THE UNIVERSITY OF
SYDNEY

School of Information Technologies

The University of Sydney

COMP5318 – Assignment 1

Classification Model

by

Cameron Wasilewsky
Kristopher Lopez
Arjun Sathasivam

Submitted: 8 May 2017

Student IDs:

cwas5136 – 470267240

klop5007 – 470236420

asat9100 – 470367366

Originality Statement

We hereby declare that this submission is our own work and to the best of our knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at The University of Sydney or any other educational institution, except where due acknowledgement is made in the document. Any contribution made to the research by others, with whom we have worked with, is explicitly acknowledged in the document. We also declare that the intellectual content of this document is the product of our own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Signed – Cameron Wasilewsky - …….....

Signed – Kristopher Lopez - …….....

Signed – Arjun Sathasivam - …….....

Date …….Monday 8th of May 2017…….....

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1 Aim

Educationally the main aim of this study was to provide exposure to a range of topics; classification, sparse data and high dimensionality. Classification problems are common within machine learning as many problems fall into associating similarity and identifying labels. The data provided for this study was high dimensional with sparse non-zero values a situation common to a text classification problem. Finally, the high dimensional~~ity of the~~ feature space provided understanding of brute force ~~gear~~ feature selection and the impact of naïve search processes (K Nearest Neighbours), encouraging researchers to focus on efficiency rather than simple accuracy.

There is also a more human driven aim of this study which can be expressed through two lenses; that of the authors and the second of society. The former is straightforward, the research aims to provide those involved with practical experience in data mining algorithms, their implementation and design [1], as aforementioned. By doing so an understanding of different models, their applications, benefits and draw backs are understood. Finally, the last individual based aim is to improve communication, of complex concepts, data and information. These aims are individual focused, however, the aim towards society is more holistic and altruistic. Such a study being undertaken by over 200 highly intelligent and motivated individuals at one of the best universities in Australia, upskills these individuals, and provides them with the skills and understanding to apply these techniques in a range of industries, ultimately driving growth, development and improved standards of living for those impacted.

2 Importance

In the last decade, there has been exponential growth in the access to data, information and processing resources. The ability to analyse and pull valuable insight from this data has become the next bottleneck. Machine Learning and Deep Learning are relatively new techniques that are showing huge value in this area. The techniques can ingest huge volumes of data, learn from it and apply this to unseen inputs. The applications of such technology are self-evident essential. These methods have the potential to solve problems that humans are yet to comprehend, and in fact are already doing so in areas, such as classification of cancerous melanomas and fraud detections [2]. It is a step in the right direction to tackling problems that will alter human life, wealth and health. This study is important because of the impact the learnt skills and developed techniques will have going forward.

This study has emphasised the importance of model selection, reinforcing that a brute force is not an option for large high dimensional data sets. Rather, critical thinking, analysis planning and assessment is required to tackle the problem, requiring domain knowledge, model benefits and costs as well as the skills and tools to bring the solution to fruition.

Chapter 2: Methods

This chapter details the methodology utilised to build a high accuracy, low processing power model, whilst ensuring that it is validated and verified, accurately and holistically compared to other models and implementations. The design of the system was split into three core processes, pre-processing, preparing the data and the classifier model architecture. Section 1 discusses two elements of pre-processing. The first is joining the data with its associated labels as well as formatting text to integer ids to allow matrix evaluation in the 'Numpy' Python Package. The second is feature extraction and reduction. The aim was to remove correlated data and reduce the high dimensionality to improve performance without impacting accuracy. Section 2 focuses on the model architecture and design utilised. Three classifiers were developed; K-Nearest Neighbours (k-NN), Naïve Bayes (NB) and Logistic Regression (LR); and later combined in an Ensemble Model.

1 Pre-processing

The data set provided was an app name and its associated description, a 'tf-idf' [3] calculation was applied to the description resulting in the final set having 20104 rows by 13627 columns. This is a huge text data set and culminated in long load times and complex calculations. To improve performance, it was essential to utilise the most efficient functions in preparing the data as well as reducing the dimensionality.

1.1 Join & Load

As the source data was 0.5 GBs, preliminary testing was done to find the quickest method of loading the data. From research, it was found that the Python package 'Pandas' was able to load the data six times faster than 'Numpy'. This method allowed for improved performance (comparison seen in Table 1).

