Diploma Project

PREDICTION OF VIRALITY AND POPULARITY OF YOUTUBE VIDEOS ON TWITTER:
MACHINE LEARNING ALGORITHMS ANALYSIS

Giorgos Demosthenous

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2017

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Prediction of Virality and Popularity of YouTube Videos on Twitter:

Machine Learning Algorithms Analysis

Giorgos Demosthenous

Supervisor

Chryssis Georgiou

This diploma project was submitted to fulfil a part of the requirements of acquiring the Computer Science Degree of the University of Cyprus

May 2017

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to Dr. Georgiou Chryssis, Associate Professor at the Department of Computer Science of the University of Cyprus for being my mentor and helping me in every stage of my thesis. I am extremely thankful and indebted to him for sharing his expertise and giving me the chance to further develop my skills in one of my favorite topics, which is machine learning.

I am also grateful to Dr. Pallis George, Assistant Professor at the Computer Science Department of University of Cyprus, for providing me with all the necessary knowledge needed to begin learning about the technologies that were necessary for the implementation of the tools in my study.

I would also like to thank Georgiou Zacharias, who was working in a different topic of the same research as me, for the way he pushed me to acquire new and very valuable knowledge for certain technologies. With his help, we were able to solve the common problem that was the collection of the data needed for our research.

Finally, I would like to thank Evi Fasouliotou for supporting me and my parents for encouraging me during my final year at the University of Cyprus.

ABSTRACT

The exponential growth of video viewing traffic on the Internet has led to the rise of the internet economy. Marketeers aiming to promote their content have turned their interest on this fairly new kind of advertisements on videos. Their main concern is that they would only like to promote their content on videos that will be viewed by as many people as possible. This requirement of knowing which video is going to attract the more viewers, has made necessary the developing of predictive mechanisms that would be able to predetermine the popularity and virality of videos.

In order to be able to predict when a video could become popular or viral, we first needed to understand under which circumstances a YouTube video is most likely to flourish. To achieve this, information about the video on the two platforms, YouTube and Twitter, was collected and analyzed. Monitoring the progress of each video for a specific amount of days, we were able to extract a set of YouTube and Twitter features to later determine if they play any role in the progress of that video. Having collected all the necessary data, the next step was to be able to predict when a video was going to become popular, viral or both. Ideally, a prediction of the oncoming popularity or virality of a video, should be made within a couple of days since its upload. To achieve this, a powerful prediction model had to be created using machine learning.

In this study, several powerful prediction models where developed using a variety of machine learning algorithms and methodologies. More specifically, multiple algorithms where tested, evaluated and compared to determine which ones produce the most accurate predictions on the study under consideration. The next step was to combine multiple machine learning algorithms and methodologies to create hybrid algorithms that were able to produce even higher accuracy predictions.

Most of the algorithms used in this research where fine-tuned to increase their performance and evaluated in detail to determine under which circumstances each algorithm thrives and how suitable the algorithm is for the classification of a video. The most powerful models where then compared to each other to determine which one produces the most desirable and accurate classifications under the current circumstances. The main objective was to come up with a bunch of machine learning algorithms that performed better in different scenarios, and finally decide which one is most suitable for this research.

Finally, another goal of this study was to create models that are able to extract the importance of each feature in order to have the ability to narrow down the features of each platform that are the most important in the classification of the video.

CONTENTS

LIST OF TABLES AND FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

DT: Decision Tree

LR: Logistic Regression

ET: Extra Trees

RF: Random Forest

KNN: K Nearest Neighbors

BDT: Bagging Decision Tree

GBDT: Gradient Boosting Decision Tree

ABDT: Ada Boosting Decision Tree

BGBDT: Bagging Gradient Boosting Decision Tree

SVM: Support Vector Machine

Chapter 1

Introduction

1.1 Motivation and Prior Work

During the last decade, a huge part of Internet usage has shifted from traditional web to media, and more specifically videos [1]. This exponential growth of video viewing traffic on the Internet has led to the rise of the Internet economy. Marketeers aiming to promote their content have turned their interest on this fairly new kind of advertisements on videos. Their main concern is that they would only like to promote their content on videos that will be viewed by as many people as possible. This need to be able to know which video is going to attract the more viewers, has made necessary the existence of a predictive mechanism that would be able to predetermine the popularity and virality of videos.

The definition of popularity and virality was given in David Vallety's previous research on a similar topic: *Characterizing and Predicting Viral-and-Popular Video Content* [2]. They define **popularity** as the inherent propensity of a video to attract views on YouTube and **virality** as its potential to elicit Twitter posts from its viewers. The most valuable and interesting videos are those that become both viral and popular. Predicting whether a video will become both viral and popular has proven to be one of the most difficult tasks.

In order to be able to predict when a video would become popular or viral, we first needed to understand under which circumstances a YouTube video is most likely to flourish. To achieve this, information about the video on the two platforms, YouTube and Twitter, was collected and analyzed. Monitoring the progress of each video for a specific amount of days, we were able to extract a set of YouTube and Twitter features to later determine if they play any role in the popularity or virality of that video.

Having collected all the necessary data, the next step was to be able to predict when a video was going to become popular, viral or both. In order to be of any value, this prediction should be made as independent as possible from the number of days the video was being monitored. Ideally, a prediction of the oncoming popularity or virality of a YouTube video, should be made within a couple of days since its upload. To achieve this need a powerful prediction model had to be created using machine learning. **Machine learning** [3] provides computers with the ability to learn without being explicitly programmed and focuses on the development of algorithms whose outcome can change when exposed to new data.

In this study, several powerful prediction models where developed using a variety of machine learning algorithms and methodologies [15]. More specifically, multiple algorithms where tested, evaluated and compared to determine which one produces the most accurate predictions. The next step was to combine multiple machine learning algorithms and methodologies to create hybrid algorithms that were able to produce even higher accuracy predictions.

Most of the algorithms used in this research where evaluated in detail in chapter 3 to determine under which circumstances each algorithm thrives and how suitable the algorithm is for the classification of a video as popular and viral. The most powerful models where then compared to each other to determine which one produces the most desirable and accurate classifications under the current circumstances. Machine learning algorithms analysis plays a huge role in the overall goal of the research. More specifically, this part of the research, which is to analyze, evaluate and compare machine learning algorithms allows us to use very powerful models to predict the popularity and virality of a YouTube video with high accuracy. This also gives us the ability to narrow down the features of each platform that are the most important in the classification of the video.

Although there have been similar studies [4] that attempted to predict the virality and popularity of videos, there wasn't any research or prior work regarding machine learning algorithms analysis that attempted to create powerful models that would make the most accurate predictions for this problem. This is the novelty of the presented study.

1.2 Goals of the Study

The ultimate goal of this study was to create powerful models that would be able to make high accuracy predictions of virality and popularity of YouTube videos.

One of the main objectives of this research was to analyze and evaluate a variety of different machine learning algorithms, in order to examine their strengths and weaknesses related to this study's problem. The algorithms had to be fine-tuned to increase their performance and make them more suitable for the current research. In order to achieve even higher accuracy classifications, hybrid algorithms had to be created.

Another important goal of the study was to compare the most promising algorithms in order to examine their behavior in relation to each other and determine the most suitable ones [5] for this problem. The objective was to come up with a bunch of machine learning algorithms that performed better in different scenarios, and finally decide which one is most suitable for this research.

Finally, another goal of this study was to create classifiers that are able to extract the importance of each feature in the classification and prediction of popularity and virality of YouTube videos.

1.3 Methodology

The following diagram presents the abstract methodology of this study.

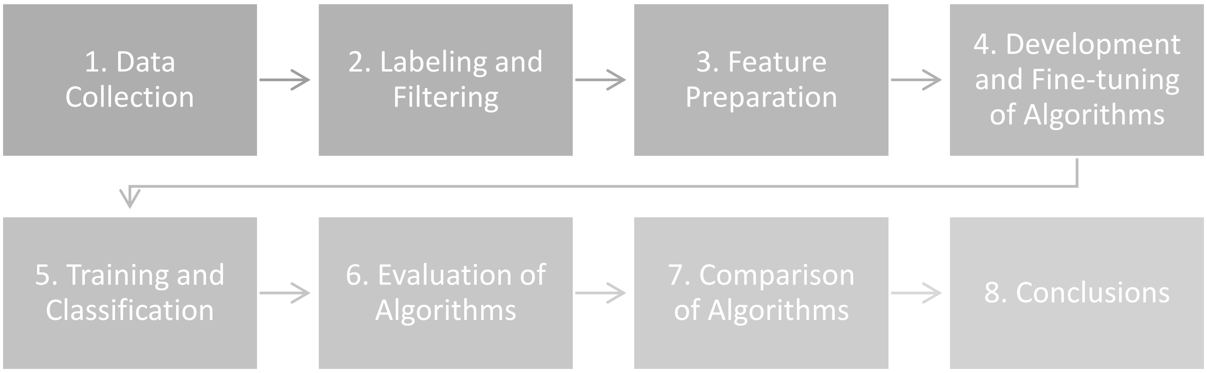

1. Data Collection
2. Labeling and Filtering
3. Feature Preparation
4. Development and Fine-tuning of Algorithms
5. Training and Classification
6. Evaluation of Algorithms
7. Comparison of Algorithms
8. Conclusions

Figure 1: Abstract diagram of the methodology used in this study

Data Collection: The general idea of how the data collection system works is described in detail in Section 2.3. In order to be able to collect the information needed, YouTube Data API

[6] and Twitter Streaming API [7] where used. The overall procedure was repeated for every video collected.

Labeling and Filtering: Only information needed for the creation of the training features where kept. Out of the 130.000 videos, only a predefined percentage is used in the training procedure. For example, in strict labeling, the percentage used is 2.5%. This means that the dataset used in the training procedure consisted of the 2.5% most popular videos, the 2.5% most viral videos, 2.5% most recent videos and 2.5% random videos. Some videos appear in multiple of the above categories, meaning that the real amount of unique videos in each category is less than 2.5%.

Feature Preparation: The raw information acquired from the datasets is processed in order to export features that will be used during the training of the classifiers. The features prepared are described in Section 2.2

Development and Fine-tuning of Algorithms: After extensive research [11], various kinds of machine learning algorithms where developed and fine-tuned to better fit the needs of this research.

Training and Classification: The dataset in an experiment is used to train each classifier, which was created using a different kind of algorithm. All the training features are fitted to the model in order to produce a final classification of the videos. The outcome is a bundle of metrics that show how well an algorithm has performed and what features where the most important.

Evaluation of Algorithms: A selection of algorithm is evaluated in order to examine their performance, strengths and weaknesses using baseline tests. A more detailed description of the evaluation methodology is presented in Section 3.1.

Comparison of Algorithms: After evaluating the algorithms, the most promising ones are compared between each other to determine when each algorithm performs better and why. The comparison methodology is described in detail in Section 4.1.

Conclusions: After evaluating and comparing the algorithms, the better performing ones are kept for further usage and development. Each algorithm selected performs better in different scenarios. The comparison conclusions are presented in detail in Section 4.7

1.4 Contributions

This study offers several contributions to the scientific community. First of all it allows any novice to rapidly pick up and understand basic machine learning knowledge. It explains in the depth how some algorithms are separated and how they operate. In addition, a variety of evaluation metrics are presented and a detailed explanation is given on how each one can be used to evaluate the performance of a machine learning algorithm.

The tools developed for the purpose of this study allow for a better understanding of how each machine learning algorithm, discussed in this document, behaves and why. More specifically, the web application developed gives the ability to see several metrics regarding a specific algorithm. You can see how it performs under different scenarios by examining its precision-recall graphs and its' F1 scores presented in the web application. In addition, it allows the user to graphically compare two algorithms to determine which one performs better under which circumstances.

The prediction models developed in this study can be used for any kind of binary classification problems and can export several kind of metrics regarding the models performance.

Finally, the evaluation and the comparison of the algorithms presented in this study depicts how each algorithm behaves in different situations. Additionally, the conclusions extracted from these evaluations and comparisons contribute heavily to the scientific community, since they can be used for further development and adjustment of the algorithms in similar problems that require binary prediction models.

1.5 Document organization

The following table contains a brief summary of what is covered in each chapter.

Chapter 2	This chapter contains some of the background knowledge and tools needed for this study. More specifically, a brief explanation is given on how each of the machine learning algorithms used works. In addition, an abstract description is given about the data collection procedure. This chapter also contains the evaluation metrics used in the research along with an explanation for each one. The training features are analyzed and described towards the end of this chapter. Finally, this chapter contains implementation details about the tools developed.
Chapter 3	In this chapter each algorithm is ankylosed and evaluated using a specified methodology. More specifically, each algorithm’s performance is tested against a set of defined baseline values in order to examine the quality of the classifiers.
Chapter 4	This chapter contains a methodology followed to compare each algorithm. A bundle of the highest performing algorithms was chosen and compared in order to extract some conclusions. Towards the end of the chapter, a set of the most suitable algorithms is chosen and analyzed for its strengths and weaknesses. A final conclusion is then made as to which algorithm performs better at which scenario, including this research.
Chapter 5	This chapter contains the overall conclusions of this study. In addition, it discusses future work that could be done based on a hypothesis that could lead to the creation of much more powerful models.

Chapter 2

Background Knowledge and Tools

2.1 Machine Learning Algorithms

There are three types of machine learning algorithms: supervised learning, unsupervised learning and reinforcement learning. For the purpose of this research supervised learning was used, more specifically classification, to test assumptions, validate data, make predictions and reach to certain conclusions regarding the popularity and virality of YouTube videos.

The machine learning algorithms chosen where separated into two groups:

1. Stand-alone Algorithms
2. Hybrid Algorithms

2.1.1 Stand-alone Algorithms

Stand-alone algorithms are standalone machine learning algorithms that make classifications using their own methodologies without any third-party assistance.

2.1.1.1 Logistic Regression

Logistic regression [9], despite its name, is a linear model for binary classification rather than regression and can be extended to multiclass classification via the OvR technique. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

The logistic function is an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits. Where “e” is the base of the natural logarithms and “x” is the actual numerical value that you want to transform. In the case of logistic regression, “x” is the linear combination of weights and sample features that can be calculated as x = W0 + W1F1 + …. + WmFm.

Due to its probabilistic nature this powerful algorithm is idle for binary classification. Strictly speaking the Logistic Regression algorithms classifies the input data as follows:

$$\text{Class} = \begin{cases} 1\ if\ f(x) \geq 0.5 \\ 0\ otherwise \end{cases}$$

The logistic regression algorithm used in this study optimizes its outcome by minimizing the following cost function:

$$\min_{w,c} \frac{1}{2}w^T w + C\sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

2.1.1.2 K Nearest Neighbors

Nearest Neighbors algorithms [9] are a family of machine learning algorithms that classify datasets based on a chosen distance metric. The K Nearest Neighbors algorithm is a typical example of a lazy learner. It is called lazy because it doesn’t learn a discriminative function from the training data but memorizes the training dataset instead.

The KNN algorithm uses a simple Euclidian distance metric to determine the class of sample data as shown in the following equation.

$$D(x,y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

Based on this distance metric, the KNN algorithm finds the k samples in the training dataset that are closest to the point that we want to classify, where k is the number of neighbors chosen. The class label of the new data point is then determined by a majority vote among its k nearest neighbors.

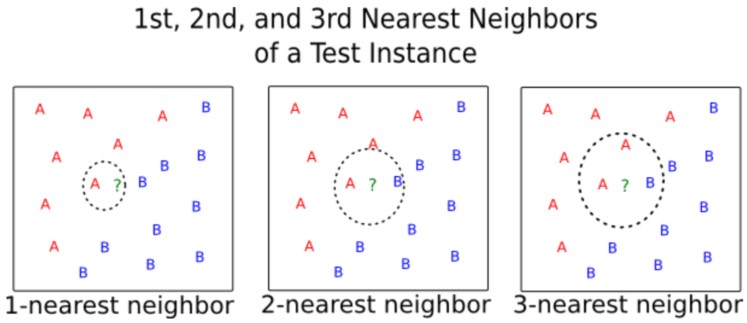


Figure: KNN algorithm takes into consideration the K nearest neighbors (1, 2, 3)

The main advantage of such a memory-based approach is that the classifier continually adapts and improves as we collect new training data. However, the downside is that the computational complexity for classifying new samples grows linearly with the number of samples in the training dataset. This means that the KNN algorithm will require more time to classify the new data as the size of the training dataset grows.

2.1.1.3 SVM – Linear SVC

Support Vector Machine (SVM) algorithms [18] aim to maximize the distance between the decision boundary, which is known as margin, and the training samples that are closest to this boundary, which are known as support vectors.

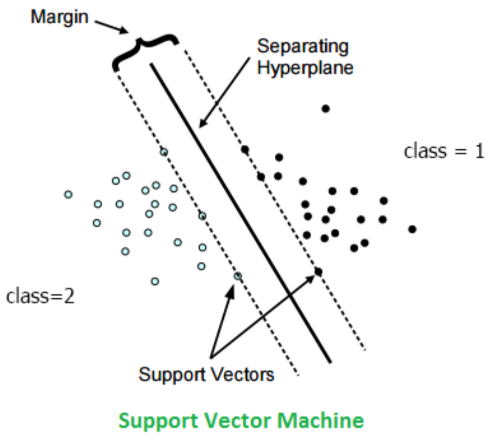


Figure 2: Two class classification using Support Vector Machine [9]

This family of machine learning algorithms is characterized by its adaptability to various kinds of problems since it can use different types of kernels to separate sample training data into classes.

Support Vector Machine algorithms tend to work more effectively in high dimensional spaces. Binary classification, which is used in this study, requires a linear kernel to be used in the SVM algorithms. For the purpose of the current research, three different SVM algorithms with linear kernels where tested: SVC, NuSVC and Linear SVC.

SVC, which stands for support vector classification, is an SVM algorithm that can use multiple kind of kernels to execute different types of classifications. In our case, a linear kernel was used for SVC. NuSVC is similar to the SVC algorithm but uses a parameter to control the number of support vectors. Linear SVC doesn't have a kernel parameter since it uses a linear kernel by default. Linear SVC was chosen over the two other algorithms since it had better overall performance and was more suitable for the needs of this research.

2.1.1.4 Decision Tree

Decision Tree classification algorithms [9] are popular models due to their interpretability. This family of algorithms operates by breaking down sample training data by making decision based on asking a series of questions. An example of a decision tree is shown in Figure 3.

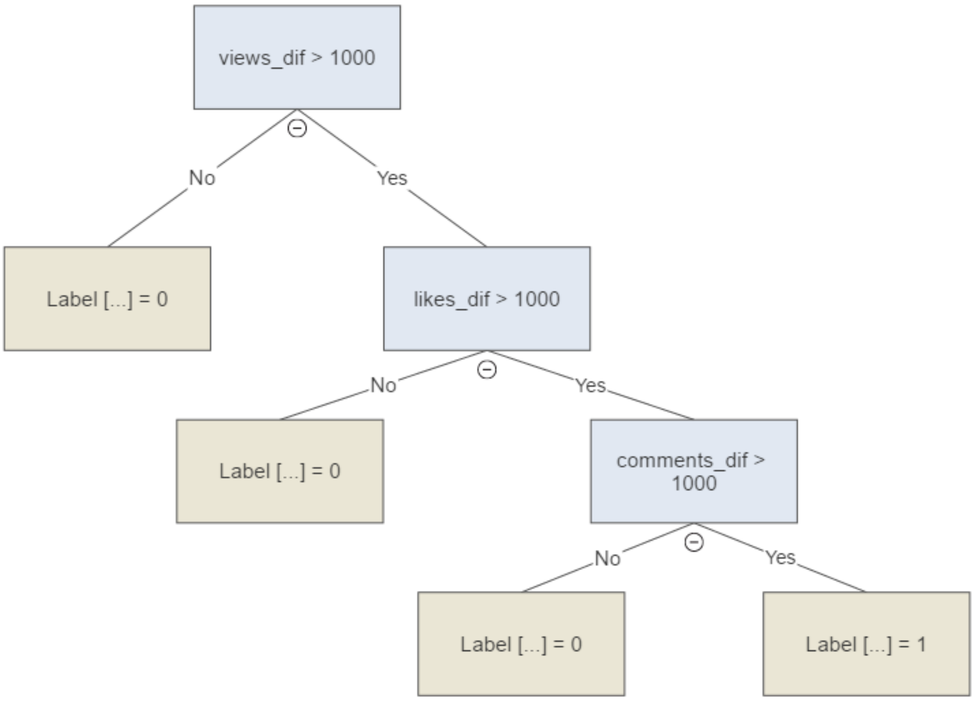


Figure 3: Example of a decision tree structure

The decision algorithm starts at the root of the tree and splits the data on the feature that results in the largest information gain. The classification process finishes when the decision tree is completely built.

Decision Tree algorithms are “weak learners” that’s why they are mostly used as part of other machine learning algorithms to improve the performance and the accuracy of a classification.

2.1.2 Hybrid Algorithms

Hybrid algorithms [20] are a combination of machine learning algorithms and methodologies that aim to enhance the accuracy of the classifications. The general idea of ensemble-hybrid learning is to combine “weak learners” to build a more robust model, a “strong learner”.

2.1.2.1 Random Forest and Extra Trees

Random forests are a collection of randomized decision trees. More specifically the algorithm chooses n random samples from the training data set with replacement and then proceeds to create a decision tree. In the decision tree created the algorithm randomly chooses a specified amount of features without replacement before moving to the splitting procedure and the classification. The algorithm repeats the procedure until a forest of a predefined number of trees is created. The final class labels are assigned using majority voting among the decision trees of the forest.

Just like Random Forests **Extra Trees** is a collection of randomized decision trees. The process of the forest creation and the classification is similar to that of the Random Forest algorithm with an exception of an additional random step. In this algorithm randomness goes one step further in the way splits are computed. As in random forests, a random subset of candidate features is used, but instead of looking for the most discriminative thresholds, thresholds are drawn at random for each candidate feature and the best of these randomly-generated thresholds is picked as the splitting rule.

