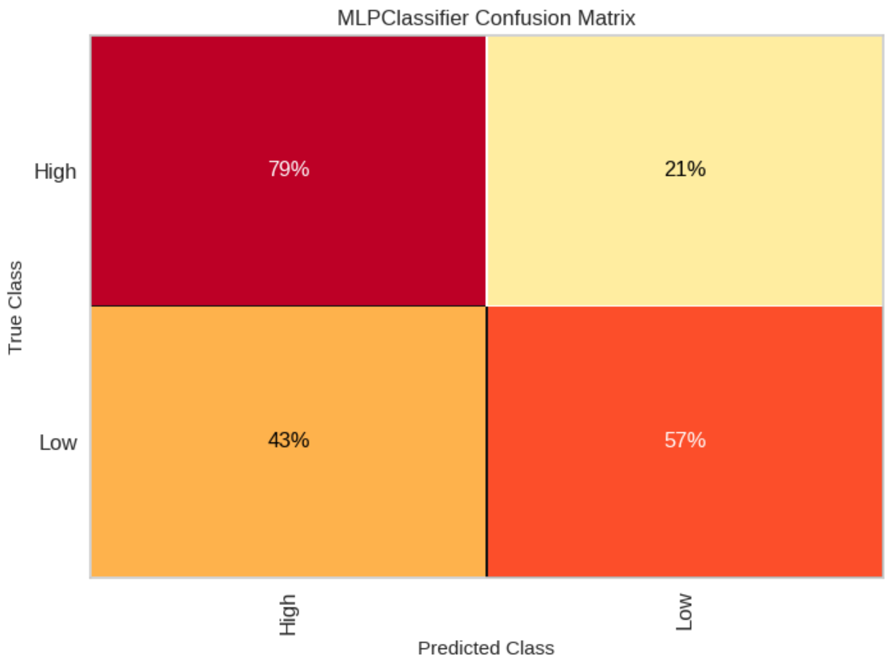


Figure 4-11 MLP confusion matrix

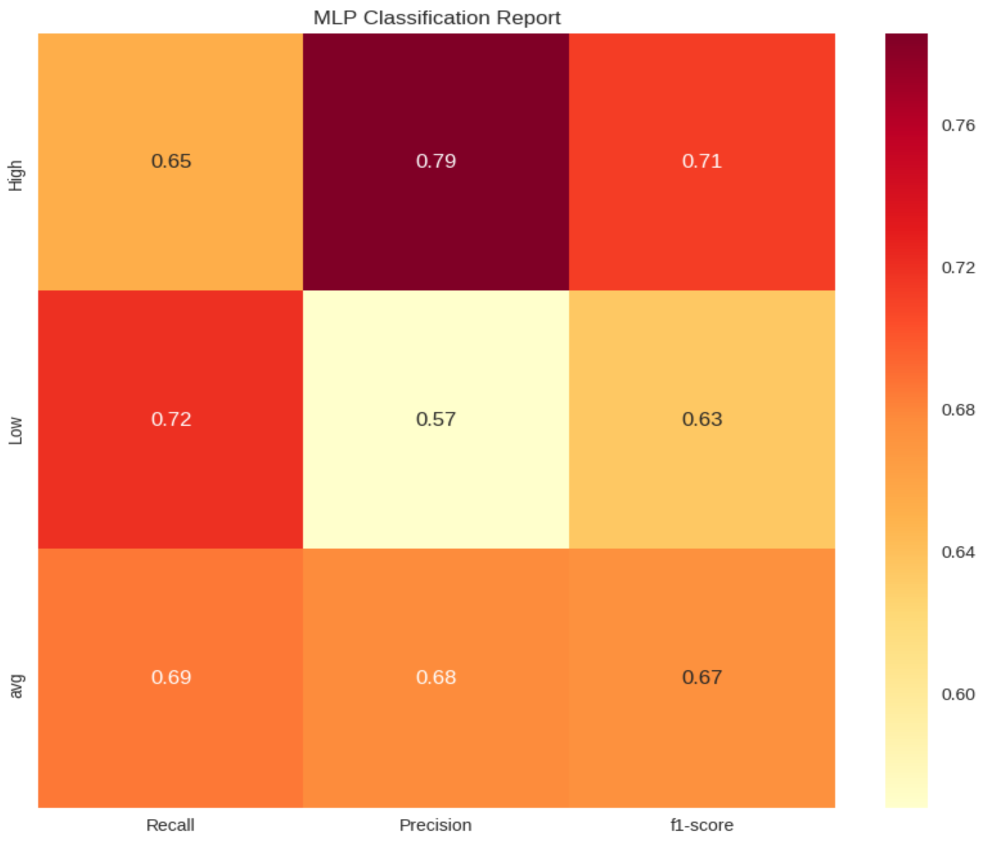


Figure 4-12 MLP classification report

Accuracy Scores

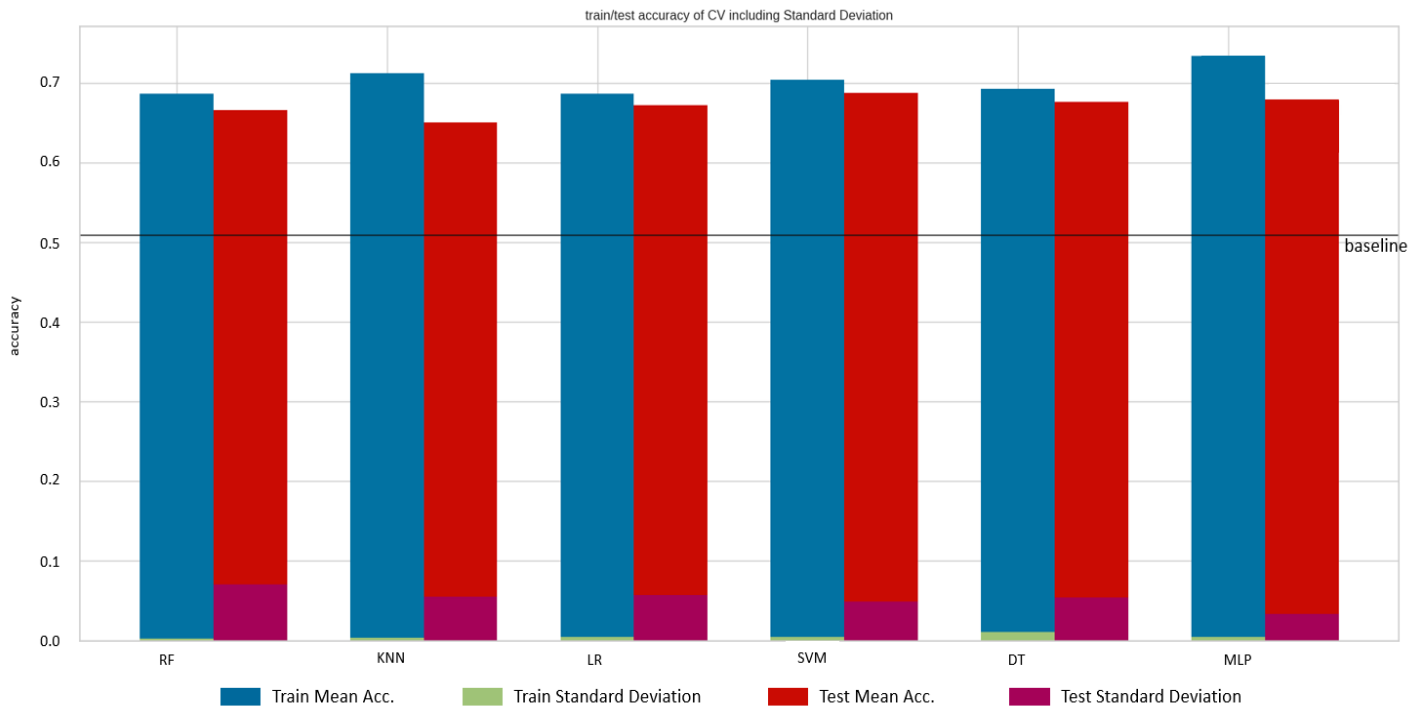


Figure 4-13 Train & Test Cross-Validation Accuracy scores

T-test

	Random Forest	k-Nearest Neighbor	Logistic Regression	Support Vector Machines	Decision Tree
k-Nearest Neighbor	t = 0.9135 p = 0.3805				
Logistic Regression	t = -0.3486 p = 0.7339	t = -0.884 p = 0.3862			
Support Vector Machines	t = -1.5248 p = 0.1555	t = -1.6954 p = 0.1043	t = -0.722 p = 0.4780		
Decision Tree	t = -2.9873 p = 0.01502	t = -1.4938 p = 0.1612	t = -0.2326 p = 0.8199	t = 0.8147 p = 0.4308	
Multi-layer Perceptron	t = -9.1173 p = 0.00055	t = -1.7181 p = 0.1133	t = -0.4274 p = 0.6772	t = 0.6074 p = 0.5556	t = -0.6457 p = 0.5315

Table 4-1 performed t-Test results

By observing the confusion matrices, classification reports, Figure 4-13 and Table 4-1 we come to the following conclusions for the binary classification results:

- The average accuracy of the binary classification algorithms appears to be 66.8%, with the worst predicting classifier being KNN (65%) and best SVM (68,5%).
- We accurately predict at least 14% cases above the baseline (on average), so our experiment is not considered a “coin-flip”.
- The models do not overfit/underfit according to Figure 4-1, as both the train and test accuracy do not differ significantly.
- The class “Low” is misclassified a lot during the predictions as the instance that acquired the most accurate prediction was 57% from MLP and worst 34% from RF. In addition, class “High” appears to be predicted significantly better, with the highest predictions being 98% (RF) and the lowest being 79% (both KNN and MLP).
- The t-Test (Table 4-1) shows that we are not able to profound any of the classifiers optimal compared to the others, as in most cases the p-value is large enough to declare that the result of the comparison is random.

Binary Classification with Threshold (threshold = 0.5)

In the subsection bellow, Figures 4-14 to 4-25 represent the confusion matrices and classification reports for each classification model used (Binary classification threshold = 0.5).