Table 1: Load function comparison across a range of systems

Computer Specification	Package	Load Time
i5-6500 3.20Ghz; 16GB Memory	numpy.loadtxt	23.46 min
i5-6500 3.20Ghz; 16GB Memory	numpy.genfromtxt	6.13 min
MacBookPro i5 2.50Ghz; 8GB Memory	numpy.loadtxt	64.32 min
MacBookPro i5 2.50Ghz; 8GB Memory	numpy.genfromtxt	6.13 min
DELL i5-6300U 2.40Ghz; 8GB Memory	Pandas	0.93 min
MacBookPro i5 2.50Ghz; 8GB Memory	Pandas	1.26 min
i5-6500 3.20Ghz; 16GB Memory	Pandas	0.78 min

The initial data set was split between labels and tf-idf values. To ensure accurate labelling Python's merge function was used to align the two data sets. App names were then removed and category labels were converted to integer ids (0-29). Finally, the data structure was in a useable format and was converted to a 'Numpy' array to be used by later functions.

Two features had 0 tf-idf values but were unchanged due to the use of SVD for both Logistic Regression and Nearest Neighbour models.

1.2 Feature Reduction

A critical part of the machine learning process is to analyse and simplify the features being used as inputs to extract the core information, and reduce the required processing power.

The feature selection process has several benefits, reducing the number of features to a more meaningful and manageable size. This is indispensable when dealing with large, sparse data sets (including data sets based on text extraction, such as the subject of this project).

Many methods for feature reduction are frequently used. Examples of these include Singular Value Decomposition (SVD), Principal Components Analysis (PCA), Linear Discriminant Analysis (LDA) and Kernel Principal Components Analysis (Kernel PCA). These methods are investigated and assessed for the purposes of this project with sklearn's "Truncated SVD" used as a benchmark.

To assess the performance of each of these methods, the data set was first transformed, then trained using a logistic regression. If the run time of the regression is at least no worse (and there was no deterioration in the f-score), then that feature selection method is considered successful.

1.2.1 PCA

The first feature selection method evaluated was Principal Components Analysis. There are multiple algorithms for calculating PCA [4].

The data matrix is first centred by subtracting the mean of each column from every value in that column. Then, a singular value decomposition is performed on the result, and the required number of eigenvalues and eigenvectors are kept. The remaining eigenvalues (and corresponding eigenvectors) are discarded.

A range of scenarios involving the number of eigenvalues to keep were tested. It was determined that 2,500 out of 13,600 components was sufficient to improve the run time of the classifier without detracting from its accuracy. 2,500 components resulted in retaining approximately 60% of the total variance inherent in the original data set, as shown in Figure 1.

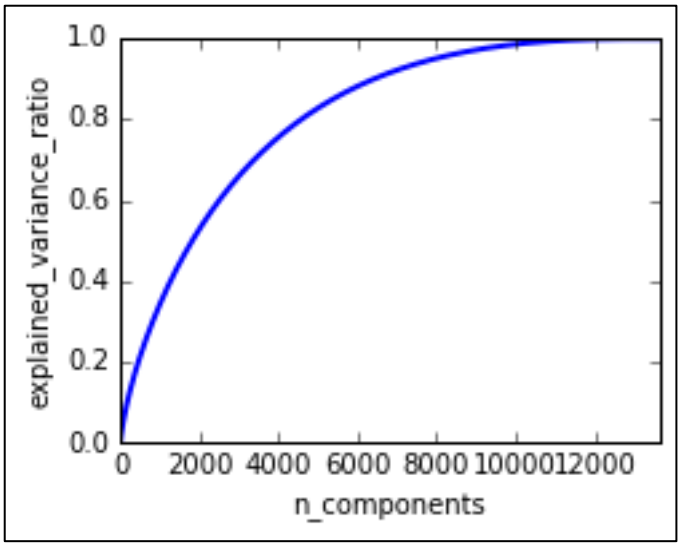

Figure 1: Explainable variance by the number of components

1.2.2 SVD

SVD follows the same process as PCA, apart from retaining the original non-centred data before performing the SVD, with clear advantage being its ease of use on non-square matrices.

SVD is more suited to tf-idf data than PCA [5], as it does not require centring the original data matrix (and hence it is able to retain inherent sparsity, which allows for more memory-efficient processing).

As with PCA, approximately 60% of the variance is explained by the first 2,500 components of the original data set.

1.2.3 LDA

Linear Discriminant Analysis is a commonly used feature reduction technique, with inherent differences to PCA and SVD.

Both PCA and SVD are unsupervised reduction tools – they only maximise variance between features and do not use any knowledge of the underlying classes. LDA, on the other hand, uses knowledge of the training classes to maximise variance between features and across classes.