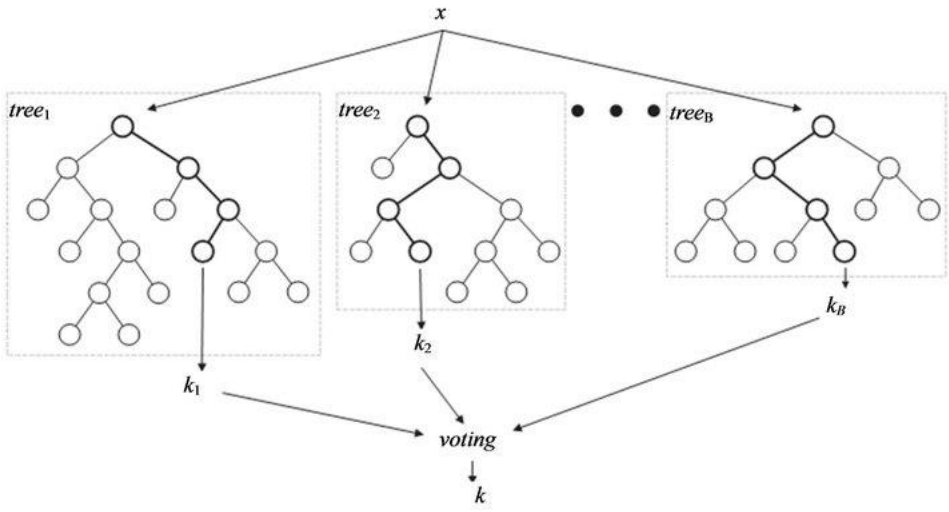


Figure 4: Random Forest and Extra Trees Algorithms structure example

Figure 4 depicts the idea behind a random forest or an extra trees algorithm. Each randomly generated decision tree classifies its data, k1… kb, and then a majority vote takes place. The voting procedure is simple. A specific video is labeled with the classification that appeared the most times in all the sub trees. If a tie occurs then the first classification in acceding order is chosen. Typically, the larger the number of trees, the better the performance of the random forest or extra trees classifier and the expense of an increased computational cost.

2.1.2.3 Bagging Decision Tree

Bagging [9] is an ensemble learning technique that aims to improve the performance of a weak learner. A predefined amount of estimators classify random parts taken from the training data sample and the final labeling occurs after majority voting takes place. Bagging methods help reduce variance and overfitting [21] caused by the individual estimators.

Figure 5 gives a fair representation of how bagging methods work. First an ensemble of decision trees with random subsets of data is created. Each decision tree contains all the features used in the dataset. After each individual classification, a majority vote takes place. Looking at all the classifications, C1…Cm, we chose the label that has appeared the most times for a specific video. If a tie occurs, the first labeling is chosen in acceding order.

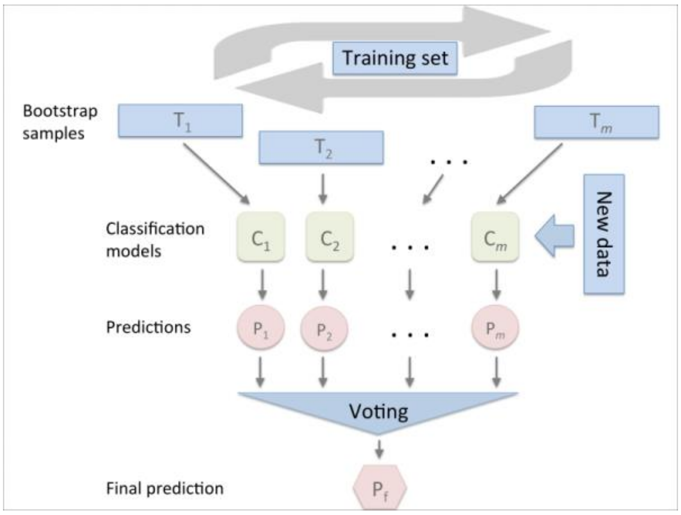


Figure 5: Representation of how the Bagging methodology works [9]

The Bagging Decision Tree algorithm was used to examine the extent at which the bagging technique contributes to the improvement of an individual estimator, in this case a Decision Tree.

2.1.2.4 Boosting Decision Trees

Boosting is the methodology that aims to optimize the overall accuracy of a weak learner by continually improving its classification performance. It starts with using a weak learner to classify some data and then adds more weak learners to the equation that focus on the wrong labels given by the previous classification. Each boosting iteration aims to improve the performance of the previous learner.

2.1.2.4.1 Gradient Boosting Decision Tree

Gradient Boosting [13] is an advance methodology that is used to boost the performance of weak learners (in this case decision trees).

The algorithm bases its functionality on three factors:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

In each boosting iteration a new weak learner is added to the ensemble and at the end of its predictions the overall model's performance is improved whereas the loss function is reduced.

The goal is to minimize the loss function every time a new decision tree is added. This is achieved by using the gradient decent methodology.

2.1.2.4.2 Ada Boosting Decision Tree

The core principle of Ada Boosting [23] is to fit an ensemble of weak learners (in this case decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction.

Each boosting iteration consists of applying weights [W1...Wn] to each of the training samples. Initially, those weights are all set to Wi = 1/N, so that the first step simply trains the weak learners on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. The training samples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. This weight change in each boosting iteration forces the week learners to focus on data that was misclassified in previous iterations.

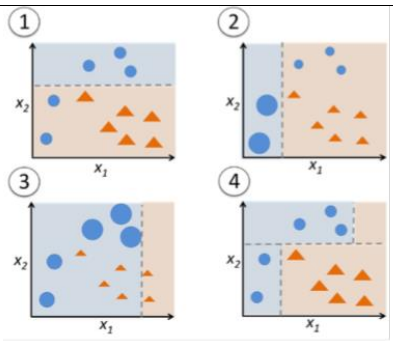


Figure 6: Adaptive boosting example [9]

2.1.2.4.3 Voting – Ada and Gradient Boosting Decision Tree

Voting [9] takes multiple estimator's classifications as input and performs a voting procedure to decide the final classification of the training data. This methodology uses two kinds of voting, hard or soft. If "hard" voting is chosen, a majority voting procedure takes place between all the classifications of the ensemble. Else if "soft" voting is chosen, the final classification is deducted base on the max of the sums of the predicted probabilities. Ada Boosting and Gradient Boosting both improve the performance of a decision tree algorithm and Voting serves as a final improvement step for ABDT and GBDT classifications, deciding the final labeling that takes place.

2.1.2.4.4 Voting – Ada and Gradient Boosting Decision Tree + Logistic Regression

In this majority vote hybrid algorithm, Ada boosting Decision Tree, Gradient Boosting Decision Tree and Logistic Regression where used in an attempt to test the boundaries of the Voting methodology.

2.1.2.4.5 Bagging Gradient Boosting Decision Tree

Gradient Boosting Decision Tree classifier is a powerful model when it comes to making predictions with binary classification. Despite its' high accuracy, GBDT has some flaws which under the Bagging model (explained in Section 2.1.2.3) seem to be reduced substantially.

2.2 Features

2.2.1 Training Features Description

In order to train the classifiers as effectively as possible 74 distinct features where used. With each additional training day the count of window sensitive features (e.g. ratios, differences etc.) is increased accordingly.

YouTube Features (31):

Static Features	
category	The category of the video assigned by YouTube
artificial_category	Some categories were grouped for the purposes of this research
duration	The duration of the video in milliseconds
comments_sentiment_neg	Statistics about video comments that were classified as sentimentally negative
comments_sentiment_neu	Statistics about video comments that were classified as sentimentally neutral

comments_sentiment_pos	Statistics about video comments that were classified as sentimentally positive
comments_sentiment_compound	Statistics about video comments that were classified as sentimentally compound
channel_uploads	The amount of video uploads in the channel the video belongs to
channel_subscribers	The amount of subscribers in the channel the video belongs to
channel_views	The amount of total views in the channel the video belongs to

Table 1: Descriptions of the static YouTube features

Differences	
views_dif	The difference between the accumulated views of a video at the first and last day of the training
likes_dif	The difference between the accumulated likes of a video at the first and last day of the training
dislikes_dif	The difference between the accumulated dislikes of a video at the first and last day of the training
comments_dif	The difference between the accumulated comments of a video at the first and last day of the training

Table 2: Descriptions of the differences between YouTube features

Accelerations - The ratio of a feature between day n and day n-1	
views_acc	The average acceleration of the views
likes_acc	The average acceleration of the likes
dislikes_acc	The average acceleration of the dislikes
comments_acc	The average acceleration of the comments

Table 3: Descriptions of the accelerations of several YouTube features

Daily Stats - (1..n)	
views	Views added to the video at the # day
likes	Likes added to the video at the #day
dislikes	Dislikes added to the video at the #day
comments	Comments added to the video at the #day

Table 4: Descriptions of the daily statistics of several YouTube features

Age ratio – (1..n)	
ageRatioViews	The ratio between the number of views of a video on day # and the date it was uploaded (in days)
ageRatioLikes	The ratio between the number of likes of a video on day # and the date it was uploaded (in days)
ageRatioDislikes	The ratio between the number of dislikes of a video on day # and the date it was uploaded (in days)
ageRatioComments	The ratio between the number of comments of a video on day # and the date it was uploaded (in days)

Table 5: Descriptions of the age ratio of several YouTube features

Ratios - (1..n)	
ratioViews	The ratio between the number of views on day # and the total likes of the video since upload
ratioLikes	The ratio between the number of likes on day # and the total likes of the video since upload
ratioDislikes	The ratio between the number of dislikes on day # and the total likes of the video since upload
ratioComments	The ratio between the number of comments on day # and the total likes of the video since upload

Table 6: Descriptions of the ratios of several YouTube features

Twitter Features (43):

Static Features	
user_followers_count	The average amount of followers users referring to the specific video have
users_verified_count	The amount of verified users referring to a video
user_friends_count	The average amount of friends users referring to the specific video have

Table 7: Descriptions of the static Twitter features

Differences	
tw_tweets_dif	The difference between the total amount of tweets referring to a specific video at the first and last day of the training
tw_orig_tweets_dif	The difference between the total amount of original tweets referring to a specific video at the first and last day of the training
tw_retweets_dif	The difference between the total amount of retweets referring to a specific video at the first and last day of the training
tw_user_favorites_dif	The difference between the total amount favorites the tweet referring to a specific video has received, at the first and last day of the training
tw_eng_dif	The difference between the total amount of tweets in English referring to a specific video at the first and last day of the training
tw_sp_dif	The difference between the total amount of tweets in Spanish referring to a specific video at the first and last day of the training
tw_user_eng_dif	The difference between the total amount of tweets whose user has set their account language to English, referring to a specific video at the first and last day of the training
tw_user_sp_dif	The difference between the total amount of tweets whose user has set their account language to Spanish, referring to a specific video at the first and last day of the training
tw_user_statuses_dif	The difference between the total amount of statuses posted by users that have referred to a specific video, at the first and last day of the training
tw_hashtags_dif	The difference between the total amount of hashtags used in tweets referring to a specific video, at the first and last day of the training

Table 8: Descriptions of the differences between Twitter features

Accelerations - The ratio of a feature between day n and day n-1	
tw_tweets_acc	The average acceleration of tweets referring to a specific video
tw_orig_tweets_acc	The average acceleration of original tweets referring to a specific video
tw_retweets_acc	The average acceleration of retweets referring to a specific video
tw_user_favorites_acc	The average acceleration of user favorites
tw_eng_acc	The average acceleration of English tweets referring to a specific video
tw_sp_acc	The average acceleration of Spanish tweets referring to a specific video
tw_user_eng_acc	The average acceleration of English users referring to a specific video
tw_user_sp_acc	The average acceleration of Spanish users referring to a specific video
tw_user_statuses_acc	The average acceleration of statuses of a User that has referred to a specific video
tw_hashtags_acc	The average acceleration of the number of hashtags used in tweets referring a specific video

Table 9: Descriptions of the accelerations of several Twitter features

Daily stats - (1..n)	
tweets_added	The number of tweets added on day #
original_tweets_added	The number of original tweets added on day #
retweets_added	The number of retweets added on day #
tweets_favorited_added	The number of tweets favored on day #
tweets_in_english_added	The number of English tweets added on day #
tweets_in_spanish_added	The number of Spanish tweets added on day #
user_eng_count	The number of English users that posted a tweet on day #
user_sp_count	The number of Spanish users that posted a tweet on day #
user_statuses_count	The number of user statuses added on day #
tweets_hashtags_added	The number of additional hashtags used on day #

Table 10: Descriptions of the daily statistics of several Twitter features

Ratios – (1..n)	
ratioTweets	The ratio between the number of tweets on day # and the total number of tweets since post.
ratioOrigTweets	The ratio between the number of original tweets on day # and the total number of tweets since post.
ratioRetweets	The ratio between the number of retweets on day # and the total number of tweets since post.
ratioUserFavorites	The ratio between the number of user favorites on day # and the total number of user favorites since post.
ratioTwEn	The ratio between the number of English tweets on day # and the total number of English tweets since post.
ratioTwSp	The ratio between the number of Spanish tweets on day # and the total number of Spanish tweets since post.
ratioUserEng	The ratio between the number of English users on day # and the total number of English users since post.
ratioUserSp	The ratio between the number of Spanish users on day # and the total number of Spanish users since post.
ratioUserStatuses	The ratio between the number of user statuses on day # and the total number of user statuses since post.
ratioHashtags	The ratio between the number of hashtags used on day # and the total number of hashtags used since post.

Table 11: Descriptions of the ratios of several Twitter features

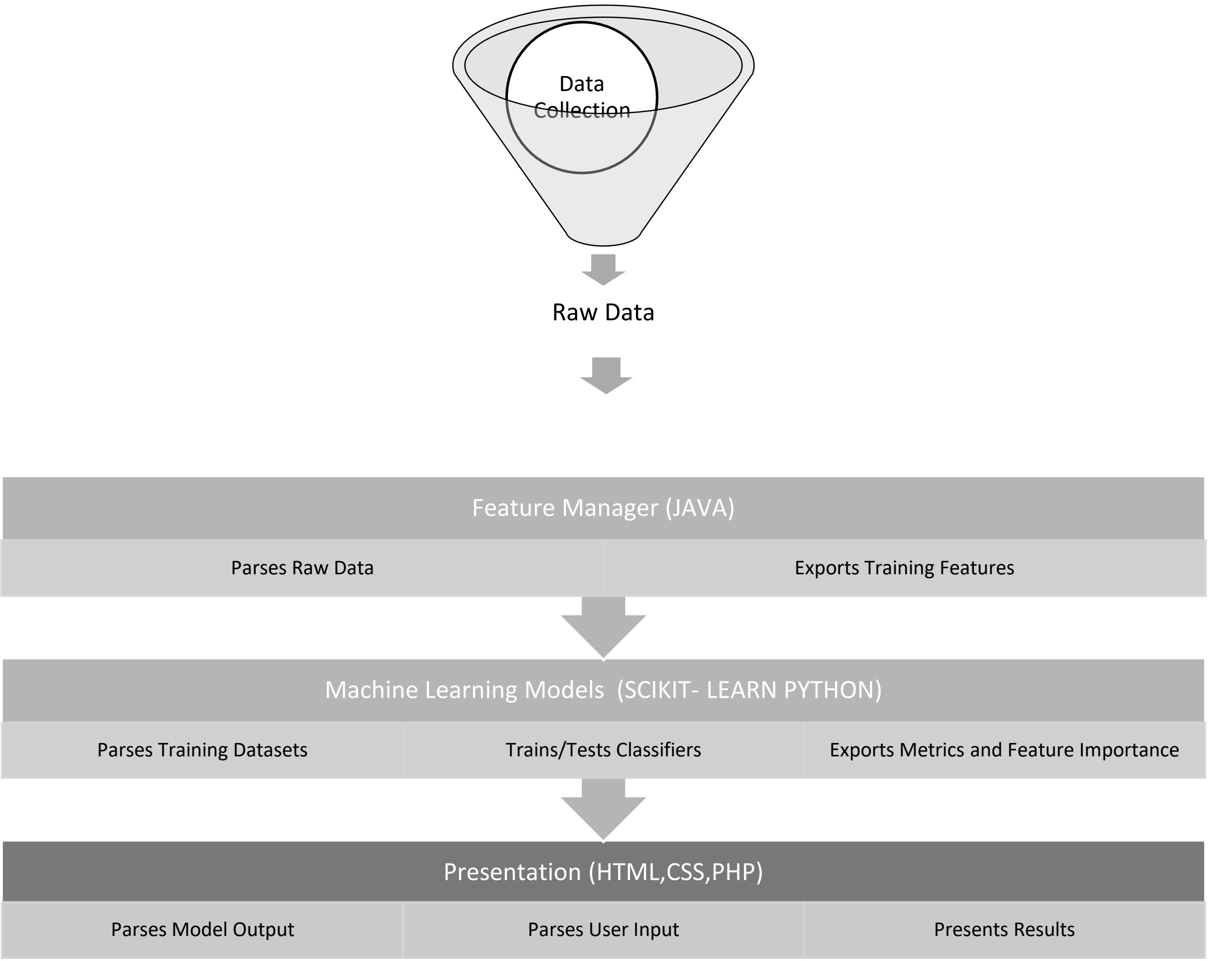
2.2.2 Feature Importance

During classification the training features are assigned multiple weights according to their importance in labeling the videos. Different features are more important in different scenarios. For this reason, the importance of each feature is exported after each classification in the form of a percentage, to examine how much it has contributed in the final labeling of the video. An example of a feature importance extraction is shown in table 12.

Feature Importance	Features	Importance
1	views_acc	12%
2	views_1	11%
3	ageRatioViews_1	9%
4	video_duration	9%
5	comments_1	5%
6	channel_uploads	5%
7	ageRatioLikes_1	4%
8	comments_acc	4%
9	channel_views	4%
10	comments_sentiment_compound	3%
11	comments_sentiment_neu	3%
12	tw_user_followers	3%
13	ageRatioComments_1	2%
14	dislikes_acc	2%
15	channel_subscribers	2%
16	comments_sentiment_pos	2%
17	comments_sentiment_neg	2%
18	category	2%
19	tw_hashtags_1	2%
20	ageRatioDislikes_1	1%
21	dislikes_1	1%
…	…	…

Table 12: Example of exported feature importance of several features

2.3 Implementation Details



The above figure summarizes the implementation details described in this chapter. First we had to collect metadata for a number of YouTube videos from YouTube and Twitter. These data had to go through a labeling and filtering procedure to come up with a dataset containing only raw information regarding each video. Three tools where then developed, a Feature Manager, the Machine Learning Models and a Web application. Each of this tools developed for the needs of this study serves a different purpose which is analyzed to an extent in the subsections of the Implementation Details section.

2.3.1 Data Collection

Despite the fact that data collection was a huge part of this research, it doesn't fit in the main context of this document, which is the analysis and comparison of machine learning algorithms for the needs of the current research. For this reason, only a brief explanation of the data collection methodology followed will be given instead of detailed one. The general idea of how the data collection system works is depicted in Figure 7. In order to be able to collect the information needed, YouTube Data API and Twitter Streaming API where used. The overall procedure was repeated for every video collected.

After receiving a random YouTube video mentioned in a tweet, the monitoring procedure took place. Each video collected was monitored for 15 consecutive days acquiring metadata about its progress on YouTube and Twitter. All the raw data collected where stored in a remote database. In addition to the metadata, a bunch of comments were collected, analyzed and stored for each video.

After collecting the desired amount of 130.000 videos, a filtering took place. More specifically, only information needed for the creation of the training features where kept. Out of the 130.000 videos, only a predefined percentage was used in the training procedure. For example, in strict labeling, the percentage used was 2.5%. This means that the dataset used in the training procedure consisted of the 2.5% most popular videos, the 2.5% most viral videos, 2.5% most recent videos and 2.5% random videos. Some videos appear in multiple of the above categories, meaning that the real amount of unique videos in each category is less than 2.5%.

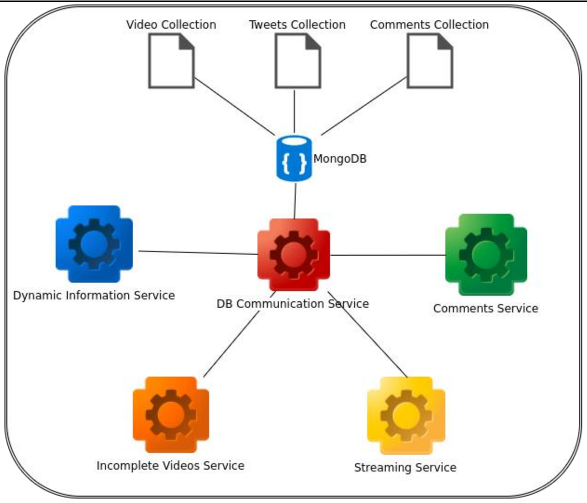


Figure 7: Data collection system

2.3.2 Feature Manager

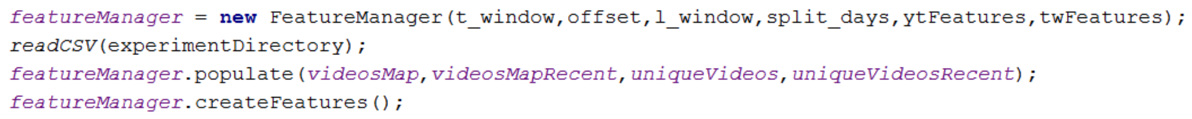
The data collected during the monitoring period consisted of raw information regarding the progress of about 130.000 videos on YouTube and Twitter. In order to be of any value, these data needed to be processed, combined and manipulated to create various training features. For this reason, a "Feature Manager" tool was developed using Java [19]. In this Section, a high level explanation is given on how this tool works.

Input:

Figure 8 contains a code snippet of a FeatureManager object being created with a set of specific parameters. More specifically, the FeatureManager object takes six parameters:

1. **t_window**: Number of training days
2. **offset**: Number of offset days
3. **l_window**: Number of labeling days
4. **split_days**: Defines after how many days since upload the video is classified as old instead of recent
5. **ytFeatures**: Binary value that specifies which YouTube features (from a predefined list) should be exported.
6. **twFeatures**: Binary value that specifies which Twitter features (from a predefined list) should be exported.

The raw data of the videos is parsed to the system through a CSV file and separated to their respective categories. Finally, the feature manager calls a procedure named "createFeatures()" that processes, combines, creates and exports the final training features.