Random Forest

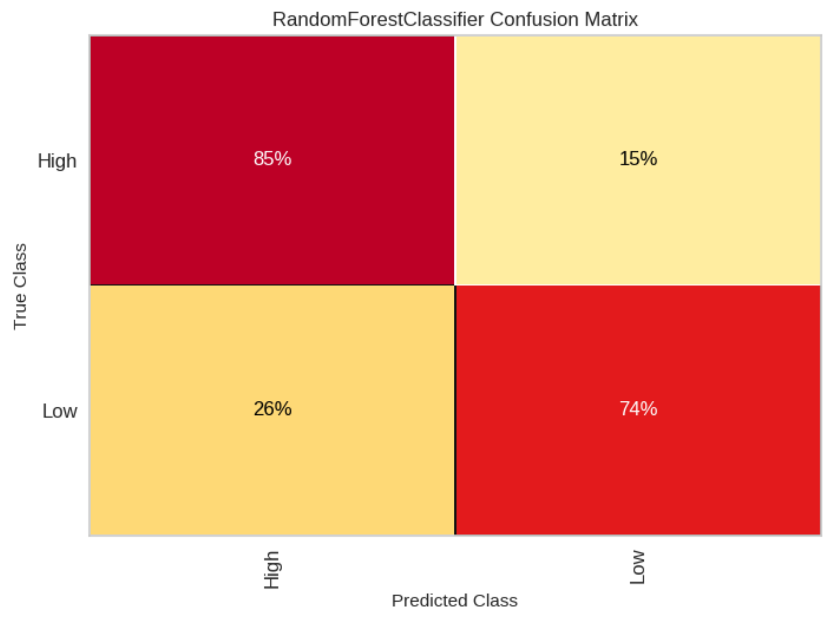


Figure 4-14 RF confusion matrix

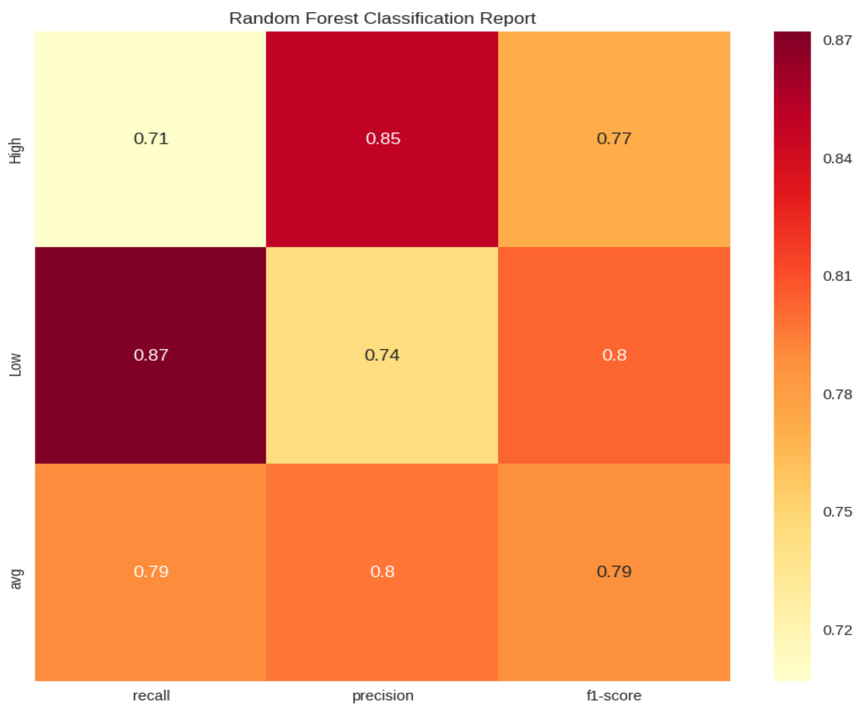


Figure 4-15 RF classification report

K-Nearest Neighbor

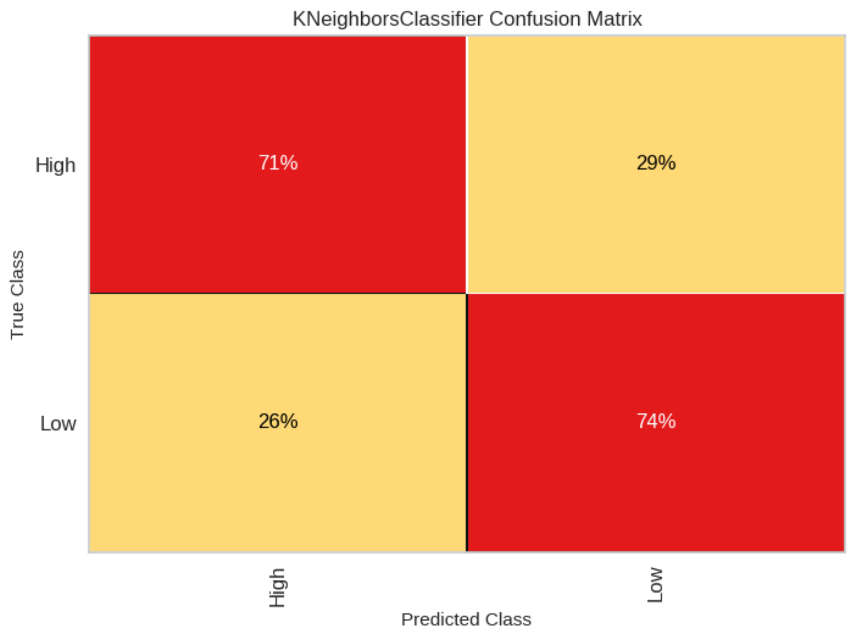


Figure 4-16 kNN confusion matrix

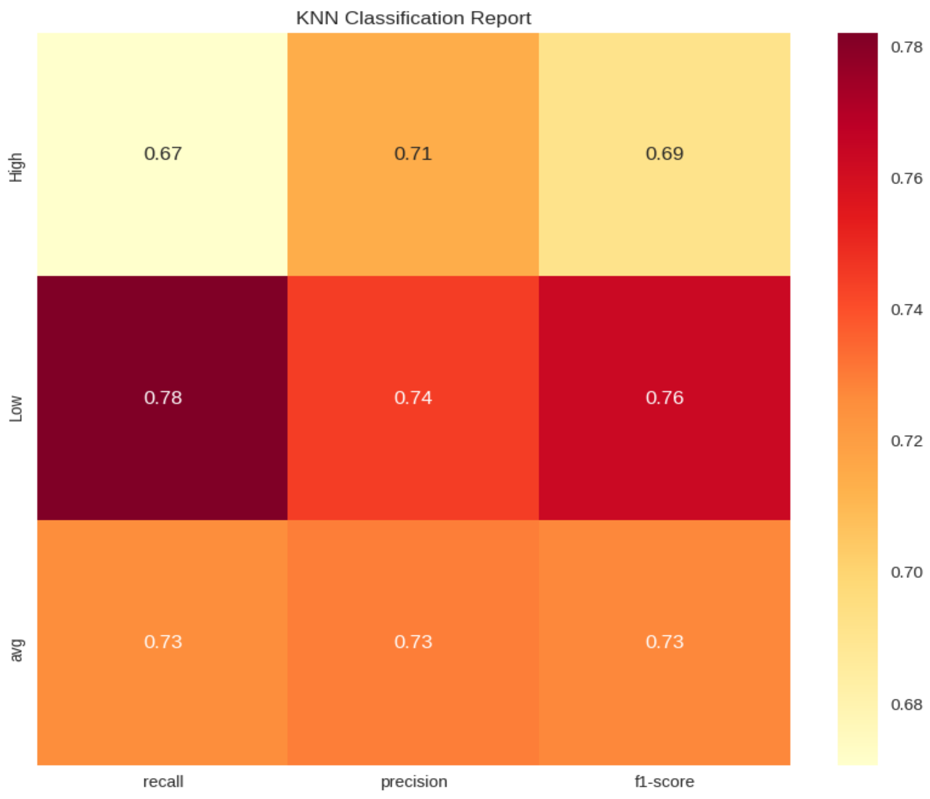


Figure 4-17 kNN classification report

Logistic Regression

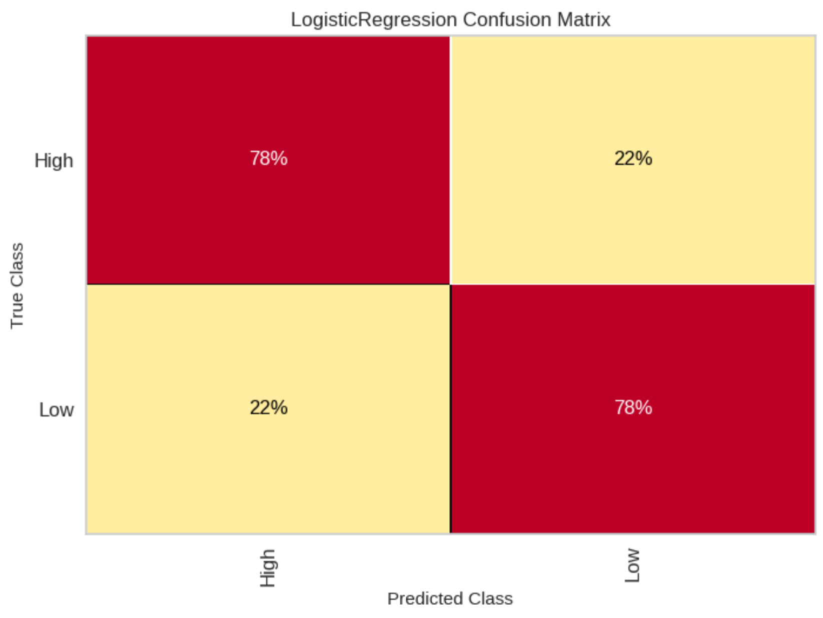


Figure 4-18 LR confusion matrix

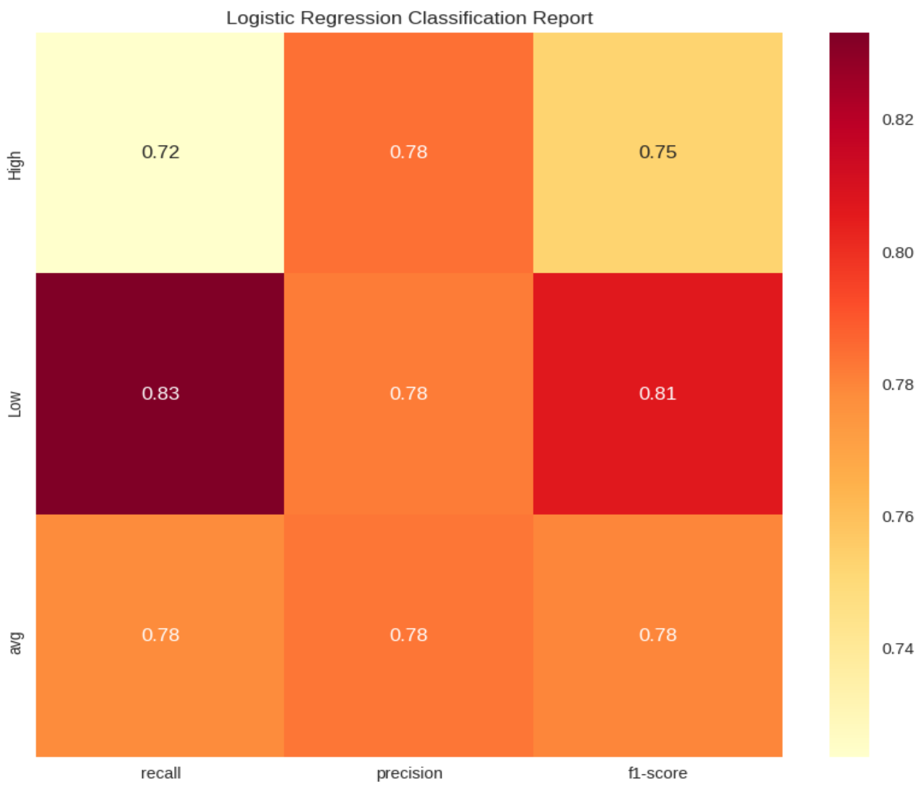


Figure 4-19 LR classification report

Support Vector Machine

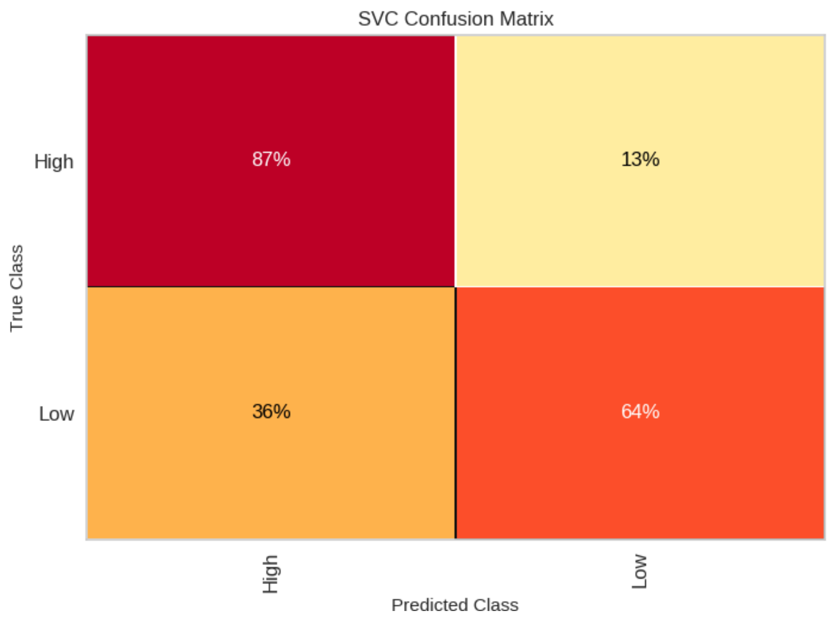


Figure 4-20 SVM confusion matrix

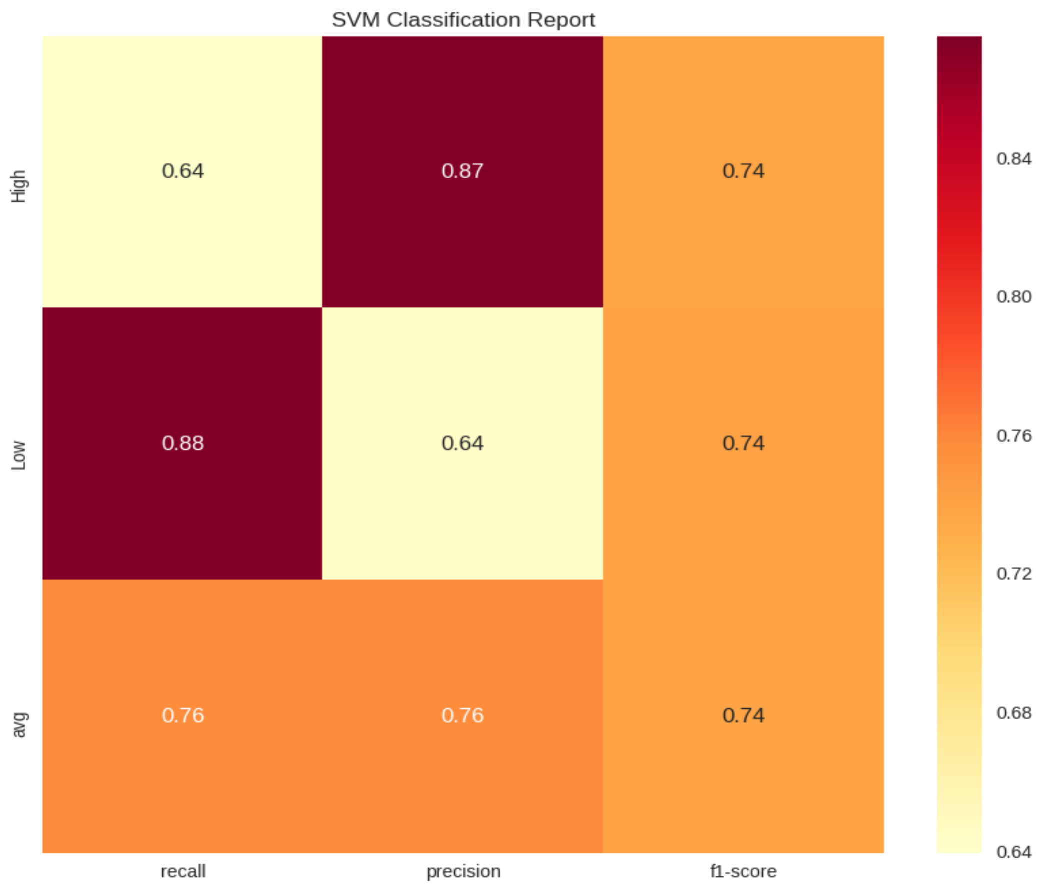


Figure 4-21 SVM classification report

Decision Tree

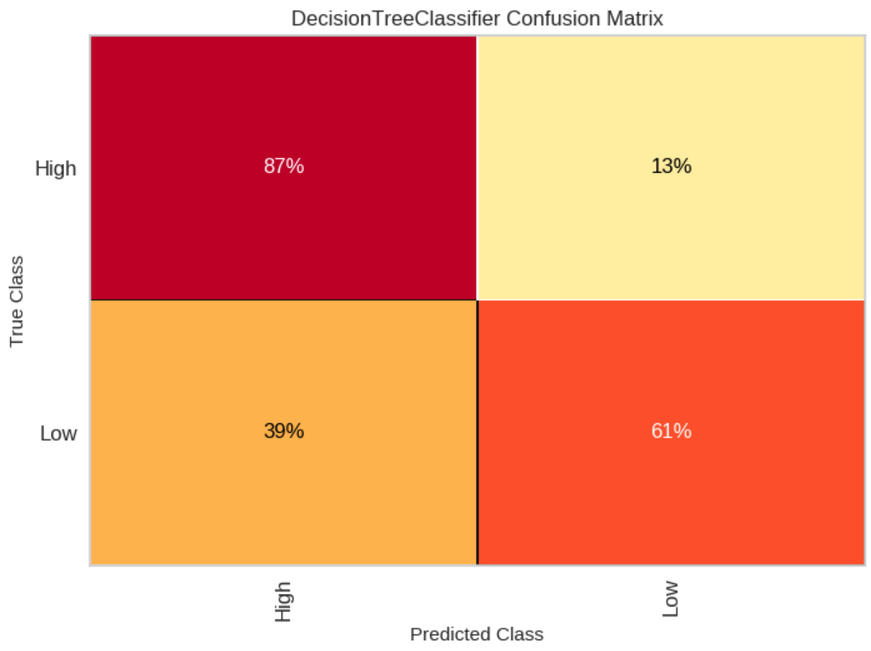


Figure 4-22 DT confusion matrix

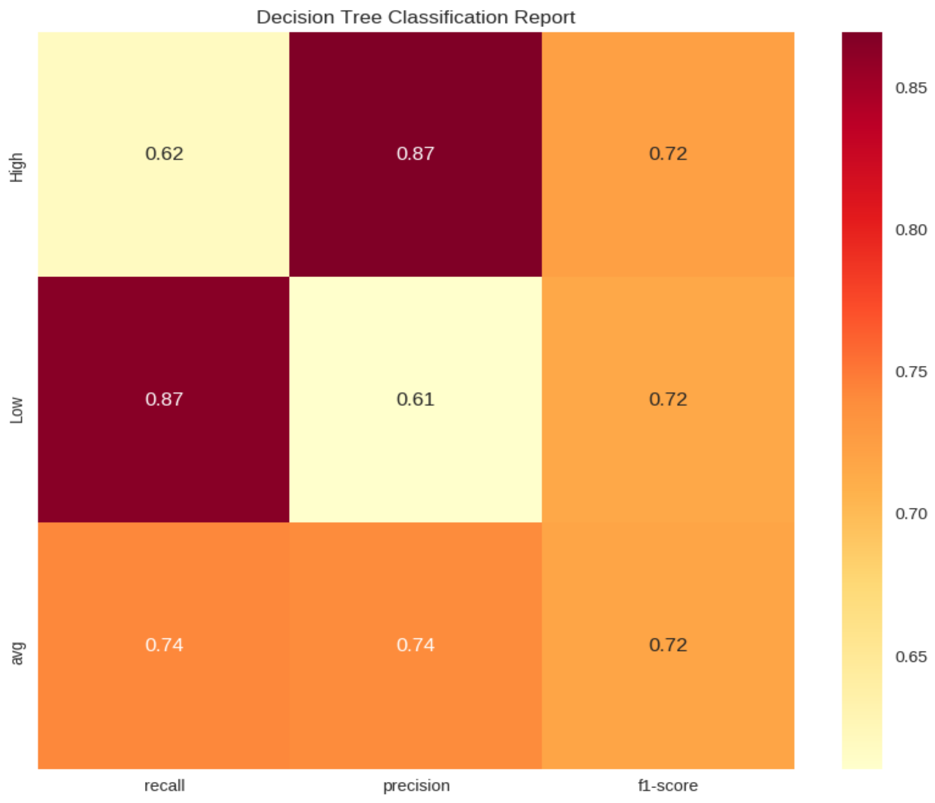


Figure 4-23 DT classification report

Multi-layer Perceptron (MLP)

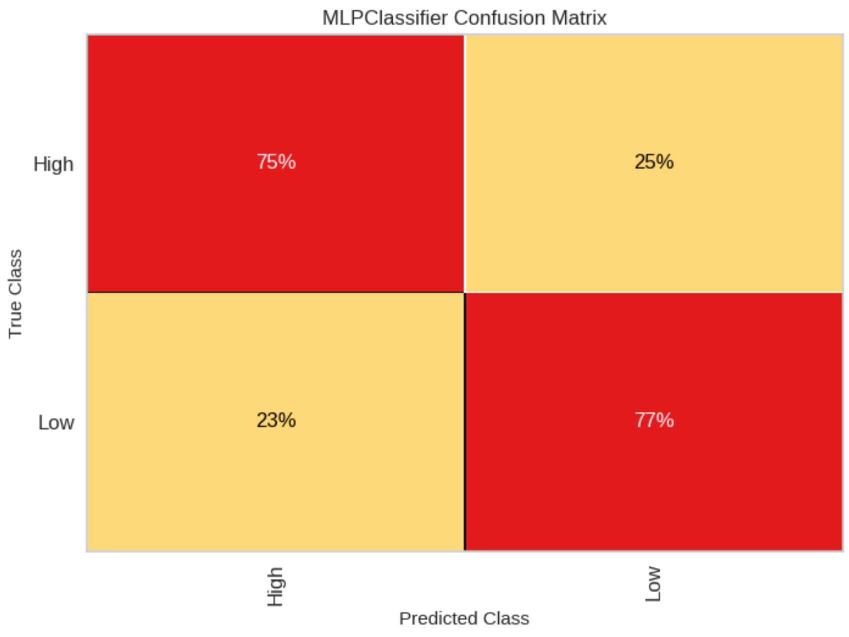


Figure 4-24 MLP confusion matrix

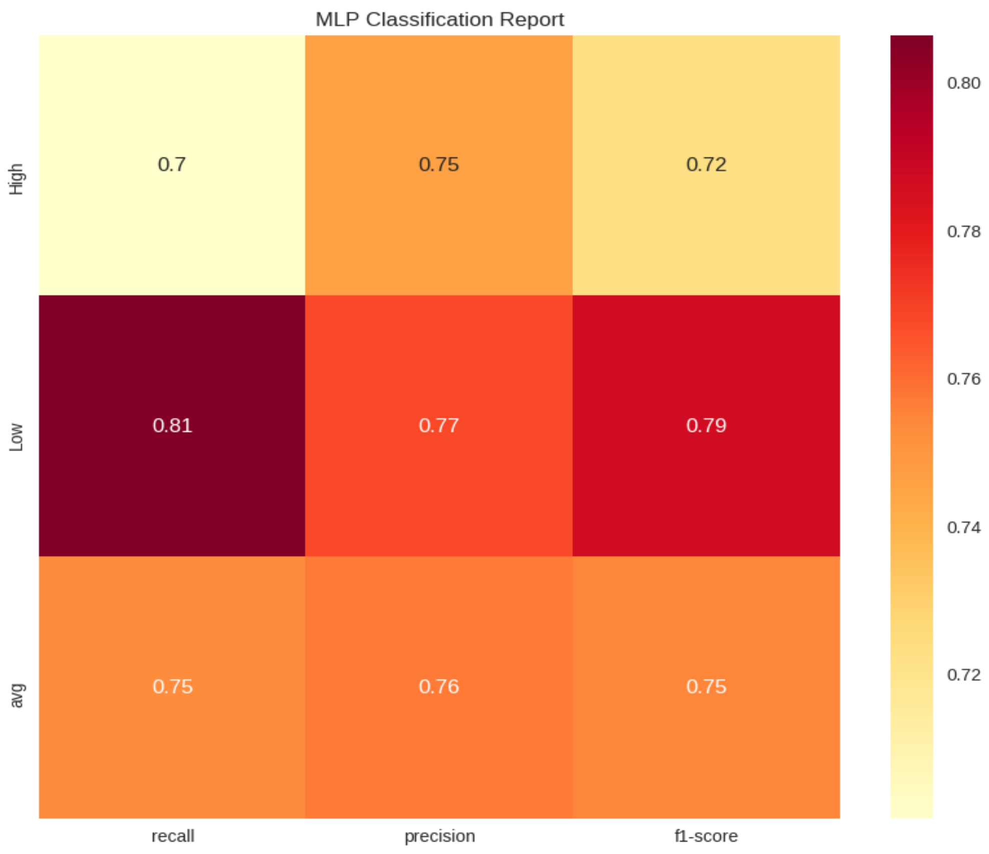


Figure 4-25 MLP classification report

Accuracy

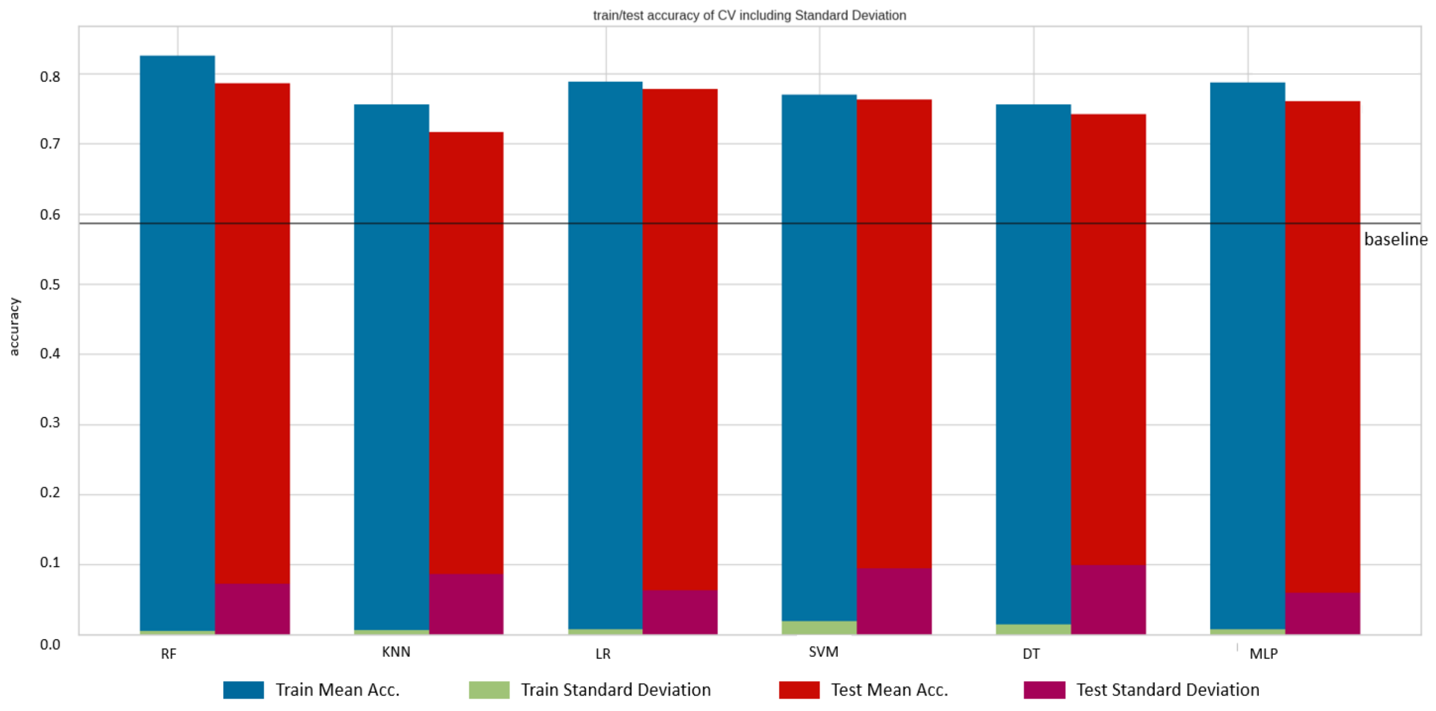


Figure 4-26 Train & Test Cross-Validation Accuracy scores (Threshold=0.5)

T-test

	Random Forest	k-Nearest Neighbor	Logistic Regression	Support Vector Machines	Decision Tree
k-Nearest Neighbor	**t = 1.6923** **p = 0.1112**				
Logistic Regression	**t = 0.2761** **p = 0.7862**	**t = -1.8911** **p = 0.0718**			
Support Vector Machines	**t = 0.5066** **p = 0.6260**	**t = -0.9066** **p = 0.3789**	**t = 0.3704** **p = 0.7162**		