Despite its advantages, LDA is fundamentally unsuitable for use in this problem. LDA inherently reduces the feature set to $C - 1$ features, where C is the number of total number of classes (in this case, 30). Reducing approximately 13,600 features to 29 results in an unacceptable

loss of information, and renders classification impossible.

1.2.4 Kernel PCA

PCA, SVD and LDA are valuable feature reduction tools, but are limited by only being able to transform the feature set into a linear subspace.

Kernel PCA makes use of the kernel trick to identify non-linear feature mappings. It works by applying a kernel transformation to the underlying data, before performing a standard PCA [6]. The transformation (when selected appropriately) allows a wide scope of non-linearity. A straightforward algorithm is described by Sebastian Raschka [7].

The kernel PCA was evaluated using existing Python packages (sklearn's KernelPCA), but its poor performance did not justify its use in final classification. The kernel used was a Gaussian radial basis function, defined as

$$k(x_i, x_j) = \exp(-\gamma \left|\left|x_i - x_j\right|\right|^2) \tag{1}$$

where x_i, x_j are pairs of features, and γ is a hyperparameter.

1.2.5 Performance comparison

A comparison of each of the feature selection methods is given in Table 2. Plots of the time taken to run against the f-score (defined as $2\times(\text{precision} \times \text{recall})/(\text{precision} + \text{recall})$) are given in Figure 2.

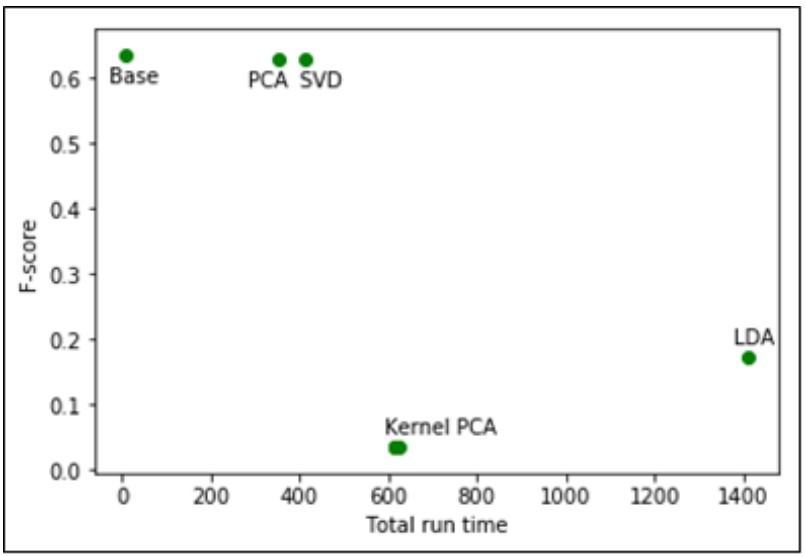

Figure 2: Comparison of run time by F-score for feature selection methods

Table 2: Feature selection comparision

Feature selection method	Hyperparameters	Time to run (seconds)	Time to train logistic regression (seconds)	Average accuracy	Average recall	f-score
None	-	-	8	0.6384	0.6319	0.6315
PCA	2500 principal components	256	112	0.6333	0.6270	0.6264
SVD	2500 components	277	110	0.6315	0.6251	0.6249
LDA	-	1418	9	0.1641	0.1644	0.1758
Kernel PCA	2500 components Gaussian RBF kernel $\gamma = 0.001$	575	48	0.3124	0.0550	0.0358
Kernel PCA	2500 components Gaussian RBF kernel $\gamma = 0.01$	564	48	0.3124	0.0550	0.0358
Kernel PCA	2500 components Gaussian RBF kernel $\gamma = 0.1$	565	48	0.3124	0.0550	0.0358
Kernel PCA	2500 components Gaussian RBF kernel $\gamma = 1$	568	48	0.3124	0.0550	0.0358
Kernel PCA	2500 components Gaussian RBF kernel $\gamma = 10$	569	48	0.3124	0.0550	0.0358

2 Classifier

Classification of data has been an area of research in computer science for many decades and thus many models have been developed. For the purposes of this research three main models were developed. They were chosen for their simplicity, interpretability and maturity. Each method has their own benefits and drawbacks, thus by implementing an ensemble of the three the intention was to bring forward the benefits averaging out their drawbacks.