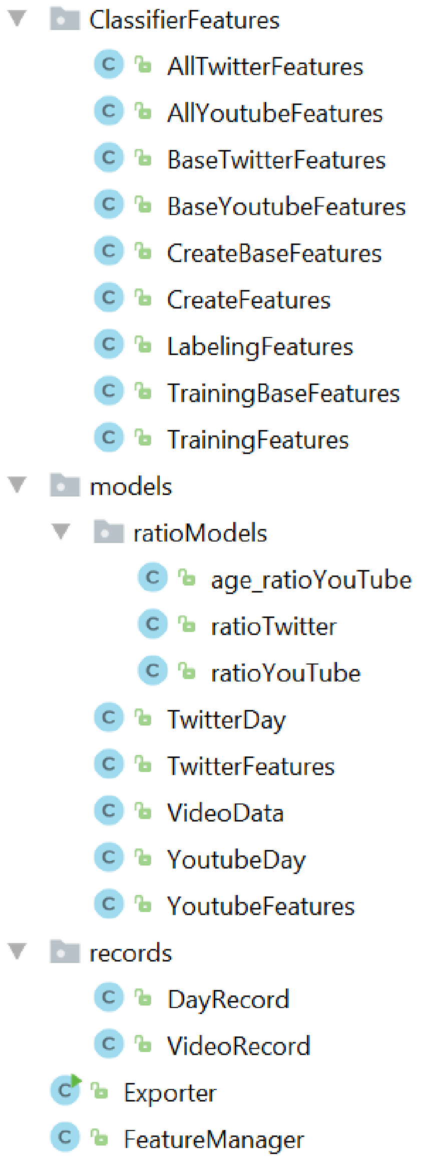


```
featureManager = new FeatureManager(t_window,offset,l_window,split_days,ytFeatures,twFeatures);
readCSV(experimentDirectory);
featureManager.populate(videosMap,videosMapRecent,uniqueVideos,uniqueVideosRecent);
featureManager.createFeatures();
```

Figure 8: Code snippet of Feature Manager input and execution

Processing:

After parsing the raw data, the feature manager goes through several methods of processing and combining the data to produce valuable information, which are converted to a set of training features at the end of the execution. Figure 9 depicts the skeleton of the Feature Manager tool.



```
▼ ClassifierFeatures
      AllTwitterFeatures
      AllYoutubeFeatures
      BaseTwitterFeatures
      BaseYoutubeFeatures
      CreateBaseFeatures
      CreateFeatures
      LabelingFeatures
      TrainingBaseFeatures
      TrainingFeatures
▼ models
   ▼ ratioModels
         age_ratioYouTube
         ratioTwitter
         ratioYouTube
      TwitterDay
      TwitterFeatures
      VideoData
      YoutubeDay
      YoutubeFeatures
▼ records
      DayRecord
      VideoRecord
   Exporter
   FeatureManager
```

The **ClassifierFeatures** package contains utilities that are responsible for creating any kind of features (baseline YouTube, all YouTube, baseline Twitter, all Twitter) and labeling the videos (popular , viral)

The **models** package contains all the objects related to specific features like: daily features, age ratios, YouTube related ratios etc. The **VideoData** object contains all the final information needed for a specific video

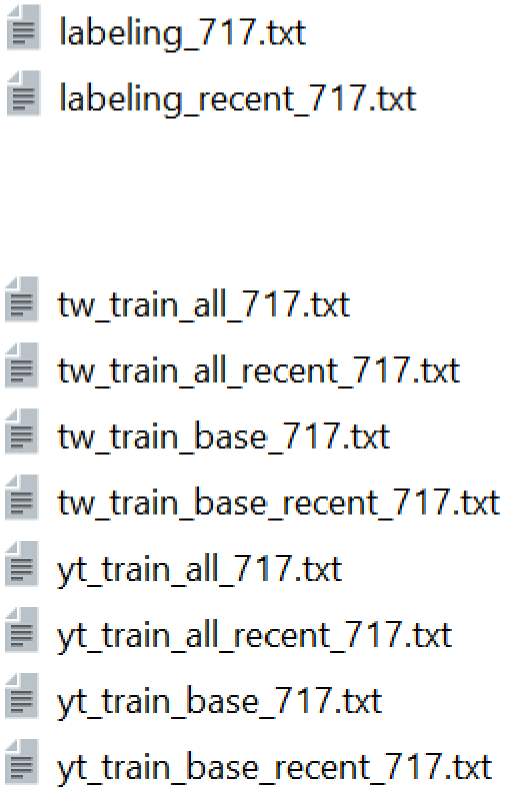
The **records** package is used to parse information into the FeatureManager from the services developed instead of a CSV file

Exporter parses the input data and creates a feature manager to export the final training features. The exporter can be used to automatically export features using different window values.

Figure 9: Feature Manager Skeleton

Output:

After all the information is created regarding a specific video, it is stored in a VideoData object. Finally, all the data from the collection of VideoData objects is exported into ten text files, for each different window combination. These files will be later used to train and test the classifiers for each of the machine learning algorithms presented in this study.



labeling_717.txt
labeling_recent_717.txt

tw_train_all_717.txt
tw_train_all_recent_717.txt
tw_train_base_717.txt
tw_train_base_recent_717.txt
yt_train_all_717.txt
yt_train_all_recent_717.txt
yt_train_base_717.txt
yt_train_base_recent_717.txt

The labeling files contain two rows of data. Each row consists of 1s and 0s. The first row contains the popularity labeling of the videos whereas the second row contains the virality labeling of the videos.

The training files are separated into YouTube training features (4) and Twitter training features (4). The training features are separated into baseline (only basic features) and all (all features). Finally, the training features are separated to those related to older videos and those related to recent videos. Each row of data inside these files, correlates to information about a specific video, whereas each column represents the feature values of that video.

Figure 10: Output files of FeatureManager that contain all the sets of training features

2.3.3 Algorithms Implementation

The machine learning algorithms, methodologies, functions and tools in this research where developed using the Scikit-learn [14] machine learning library.

Scikit – Learn (aka sklearn) is an open source collection of tools for data mining and data analysis. It was developed using the python programming language and it was built on top of other popular open source projects like NumPy, SciPy and matplotlib. Its' usage is fairly permissive since it is covered by a BSD license.

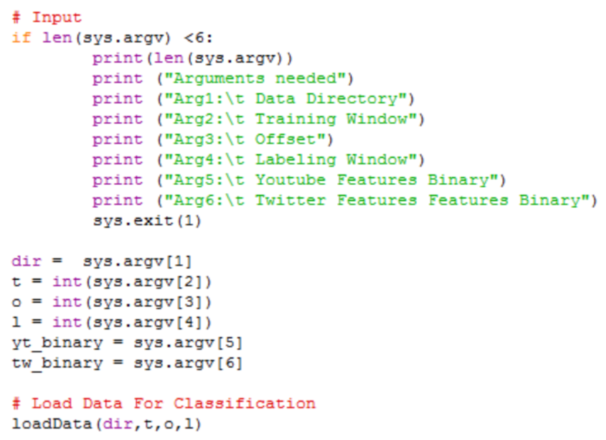
For the purpose of this research sklearn was used to deploy a variety of machine learning classification algorithms with the goal of finding the most suitable classifier for the needs of popularity and virality labeling in the context of the current research. Sklearn also comes with a bunch of useful validation tools that assist in the evaluation of various models.

The Scikit – learn environment doesn't require in-depth knowledge of machine learning algorithms. It is ideal for researchers that need to use the algorithms for classification and model validation without the need of developing the algorithms from scratch. The user has the ability to fine-tune the algorithms in any way needed to fit them to the context of his own project.

Classification Procedure:

In this Section, a high level explanation of how each algorithm was developed and how it works, will be given. The code snippets and examples presented in this Section where taken from the Gradient Boosting Decision Tree algorithm.

Input:



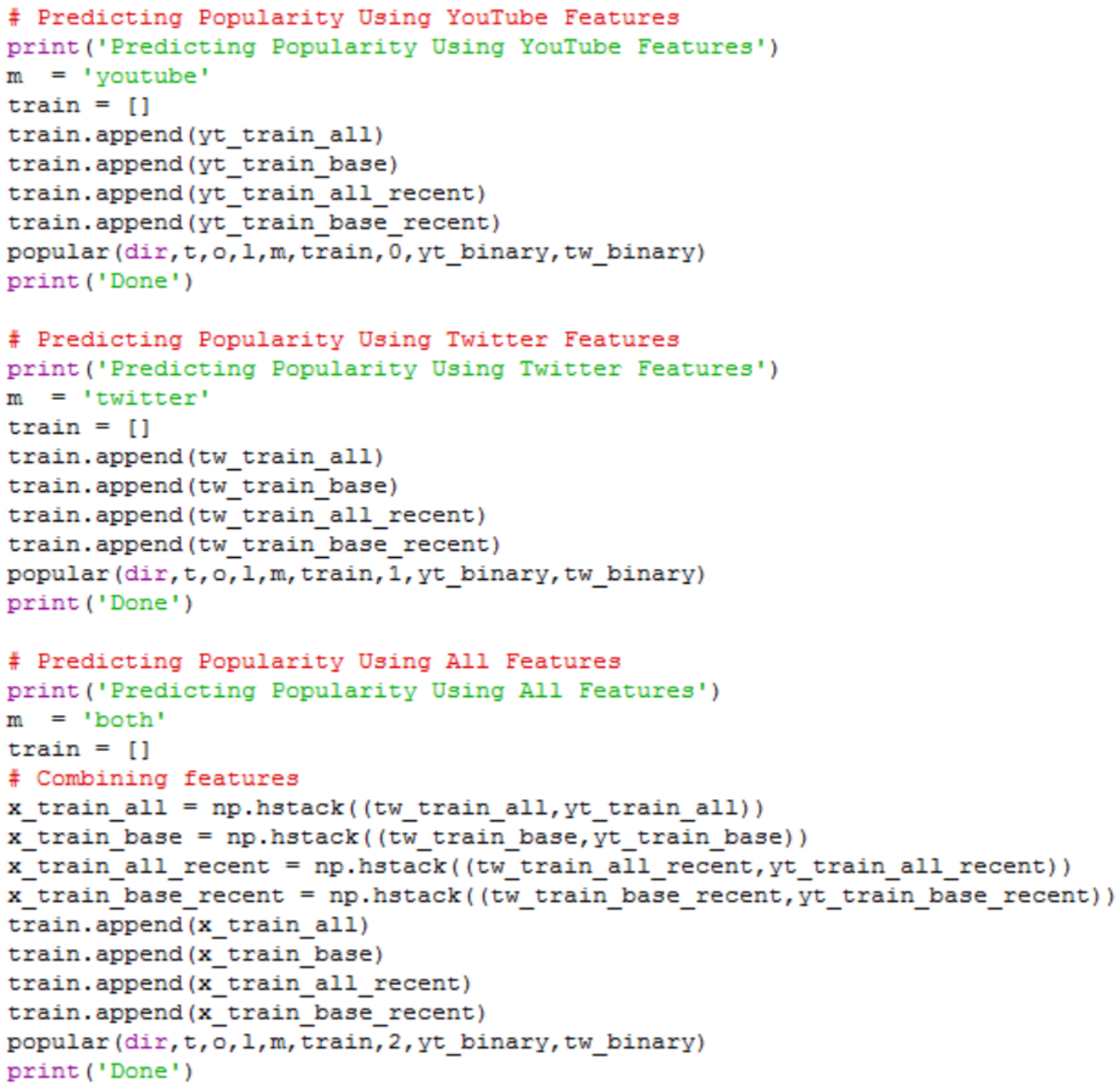
```
# Input
if len(sys.argv) <6:
        print(len(sys.argv))
        print ("Arguments needed")
        print ("Arg1:\t Data Directory")
        print ("Arg2:\t Training Window")
        print ("Arg3:\t Offset")
        print ("Arg4:\t Labeling Window")
        print ("Arg5:\t Youtube Features Binary")
        print ("Arg6:\t Twitter Features Features Binary")
        sys.exit(1)

dir =  sys.argv[1]
t = int(sys.argv[2])
o = int(sys.argv[3])
l = int(sys.argv[4])
yt_binary = sys.argv[5]
tw_binary = sys.argv[6]

# Load Data For Classification
loadData(dir,t,o,l)
```

In order to run the algorithm, all the necessary information need to be parsed first. The information passed to the classifier are: training window, offset, labeling window, YouTube binary, Twitter binary and directory of training features files. The data are then loaded into the classifier using the loadData function and the DataLoader tool developed.

Figure 11: Code snippet of the input data passed into the GBDT.py classifier

Training set preparation:

After parsing all the information needed, the data sets are prepared in the loadData(…) function which uses the DataLoader tool developed for the purposes of this study. Before each classification begins, the necessary data are loaded into a “train” structure [8] to pass into the classification function. Figure 12 depicts three different classification function calls for popularity classification using the respective datasets.

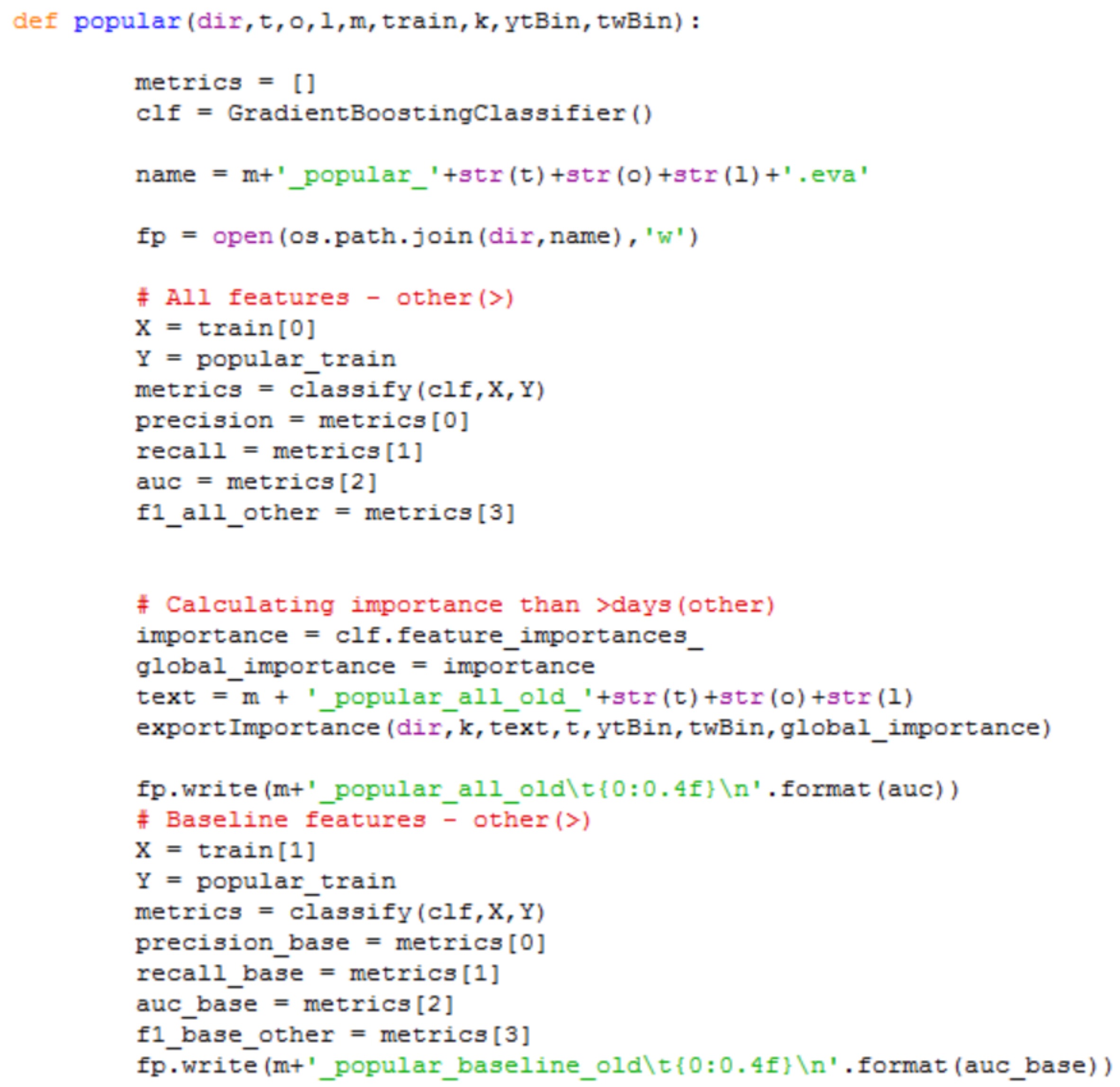


```
# Predicting Popularity Using YouTube Features
print('Predicting Popularity Using YouTube Features')
m  = 'youtube'
train = []
train.append(yt_train_all)
train.append(yt_train_base)
train.append(yt_train_all_recent)
train.append(yt_train_base_recent)
popular(dir,t,o,l,m,train,0,yt_binary,tw_binary)
print('Done')

# Predicting Popularity Using Twitter Features
print('Predicting Popularity Using Twitter Features')
m  = 'twitter'
train = []
train.append(tw_train_all)
train.append(tw_train_base)
train.append(tw_train_all_recent)
train.append(tw_train_base_recent)
popular(dir,t,o,l,m,train,1,yt_binary,tw_binary)
print('Done')

# Predicting Popularity Using All Features
print('Predicting Popularity Using All Features')
m  = 'both'
train = []
# Combining features
x_train_all = np.hstack((tw_train_all,yt_train_all))
x_train_base = np.hstack((tw_train_base,yt_train_base))
x_train_all_recent = np.hstack((tw_train_all_recent,yt_train_all_recent))
x_train_base_recent = np.hstack((tw_train_base_recent,yt_train_base_recent))
train.append(x_train_all)
train.append(x_train_base)
train.append(x_train_all_recent)
train.append(x_train_base_recent)
popular(dir,t,o,l,m,train,2,yt_binary,tw_binary)
print('Done')
```

Figure 12: Code snippet of three popularity classification function calls, using different datasets each time

Popularity Classification and Metrics:

In the popular(…) function, the classifier is defined and the classification procedure begins. The classifier along with the training and labeling data are passed to the classify(…) function in order to run the algorithm and export various metrics regarding the quality of the classification. Inside the popular(…) function the feature importance is calculated as well and exported by calling the exportImportance(…) function. Finally, the F1 scores received from the metrics [] are exported into the respective files and the precision-recall graphs are plotted and exported as well. Figure 13 shows **only a small code snippet** of the popular(…) function to give an abstract idea of how it works.

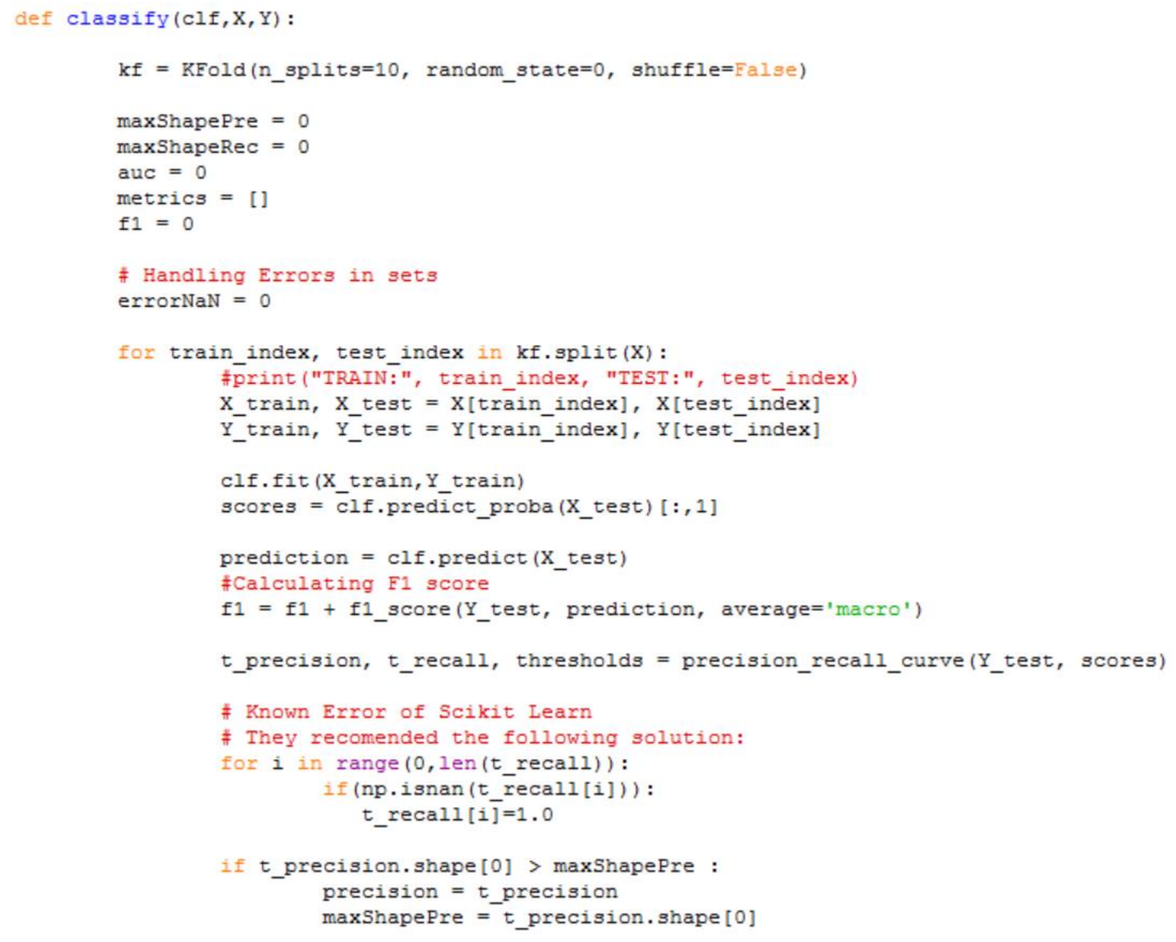


```
def popular(dir,t,o,l,m,train,k,ytBin,twBin):

        metrics = []
        clf = GradientBoostingClassifier()

        name = m+'_popular_'+str(t)+str(o)+str(l)+'.eva'

        fp = open(os.path.join(dir,name),'w')

        # All features - other(>)
        X = train[0]
        Y = popular_train
        metrics = classify(clf,X,Y)
        precision = metrics[0]
        recall = metrics[1]
        auc = metrics[2]
        f1_all_other = metrics[3]


        # Calculating importance than >days(other)
        importance = clf.feature_importances_
        global_importance = importance
        text = m + '_popular_all_old_'+str(t)+str(o)+str(l)
        exportImportance(dir,k,text,t,ytBin,twBin,global_importance)

        fp.write(m+'_popular_all_old\t{0:0.4f}\n'.format(auc))
        # Baseline features - other(>)
        X = train[1]
        Y = popular_train
        metrics = classify(clf,X,Y)
        precision_base = metrics[0]
        recall_base = metrics[1]
        auc_base = metrics[2]
        f1_base_other = metrics[3]
        fp.write(m+'_popular_baseline_old\t{0:0.4f}\n'.format(auc_base))
```

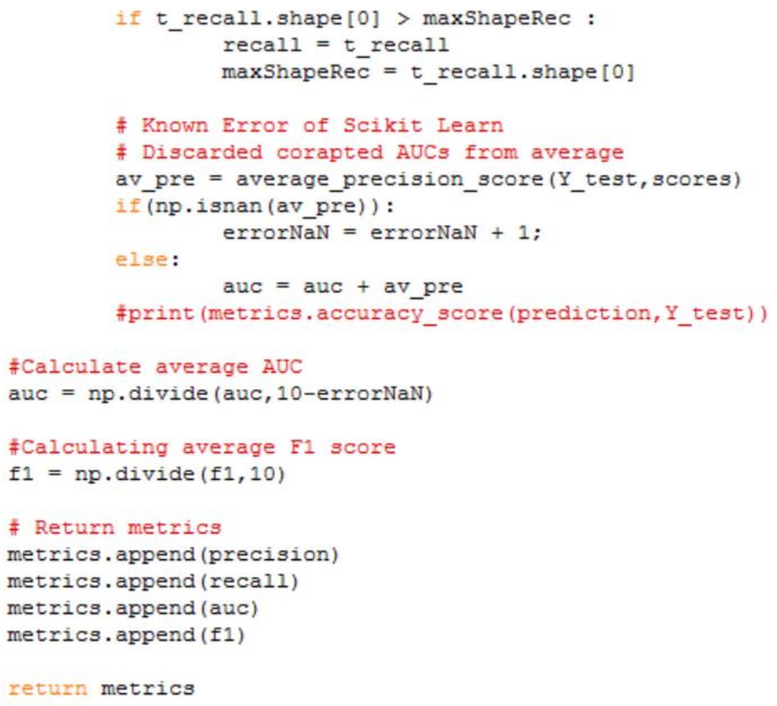
Figure 13: Code snippet showing a small part of the popular(…) function.