Decision Tree	**t = 12.9703** **p = 7.42e-05**	**t = -0.8857** **p = 0.3862**	**t = 1.683** **p = 0.1079**	**t = 0.6257** **p = 0.5423**	
Multi-layer Perceptron	**t = 15.1455** **p = 9.81e-05**	**t = -1.0611** **p = 0.3053**	**t = 0.5781** **p = 0.5717**	**t = -0.0421** **p = 0.967**	**t = -1.0621** **p = 0.3049**

Table 4-2 performed t-Test results

By observing the confusion matrices, classification reports, Figure 4-26 and Table 4-2 we come to the following conclusions for the binary classification results:

- The average accuracy of the binary classification with threshold = 0.5 appears to be 75.9%, with the worst predicting classifier being KNN (72.5%) and best RF (79.5%).
- We accurately predict at least 16% cases above the baseline (on average) in this solution too, so our experiment is not considered a “coin-flip”.
- The models do not overfit/underfit according to Figure 4-2, as both the train and test accuracy do not differ significantly.
- The classifiers seem to predict both “High” and “Low” classes correctly with almost the same rate. The accuracy for class “High” ranged between 71% (KNN) and 87% (DT), as for class “Low” it ranged between 61% (DT) and 78% (LR and MLP).
- The t-Test (Table 4-2) shows that we are not able to profound any of the classifiers optimal compared to the others, as in most cases the p-value is large enough to declare that the result of the comparison is random.

Binary Classification with Threshold (threshold = 1.0)

In the subsection bellow, Figures 4-27 to 4-38 represent the confusion matrices and classification reports for each classification model used (Binary classification threshold = 1.0).