Classification Function

The classify(…) function depicted in Figure 14 contains the main functionality of the classification algorithm. The parameters of this function are the algorithm, the training set and the labeling set. It works for almost all types of algorithms used in this research, with minor changes. The use of the KFold validation methodology increases the accuracy of the classification. Within the loop, the training and labeling data are fitted to the classifier and then the algorithm makes a prediction. The precision and recall values are exported [17] from that prediction in order to calculate the F1 scores and the AUC values and later plot the precision-recall graph. All the metrics calculated from the classification are stored into the metrics[] array and returned to the main program.



```
def classify(clf,X,Y):

        kf = KFold(n_splits=10, random_state=0, shuffle=False)

        maxShapePre = 0
        maxShapeRec = 0
        auc = 0
        metrics = []
        f1 = 0

        # Handling Errors in sets
        errorNaN = 0

        for train_index, test_index in kf.split(X):
                #print("TRAIN:", train_index, "TEST:", test_index)
                X_train, X_test = X[train_index], X[test_index]
                Y_train, Y_test = Y[train_index], Y[test_index]

                clf.fit(X_train,Y_train)
                scores = clf.predict_proba(X_test)[:,1]

                prediction = clf.predict(X_test)
                #Calculating F1 score
                f1 = f1 + f1_score(Y_test, prediction, average='macro')

                t_precision, t_recall, thresholds = precision_recall_curve(Y_test, scores)

                # Known Error of Scikit Learn
                # They recomended the following solution:
                for i in range(0,len(t_recall)):
                        if(np.isnan(t_recall[i])):
                           t_recall[i]=1.0

                if t_precision.shape[0] > maxShapePre :
                        precision = t_precision
                        maxShapePre = t_precision.shape[0]
```

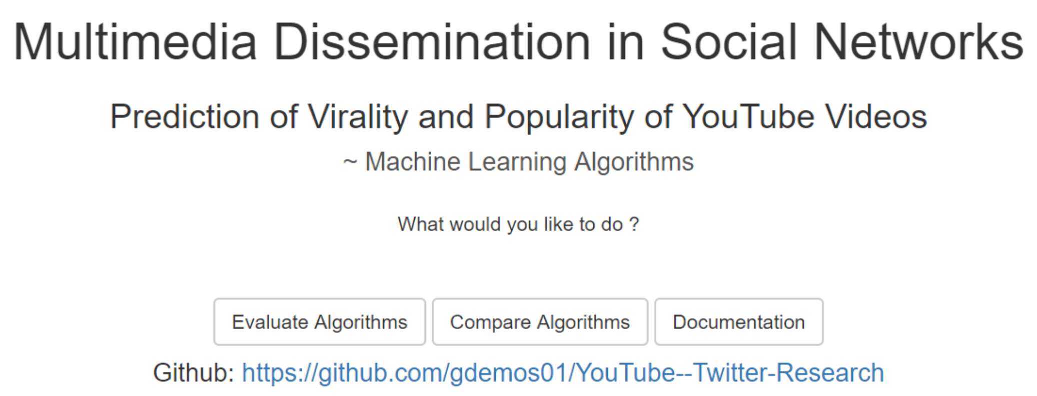


```
        if t_recall.shape[0] > maxShapeRec :
                recall = t_recall
                maxShapeRec = t_recall.shape[0]

        # Known Error of Scikit Learn
        # Discarded corapted AUCs from average
        av_pre = average_precision_score(Y_test,scores)
        if(np.isnan(av_pre)):
                errorNaN = errorNaN + 1;
        else:
                auc = auc + av_pre
        #print(metrics.accuracy_score(prediction,Y_test))

#Calculate average AUC
auc = np.divide(auc,10-errorNaN)

#Calculating average F1 score
f1 = np.divide(f1,10)

# Return metrics
metrics.append(precision)
metrics.append(recall)
metrics.append(auc)
metrics.append(f1)

return metrics
```

Figure 14: Code snippet of the classify(…) function used in all the algorithms with minor changes

2.3.4 Presentation

In order to assist with the evaluation and the comparison of the machine learning algorithms, a web application was created. This tool was developed using HTML, CSS and PHP [22]. As shown in Figure 15, this web application allows the user to evaluate or compare the algorithms and read the corresponding documentation. Also, a link to the entire code if this research is provided.



Multimedia Dissemination in Social Networks

Prediction of Virality and Popularity of YouTube Videos

~ Machine Learning Algorithms

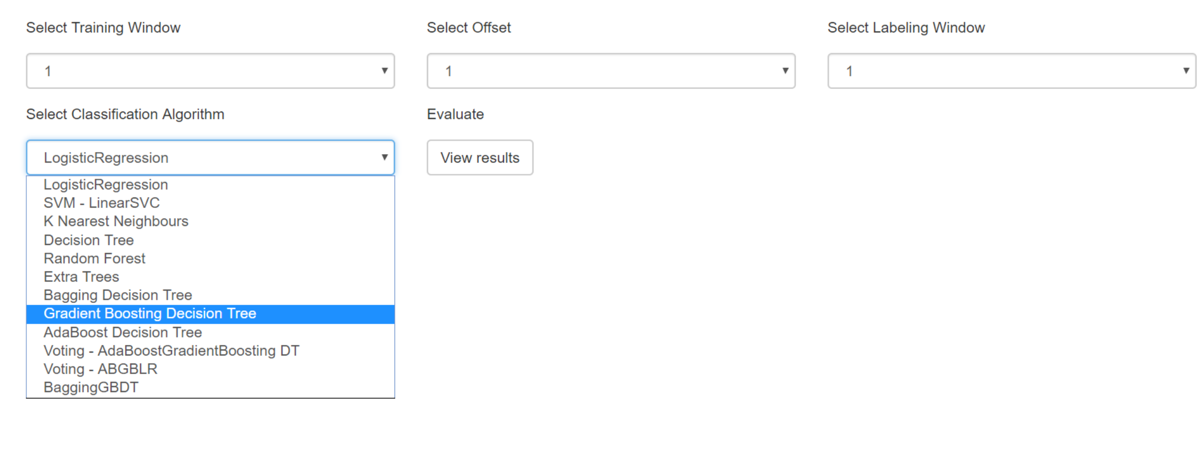
What would you like to do ?

Github: https://github.com/gdemos01/YouTube--Twitter-Research

Figure 15: Web application home page for evaluation and comparison of machine learning algorithms

Evaluation of algorithms:

The user can choose an algorithm from the drop down list and enter the windows needed in his case, as shown in Figure 16, and then click “view results” to see the outcome of the classification as shown in Figure 17.



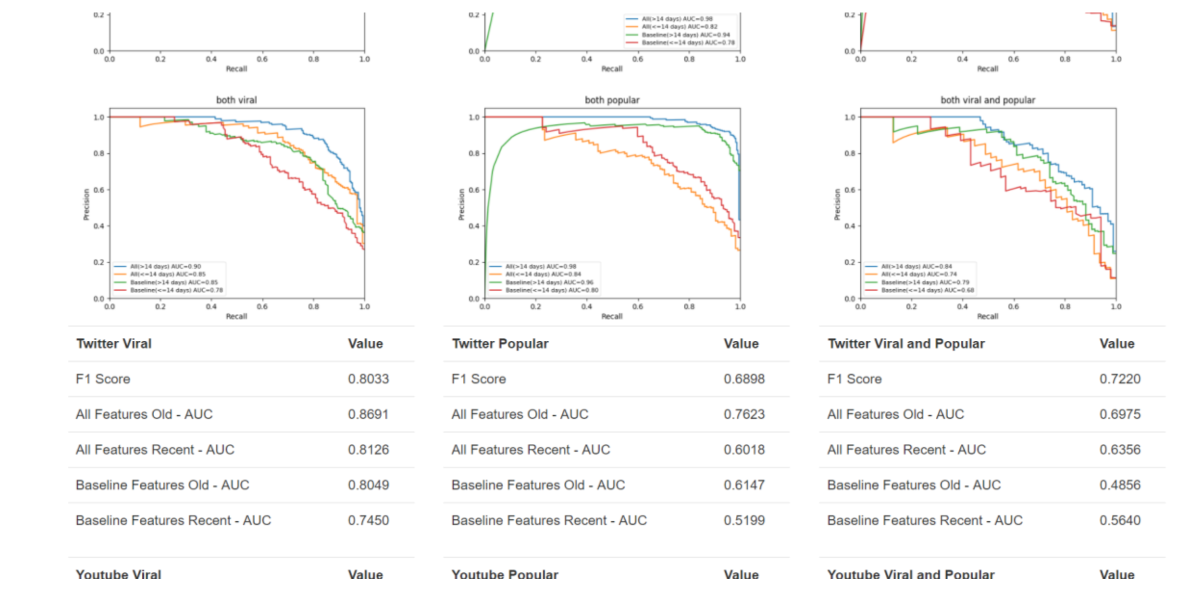
Select Training Window: 1

Select Offset: 1

Select Labeling Window: 1

Select Classification Algorithm: LogisticRegression

Figure 16: Web application snapshot of the algorithm evaluation window



Twitter Viral	Value
F1 Score	0.8033
All Features Old - AUC	0.8691
All Features Recent - AUC	0.8126
Baseline Features Old - AUC	0.8049
Baseline Features Recent - AUC	0.7450

Twitter Popular	Value
F1 Score	0.6898
All Features Old - AUC	0.7623
All Features Recent - AUC	0.6018
Baseline Features Old - AUC	0.6147
Baseline Features Recent - AUC	0.5199

Twitter Viral and Popular	Value
F1 Score	0.7220
All Features Old - AUC	0.6975
All Features Recent - AUC	0.6356
Baseline Features Old - AUC	0.4856
Baseline Features Recent - AUC	0.5640

Youtube Viral	Value

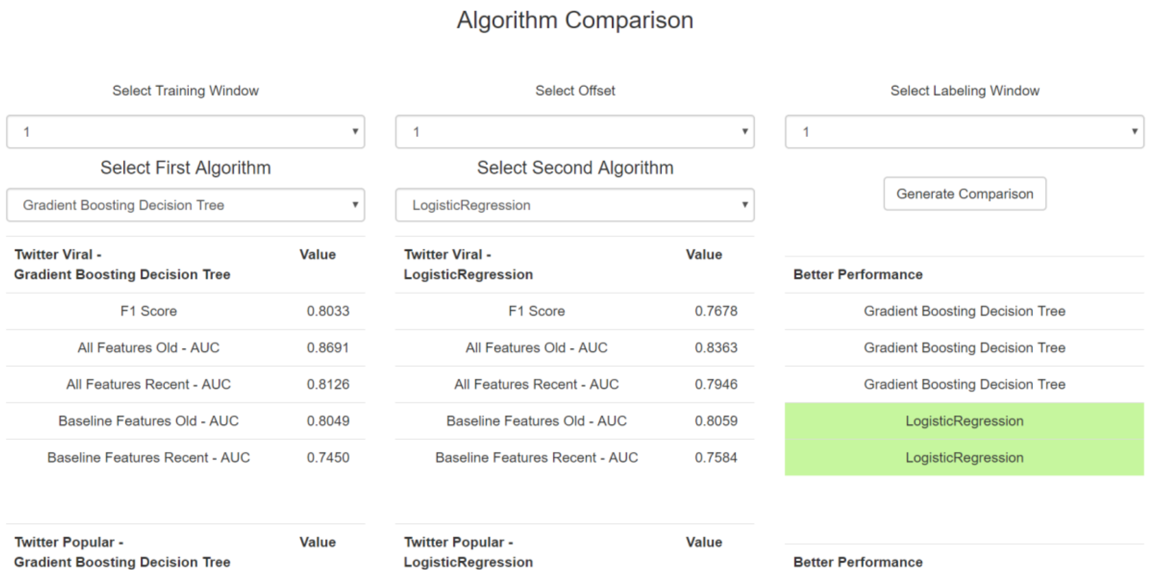
Youtube Popular	Value

Youtube Viral and Popular	Value

Figure 17: Web application snapshot of the classification results

Comparison of algorithms:

The comparison functionality of this web application has been very useful since the comparison of several algorithms has led to the extraction of valuable and interesting information. The user can select two algorithms that he wishes to compare. After clicking the "Generate Comparison" button the metrics of the algorithms are compared. The final column of this window contains the algorithm that has better performance for each metric in comparison. Figure 18 shows an example of a comparison between GBDT and Logistic Regression algorithms.



Algorithm Comparison

Select Training Window: 1

Select Offset: 1

Select Labeling Window: 1

Select First Algorithm: Gradient Boosting Decision Tree

Select Second Algorithm: LogisticRegression

Twitter Viral - Gradient Boosting Decision Tree	Value	Twitter Viral - LogisticRegression	Value	Better Performance
F1 Score	0.8033	F1 Score	0.7678	Gradient Boosting Decision Tree
All Features Old - AUC	0.8691	All Features Old - AUC	0.8363	Gradient Boosting Decision Tree
All Features Recent - AUC	0.8126	All Features Recent - AUC	0.7946	Gradient Boosting Decision Tree
Baseline Features Old - AUC	0.8049	Baseline Features Old - AUC	0.8059	LogisticRegression
Baseline Features Recent - AUC	0.7450	Baseline Features Recent - AUC	0.7584	LogisticRegression

Twitter Popular - Gradient Boosting Decision Tree	Value	Twitter Popular - LogisticRegression	Value	Better Performance

Figure 18: Web application snapshot of comparison between GBDT and Logistic Regression algorithms

2.4 Model Evaluation

Confusion Matrix

The confusion matrix [12] is a square matrix that represents the performance of a machine learning algorithm. It consists of true positive, true negative, false positive and false negative predictions of a classifier as shown in table 13

True positives (TP): Labeled 1 and their true value is 1.
True negatives (TN): Labeled 0 and their true value is 0.
False positives (FP): Labeled 1 but their true value is 0.
False negatives (FN): Labeled 0 but their true value is 1.

Confusion Matrix		Predicted Class	
		0	1
True Class	0	TN	FP
	1	FN	TP

Table 13: Confusion Matrix

Precision [24] is defined as the number of true positives (TP) over the number of true positives plus false positives (FP). It represents the frequency of correct predictions when a positive value (1) is predicted.

$$P = \frac{TP}{TP + FP}$$

Recall [24] is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (FN). It represents the frequency of correct predictions when the actual value is positive (1).

$$R = \frac{TP}{TP + FN}$$

In order to evaluate the performance of the classification algorithms a **precision-recall graph** is formed. The higher the AUC in the precision-recall graph the more accurate the classification is. An example of a precision-recall graph is shown at Figure 19, which depicts the performance of four different classification algorithms.

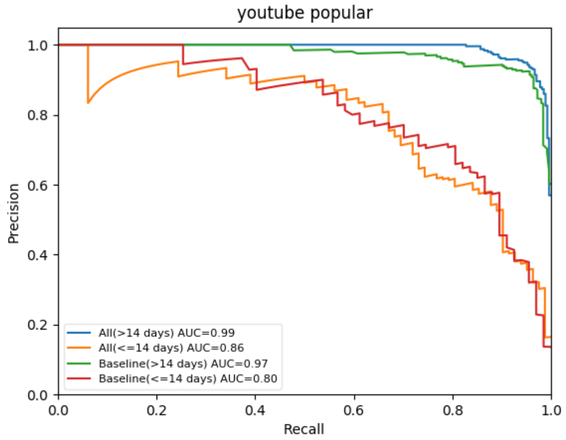


Figure 19: Example of a precision-recall graph

F1 score [24] is an additional metric that is related to precision and recall and it is used to evaluate the performance of the classification algorithms. F1 Score is defined as the harmonic average of precision and recall. A high F1 value shows high classification accuracy.

$$F1 = 2\frac{PXR}{P+R}$$

K-fold cross-validation

K-fold cross-validation [9] randomly splits the training dataset into k folds without replacement. K-1 folds are used for training the classifier and one fold is used for testing the classification. This procedure is repeated k times to obtain k different classifications on the entire dataset. The final classification is a result of the average performance of all the estimators.

For the purpose of this research a 10-fold cross-validation is used as shown in Figure 20. The usage of this technique improves the quality of the final classification and reduces the variance of the individual classifications coming from each estimator.

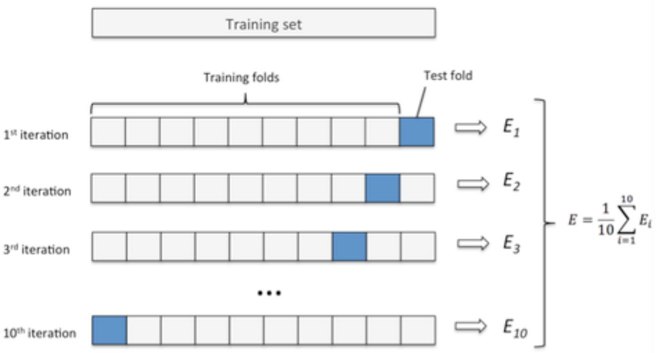


Figure 20: 10-fold cross-validation explanation

Chapter 3

Evaluation and Analysis

3.1 Evaluation Methodology

Figure 21 depicts the different kinds of datasets that were used to train the classifiers and later evaluate the performance of the respective classification algorithm. The labeling percentage was separated into two categories: **Strict and soft labeling**. Using strict labeling means that only 2.5% of the most Popular and Viral videos are kept from the entire dataset. Two more groups of videos are then added, Recent and Random, again each group containing at most 2.5% of the entire dataset. In soft labeling, the percentage of each group is increased to 5% meaning that the new dataset contains more videos but with a softer definition of popularity and virality. It should be noted that some of the videos can appear in multiple categories.

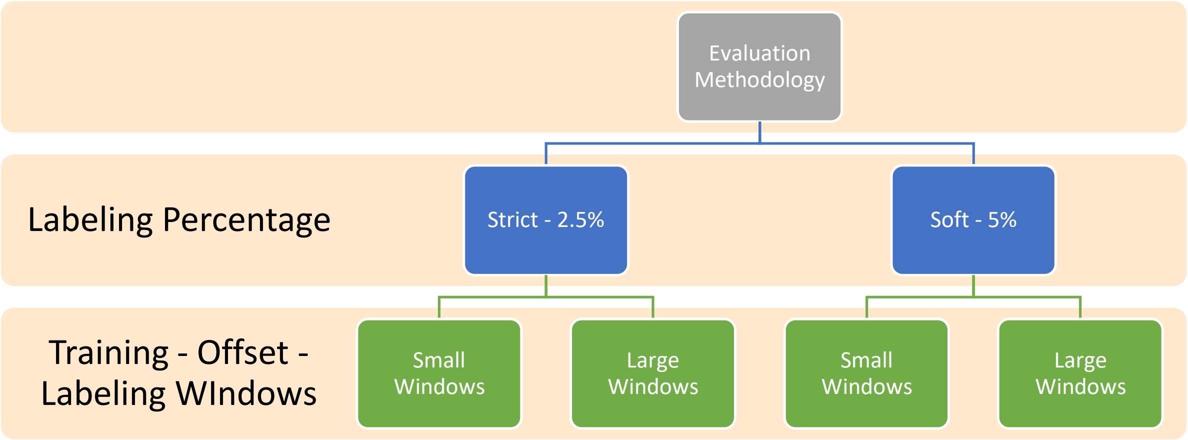


Figure 21: Datasets used in the evaluation methodology

In order to be able to examine an algorithm's window sensitivity, datasets with small and large windows where produced as shown in table 14.

	Training	Offset	Labeling
Small Windows	1	1	1
Large Windows	7	1	7

Table 14: Training, offset and labeling windows used to examine window sensitivity of algorithms

The final amount of videos included in the four training datasets are shown in table 15.

Total Videos = 130.000	**Popular + Viral + Recent + Random**	
	Small Windows	**Large Windows**
Strict Labeling – 2.5%	11039	11344
Soft Labeling – 5 %	21160	21761

Table 15: The final amount of videos included in the four training datasets

Evaluation Baselines

In order to decide if a classification algorithm is suitable for the purpose of the current research, three important baselines where established:

1. Average F1 scores of the classifications should be 0.75 at minimum.
2. Average F1 scores shouldn’t increase more than 8% when windows are changed from small to large. In other words, algorithms should have low window sensitivity.
3. All the classifications should have an AUC larger than 0.80.

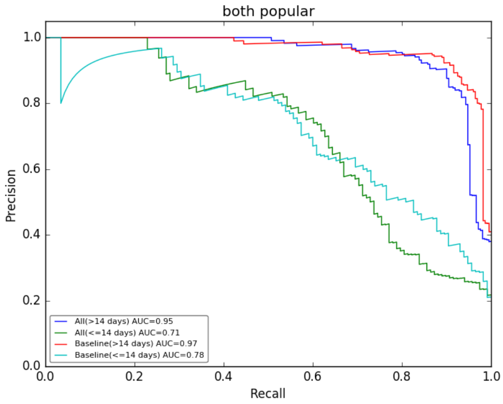
In addition to the baseline test, we evaluate the running time of each algorithm in the prediction models.

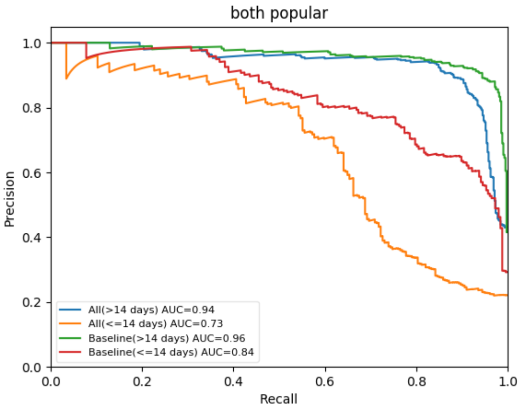
Stand-alone algorithms where evaluated at first in order to decide upon which algorithm would be chosen for further development and performance improvement with the goal of producing more accurate classifications.

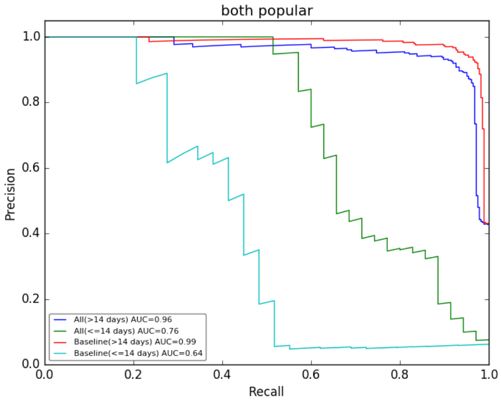
3.2. Algorithm Evaluation and Analysis

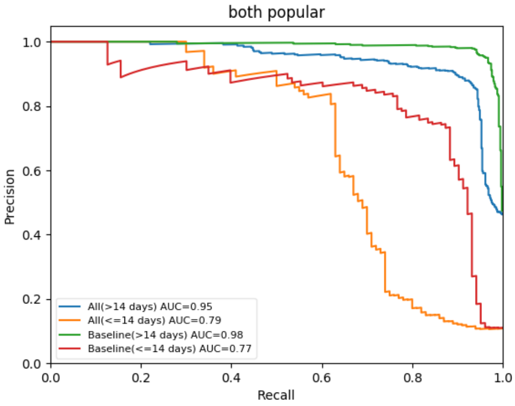
This Section contains the evaluation and analysis of various machine learning algorithms. All the algorithms are tested using the evaluation baselines to determine their strengths and weaknesses. As the evaluation of the algorithms progresses, different methodologies and algorithms are used in order to exploit more information regarding binary classifications in the context of this research. The precision-recall graphs presented in the evaluations correspond only to popularity predictions using both YouTube and Twitter features.

3.2.1 Logistic Regression Evaluation

<table>
<tr><th></th><th>2.5%</th><th>5%</th></tr>
<tr><td>111</td><td>


</td><td>


</td></tr>
<tr><td>717</td><td>


</td><td>


</td></tr>
</table>

Figure 22: Logistic Regression algorithm precision-recall graphs

Precision-Recall graph explanation:

Looking at the precision-recall graphs in Figure 22, it seems that the Logistic Regression algorithm produces extremely accurate classifications for videos that are older but mediocre classifications for videos that are recent. In addition, this algorithm seems to be ideal for classifications containing only baseline features since it reaches AUC values up 0.99. Finally, increasing the amount of videos in the dataset, by using soft labeling, hasn't improved the accuracy of the classifications.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7894	0.5078	0.5870
YouTube Features	0.4695	0.5563	0.6068
All Features	0.6769	0.6812	0.6478
Mean F1 Score	0.6136		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7962	0.5795	0.6163
YouTube Features	0.5825	0.7601	0.6651
All Features	0.6755	0.8015	0.7104
Mean F1 Score	0.6875		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7678	0.5304	0.5925
YouTube Features	0.5198	0.5574	0.6635
All Features	0.7048	0.6048	0.6773
Mean F1 Score	0.6243		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7874	0.5805	0.6274
YouTube Features	0.5838	0.7583	0.6453
All Features	0.7098	0.8227	0.7002
Mean F1 Score	0.6906		

Figure 23: F1 scores for Logistic Regression algorithm evaluation

F1 Scores explanation:

Looking at the F1 scores of all the classifications using Logistic Regression, it is clear that the algorithm underperforms mostly in cross-platform evaluations. The F1 scores exposed the low accuracy classifications of logistic regression which weren't visible from the precision-recall graphs.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Failed
All the datasets in strict and soft labeling fail to achieve a minimum average F1 score of 0.75 for any window.

(2) Window Sensitivity (<=8%) – Failed
The average F1 score in strict datasets was increased by 12% and in soft datasets it was increased by 11% meaning that the logistic regression algorithm is very sensitive to window changes.

(3) High AUC (>=0.80) – Failed
There is at least one classification with low AUC making the algorithm unsuitable for the current problem. More specifically, the algorithm seems to perform poorly when it comes to classifications of recent videos, which are very important for the purpose of this research.

Running Time (111, 2.5%): 17.37 seconds. The logistic regression is an extremely fast classification algorithm.

Additional Comments:

Due to the probabilistic nature of the logistic regression algorithm, it was expected that a high accuracy classification would be produced. Despite the fact that this algorithm performs fairly well in binary classification problems, it failed to fit the minimum needs of this research.

3.2.2 K Nearest Neighbors Evaluation

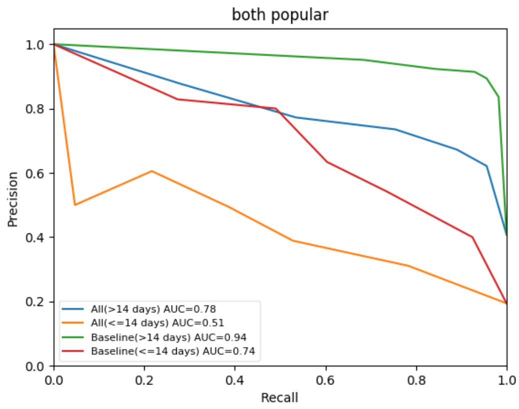
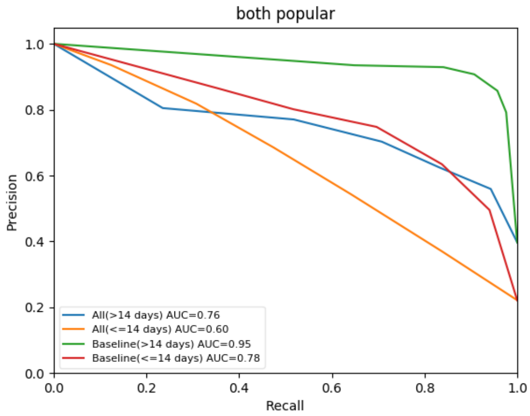
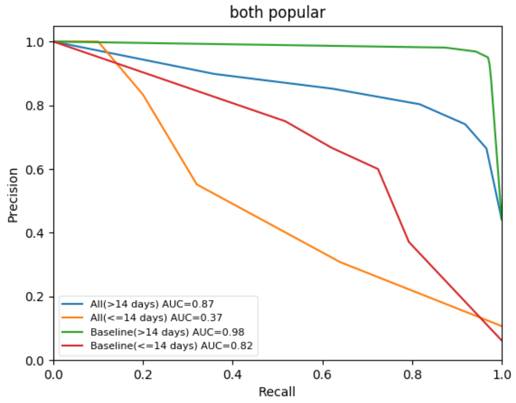
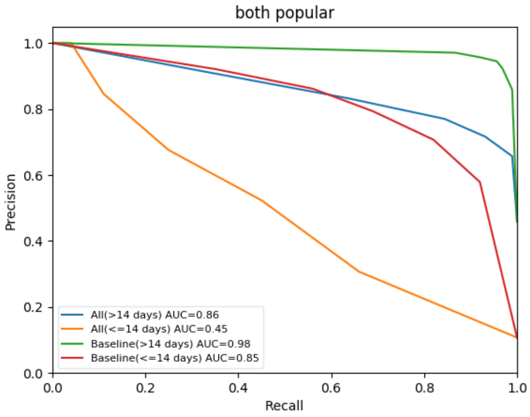
	2.5%	5%
111		
717		

Figure 24: K Nearest Neighbors algorithm precision-recall graphs

Precision-Recall graph explanation:

After looking at the precision-recall graphs of Figure 24, it can be deducted that KNN algorithm fails to produce high accuracy classifications for most of the cases. Curiously enough, KNN performs much better when only using baseline features. Increasing the amount of videos in the dataset, by using soft labeling, doesn’t have any effect on the accuracy of the classifications.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7126	0.6225	0.6231
YouTube Features	0.5553	0.7842	0.6371
All Features	0.5903	0.7843	0.6373
Mean F1 Score	0.6607		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8008	0.6691	0.7177
YouTube Features	0.6008	0.8127	0.6829
All Features	0.6394	0.8127	0.6830
Mean F1 Score	0.7132		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7208	0.6317	0.6399
YouTube Features	0.5737	0.7951	0.6658
All Features	0.6046	0.7950	0.6659
Mean F1 Score	0.6769		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8082	0.6631	0.7329
YouTube Features	0.6058	0.8226	0.6972
All Features	0.6477	0.8201	0.6983
Mean F1 Score	0.7218		

Figure 25: F1 scores for K Nearest Neighbors algorithm evaluation

F1 Scores explanation:

The F1 scores produced are generally low for most of the classifications. It is very interesting to notice that the amount of features doesn’t play any role at all in the quality of the classification using the KNN algorithm. This can be deducted by looking at the “Viral and Popular” f1 scores that hardly change when different amounts of features are used.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Failed

All the datasets in strict and soft labeling fail to achieve a minimum average F1 score of 0.75 for any window.

(2) Window Sensitivity (<=8%) – Succeeded

The average F1 score in strict datasets was increased by 8% and in soft datasets it was increased by 6.5% meaning that the "K Nearest Neighbors" algorithm has low window sensitivity.

(3) High AUC (>=0.80) – Failed

Most of the classifications fail to achieve the minimum AUC requirements of this research making the algorithm extremely unsuitable for the purpose of the current problem.

Running Time (111, 2.5%): 11.29 seconds. The K Nearest Neighbors is an extremely fast classification algorithm.

Additional Comments:

The "K Nearest Neighbors" performance was as expected. The algorithm doesn't seem to be highly dependent on window changes. The reason for this behavior is that this algorithm relies more on the number of neighbors (k) that it takes into account, rather than the number of training and labeling days or the size of the dataset.

3.2.3 SVM – Linear SVC Evaluation

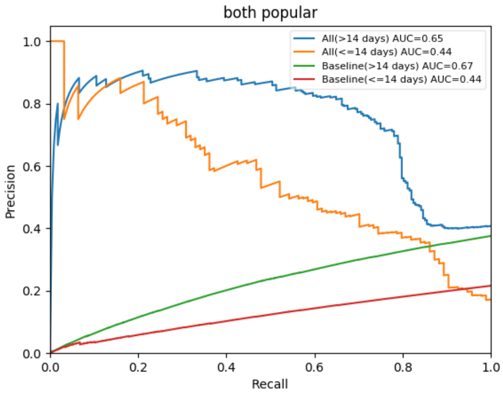
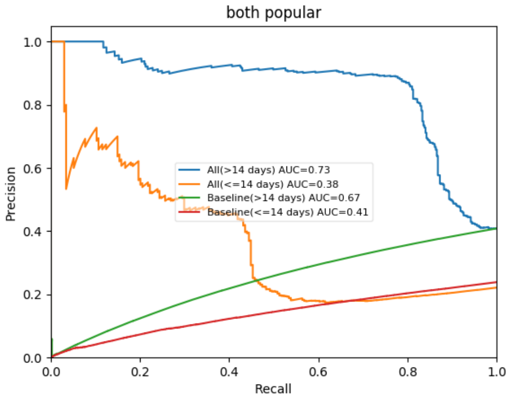
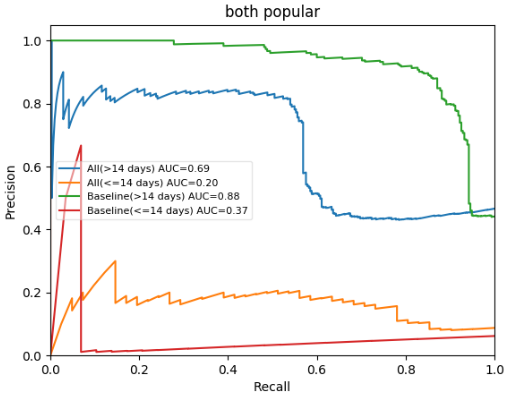
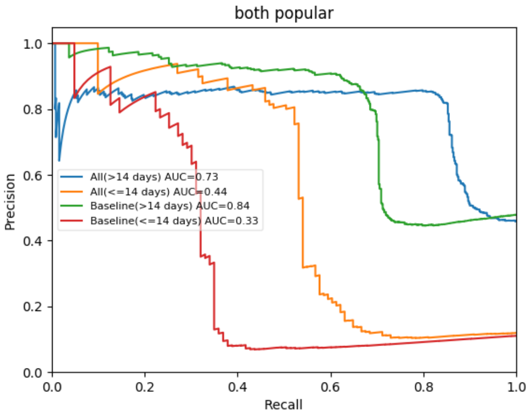
<table>
<tr><th></th><th>2.5%</th><th>5%</th></tr>
<tr><td>111</td><td>
</td><td>
</td></tr>
<tr><td>717</td><td>
</td><td>
</td></tr>
</table>

Figure 26: SVM – Linear SVC algorithm precision-recall graphs

Precision-Recall graph explanation:

It is obvious from Figure 26, that the SVM – Linear SVC algorithm produces “chaotic” classifications when attempting to predict popularity of the videos. It appears to produce a good classification only for older videos using baseline features. Other than that, this algorithm doesn’t seem to fit to the problem of this research.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7344	0.5404	0.5932
YouTube Features	0.4454	0.6053	0.5026
All Features	0.5816	0.6226	0.5231
Mean F1 Score	0.5721		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7611	0.5950	0.6446
YouTube Features	0.4609	0.6311	0.5446
All Features	0.5680	0.6630	0.5571
Mean F1 Score	0.6028		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7303	0.5660	0.6077
YouTube Features	0.4553	0.6488	0.5744
All Features	0.5749	0.6136	0.5101
Mean F1 Score	0.5868		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7580	0.5860	0.6557
YouTube Features	0.4896	0.6508	0.5893
All Features	0.5376	0.7072	0.5841
Mean F1 Score	0.6176		

Figure 27: F1 scores for SVM – Linear SVC algorithm evaluation

F1 Scores explanation:

Looking at the F1 scores in Figure 27, it can be denoted that SVM – Linear SVC produces low accuracy classifications for all the datasets used. It doesn’t seem to be affected by the amount of features or the total number of videos in the dataset.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Failed

All the datasets in strict and soft labeling fail to achieve a minimum average F1 score of 0.75 for any window. Moreover the algorithm appears to perform extremely poorly.

(2) Window Sensitivity (<=8%) – Succeeded

The average F1 score in strict datasets was increased by 5.4% and in soft datasets it was increased by 5.2% meaning that the SVM algorithm has extremely low window sensitivity. This was expected due to the consistency of the algorithm.

(3) High AUC (>=0.80) – Failed

There's no need for any deep analysis of the data, since it's obvious from the precession-recall graphs that SVM performs extremely poorly when it comes to classifying videos for the purpose of this research.

Running Time (111, 2.5%): 138.91 seconds. The SVM – Linear SVC algorithm is takes has a moderate execution time cost.

Additional Comments:

Despite the fact that SVM is one of the most popular classification algorithms for binary problems, it doesn't seem to be suitable for the purpose of this research. Various SVM algorithms where tested with different kernels but no one managed to exceed the minimum needs for the solution of this problem.

3.2.4 Decision Tree Evaluation

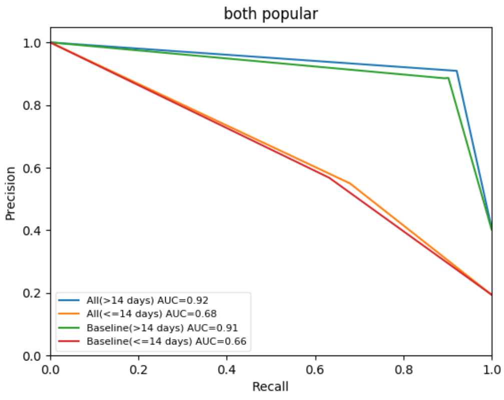
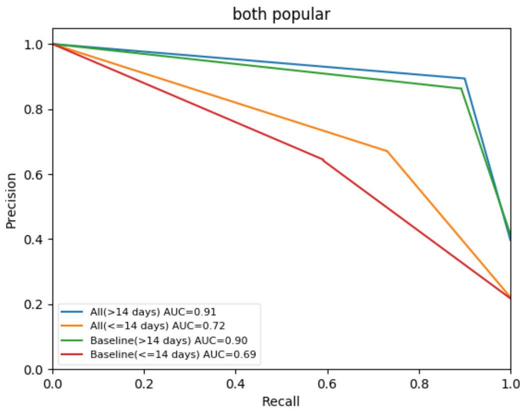
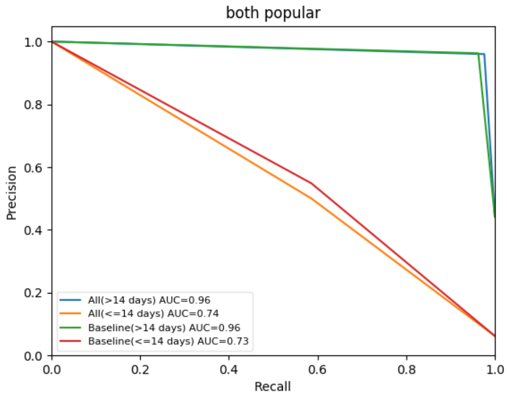
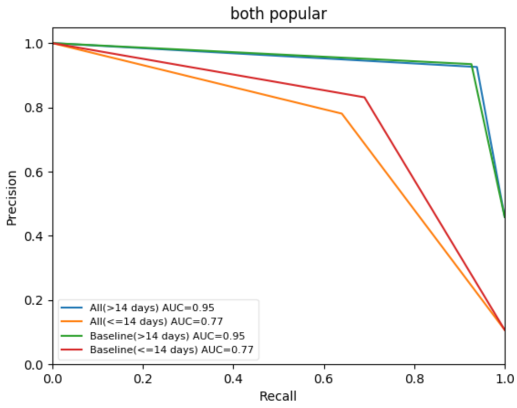
	2.5%	5%
111		
717		

Figure 28: Decision Tree algorithm precision-recall graphs

Precision-Recall graph explanation:

Figure 28 shows that the Decision Tree algorithm produces high accuracy classifications for older videos using any amount of features. In addition, it produces decent classifications for the rest of the cases as well. The total number slightly improves the accuracy of the classifications. What is important to note is that the Decision Tree algorithm produces much higher AUC values, therefore better classifications for larger windows.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7639	0.6498	0.6806
YouTube Features	0.5965	0.8308	0.6941
All Features	0.7671	0.8368	0.7673
Mean F1 Score	0.7319		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8097	0.6715	0.7546
YouTube Features	0.6486	0.8981	0.7573
All Features	0.8319	0.9018	0.8567
Mean F1 Score	0.7922		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7657	0.6655	0.6925
YouTube Features	0.5988	0.8420	0.7129
All Features	0.7406	0.8430	0.7720
Mean F1 Score	0.737		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8123	0.6717	0.7596
YouTube Features	0.6422	0.9033	0.7544
All Features	0.8219	0.9011	0.8473
Mean F1 Score	0.7904		

Figure 29: F1 scores for Decision Tree algorithm evaluation

F1 Scores explanation:

The F1 scores of the Decision Tree algorithm classifications shown in Figure 29, correlate with the AUC values in the precision-recall graphs. This is a reassurance that the Decision Tree algorithm produces high accuracy classifications for almost all the cases. When it comes to cross-platform predictions, the DT algorithm has lower F1 scores, therefore produces lower quality classifications. Finally, it appears that this algorithm performs better while using larger amounts of features. This behavior was expected due to the tree structure of the algorithm.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Partly Failed

Barely fails at small windows but succeeds at large windows.

(2) Window Sensitivity (<=8%) – Partly Failed

The average F1 score in strict datasets was increased by 8.2% and in soft datasets it was increased by 7.2% meaning that the Decision Tree algorithm is relatively sensitive to window changes but is very close to succeeding the minimum baseline tests.

(3) High AUC (>=0.80) – Partly Failed

The decision tree algorithm produces AUC slightly below the baseline causing the algorithm to fail the tests. Although it produces high accuracy classifications at most cases, the algorithm falls below baseline when it comes to labeling recent videos.

Running Time (111, 2.5%): 18.98 seconds. The Decision Tree is an extremely fast classification algorithm.

Additional Comments:

From the window sensitivity test it is derived that while using strict datasets, the decision tree algorithm is slightly oversensitive (8.2%). For this reason it barely fails the three predefined tests. When a soft dataset is used, the algorithm succeeds in all three baseline tests, making it suitable for the needs of this research.