Random Forest

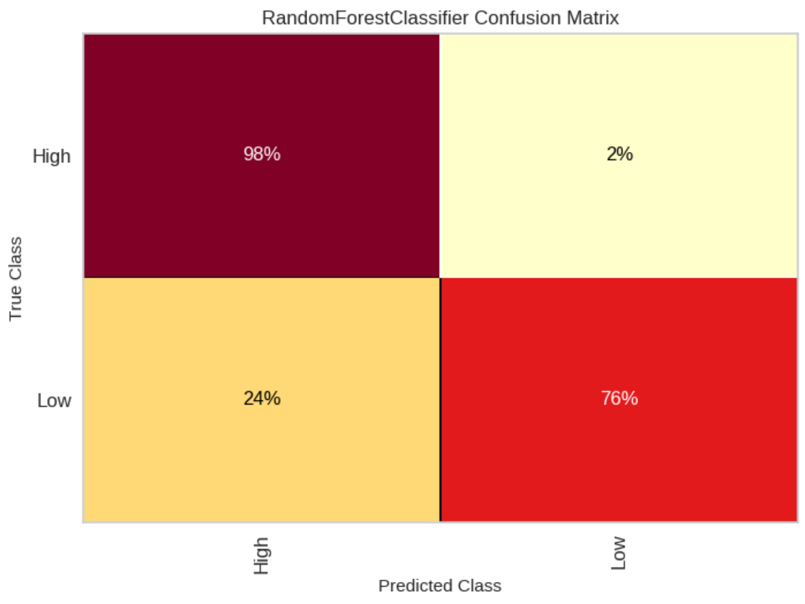


Figure 4-27 RF confusion matrix

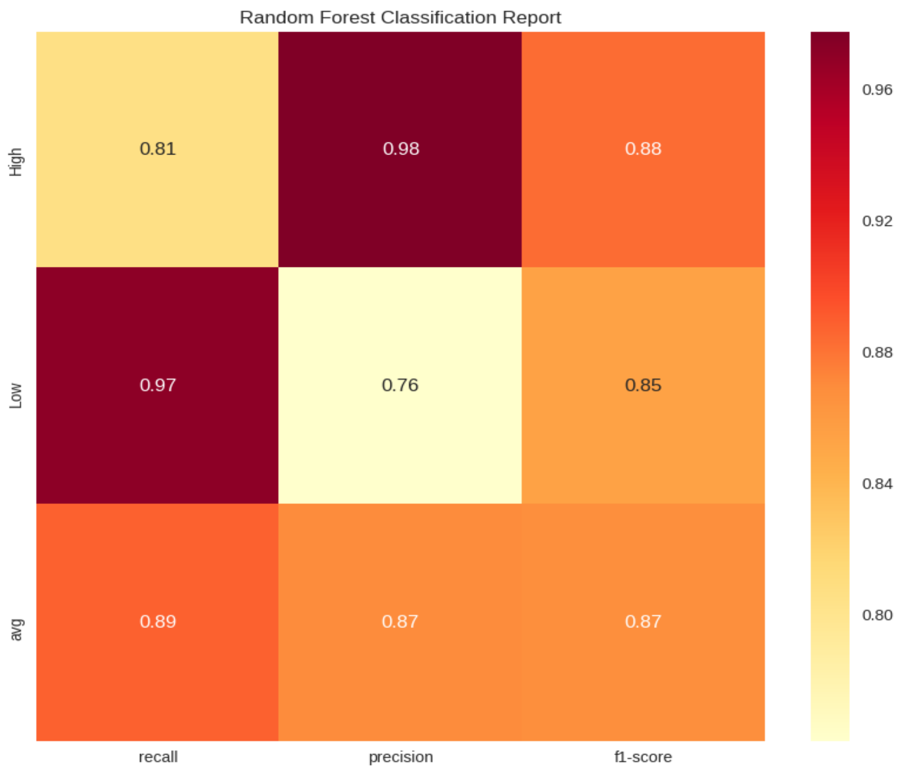


Figure 4-28 RF classification report

K-Nearest Neighbor

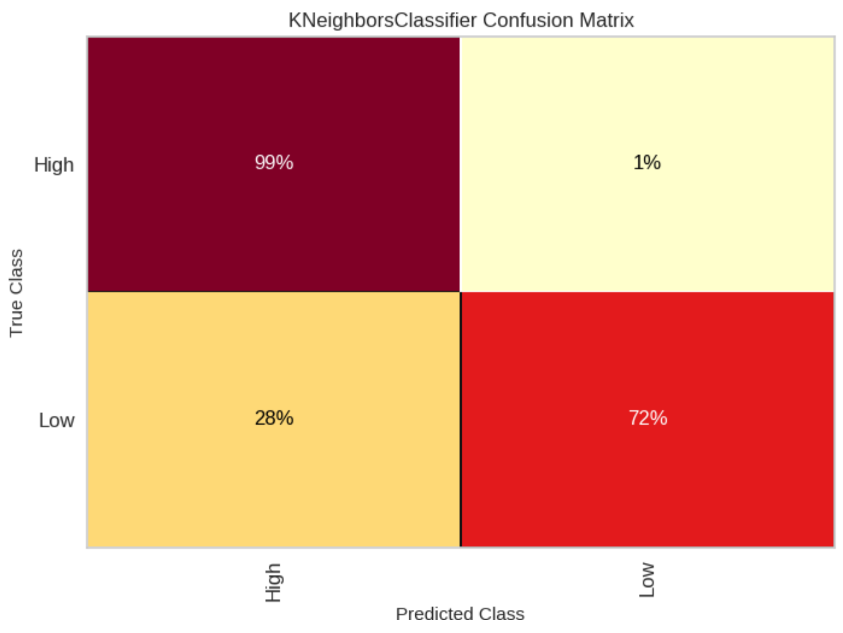


Figure 4-29 kNN confusion matrix

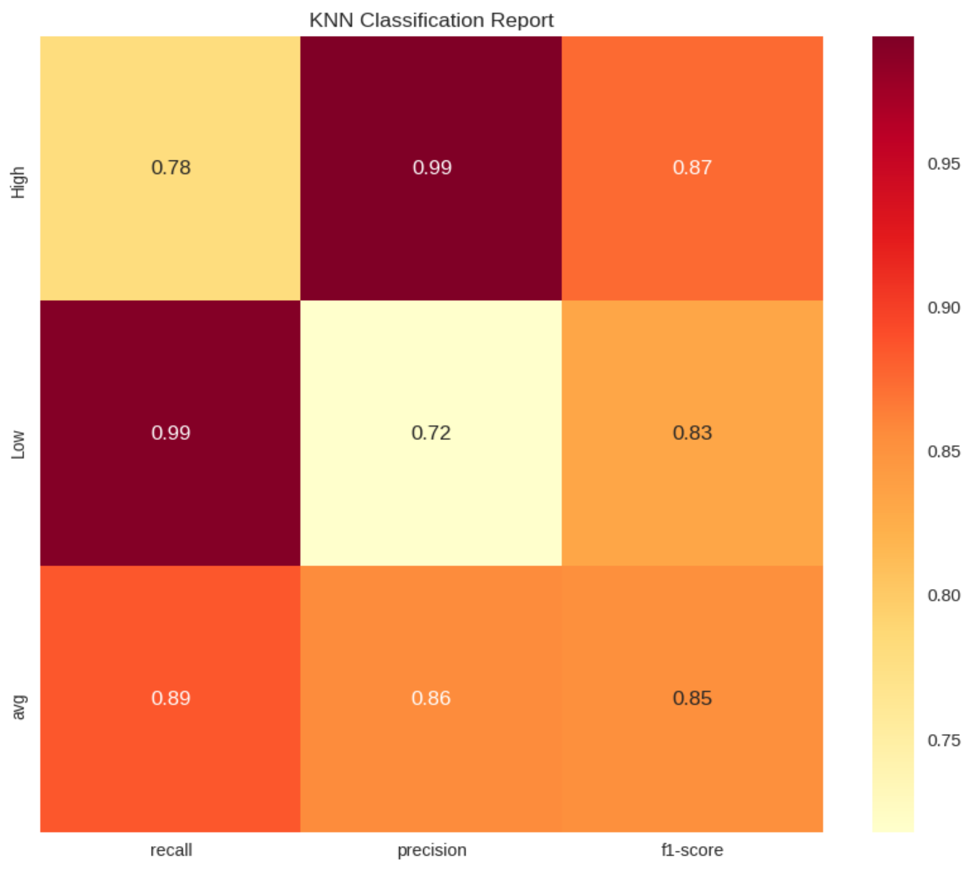


Figure 4-30 kNN classification report

Logistic Regression

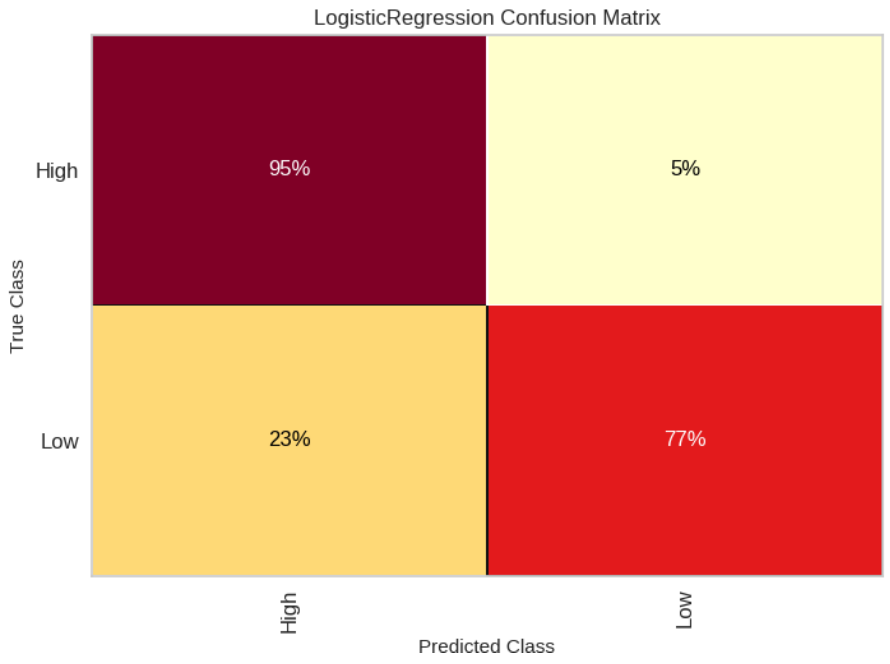


Figure 4-31 LR confusion matrix

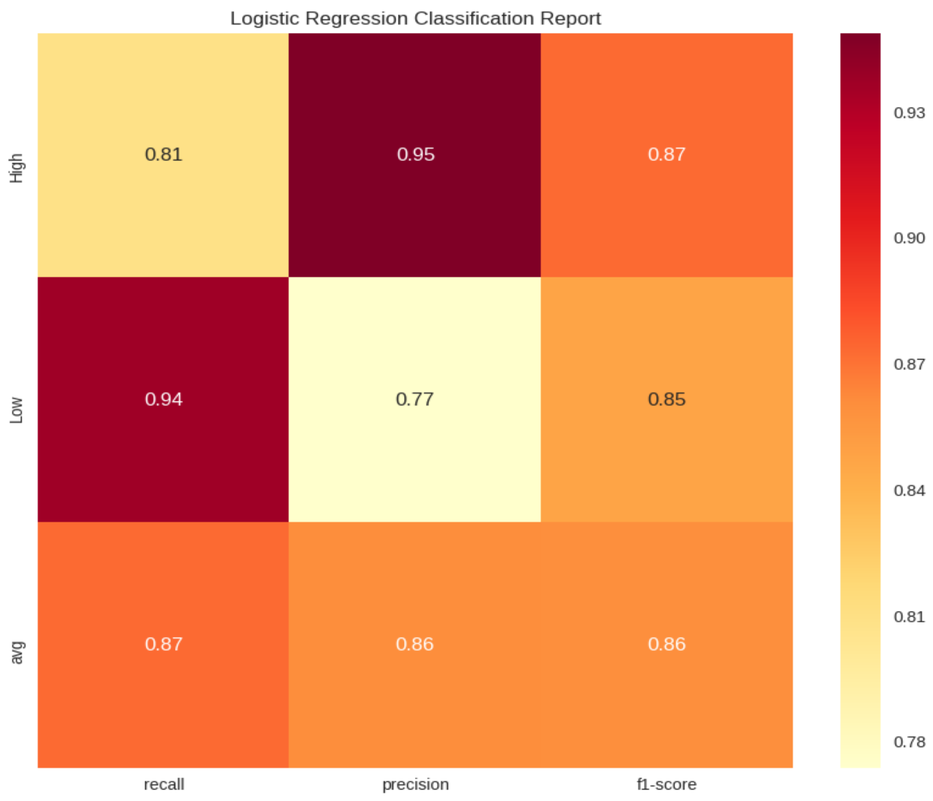


Figure 4-32 LR classification report

Support Vector Machine

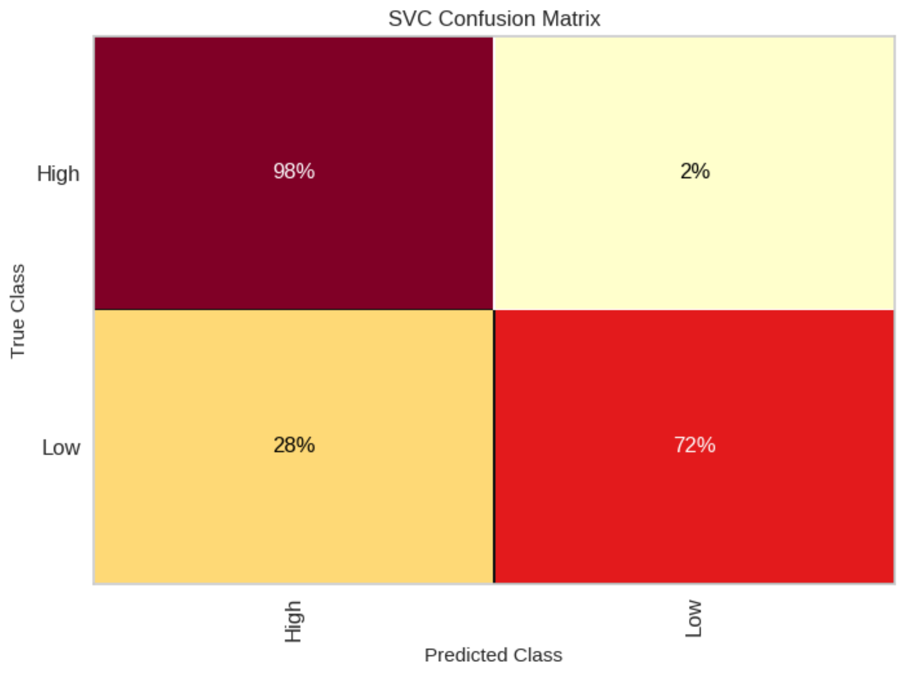


Figure 4-33 SVM confusion matrix

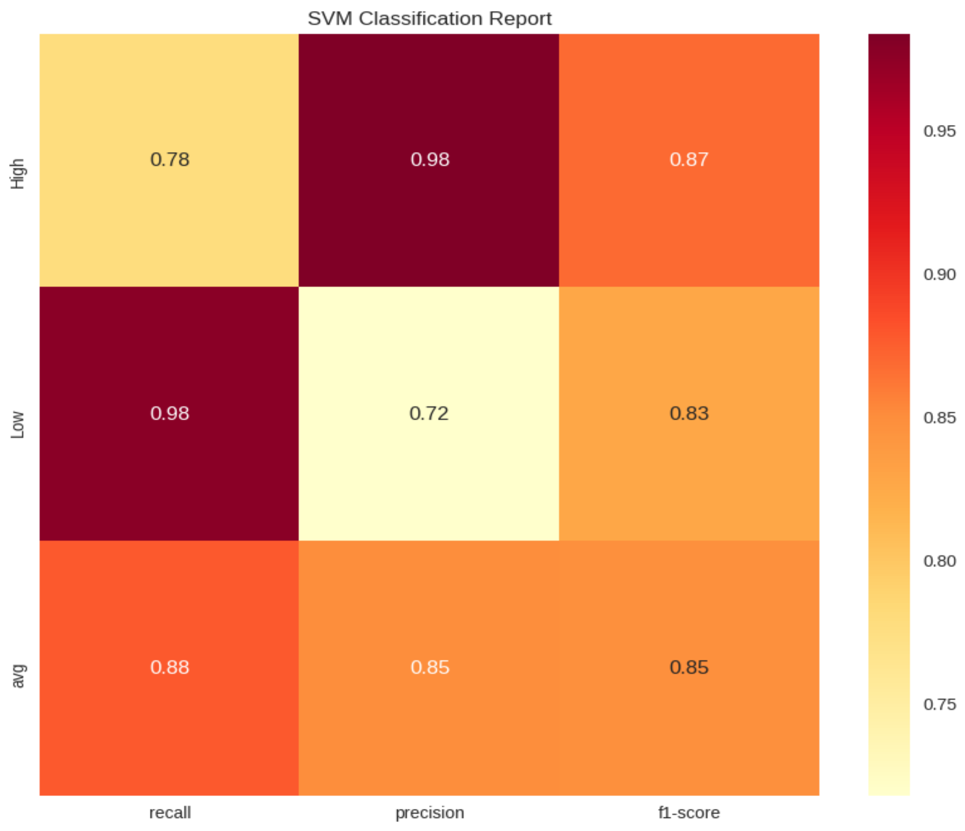


Figure 4-34 SVM classification report

Decision Tree

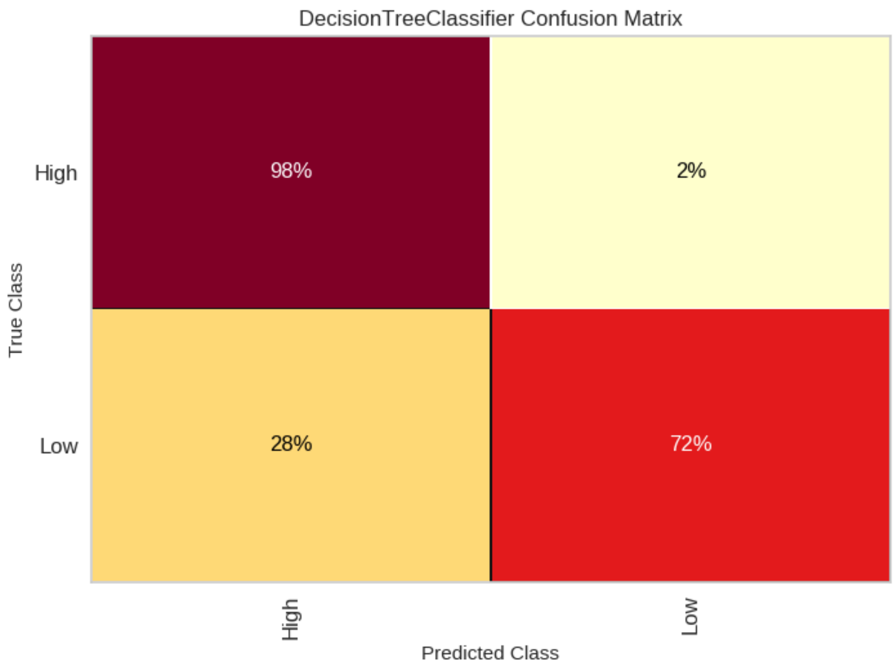


Figure 4-35 DT confusion matrix

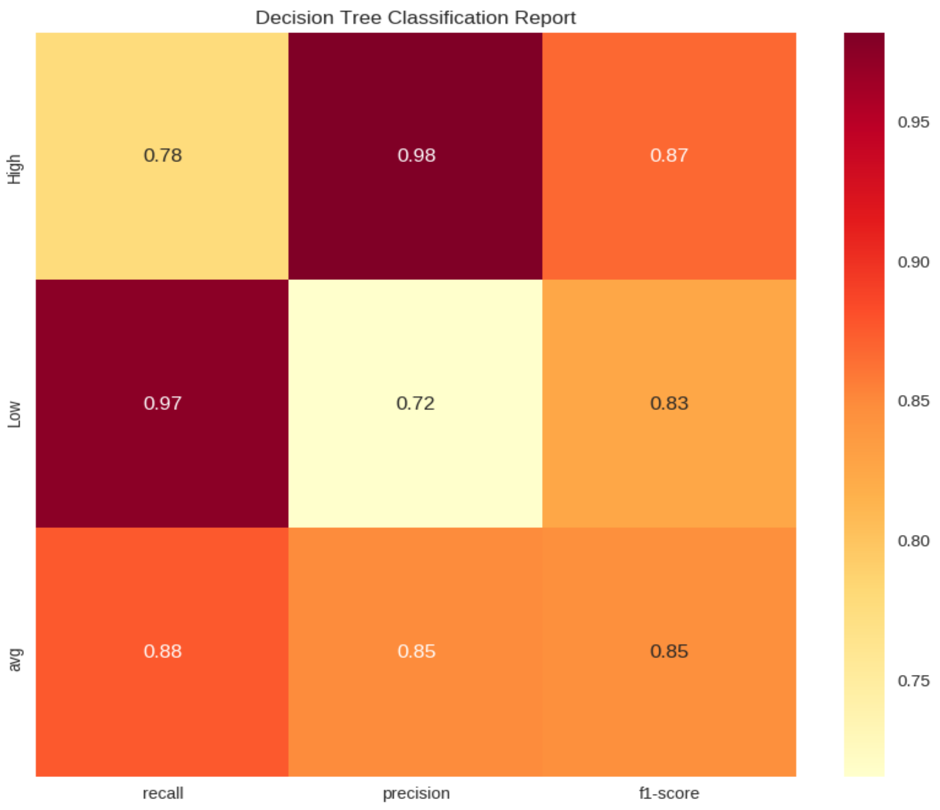


Figure 4-36 DT classification report