After comparison of all the stand-alone algorithms, the decision tree algorithm was considered to be the most suitable for the purpose of this research. All hybrid algorithms where developed on top of decision trees to maximize performance and classification accuracy. Detailed comparison of all the stand-alone algorithms can be found in Section 4.2.

3.2.6 Random Forest Evaluation

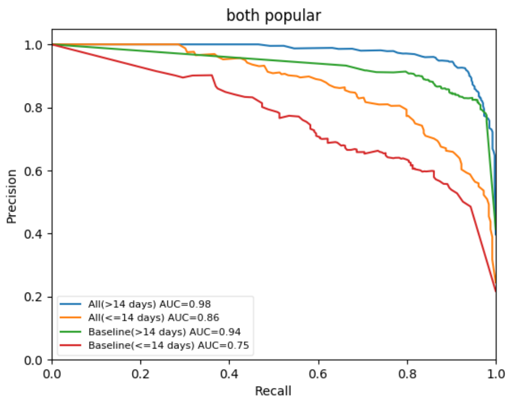
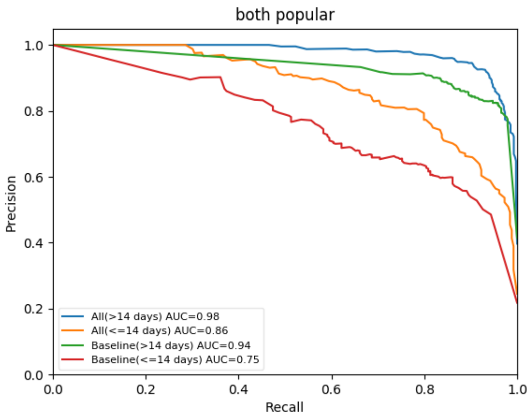
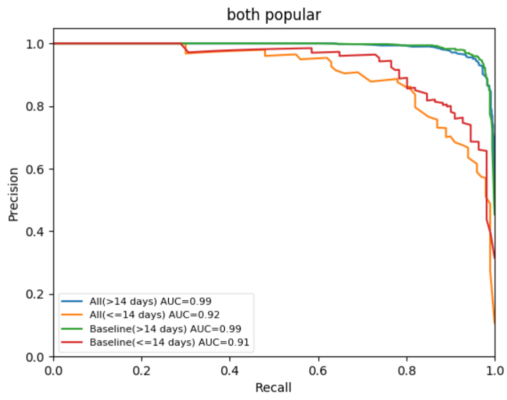
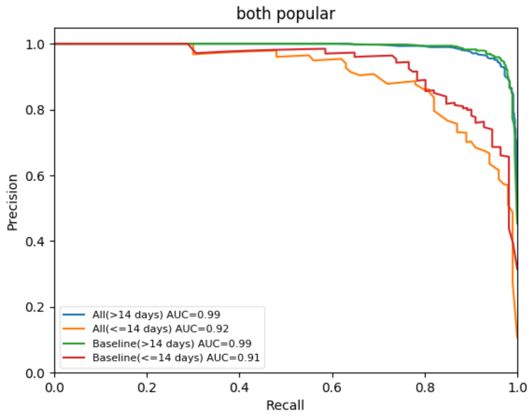
	2.5%	5%
111		
717		

Figure 30: Random Forest algorithm precision-recall graphs

Precision-Recall graph explanation:

The precision-recall graphs in Figure 30 show that the Random Forest algorithm produces very high AUC values, which could be an indicator of high accuracy classifications. The total number of videos doesn't seem to affect the outcome of the classification. The window number on the other hand, increases substantially the AUC values, especially for classifications using only baseline features.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7994	0.6818	0.7167
YouTube Features	0.6261	0.8513	0.7197
All Features	0.8061	0.8720	0.8020
Mean F1 Score	0.7639		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8662	0.7338	0.8039
YouTube Features	0.6901	0.9298	0.7883
All Features	0.8885	0.9322	0.8947
Mean F1 Score	0.8364		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7955	0.6986	0.7243
YouTube Features	0.6304	0.8634	0.7367
All Features	0.7815	0.8710	0.8096
Mean F1 Score	0.7679		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8545	0.7212	0.8071
YouTube Features	0.6916	0.9300	0.8008
All Features	0.8739	0.9325	0.8902
Mean F1 Score	0.8335		

Figure 31: F1 scores for Random Forest algorithm evaluation

F1 Scores explanation:

Looking at the F1 scores of Figure 31, it can be denoted that the Random Forest algorithm produces high accuracy classifications for most cases. It is clear from the "Viral and Popular" F1 scores that the amount of features plays a big role in the quality of the classification.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

The random algorithm produces fairly high average F1 scores for all the datasets, surpassing the minimum required score of the test.

(2) Window Sensitivity (<=8%) – Failed

The average F1 score in strict datasets was increased by 9.5% and in soft datasets it was increased by 8.5% meaning that the Random Forest algorithm is very sensitive to window changes.

(3) High AUC (>=0.80) – Partly Failed

In strict labeling datasets the algorithm produces AUC below the baseline when it comes to classifying recent videos using only the basic features. In soft labeling datasets though, the algorithm succeeds in all the classifications. This behavior was to be expected due to the very high window sensitivity of the algorithm.

Running Time (111, 2.5%): 248.38 seconds. The Random Forest algorithm has a moderate execution time cost.

Additional Comments:

The Random Forest algorithm substantially improves the performance and classification accuracy of a Decision Tree algorithm, but it suffers from very high window sensitivity making it unsuitable for the minimum needs of the research. It also appears that this algorithm performs better when using both YouTube and Twitter features. This could be an indication that, the Random Forest algorithm is more suitable for a higher amount of features. This behavior was expected since the algorithm is built on top of a Decision Tree.

3.2.5 Extra Trees Evaluation

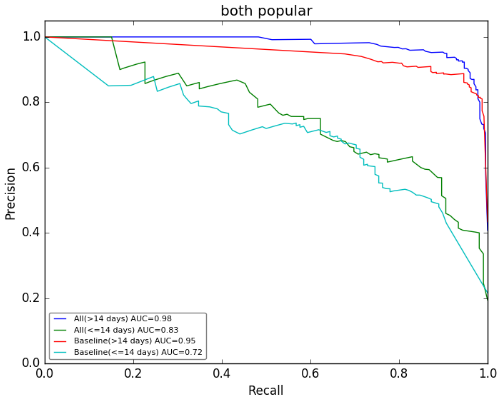
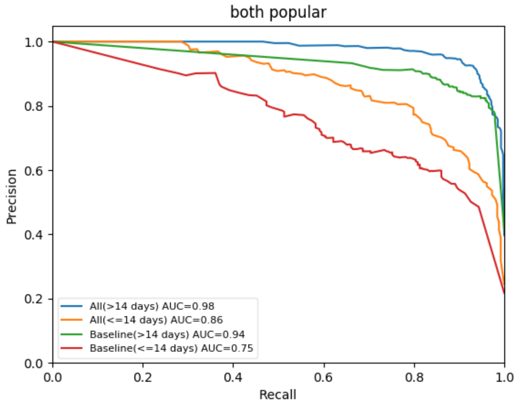
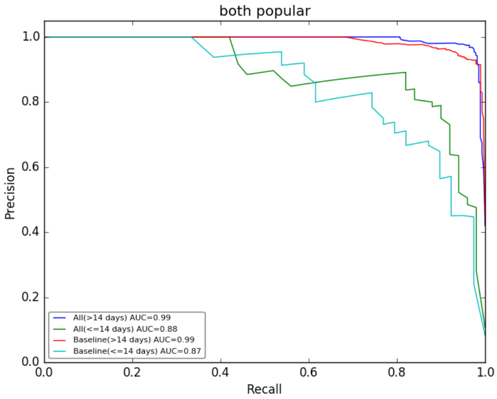
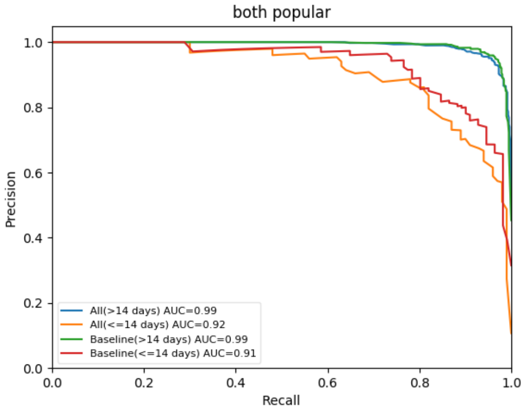
	2.5%	5%
111		
717		

Figure 32: Extra Trees algorithm precision-recall graphs

Precision-Recall graph explanation:

Looking at Figure 32, it seems that the Extra Trees algorithm produces precision-recall graphs with extremely high AUC value. This could be an indication of a high accuracy classification. Also, it appears that the performance of the algorithm is improved when the total number of videos in the dataset is increased or when the window number is increased.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7915	0.6734	0.6991
YouTube Features	0.6232	0.8515	0.7231
All Features	0.7996	0.8683	0.7932
Mean F1 Score	0.7581		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8659	0.7340	0.8058
YouTube Features	0.6825	0.9282	0.7851
All Features	0.8850	0.9285	0.8951
Mean F1 Score	0.8345		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7890	0.6875	0.7101
YouTube Features	0.6293	0.8626	0.7348
All Features	0.7800	0.8684	0.8016
Mean F1 Score	0.7626		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8548	0.7187	0.8099
YouTube Features	0.6889	0.9301	0.7989
All Features	0.8699	0.9321	0.8871
Mean F1 Score	0.8323		

Figure 33: F1 scores for Extra Trees algorithm evaluation

F1 Scores explanation:

Figure 33 shows that the Extra Trees algorithms produces very high F1 scores for almost all the cases. It slightly underperforms in cross-platform predictions. It’s worth noting that the quality of the classifications is substantially increased when the total amount of features is increased.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

The extra trees algorithm produces fairly high average F1 scores for all the datasets, surpassing the minimum required score of the test.

(2) Window Sensitivity (<=8%) – Failed

The average F1 score in strict datasets was increased by 10.1% and in soft datasets it was increased by 9.1% meaning that the Extra Trees algorithm is extremely sensitive to window changes.

(3) High AUC (>=0.80) – Partly Failed

In strict labeling datasets the algorithm produces AUC below the baseline when it comes to classifying recent videos using only the basic features. In soft labeling datasets though, the algorithm succeeds in all the classifications. This behavior was to be expected due to the extremely high window sensitivity of the algorithm.

Running Time (111, 2.5%): 166.68 seconds. The Extra Trees algorithm has a moderate execution time cost.

Additional Comments:

The extra trees algorithm substantially improves the performance and classification accuracy of a Decision Tree algorithm, but it suffers from extremely high window sensitivity making it unsuitable for the minimum needs of the research. The major increase in the accuracy of the classification when more features are used is mostly attributed to the fact that Extra Trees consist of a bundle of Decision Trees

3.2.7 Bagging Decision Tree Evaluation

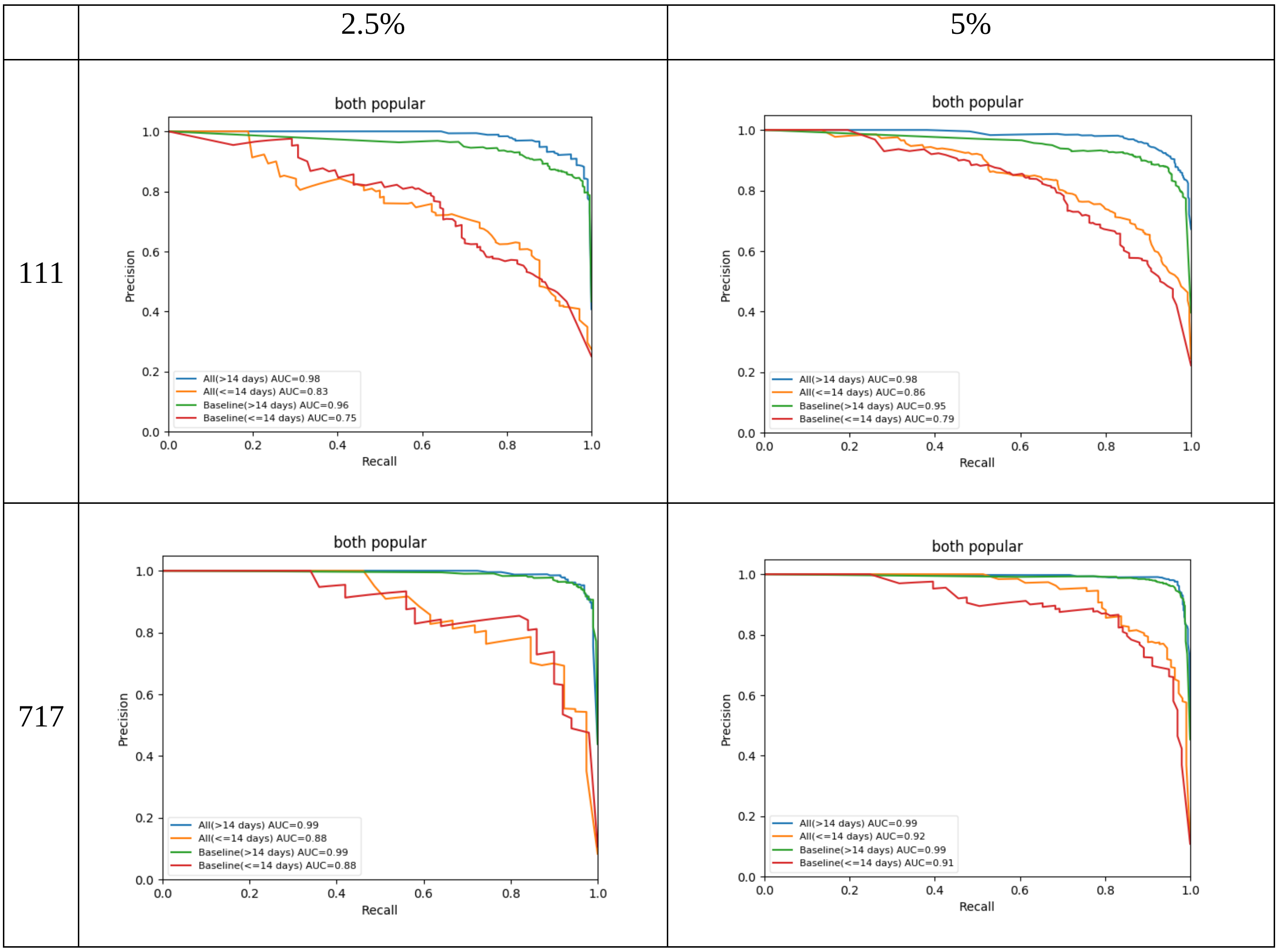


Figure 34: Bagging Decision Tree algorithm precision-recall graphs

Precision-Recall graph explanation:

The precision-recall graphs in Figure 34 show a slight increase in the AUC values when the total amount of videos was increase. It also shows a major increase in the AUC values when the larger windows where used.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7978	0.6813	0.7157
YouTube Features	0.6290	0.8504	0.7203
All Features	0.8085	0.8734	0.8043
Mean F1 Score	0.7645		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8632	0.7255	0.8049
YouTube Features	0.6880	0.9295	0.7882
All Features	0.8866	0.9318	0.8954
Mean F1 Score	0.8347		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7952	0.6995	0.7268
YouTube Features	0.6328	0.8629	0.7367
All Features	0.7830	0.8763	0.8137
Mean F1 Score	0.7697		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8534	0.7196	0.8097
YouTube Features	0.6971	0.9292	0.8017
All Features	0.8731	0.9338	0.8885
Mean F1 Score	0.8340		

Figure 35: F1 scores for Bagging Decision Tree algorithm evaluation

F1 Scores explanation:

Looking at the f1 scores in Figure 35, it appears the BDT algorithm produces very high F1 scores, meaning that the accuracy of most of the classifications is high. Like most of the algorithms it stumbles on cross-platform predictions.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

The bagging methodology improved the performance of the decision tree, producing average F1 scores greater than the baseline.

(2) Window Sensitivity (<=8%) – Failed

The average F1 score in strict datasets was increased by 9.2% and in soft datasets it was increased by 8.4% meaning that the Bagging Decision Tree algorithm is very sensitive to window changes.

(3) High AUC (>=0.80) – Partly Failed

In strict labeling datasets the algorithm produces AUC below the baseline when it comes to classifying recent videos using only the basic features. In soft labeling datasets though, the algorithm succeeds in all the classifications. This behavior was to be expected due to the very high window sensitivity of the algorithm.

Running Time (111, 2.5%): 1013.85 seconds. The Bagging Decision Tree algorithm has an extremely high execution time cost.

Additional comments:

The Bagging Decision Tree algorithm improves substantially the performance and classification accuracy of a Decision Tree algorithm, but it suffers from very high window sensitivity making it unsuitable for the minimum needs of the research. The algorithm is very window sensitive due to the randomness embedded in the bagging methodology. The performance of the BDT algorithm is increased when more features are used due to the use of the Decision Tree structure. It is particularly interesting that the bagging methodology comes with an extremely high execution time cost.

3.2.8 Gradient Boosting Decision Tree Evaluation

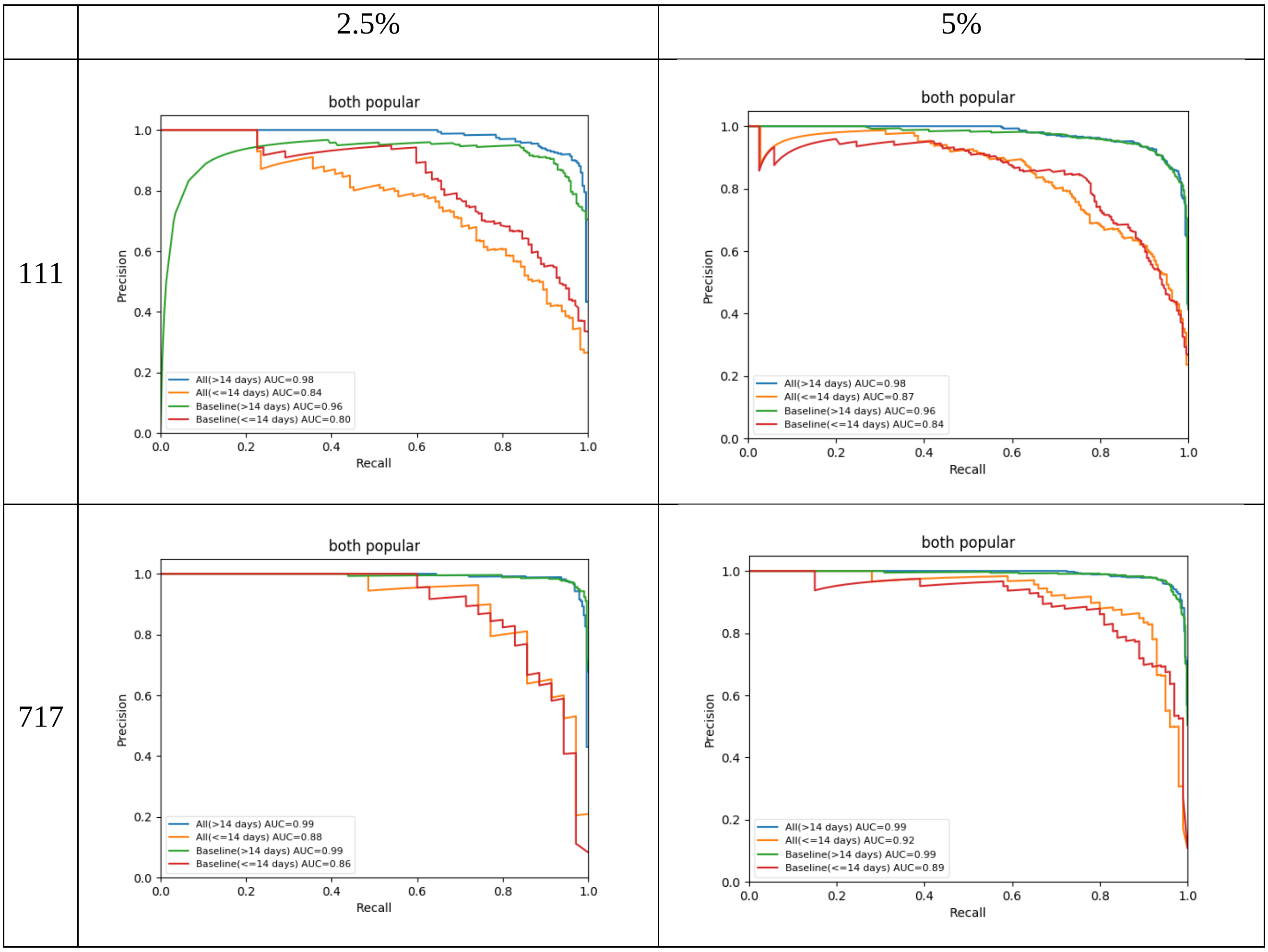


Figure 36: Gradient Boosting Decision Tree algorithm precision-recall graphs

Precision-Recall graph explanation:

Figure 36 shows that the boosting methodology has improved the precision-recall graphs. When predicting older videos, the GBDT algorithm produces almost perfect AUC values. It appears that when the total number of videos is increased or when using larger windows, the GBDT algorithm performs much better.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8033	0.6898	0.7220
YouTube Features	0.6109	0.8755	0.7307
All Features	0.8292	0.8849	0.8161
Mean F1 Score	0.7736		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8649	0.7363	0.8079
YouTube Features	0.6837	0.9269	0.7969
All Features	0.8842	0.9310	0.8931
Mean F1 Score	0.8361		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8005	0.7053	0.7268
YouTube Features	0.6265	0.8853	0.7498
All Features	0.8097	0.8891	0.8285
Mean F1 Score	0.7802		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8602	0.7222	0.8145
YouTube Features	0.6928	0.9319	0.7986
All Features	0.8747	0.9346	0.8939
Mean F1 Score	0.8359		

Figure 37: F1 scores for Gradient Boosting Decision Tree algorithm evaluation

F1 Scores explanation:

It is clear from the F1 scores in Figure 37 that the boosting methodology has substantially improved the performance of the Decision Tree. It produces very high F1 scores, with the exception of cross-platform predictions. The correlation between the accuracy of the classifier and the amount of features is clearly visible from the “Viral and Popular” F1 scores.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

Gradient boosting improved significantly the performance of the decision tree, producing average F1 scores much higher than the baseline.