Multi-layer Perceptron (MLP)

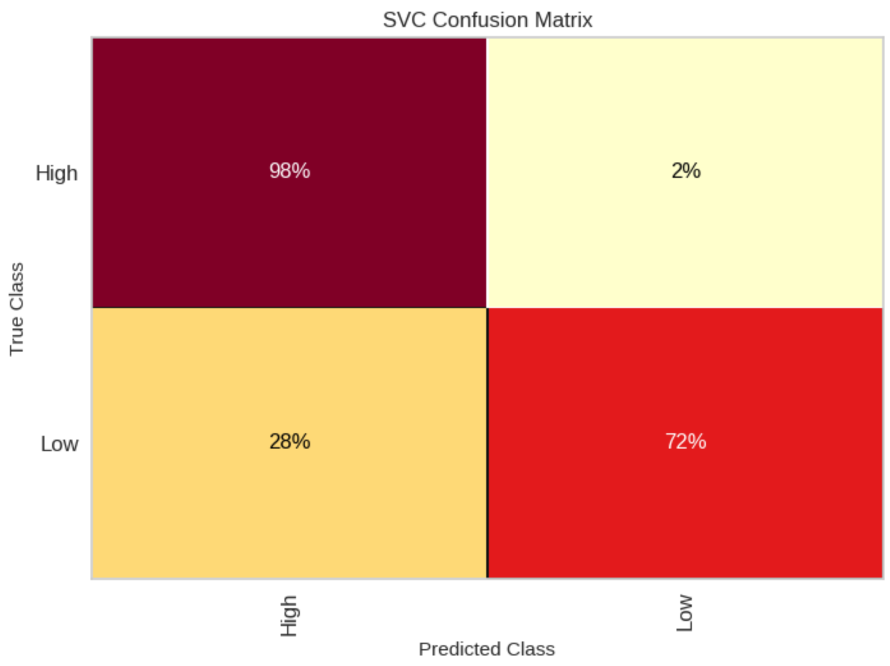


Figure 4-37 MLP confusion matrix

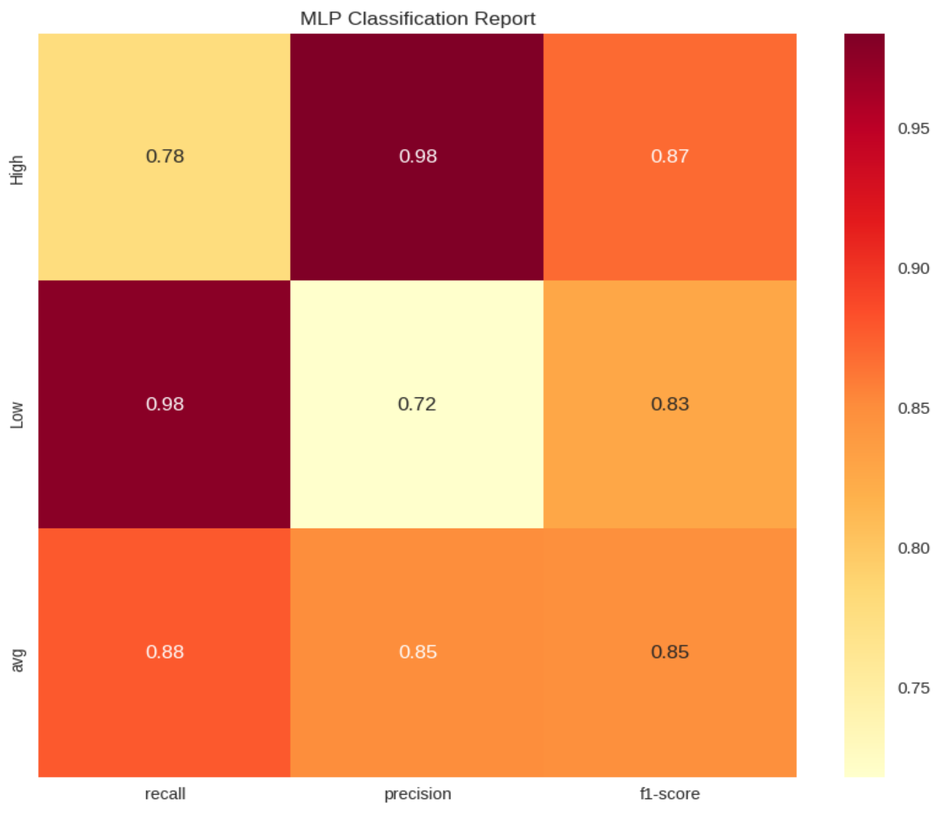


Figure 4-38 MLP classification report

Accuracy

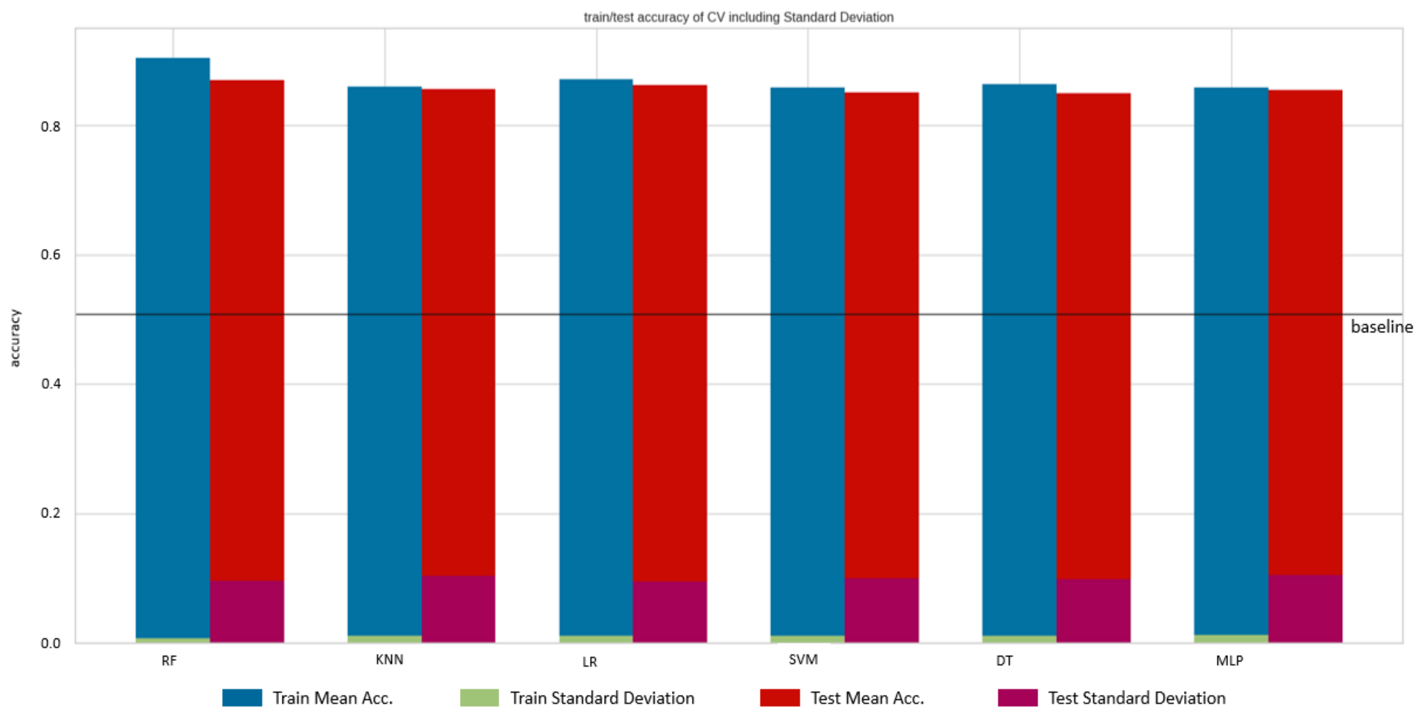


Figure 4-39 Train & Test Cross-Validation Accuracy scores (Threshold=1.0)

T-test

	Random Forest	k-Nearest Neighbor	Logistic Regression	Support Vector Machines	Decision Tree
k-Nearest Neighbor	t = 0.2701 p = 0.7912				
Logistic Regression	t = 0.1819 p = 0.8584	t = -0.1105 p = 0.9131			
Support Vector Machines	t = 0.3864 p = 0.7053	t = 0.1076 p = 0.9154	t = 0.2249 p = 0.8245		
Decision Tree	t = 9.4828 p = 3e-07	t = 0.2094 p = 0.8364	t = 0.3928 p = 0.6990	t = 0.0618 p = 0.9513	
Multi-layer Perceptron	t = 19.4002 p = 4.51e-07	t = 0.0415 p = 0.9674	t = 0.1591 p = 0.8759	t = -0.0644 p = 0.9496	t = -0.2605 p = 0.7985

Table 4-3 performed t-Test results.

By observing the confusion matrices, classification reports, Figure 4-39 and Table 4-3 we come to the following conclusions for the binary classification results:

- The average accuracy of the binary classification with threshold = 1.0 appears to be 85.6%, with the worst predicting classifier being SVM, DT and MLP (85%) and best RF (87%).
- We accurately predict at least 35% cases above the baseline (on average), so our experiment is not considered random, presenting the highest gain so far.
- The models do not overfit/underfit according to Figure 4-3, as both the train and test accuracy do not differ significantly.
- The classifiers seem to predict both “High” and “Low” classes satisfactorily, although class “High” nearly gets absolute predictions in terms of accuracy. The accuracy for class “High” ranged between 95% (LR) and 99% (KNN), as for class “Low” it ranged between 72% (KNN, SVM, DT and MLP) and 77% (LR).
- The t-Test (Table 4-3) shows that we are not able to profound any of the classifiers optimal compared to the others, as in most cases the p-value is large enough to declare that the result of the comparison is random.

Multi-class Classification (Three classes)

In the subsection bellow, Figures 4-40 to 4-51 represent the confusion matrices and classification reports for each classification model used (Multiclass classification).

Random Forest

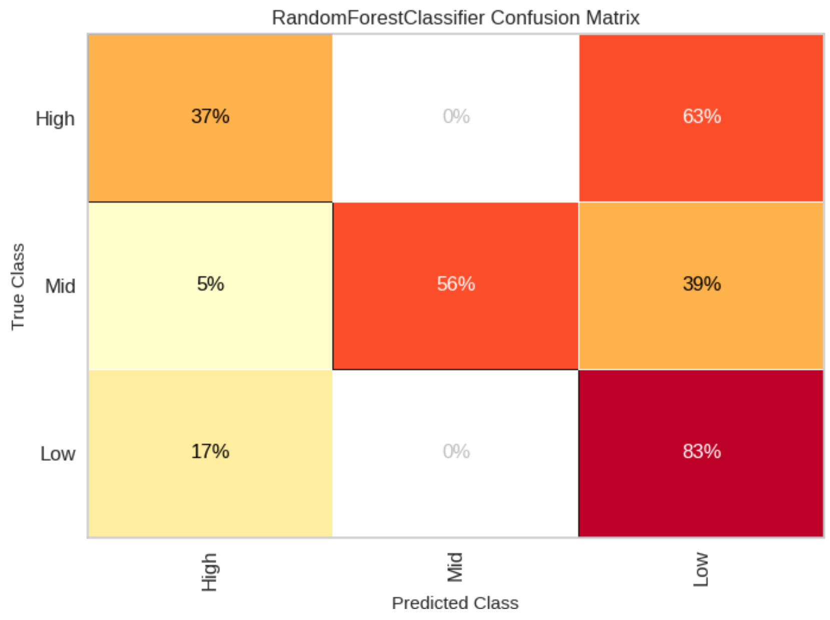


Figure 4-40 RF confusion matrix

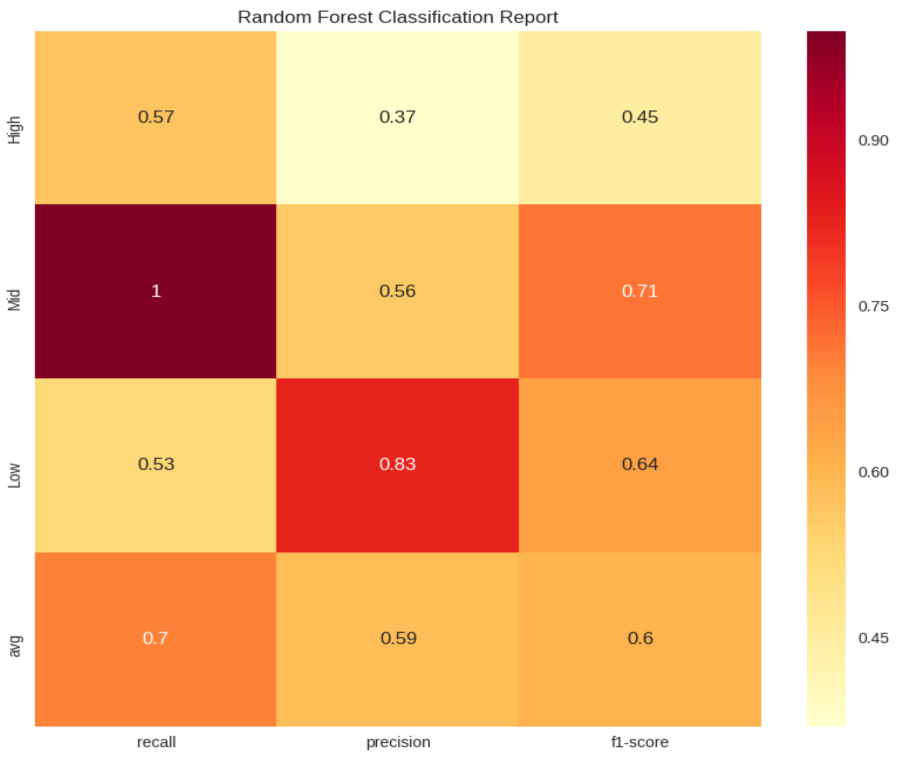


Figure 4-41 RF classification report

K-Nearest Neighbor

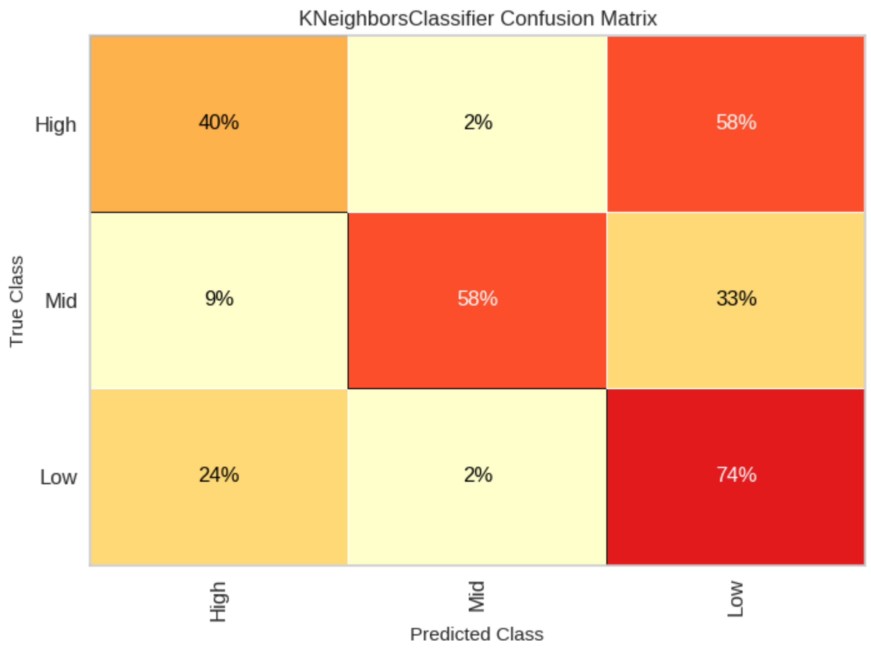


Figure 4-42 kNN confusion matrix

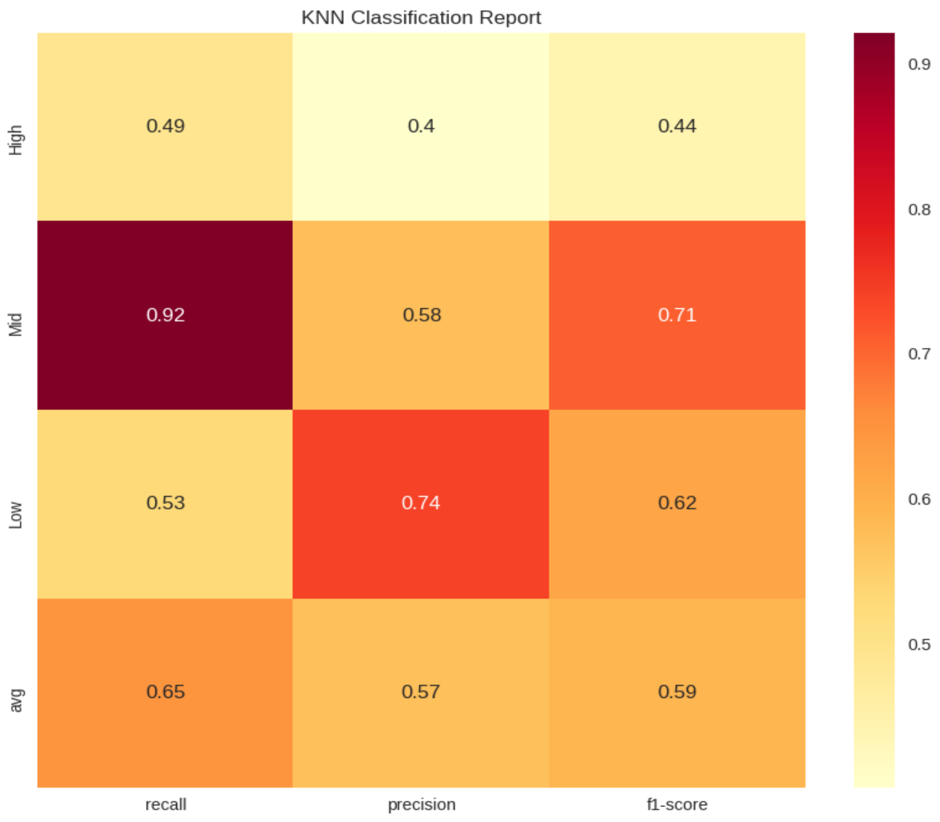


Figure 4-43 kNN classification report

Logistic Regression

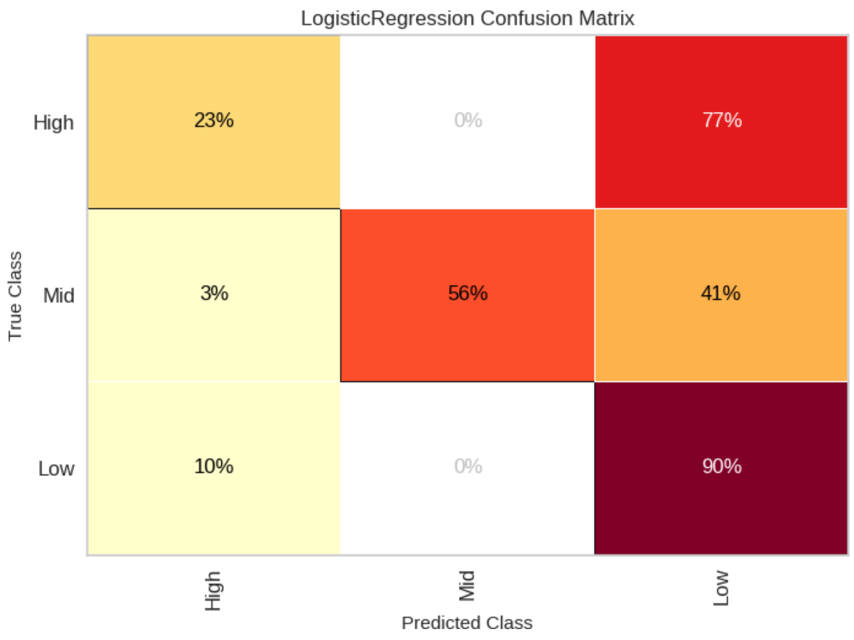


Figure 4-44 LR confusion matrix

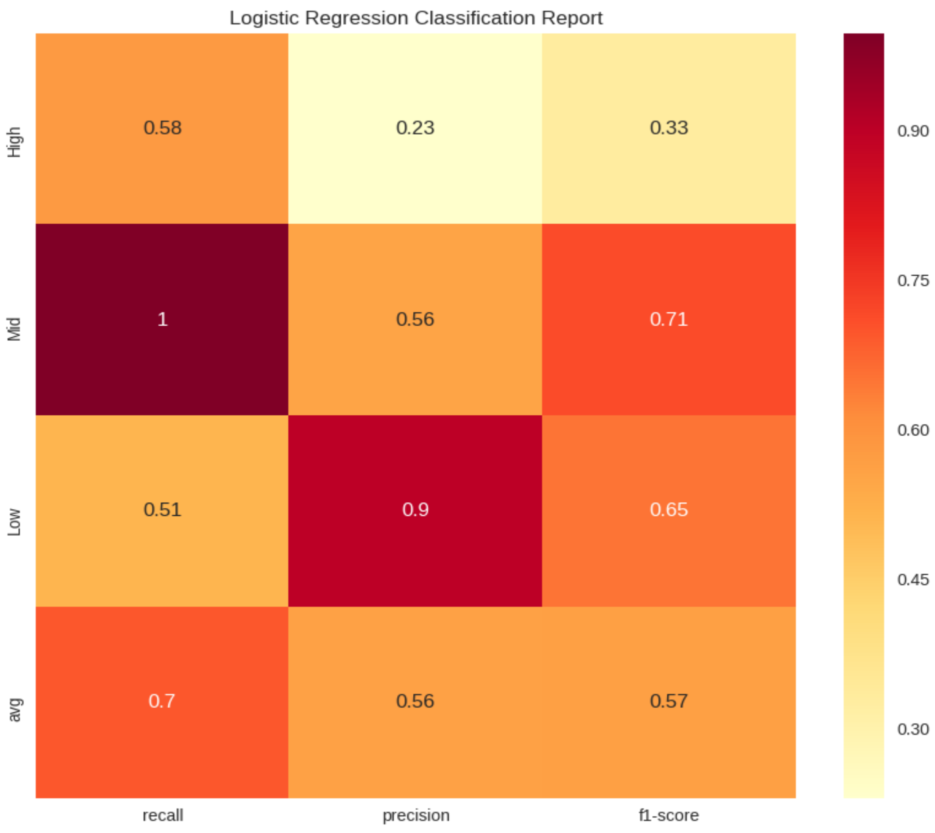


Figure 4-45 LR classification report

Support Vector Machine

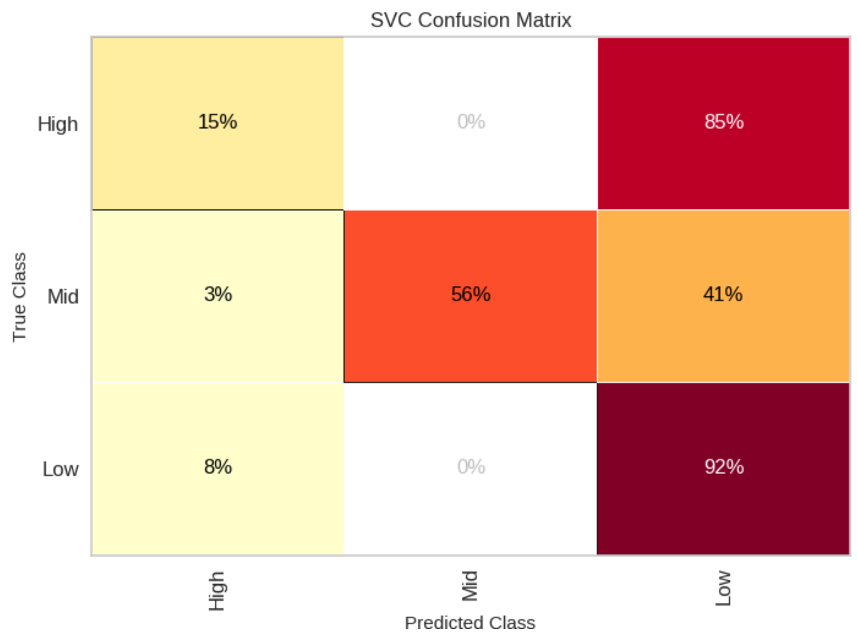


Figure 4-46 SVM confusion matrix

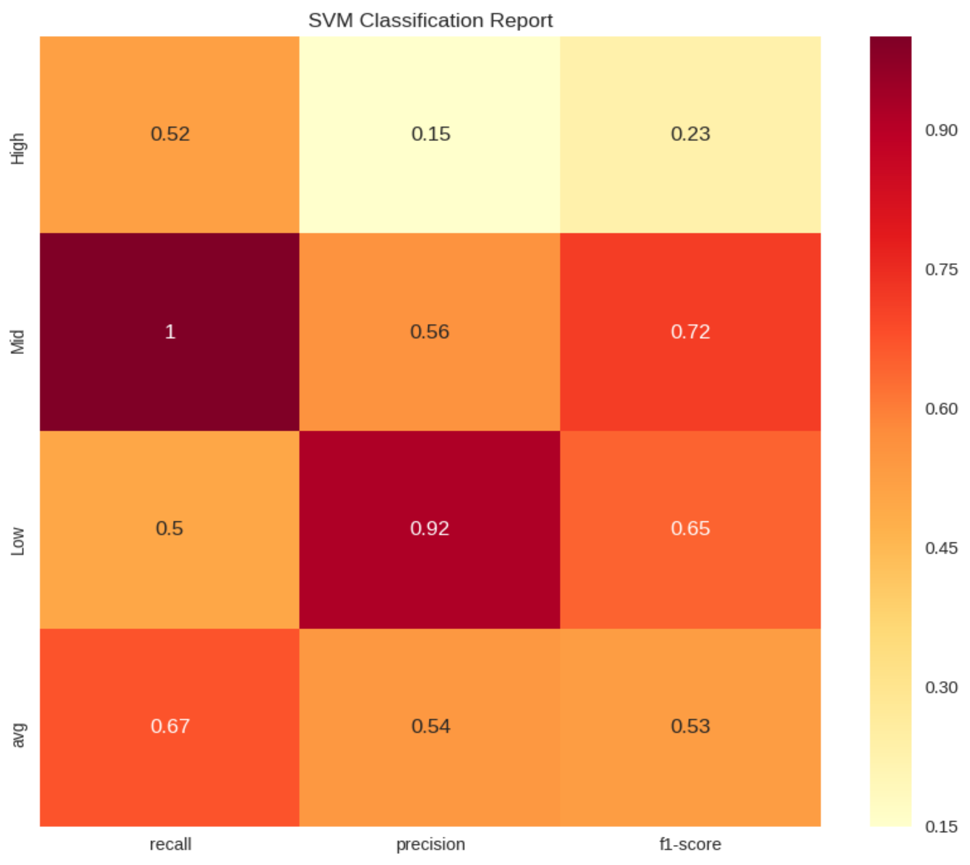


Figure 4-47 SVM confusion matrix

Decision Tree

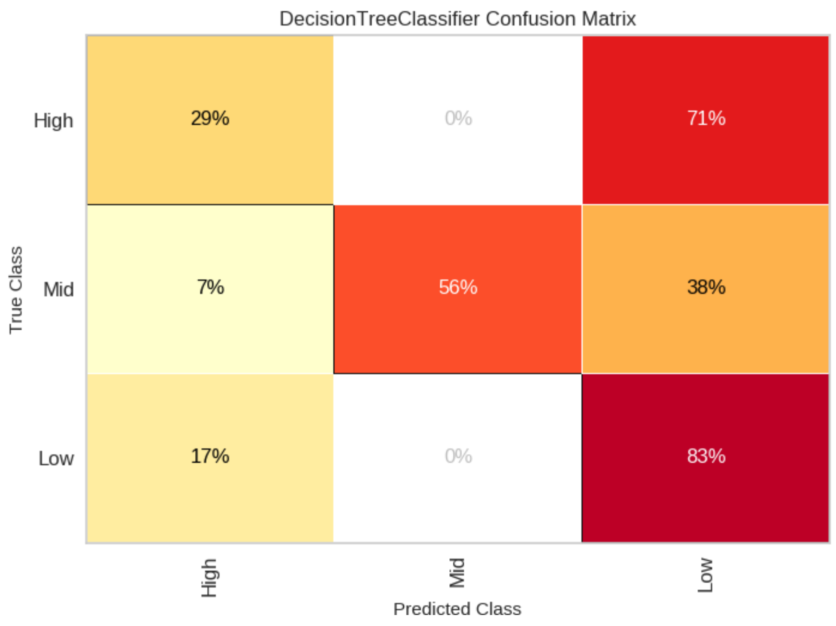


Figure 4-48 DT confusion matrix

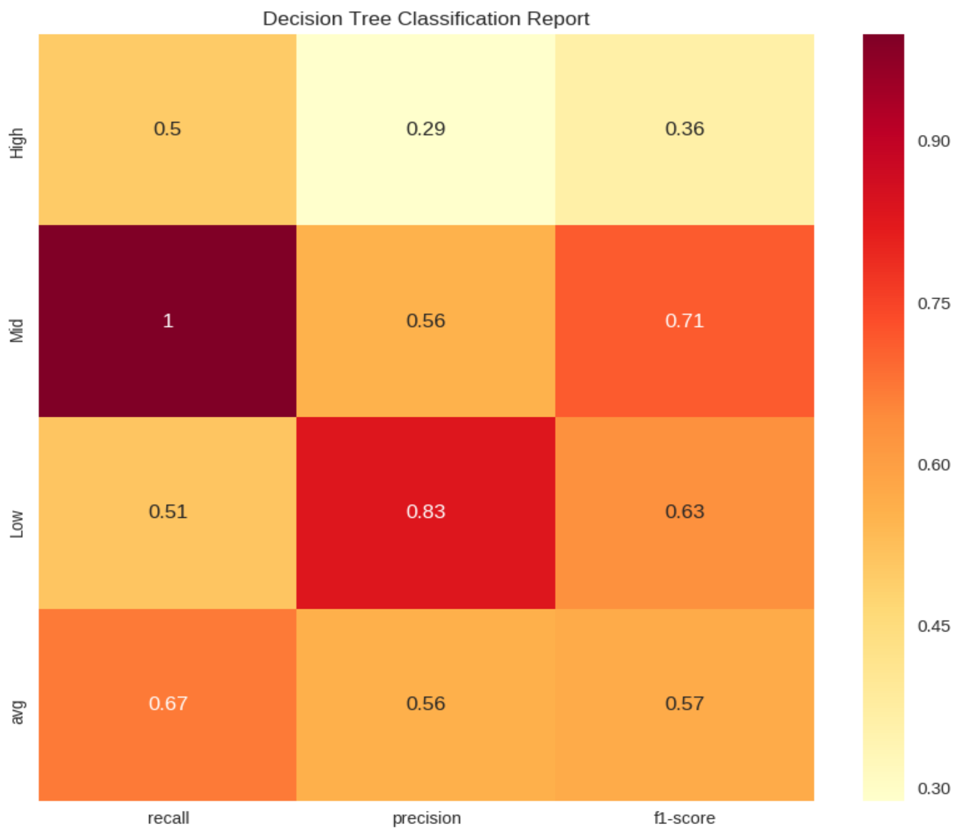


Figure 4-49 DT classification report

Multi-layer Perceptron (MLP)