(2) Window Sensitivity (<=8%) – Partly Failed

The average F1 score in strict datasets was increased by 8.1% and in soft datasets it was increased by 7.1% meaning that the Gradient boosting decision tree algorithm is mildly sensitive to window changes causing it to fail the baseline test when strict datasets are used.

(3) High AUC (>=0.80) – Succeeded

The gradient boosting decision tree algorithm produces very high AUC values despite its mild window oversensitivity.

Running Time (111, 2.5%): 198.48 seconds. The Gradient Boosting Decision Tree algorithm has a moderate execution time cost.

Additional comments:

High average F1 scores and AUC values are an indicative of a very accurate classifier. The gradient boosting decision tree algorithm produces high quality classifications making it one of the best choices for the purpose of this research. A small drawback of this classifier is its mild oversensitivity to window changes. To overcome this problem some other hybrid algorithms are used along with GBDT which lower the classifier's window sensitivity. These hybrid algorithms are evaluated in detail in the following Sections. Finally, the algorithm's dependence in the amount of features used is due to the decision tree structure.

3.2.9 Ada Boosting Decision Tree Evaluation

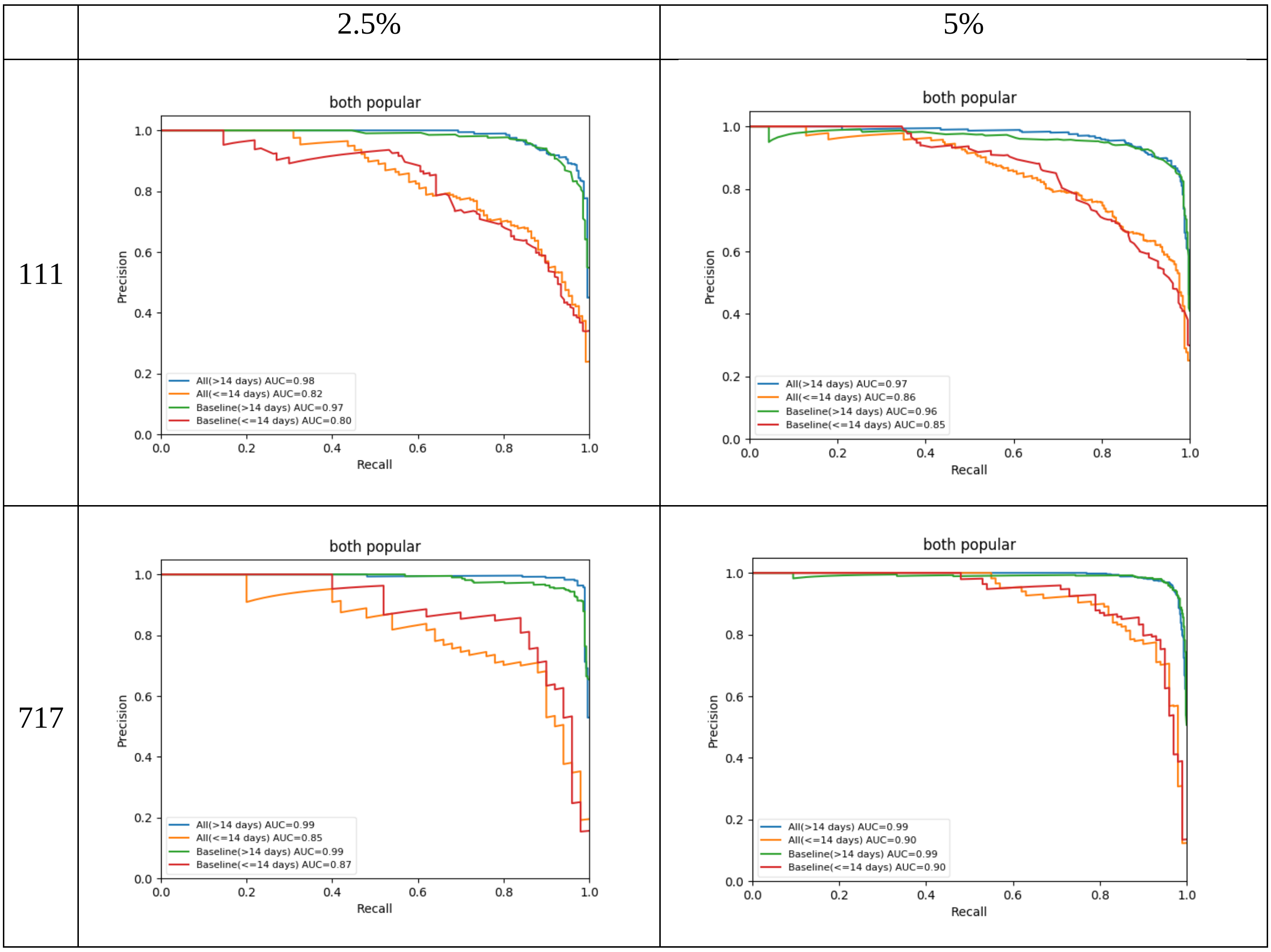


Figure 38: Ada Boosting Decision Tree algorithm precision-recall graphs

Precision-Recall graph explanation:

Figure 38 shows that adaptive boosting produces high AUC values in precision-recall graphs. This could be an indicator of a high accuracy classification. Also, the algorithm seems to perform slightly better when the total number of videos is used or when larger windows are applied.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7992	0.6858	0.7257
YouTube Features	0.6077	0.8724	0.7324
All Features	0.8261	0.8809	0.8125
Mean F1 Score	0.7714		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8560	0.7311	0.7951
YouTube Features	0.6744	0.9209	0.7895
All Features	0.8770	0.9215	0.8832
Mean F1 Score	0.8276		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8030	0.6996	0.7261
YouTube Features	0.6278	0.8827	0.7457
All Features	0.8061	0.8866	0.8254
Mean F1 Score	0.7781		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8551	0.7158	0.8032
YouTube Features	0.6830	0.9294	0.7954
All Features	0.8665	0.9291	0.8758
Mean F1 Score	0.8281		

Figure 39: F1 scores for Ada Boosting Decision Tree algorithm evaluation

F1 Scores explanation:

Figure 39 shows the F1 scores of the Ada Boosting Decision Tree algorithm. It appears that adaptive boosting produces very high F1 scores with the exception of cross-platform predictions where it underperforms. It particularly interesting that the ABDT algorithm has reduced dependence on the amount of features despite the Decision Tree structure used within the algorithm. It is also worth noting that the average F1 score doesn't change much when larger windows are used.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

Adaptive boosting improved significantly the performance of the decision tree, producing average F1 scores much higher than the baseline.

(2) Window Sensitivity (<=8%) – Succeeded

The average F1 score in strict datasets was increased by 7.3% and in soft datasets it was increased by 6.4% meaning that the Ada boosting decision tree algorithm has extremely low window sensitivity.

(3) High AUC (>=0.80) – Succeeded

The adaptive boosting decision tree algorithm produces very high AUC values for all the datasets and windows of the experiments.

Running Time (111, 2.5%): 238.77 seconds. The Ada Boosting Decision Tree algorithm has a moderate execution time cost.

Additional comments:

The Ada boosting decision tree algorithm managed to pass all the baseline tests, making it ideal for the purpose of this research. Looking at the average F1 scores and the AUC values, it can be concluded that ABDT produces high accuracy classifications. In addition to its' high performance, ABDT has very low sensitivity to window changes making it ideal for future usage of the algorithm in variable windows to make popularity and virality predictions.

3.2.10 Voting – Ada and Gradient Boosting Decision Tree Evaluation

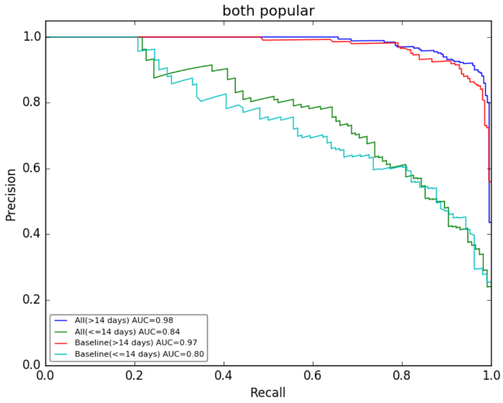
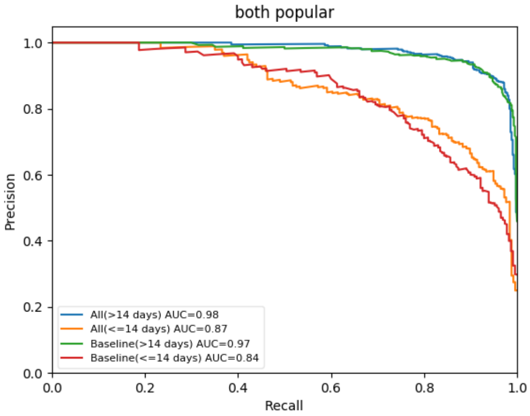
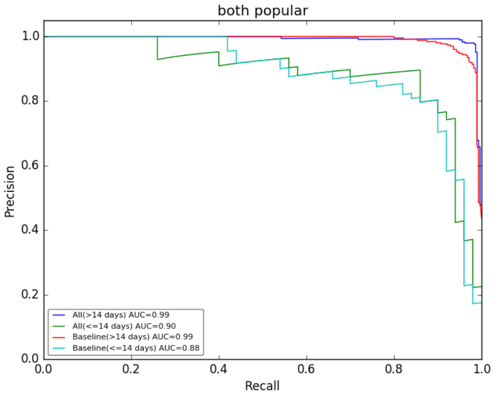
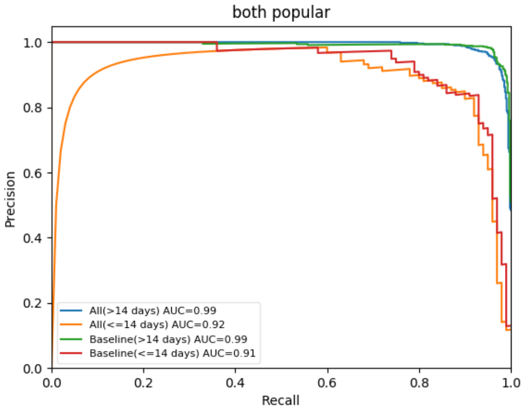
	2.5%	5%
111		
717		

Figure 40: Voting – Ada and Gradient Boosting Decision Tree algorithm precision-recall graphs

Precision-Recall graph explanation:

The precision-recall graphs in Figure 40 shows that the voting between ABDT and GBDT has produced very high AUC values which is an indicator of a high quality classification. This hybrid algorithm still has a small dependence to the number of videos in the dataset and the size of the windows.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8031	0.6898	0.7225
YouTube Features	0.6110	0.8757	0.7318
All Features	0.8294	0.8851	0.8170
Mean F1 Score	0.7739		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8666	0.7353	0.8062
YouTube Features	0.6822	0.9273	0.7964
All Features	0.8858	0.9314	0.8935
Mean F1 Score	0.8361		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8006	0.7056	0.7266
YouTube Features	0.6263	0.8857	0.7498
All Features	0.8098	0.8895	0.8289
Mean F1 Score	0.7803		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8603	0.7224	0.8140
YouTube Features	0.6930	0.9318	0.7982
All Features	0.8752	0.9346	0.8948
Mean F1 Score	0.8360		

Figure 41: F1 scores for Voting – Ada and Gradient Boosting Decision Tree algorithm evaluation

F1 Scores explanation:

The F1 scores in Figure 41 show stunning results. The voting between the two boosting algorithm has produced very high F1 scores, which in combination with the high AUC values proves that this model is very powerful. It has even managed to slightly increase cross-platform prediction scores. Its dependence on the amount of features used is slightly reduce, as it seems from the “Virality and Popularity” columns.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded
Voting – (ABDT, GBDT) improved significantly the performance of the decision tree, producing average F1 scores much higher than the baseline.

(2) Window Sensitivity (<=8%) – Succeeded
The average F1 score in strict datasets was increased by 8% and in soft datasets it was increased by 7.1% meaning that the Voting – (ABDT, GBDT) algorithm has a moderate window sensitivity but lies below the predefined baseline.

(3) High AUC (>=0.80) – Succeeded
The Voting – (ABDT, GBDT) algorithm produces very high AUC values.

Running Time (111, 2.5%): 303.5 seconds. The Voting – (ABDT, GBDT) algorithm has a very high execution time cost.

Additional comments:

GBDT produced extremely high F1 scores and AUC values but suffered from mild window sensitivity. ABDT produced very high F1 scores and AUC values as well but was characterized by a very low sensitivity to window changes. Voting – (ABDT, GBDT) is a hybrid algorithm which uses the majority vote methodology to produce high accuracy values but at the same time achieve low window sensitivity. It is clear from the experiment results that voting between the two classification algorithms has produced a very positive outcome since it has succeeded in all three baseline tests. The only drawback of this algorithm is its very high computational cost.

3.2.11 Voting – Ada and Gradient Boosting Decision Tree with Logistic Regression Evaluation

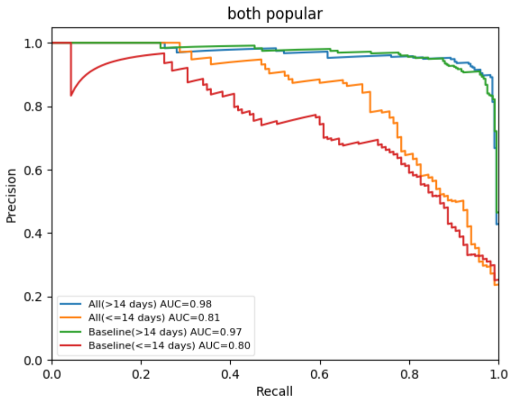
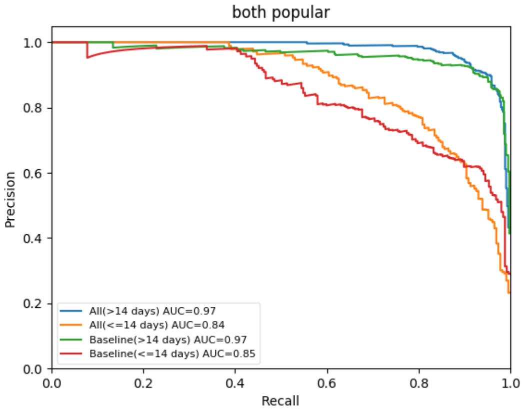
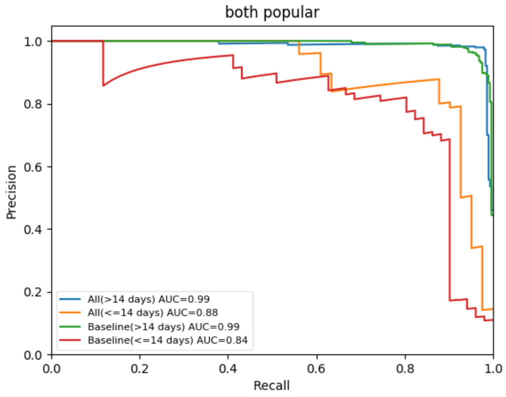
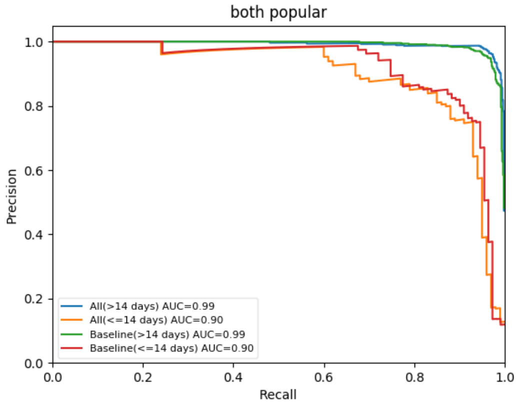
	2.5%	5%
111		
717		

Figure 42: Voting – (ABDT, GBDT, LR) algorithm precision-recall graphs

Precision-Recall graph explanation:

The precision-recall graphs shown in Figure 42 are an indication of a high quality classification model. The AUC values produced are very high for almost all the cases. When the number of videos is increased or larger windows are used, the AUC values are slightly increased.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8010	0.6466	0.6501
YouTube Features	0.5980	0.8726	0.6936
All Features	0.8243	0.8789	0.7874
Mean F1 Score	0.7503		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8625	0.7106	0.7285
YouTube Features	0.6703	0.9229	0.7835
All Features	0.8752	0.9241	0.8754
Mean F1 Score	0.8170		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.7908	0.6716	0.6616
YouTube Features	0.6150	0.8815	0.7146
All Features	0.8009	0.8856	0.7988
Mean F1 Score	0.7578		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8560	0.7032	0.7492
YouTube Features	0.6739	0.9271	0.7820
All Features	0.8650	0.9309	0.8785
Mean F1 Score	0.8184		

Figure 43: F1 scores for Voting – (ABDT, GBDT, LR) algorithm evaluation

F1 Scores explanation:

The F1 scores shown in Figure 43 are a solid proof of how deceiving precision-recall graphs can sometimes be [10, 16]. For this reason a combination of the F1 score and the AUC values is used in order to label a classifier as a “powerful model”. Looking at the F1 scores it is easily noticeable that in some cases the algorithm performs relatively bad. More specifically, it underperforms in cross-platform predictions and in “Viral and Popular” predictions.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

Voting – (ABDT, GBDT, LR) improved the performance of the decision tree, producing average F1 scores slightly higher than the baseline.

(2) Window Sensitivity (<=8%) – Partly Failed

The average F1 score in strict datasets was increased by 8.9% and in soft datasets it was increased by 8% meaning that the Voting – (ABDT, GBDT, LR) algorithm has a high window sensitivity when it comes to more strict datasets.

(3) High AUC (>=0.80) – Succeeded

The Voting – (ABDT, GBDT, LR) algorithm produces high AUC values and manages to pass the baseline in all the datasets despite its high sensitivity to window changes.

Running Time (111, 2.5%): 334.89 seconds. The Voting – (ABDT, GBDT, LR) algorithm has a very high execution time cost.

Additional comments:

Logistic regression was added as part of the voting algorithm in order to examine how far can majority vote improve the quality of the classification. It is clear that when using too many different estimators, voting can lead to low quality classifications due to overfitting. More estimators doesn’t always mean better classifications.

3.2.12 Bagging Gradient Boosting Decision Tree Evaluation

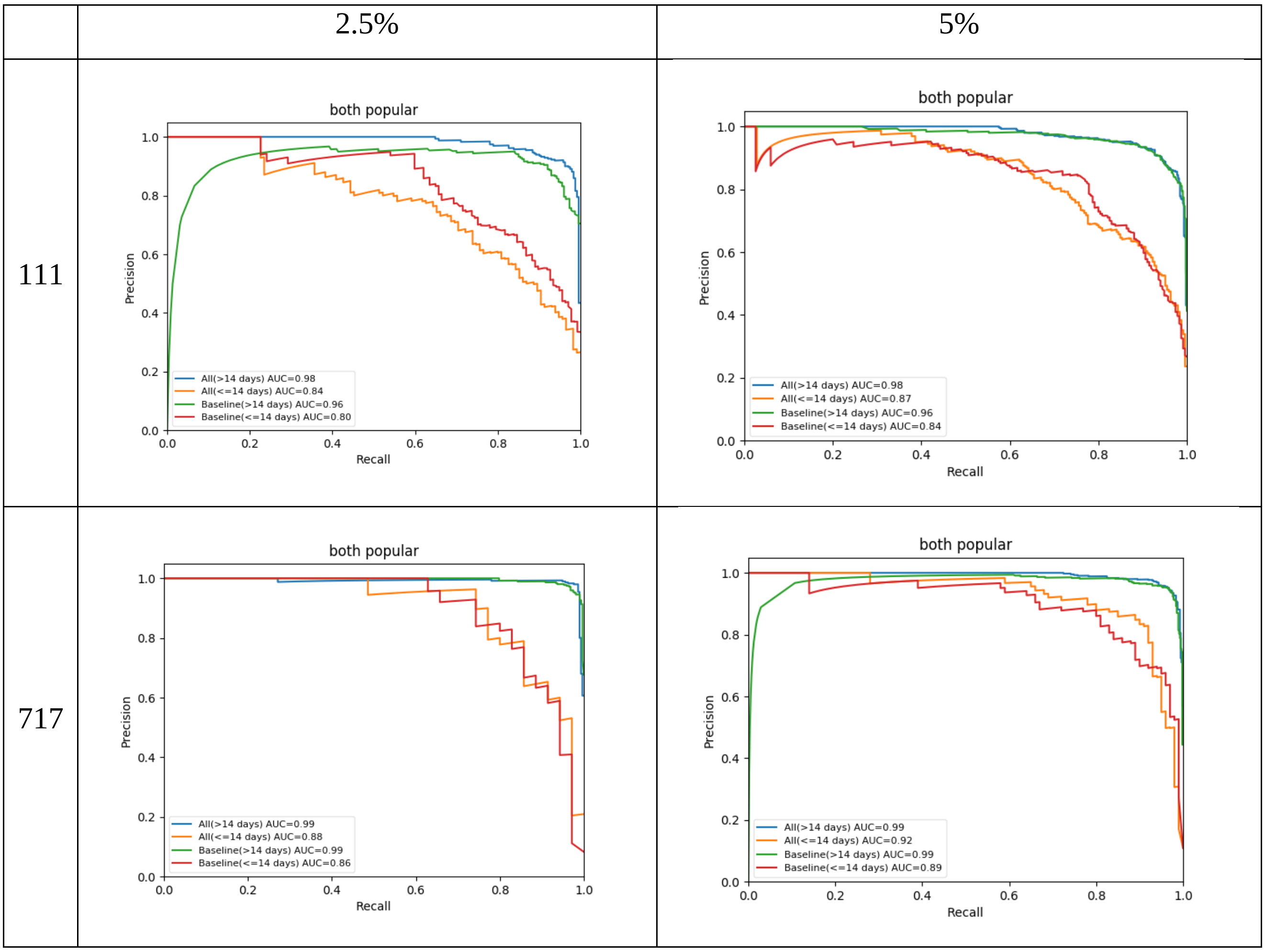


Figure 44: Bagging Gradient Boosting Decision Tree algorithm precision-recall graphs

Precision-Recall graph explanation:

As it seems from the precision-recall graphs in Figure 44, the Bagging Gradient Boosting Decision Tree algorithm produces very high AUC values with a smaller dependence on the number of videos used and the size of the window.