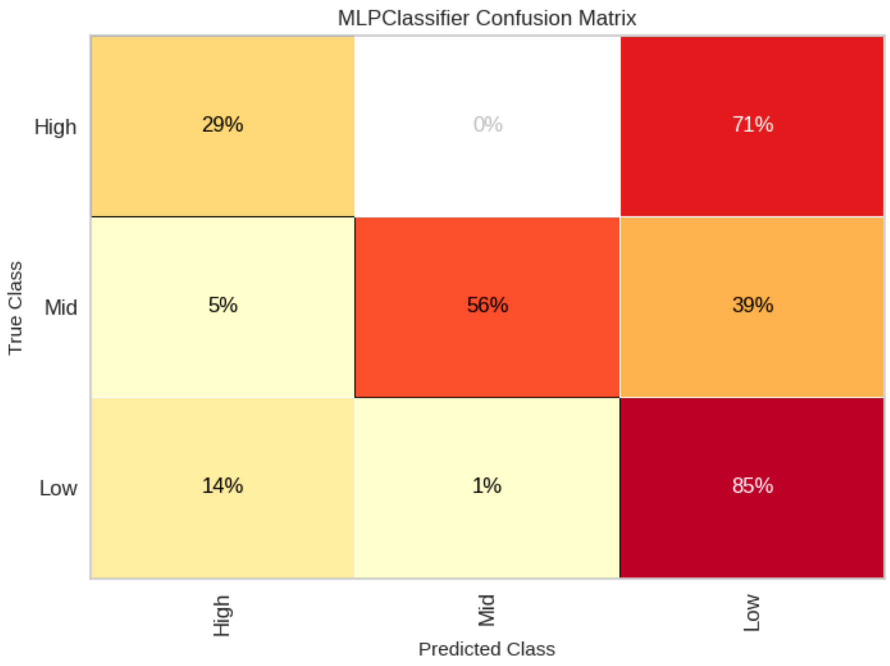


Figure 4-50 MLP confusion matrix

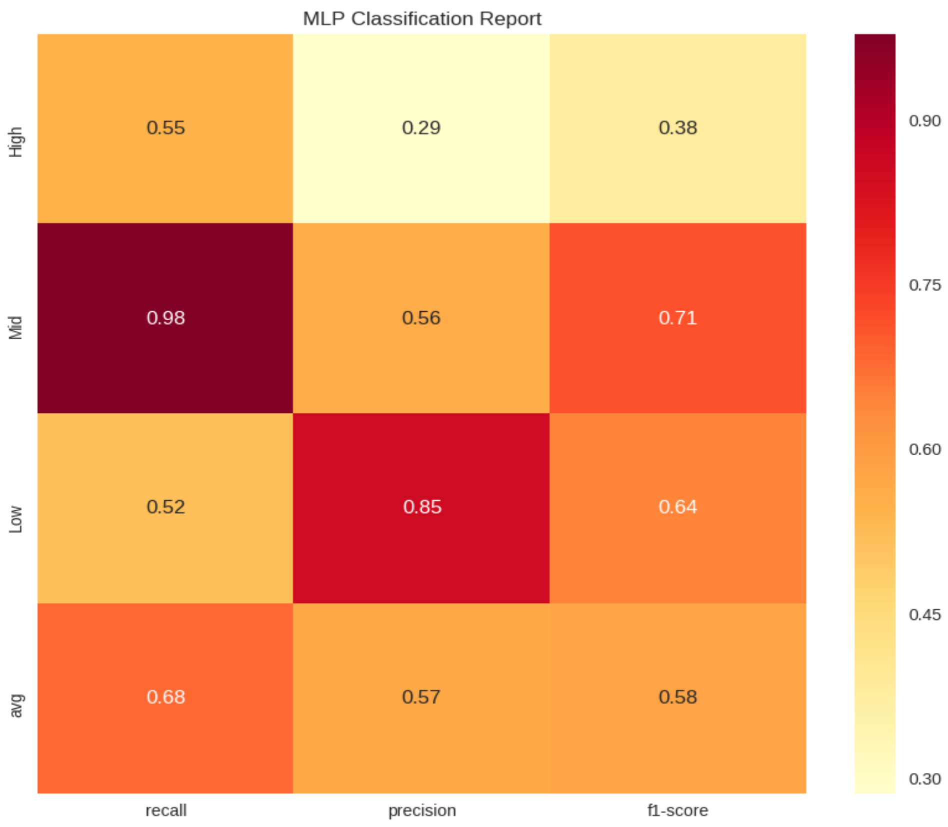


Figure 4-51 MLP classification report

Accuracy

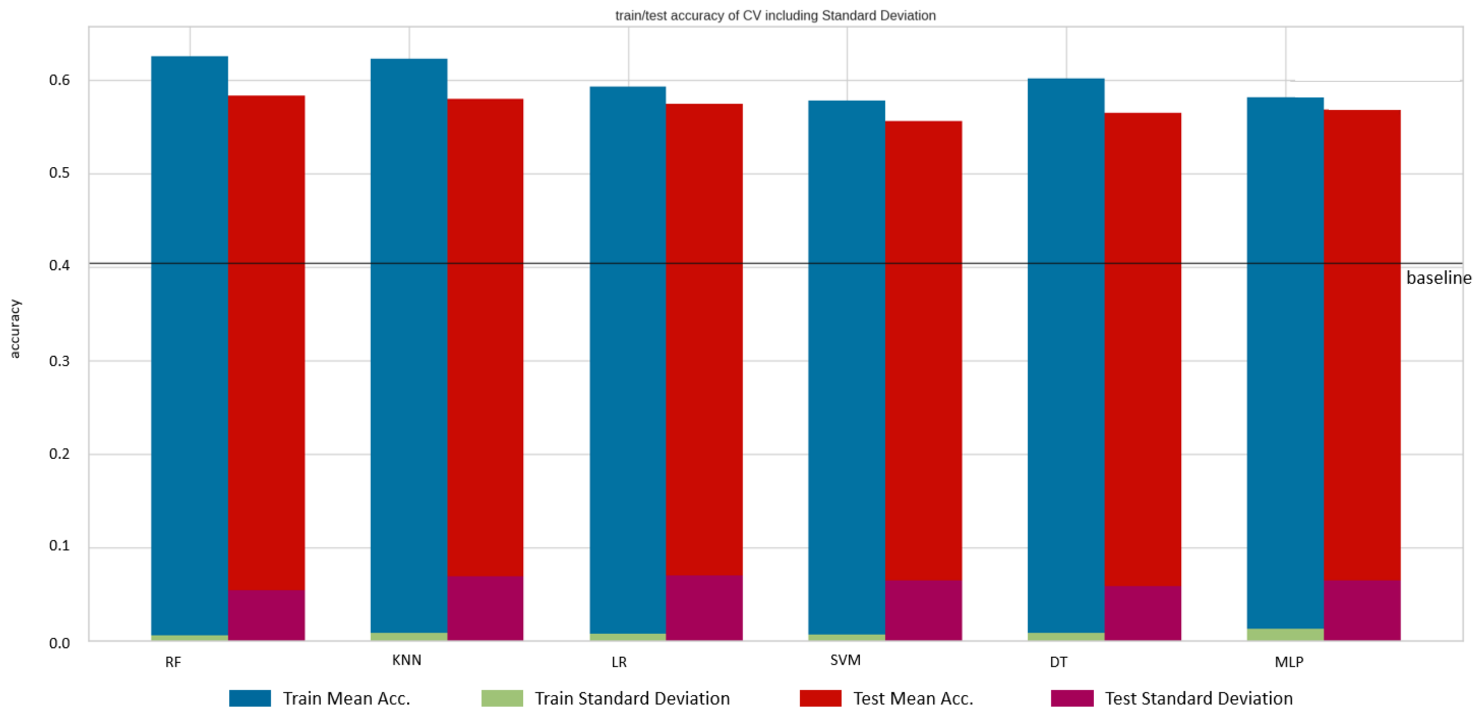


Figure 4-52 Train & Test Cross-Validation Accuracy scores Multiclass

T-test

	Random Forest	k-Nearest Neighbor	Logistic Regression	Support Vector Machines	Decision Tree
k-Nearest Neighbor	**t = 0.1268** **p = 0.9018**				
Logistic Regression	**t = 0.3706** **p = 0.7194**	**t = 0.1733** **p = 0.8643**			
Support Vector Machines	**t = 1.2534** **p = 0.2416**	**t = 0.761** **p = 0.4565**	**t = 0.5793** **p = 0.5695**		
Decision Tree	**t = 5.3382** **p = 0.00045**	**t = 0.6453** **p = 0.5341**	**t = 0.3998** **p = 0.6832**	**t = -0.4113** **p = 0.6899**	
Multi-layer Perceptron	**t = 7.4337** **p = 0.00153**	**t = 0.5066** **p = 0.6244**	**t = 0.2601** **p = 0.8004**	**t = -0.5684** **p = 0.5834**	**t = -1.0843** **p = 0.3055**

Table 4-4 performed t-Test results

By observing the confusion matrices, classification reports, Figure 4-52 and Table 4-4 we come to the following conclusions for the binary classification results:

- The average accuracy of the multiclass classification appears to be 56.5%, with the worst predicting classifier being SVM (54.3%) and best RF (58.5%).
- We accurately predict at least 15% cases above the baseline (on average) once again, so our experiment is providing some level of predictions.
- The models do not overfit/underfit according to Figure 4-4, as both the train and test accuracy do not significantly differ.
- The classifiers in this case present the abnormality, that "High" labels are strongly misclassified as "Low". Thereafter, "Low" labels misclassified as "High", but not that intensively. As for "Mid" labels, they are either classified correctly or "Low" and rarely "High". Although, almost no labels (nor "High" or "Low") are falsely classified as "Mid".
- The t-Test (Table 4-4) shows that we are not able to profound any of the classifiers optimal compared to the others, as in most cases the p-value is large enough to declare that the result of the comparison is random.

Results Overview

After examining each solution independently, we come to compare the solutions and point out a number of key conclusions. First of all, we observe that within each classification approach we followed there doesn't appear a winner-takes-it-all (optimal) algorithm, that significantly outperforms all others. Another important finding is that it seems that none of the models either overfits or underfits in any of the solutions. Overall, the threshold we created appears to increase the efficiency of the models, with the best performing solution amongst all being the binary classification with threshold = 1.0.

Despite the fact that the accuracy levels seem to differ for each solution, a compelling notice is the fact that the solutions' average gain considering the baseline is about the same on each case. The exception to the outcome mentioned above is the binary classification with threshold = 1.0, where the gain appears to be doubled compared to the other solutions. Thereafter, we came to realize that this classification approach performs best. We underline at this point that all of the models come with a certain level of accurate predictions, high enough to reject the possibility of the experiments being "coin-flips".

CHAPTER 5

CONCLUSION

5.1 Summary of Results and Contributions

Going back to our basic research question of Chapter 1, "Can we efficiently predict the user Ratings of applications in Google Play, using the information within their features?" the answer appears to be affirmative. As discussed in Results (Chapter 4), a solution capable of predicting satisfactorily ratings based on the other features of applications, is plausible reaching up to 87% accuracy. Overall, it seems we can predict user ratings to a good degree, focusing mainly on creating a good solution model rather than aiming to finding the classifier that optimizes the output to the maximum in this specific case. As six of the most-widely used classifiers were utilized, the results show that well-tuned algorithms would solve our problem achieving almost the same results. In summary it appears that we can offer an efficient solution on predicting ratings, taking advantage of a different perspective compared to existing studies (see Related Work Section 2.5).

5.2 Limitations

The methods utilized in this thesis are general and commonly used in a variety of similar Machine Learning problems. Both the technique of creating a threshold and the algorithms used, however, were modified and adapted to solve the problem of ratings prediction. Thresholds and algorithms are problem dependent, therefore proper parametrization is mandatory in each different case study or domain we tackle. On the other hand, the evaluation techniques and metrics can be applied on every problem as they use the information acquired from the tuned step and do not have to be modified. Finally, the obtained results appear to be solid within this dataset; we do not know, however, to which extend this approach would be applicable to other datasets containing user ratings about apps or other domains (beyond apps) containing user ratings.

5.3 Future Work

In this final section of the thesis we suggest a number of possible areas for the extension of our approach in the short-term future..

- Google Play is not the only platform used to market applications. The markets vary depending on the devices’ manufacturer, thereafter the existence of App Store and App Gallery could not be ignored. As a future step, a similar research could take place using datasets referred to other markets. Therefore, our aims is to compare the results for each given case and conclude firstly on how feasible is to create a marketable application amongst all the different platforms and secondly to derive the common factors that make an application marketable in this case.
- Besides Google Play, other platforms take advantage of similar rating systems. Another possible approach would be to try applying our frameworks with proper tuning on different markets (e.g. AirBnB) and examine the appropriateness and generality of our approach beyond software apps.
- A study we consider for future work is to approach this problem from a different (ordinal) perspective using Preference Learning, —which is another subfield of Machine (supervised) Learning. Preference learning is a classification method based on observed preference information which builds and learns from ordinal data. In such a case, user ratings (ordinal by nature) would be converted to ordinal data and treated as pairwise preferences. According to [5][46][47], Preference Learning is far more suitable than classification in instances of modeling experience. Preference learning methods are stated to lead to more efficient, generic and robust models which capture more information about the ideal expected result.

REFERENCES

[1] Damodaran, Aswath. "Marketability and value: Measuring the illiquidity discount." *Available at SSRN 841484* (2005).

[2] Srinivasan, V., William S. Lovejoy, and David Beach. "Integrated product design for marketability and manufacturing." *Journal of Marketing Research* 34.1 (1997).

[3] Knotts, Tami L., Stephen C. Jones, and Gerald G. Udell. "Innovation evaluation and product marketability." *Marketing Management Journal, Fall* 19.2 (2009).

[4] Tian, Yuan, et al. "What are the characteristics of high-rated apps? a case study on free android applications." *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015.

[5] Yannakakis, Georgios N., and Julian Togelius. *Artificial intelligence and games*. Vol. 2. New York: Springer, 2018.

[6] Chakraborty, Supriya, and Nabendu Chaki. "A survey on the semi-structured data models." *Computer Information Systems–Analysis and Technologies*. Springer, Berlin, Heidelberg, 2011. 257-266.

[7] Arasu, Arvind, and Hector Garcia-Molina. "Extracting structured data from web pages." *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003.

[8] Blumberg, Robert, and Shaku Atre. "The problem with unstructured data." *Dm Review* 13.42-49 (2003): 62.

[9] Buneman, Peter. "Semistructured data." *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 1997.

[10] Van den Broeck, Jan, et al. "Data cleaning: detecting, diagnosing, and editing data abnormalities." *PLoS Med* 2.10 (2005): e267.

[11] Osborne, Jason W. *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data*. Sage, 2013.

[12] Angeles, Pilar, and Lachlan M. MacKinnon. "Detection and Resolution of Data Inconsistencies, and Data Integration using Data Quality Criteria." *QUATIC 2004: Conference for Quality in Information and Communications Technology*. 2004.

[13] Heeringa, Steven G., Brady T. West, and Patricia A. Berglund. *Applied survey data analysis*. CRC press, 2017.

[14] Alpaydin, Ethem. *Introduction to machine learning*. MIT press, 2020.

[15] Mitchell, T. "The discipline of machine learning (Technical Report CMUML-06-108)." (2006).

[16] Muhammad, Iqbal, and Zhu Yan. "SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY." *ICTACT Journal on Soft Computing* 5.3 (2015).

[17] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. "Foundations of machine learning.[Sl]." (2012): 18.

[19] Safavian, S. Rasoul, and David Landgrebe. "A survey of decision tree classifier methodology." *IEEE transactions on systems, man, and cybernetics* 21.3 (1991): 660-674.

[20] Shalev-Shwartz, Shai, and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[21] Breiman, Leo. "Random forests." *Machine learning* 45.1 (2001): 5-32.

[22] Bhatia, Nitin. "Survey of nearest neighbor techniques." *arXiv preprint arXiv:1007.0085* (2010).

[23] Aly, Mohamed. "Survey on multiclass classification methods." *Neural Netw* 19 (2005): 1-9.

[24] Kumar, Raj, and Rajesh Verma. "Classification algorithms for data mining: A survey." *International Journal of Innovations in Engineering and Technology (IJIET)* 1.2 (2012): 7-14.

[25] Tolles, Juliana, and William J. Meurer. "Logistic regression: relating patient characteristics to outcomes." *Jama* 316.5 (2016): 533-534.

[26] An, Anthony B. "Performing logistic regression on survey data with the new SURVEYLOGISTIC procedure." *Proceedings of the twenty-seventh annual SAS® users group international conference*. SAS Institute Inc. Cary, NC, 2002.

[27] Nayak, Janmenjoy, Bighnaraj Naik, and H. Behera. "A comprehensive survey on support vector machine in data mining tasks: applications & challenges." *International Journal of Database Theory and Application* 8.1 (2015): 169-186.

[28] Pradhan, Ashis. "Support vector machine-a survey." *International Journal of Emerging Technology and Advanced Engineering* 2.8 (2012): 82-85.

[29] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.

[30] Wankhede, Sonali B. "Analytical study of neural network techniques: SOM, MLP and classifier-a survey." *IOSR Journal of Computer Engineering* 16.3 (2014): 86-92.

[31] Arlot, Sylvain, and Alain Celisse. "A survey of cross-validation procedures for model selection." *Statistics surveys* 4 (2010): 40-79.

[32] Moreno-Torres, Jose García, José A. Sáez, and Francisco Herrera. "Study on the impact of partition-induced dataset shift on $ k $-fold cross-validation." *IEEE Transactions on Neural Networks and Learning Systems* 23.8 (2012): 1304-1312.

[33] Hossin, Mohammad, and M. N. Sulaiman. "A review on evaluation metrics for data classification evaluations." *International Journal of Data Mining & Knowledge Management Process* 5.2 (2015): 1.

[34] Lever, Jake, Martin Krzywinski, and Naomi Altman. "Classification evaluation." (2016): 603-604.

[35] Feurer, Matthias, and Frank Hutter. "Hyperparameter optimization." *Automated Machine Learning*. Springer, Cham, 2019. 3-33.

[36] Monett, Dagmar, and Hermann Stolte. "Predicting star ratings based on annotated reviews of mobile apps." *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2016.

[37] Xu, Qing-Song, and Yi-Zeng Liang. "Monte Carlo cross validation." *Chemometrics and Intelligent Laboratory Systems* 56.1 (2001): 1-11.

[38] Islam, Mir Riyanul. "Numeric rating of Apps on Google Play Store by sentiment analysis on user reviews." *2014 International Conference on Electrical Engineering and Information & Communication Technology*. IEEE, 2014.

[39] Zhu, Ciyou, et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization." *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997): 550-560.

[40] Mahmood, Ahsan. "Identifying the influence of various factor of apps on google play apps ratings." *Journal of Data, Information and Management* 2.1 (2020): 15-23.

[41] Meng, Jingke, et al. "User-specific rating prediction for mobile applications via weight-based matrix factorization." *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 2016.

[42] Christian, Hans, Mikhael Pramodana Agus, and Derwin Suhartono. "Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF)." *ComTech: Computer, Mathematics and Engineering Applications* 7.4 (2016): 285-294.

[43] Imran, Saif, et al. "Depth coefficients for depth completion." *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019.

[44] Zhou, Tinghui, et al. "Kernelized probabilistic matrix factorization: Exploiting graphs and side information." *Proceedings of the 2012 SIAM international Conference on Data mining*. Society for Industrial and Applied Mathematics, 2012.

[45] Mnih, Andriy, and Russ R. Salakhutdinov. "Probabilistic matrix factorization." *Advances in neural information processing systems*. 2008.

[46] Yannakakis, Georgios N., Roddy Cowie, and Carlos Busso. "The ordinal nature of emotions: An emerging approach." *IEEE Transactions on Affective Computing* (2018).

[47] Melhart, David, et al. "Your gameplay says it all: modelling motivation in Tom Clancy's the division." *2019 IEEE Conference on Games (CoG)*. IEEE, 2019.