F1 Scores (111, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8034	0.6898	0.7218
YouTube Features	0.6108	0.8757	0.7307
All Features	0.8291	0.8849	0.8168
Mean F1 Score	0.7737		

F1 Scores (717, 2.5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8655	0.7357	0.8081
YouTube Features	0.6839	0.9267	0.7943
All Features	0.8845	0.9305	0.8928
Mean F1 Score	0.8358		

F1 Scores (111, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8005	0.7055	0.7267
YouTube Features	0.6265	0.8854	0.7499
All Features	0.8096	0.8892	0.8283
Mean F1 Score	0.7802		

F1 Scores (717, 5%)	Viral	Popular	Viral and Popular
Twitter Features	0.8603	0.7220	0.8138
YouTube Features	0.6929	0.9318	0.7990
All Features	0.8747	0.9346	0.8937
Mean F1 Score	0.8357		

Figure 45: F1 scores for Bagging Gradient Boosting Decision Tree algorithm evaluation

F1 Scores explanation:

The F1 scores shown in Figure 45 give a clearer image of how accurate the classifications of the BGBDT algorithm are. The algorithm underperforms in relation to other hybrid algorithms but it is still a descent model for binary classifications. Looking at the “Viral and Popular” column of the Figure it is clear that the model performs much better when larger amounts of features are used.

Baseline Testing:

(1) Average F1 Scores (>=0.75) – Succeeded

Bagging GBDT significantly improved the performance of the decision tree, producing average F1 scores much higher than the baseline.

(2) Window Sensitivity (<=8%) – Succeeded

The average F1 score in strict datasets was increased by 8% and in soft datasets it was increased by 7.1% meaning that the Bagging GBDT algorithm has a moderate window sensitivity but lies below the predefined baseline.

(3) High AUC (>=0.80) – Succeeded

The Bagging GBDT produces very high AUC values making it one of the most accurate classifiers in this research.

Running Time (111, 2.5%): 1618.67 seconds. The Bagging Gradient Boosting Decision Tree algorithm has an extremely high execution time cost.

Additional comments:

In order to improve the performance of a gradient boosting decision tree algorithm and at the same time reduce its' window sensitivity, the bagging methodology was applied. The new Bagging GBDT hybrid algorithm produced very promising results since it appears to have improved the overall performance of the GBDT classifier. Despite its' high quality performance, the bagging GBDT is still characterized by a moderate but acceptable window sensitivity. The combination of the Gradient boosting methodology and the bagging methodology produces very accurate classifications but the execution time required is extremely high.

Chapter 4

Comparison

4.1 Comparison Methodology

In order to examine which algorithm better suits the needs of a certain research, a comparison methodology was constructed. The comparison procedure consists of three different metrics used in the evaluation of the algorithms. Depending on the kind and needs of a research, different weights can be assigned to the following metrics, making one more important than the other. In the context of this research, average F1 score is the most important metric.

The following tables depict how the algorithms chosen will be compared. **The "Comparison" column contains the name of the best performing algorithm.**

Average F1 Scores	Algorithm 1	Algorithm 2	Comparison
111 – 2.5%			
717 – 2.5%			
111 – 5%			
717 – 5%			

Table 16: Example of an average F1 score comparison table.

Window Sensitivity	Algorithm 1	Algorithm 2	Comparison
Strict Labeling			
Soft Labeling			

Table 17: Example of window sensitivity in strict and soft labeling comparison table

AUC	Algorithm 1	Algorithm 2	Comparison
All Features Old - AUC			
All Features Recent - AUC			
Baseline Features Old - AUC			
Baseline Features Recent - AUC			

Table 18: Example of AUC values for Both Popular graphs with (111, 2.5%) comparison table

Running Time	Algorithm 1	Algorithm 2	Comparison
(111, 2.5%)			

Table 19: Example of Running Time for (111, 2.5%) comparison table

4.2 Stand-alone Algorithms Comparison

Average F1 Scores	LR	KNN	SVM	DT	Comparison
111 – 2.5%	0.6136	0.6607	0.5721	0.7319	DT
717 – 2.5%	0.6875	0.7132	0.6028	0.7922	DT
111 – 5%	0.6243	0.6769	0.5868	0.737	DT
717 – 5%	0.6906	0.7218	0.6176	0.7904	DT

Window Sensitivity	LR	KNN	SVM	DT	Comparison
Strict Labeling	12%	8%	5.4%	8.2%	SVM
Soft Labeling	11%	6.5%	5.2%	7.2%	SVM

AUC	LR	KNN	SVM	DT	Comparison
All Features Old - AUC	0.95	0.70	0.65	0.92	LR
All Features Recent - AUC	0.71	0.51	0.44	0.68	LR

Baseline Features Old - AUC	0.97	0.94	0.67	0.91	LR
Baseline Features Recent - AUC	0.78	0.74	0.44	0.66	LR

Running Time	LR	KNN	SVM	DT	Comparison
(111, 2.5%)	17.37	11.29	138.9	18.98	KNN

Figure 46: Comparison between all stand-alone algorithms (LR, KNN, SVM, DT)

Comparison comments on Figure 46:

In this Section a comparison between all the stand-alone algorithms used takes place in order to examine which algorithm is more promising for further development.

The stand-alone algorithms compared behaved exactly as expected. What is particularly interesting is that each algorithm dominated a different metric, with the exception of KNN. The Decision Tree algorithm produced much higher average F1 scores from the algorithms in comparison, proving that it is the most accurate algorithm when it comes to binary classifications.

The SVM algorithm has the lowest window sensitivity compared to the rest of the algorithms, whereas Logistic Regression has the highest. SVM low window sensitivity is a product of its consistency as an algorithm, whereas Logistic Regression high window sensitivity is an outcome of its’ probabilistic nature.

Logistic Regression produced the highest AUC values, showing that it performs well when it comes to precision-recall. What this means is that Logistic Regression appears to be a good standalone algorithm for binary classification but its’ poor performance on F1 Scores proves that AUC values could be misleading.

Since the most reliable and important metric for this research is the F1 score, the Decision Tree algorithm was chosen for further development as it performed substantially better than the others. Another reason for choosing the Decision Tree is its execution time cost which is very low.

4.3 Extra Trees VS Random Forest VS Bagging Decision Tree

Average F1 Scores	ET	RF	BDT	Comparison
111 – 2.5%	0.7581	0.7639	0.7645	BDT
717 – 2.5%	0.8345	0.8364	0.8347	RF
111 – 5%	0.7626	0.7679	0.7697	BDT
717 – 5%	0.8323	0.8335	0.8340	BDT

Window Sensitivity	ET	RF	BDT	Comparison
Strict Labeling	10.1%	9.5%	9.2%	BDT
Soft Labeling	9.1%	8.5%	8.4%	BDT

AUC	ET	RF	BDT	Comparison
All Features Old - AUC	0.98	0.98	0.98	-
All Features Recent - AUC	0.83	0.86	0.83	RF
Baseline Features Old - AUC	0.95	0.94	0.96	BDT
Baseline Features Recent - AUC	0.72	0.75	0.75	RF/BDT

Running Time	ET	RF	BDT	Comparison
(111, 2.5%)	166.68	248.37	1013.85	ET

Figure 47: Comparison between all the algorithms that embed randomness to their procedure

Comparison comments on Figure 47:

The first hybrid algorithms produced using Decision Trees where based on methodologies that integrate random characteristics and/or random subsets of data. Those ensemble algorithms where Extra Trees, Random Forest and Bagging Decision Tree. Each algorithm integrates randomness to its methodology in a different extent.

Comparing the average F1 Scores and AUC values of these algorithms, it is clear that both Bagging Decision Tree and Random Forest perform slightly better than the Extra Trees algorithm.

It is no coincidence that the Extra Trees algorithm has the highest window sensitivity of all three whereas the Bagging Decision Tree algorithm has the lowest. This behavior can be attributed to the extent of randomness in each algorithm. The Extra Trees algorithm integrates the most randomness in its methodology, therefore it is the most sensitive to window changes.

It is worth noting that the higher the accuracy the more running time required. The bagging methodology appears to be very time costly. Despite its running time, the Bagging Decision Tree algorithm has the best overall performance out of the three, so it was chosen for further comparison with different hybrid algorithms implemented using various methodologies.

4.4 BDT VS GBDT VS ABDT

Average F1 Scores	BDT	GBDT	ABDT	Comparison
111 – 2.5%	0.7645	0.7736	0.7714	GBDT
717 – 2.5%	0.8347	0.8361	0.8276	GBDT
111 – 5%	0.7697	0.7802	0.7781	GBDT
717 – 5%	0.8340	0.8359	0.8281	GBDT

Window Sensitivity	BDT	GBDT	ABDT	Comparison
Strict Labeling	9.2%	8.1%	7.3%	ABDT
Soft Labeling	8.4%	7.1%	6.4%	ABDT

AUC	BDT	GBDT	ABDT	Comparison
All Features Old - AUC	0.98	0.98	0.98	-
All Features Recent - AUC	0.83	0.84	0.82	GBDT
Baseline Features Old - AUC	0.96	0.96	0.97	ABDT
Baseline Features Recent - AUC	0.75	0.80	0.80	GBDT/ABDT

Running Time	BDT	GBDT	ABDT	Comparison
(111, 2.5%)	1013.85	198.48	238.77	GBDT

Figure 48: Comparison between the boosting algorithms and the Bagging Decision Tree algorithm

Comparison comments on Figure 48:

The Bagging Decision Tree algorithm is compared with two ensemble algorithms that apply some kind of boosting on top of the decision tree weak learner. More specifically Adaptive Boosting and Gradient Boosting respectively.

Looking at the comparison tables, it is clear that boosting methodologies produce better overall results than the Bagging Decision Tree algorithm. It is worth noting that since GBDT and ABDT don’t integrate any kind of randomness in their methodologies, they are much less sensitive to window changes than BDT.

Analyzing the results it is not clear which algorithm, Gradient Boosting Decision Tree or Ada Boosting Decision Tree, performs better.

The GBDT algorithm produces slightly higher average F1 scores and AUC values than the ABDT algorithm. The ABDT algorithm on the other hand is substantially less sensitive to window changes than GBDT, which makes it particularly useful when it comes to scaling the problem in discussion. The running time of the two boosting algorithms is similar with the GBDT algorithm requiring a little less time to run than the ABDT algorithm.

Depending on the nature of the problem either one of these two algorithm be chosen.

4.5 GBDT VS Voting – (GBDT, ABDT)

Average F1 Scores	GBDT	Voting – (GBDT, ABDT)	Comparison
111 – 2.5%	0.7736	0.7739	Voting
717 – 2.5%	0.8361	0.8361	-
111 – 5%	0.7802	0.7803	Voting
717 – 5%	0.8359	0.8360	Voting

Window Sensitivity	GBDT	Voting – (GBDT, ABDT)	Comparison
Strict Labeling	8.1%	8%	Voting
Soft Labeling	7.1%	7.1%	-

AUC	GBDT	Voting – (GBDT, ABDT)	Comparison
All Features Old - AUC	0.98	0.98	-
All Features Recent - AUC	0.84	0.84	-
Baseline Features Old - AUC	0.96	0.97	Voting
Baseline Features Recent - AUC	0.80	0.80	-

Running Time	GBDT	Voting – (GBDT, ABDT)	Comparison
(111, 2.5%)	198.48	303.50	GBDT

Figure 49: Comparison between GBDT and Voting – (GBDT, ABDT)

Comparison comments on Figure 49:

Since the GBDT and the ABDT algorithms performed similarly, the Voting methodology was used to try and create a hybrid algorithm with higher average F1 Scores and AUC values and lower window sensitivity.

The Voting – (GBDT, ABDT) hybrid algorithm managed to produce higher average F1 Scores from GBDT, while at the same time reducing slightly the window sensitivity. The AUC values where overall the same with a small exception where Voting – (GBDT, ABDT) performed slightly better.

In addition to the above results, further comparison took place between the two algorithms, which contained all the scenarios examined in this research. The conclusion was that Voting – (GBDT, ABDT) performs better than GBDT in almost all the scenarios. The only throwback of Voting – (GBDT, ABDT) algorithm is its computational cost that is much higher than that of the GBDT algorithm. Since the GBDT uses 100 estimators whereas Voting – (GBDT, ABDT) uses 150, it was expected that the voting algorithm would require at least 1.5 times more running times than the GBDT algorithm. This hypothesis was justified by the running time results presented in the comparison table.

4.6 Voting – (GBDT, ABDT) VS Bagging GBDT

Average F1 Scores	Voting – (GBDT, ABDT)	Bagging GBDT	Comparison
111 – 2.5%	0.7739	0.7737	Voting
717 – 2.5%	0.8361	0.8358	Voting
111 – 5%	0.7803	0.7802	Voting
717 – 5%	0.8360	0.8357	Voting

Window Sensitivity	Voting – (GBDT, ABDT)	Bagging GBDT	Comparison
Strict Labeling	8%	8%	-
Soft Labeling	7.1%	7.1%	-

AUC	Voting – (GBDT, ABDT)	Bagging GBDT	Comparison
All Features Old - AUC	0.98	0.98	-
All Features Recent - AUC	0.84	0.84	-
Baseline Features Old - AUC	0.97	0.96	Voting
Baseline Features Recent - AUC	0.80	0.80	-

Running Time	Voting – (GBDT, ABDT)	Bagging GBDT	Comparison
(111, 2.5%)	303.50	1618.67	Voting – (GBDT, ABDT)

Figure 50: Comparison between Voting – (GBDT, ABDT) and Bagging GBDT algorithms

Comparison comments on Figure 50:

The Bagging GBDT hybrid algorithm combines randomness and boosting in an attempt to produce a high quality classifier that performs better than all the other algorithms. Since it integrates two different kinds of demanding ensemble methodologies, it is obvious that Bagging GBDT comes with a very high computational cost.

Despite the fact that the Bagging GBDT algorithm performed very well, it failed to surpass the Voting – (GBDT, ABDT) algorithm in any of the metrics. It “tied” in window sensitivity and AUC values but underperformed in the most important metric, which is the average F1 Scores. Bagging GBDT has an extremely high computational cost. It requires five times more running time than the Voting algorithm. This execution time cost doesn’t justify the final results, therefore there is no reason to use the Bagging GBDT algorithm instead of the Voting – (GBDT, ABDT) algorithm.

4.7 Comparison Conclusion

After extensive research, experiments and comparison between multiple algorithms, a list with the most high performing hybrid algorithms was produced.

The list consists of the Gradient Boosting Decision Tree, Ada Boosting Decision Tree and Voting – (GBDT, ABDT) hybrid algorithms. These algorithms appear to be the most suitable for this kind of research problems or any similar binary classification problems.

The **Gradient Boosting Decision Tree algorithm** produces extremely accurate classifications. It’s the strongest boosting algorithm and has a moderate computational cost. It’s superiority to any other boosting algorithm is mostly denoted by the high F1 Scores and AUC values produced. Its **drawback** is that GBDT appears to be slightly “oversensitive” to window changes.

The **Ada Boosting Decision Tree algorithm** produces very accurate classifications. It has a moderate computational cost. This hybrid algorithm is particularly powerful when window sensitivity is very important to the research. Its **drawback** is that it produces slightly lower F1 Scores and AUC values than Gradient Boosting Decision Tree.

The **Voting – (GBDT, ABDT) algorithm** produces extremely accurate classifications. It has the highest F1 Scores from all the other algorithms compared. It has moderate window sensitivity. Its **drawback** is that it comes with a very high computational cost.

All the metrics mentioned in this document where taken into consideration for the final choice of the most suitable algorithm for this research. It was concluded that the **Voting – (GBDT, ABDT) hybrid algorithm** fits the most important needs of this kind of research and was chosen as the main algorithm for future work.

Chapter 5

Discussion

5.1 Summary and Final Thoughts

Stand-alone algorithms are generally weak learners, meaning that they produce poor classifications. Decision Tree, which is a weak learner, was a good algorithm to build stronger and more accurate, hybrid algorithms on top of.

Hybrid algorithms, which is a combination of weak learners and various methodologies, produced high accuracy classifications but with some drawbacks in some cases. For example, it was concluded that any kind of randomization procedures within the algorithm itself, increases its sensitivity to window changes. This oversensitivity is undesirable since the aim of this research is to be able to produce powerful classifications independently of the training, offset and labeling window.

Looking at the training features, it is clear that some are more important than others. More specifically, any kind of acceleration or ratios appears to assist more in the classification than most of the "static" features.

For the needs of this research, it was concluded that the Voting – (GBDT, ABDT) hybrid algorithm is the most suitable. It is worth noting that a set of other hybrid algorithms analyzed and evaluated in this study, produced high quality classifications as well and could be most suitable in scenarios where other metrics are more important (e.g. window sensitivity). In short, similar problems with slightly different needs could use one of the other powerful models developed and evaluated in this study.

Table 20 contains the Voting – (GBDT, ABDT) classifier's F1 scores for a sample of test cases of this research. It is clear that there's still room for improvement, especially in cross platform classifications. Looking at this table, a particularly interesting hypothesis was made. It seems that a different algorithms could be assigned in different subsections of the problem in hope of producing better classifications.

F1 Scores	Viral	Popular	Viral and Popular
Twitter Features	0.8031	0.6898	0.7225
YouTube Features	0.6110	0.8757	0.7318
All Features	0.8294	0.8851	0.8170

Table 20: Sample of Voting – (GBDT, ABDT) classifier's F1 scores

5.2 Future Work

Following the hypothesis that was made from looking at table 20 in Section 5.1, it appears that dividing the problem into various subsections, could indeed produce much better results. More specifically, the overall problem could be divided into four sections:

1. Predictions using Twitter Features
2. Predictions using YouTube features
3. Predictions using All Features
4. Cross-platform predictions.

A different kind of hybrid algorithm could be assigned to each subsection. Each algorithm could then be fine-tuned and adjusted to the specific needs of each section in order to produce more accurate classifications. This approach to the problem is feasible and very promising. The outcome could be an extremely powerful model that could make high accuracy, platform independent, predictions. It has been noted that when the features are "weak", meaning that they don't offer much information gain to the classifier, a more randomized approach might be more suitable. For example, in the case of predictions using Twitter features, which don't offer much information gain, a Random Forest algorithm could be more preferable than a Boosting algorithm.

Finally, further experimentation could take place regarding the training features. More specifically, features that don't seem to be important to the classification could be removed in the future, whereas features that are important, could be combined to produce even more powerful training features.

References

[1] Chris Anderson and Michael Wolff: *The Web Is Dead*. *Long Live the Internet*. URL: https://www.wired.com/2010/08/ff_webrip/ [Last Accessed 14 May 2017]

[2] David Vallety, Shlomo Berkovskyz, Sebastien Ardony, Anirban Mahantiy, Mohamed Ali Kaafary: *Characterizing and Predicting Viral-and-Popular Video Content*.

[3] What is: Machine Learning? URL: http://whatis.techtarget.com/definition/machine-learning [Last Accessed: 14 May 2017]

[4] Πούης Λούκας, Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής Πανεπιστημίου Κύπρου 2016: *Μελέτη Της Διάδοσης Των Βίντεο Του YouTube Στο Μέσο Κοινωνικής δικτύωσης Twitter.*

[5] Quora: What are the top 10 data mining or machine learning algorithms? URL: https://www.quora.com/What-are-the-top-10-data-mining-or-machine-learning-algorithms [Last Accessed: 14 May 2017]

[6] YouTube Data API V3.0. URL: https://developers.google.com/youtube/v3/ [Last Accessed: 14 May 2017]

[7] Twitter Developer Documentation – Streaming APIs. URL: https://dev.twitter.com/streaming/overview [Last Accessed: 14 May 2017]

[8] Scikit-learn, Machine Learning in Python. URL: http://scikit-learn.org/stable/ [Last Accessed: 14 May 2017]

[9] Raschka, S. (2015). *Python Machine Learning* (Book). Birmingham: PACKT.

[10] *What are the differences between AUC and F1-score?* URL: https://stats.stackexchange.com/questions/123036/what-are-the-differences-between-auc-and-f1-score [Last Accessed: 14 May 2017]

[11] Microsoft: *How to choose algorithms for Microsoft Azure Machine Learning*. URL: https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-choice [Last Accessed: 14 May 2017]

[12] Data School: *Simple guide to confusion matrix terminology*. URL: http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/ [Last Accessed: 14 May 2017]

[13] Machine Learning Mastery: *A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning*. URL*:* http://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/ [Last Accessed: 14 May 2017]

[14] Introduction to machine learning with scikit-learn. URL: https://github.com/justmarkham/scikit-learn-videos [Last Accessed: 14 May 2017]

[15] Coursera: Machine Learning by Andrew Ng. URL: https://www.coursera.org/learn/machine-learning [Last Accessed: 14 May 2017]

[16] How to choose between ROC AUC and F1 score? URL: https://stats.stackexchange.com/questions/210700/how-to-choose-between-roc-auc-and-f1-score [Last Accessed: 14 May 2017]

[17] Scikit-learn issue: *average_precision_score does not return correct AP when all negative ground truth labels*. URL: https://github.com/scikit-learn/scikit-learn/issues/8245 [Last Accessed: 14 May 2017]

[18] Scikit-learn: *Support Vector Machines*. URL: http://scikit-learn.org/stable/modules/svm.html [Last Accessed: 14 May 2017]

[19] Bruce Eckel: *Thinking in Java* (Book). Prentice Hall, Pearson Education.

[20] Scikit-learn: *Ensemble Methods*. URL: http://scikit-learn.org/stable/modules/ensemble.html#ensemble [Last Accessed: 14 May 2017]

[21] Ian Goodfellow, Yoshua Bengio and Aaron Courville: Deep Learning Book. URL: http://www.deeplearningbook.org/ [Last Accessed: 14 May 2017]

[22] W3 Schools: The world's largest web developer site. URL: https://www.w3schools.com/ [Last Accessed: 14 May 2017]

[23] Analytics Vidhya: *Quick Introduction to Boosting Algorithms in Machine Learning*. URL: https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/ [Last Accessed: 14 May 2017]

[24] Scikit-learn: Precision - Recall - F1 Score Metrics. URL: http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html [Last Accessed: 14 May 2017]