There are many classification models available, when selecting which model to develop a range of attributes were explored. To do this exploration 'sklearn' was used to determine which models would:

1. Run in a reasonable amount of time (<30 minutes)
2. Provide high accuracy (60+%)
3. Developed in a reasonable amount of time
4. Not prone to overfitting

The models that were tested with 'sklearn' were K Nearest Neighbours, Nearest Centroid Classifier, Multinomial Naïve Bayes, Gaussian Naïve Bayes, Logistic Regression, Ridge Classifier, SGD Classifier, Linear SVC (with hinge loss – approximates SVM) and Random Forest. XGBoost was also extensively applied to the dataset to determine if this state-of-the-art ensemble classifier would be able to set a high benchmark. Each model was also tested across varied numbers of dimensions reduced by Singular Value Decomposition and a variety of different hyperparameters.

Excelling in both run-time and accuracy, the Naïve Bayes and Logistic Regression (with L2 regularisation) classifiers were selected for development. K-Nearest Neighbours was also chosen due to ease of implementation and possible benefit within an ensemble model.

2.1 K-Nearest Neighbours (k-NN)

Nearest Neighbours is a widely-used algorithm given that it is easy to implement and interpret. It is also a 'lazy learner', not able to learn a function from the training data but needing to scan through the training set to generate each new prediction [8]. For high dimensional text classification problems such as the one in question, the training step can require considerable runtime, and accuracy can suffer due to the 'curse of dimensionality' [9].

A naïve implementation of nearest neighbours requires *O(ndk)* computations. Where *n* is observations in training set, *d* is the number of dimensions and *k* the number of neighbours. This therefore leads to many calculations. The KNN calculation process is as follows

- For each observation in the test set
    - Loop through the training set and calculate the Euclidean distance between the training set and training set observation.
    - Record each distance in an array, sort the array from the largest to smallest and return the indices of the first k entries [10].

Nearest Neighbours uses Euclidean distance to classify, and is therefore susceptible to the 'curse of dimensionality'. The curse of dimensionality occurs in high-dimensional spaces, such as the text classification problem in question. High-dimensional space forces the algorithm to traverse mostly empty space and degrades the potential 'nearness' between two points.

2.2 Naïve Bayes (NB)

The Naive Bayes classifier is widely used in text classification with applications in email spam detection and document categorization [11]. In a text classification setting, Naïve Bayes calculates the probability that a sample belongs to a class given the data within that sample. This posterior probability is then calculated for each class and the class with the highest probability selected for prediction. This posterior probability, by Bayes Theorem, is calculated by the product of the conditional probability of data within that sample given a class multiplied by the prior probability for that class (equations 1 and 2) [12].

$$P(C_j | \boldsymbol{x}) = \frac{P(\boldsymbol{x} | C_j) \cdot P(C_j)}{P(\boldsymbol{x})} \tag{2}$$

where Cj = 'jth' Class; x = sample vector

$$posterior\ probability = \frac{conditional\ probability \cdot prior\ probability}{evidence} \tag{3}$$

2.2.1 Conditional probabilities

The conditional probability of the vector given a class is the product of the conditional probabilities of each feature within that sample given a class. This is the assumption that gives the classifier its 'naïve' moniker, by assuming features are conditionally independent of one another [5].

$$P(\boldsymbol{x} \mid C_j) = P(x_1 \mid C_j) \cdot P(x_2 \mid C_j) \cdot \ldots \cdot P(x_d \mid C_j) = \prod_{i=1}^{d} P(x_i \mid C_j) \text{ [12]} \tag{4}$$

In the general text classification, the class conditional probabilities of each feature are calculated as the count of feature x and class C occurring divide by the count of class C (equation 4) [13]:

$$P(x_i \mid C_j) = \frac{Count(x_i \text{ and } C_j)}{Count(C_j)} \tag{5}$$

Given that only tf-idfs were provided, it was converted to the sum of tf-dfs of a feature within a class divide by the sum of tf-idfs of that class (equation 5):

$$P(x_i \mid C_j) = \frac{Sum(x_i \text{ given } C_j)}{Sum(C_j)} \tag{6}$$

To prevent the conditional probabilities from going to zero, and therefore creating a bias toward the class associated with the zero probability, Laplace smoothing has been adapted and applied (equation 6) [12]:

$$P(x_i \mid C_j) = \frac{Sum(\ x_i\ given\ C_j) + 1}{Sum(C_j) + V} \tag{7}$$

where V, the vocabulary, is equivalent to the number of features within the dataset.

2.2.2 Prior probabilities

The prior probabilities are traditionally calculated as the count of the class divide by the number of samples within the data set (equation 7):

$$P(C_j) = \frac{Count(samples\ in\ C_j)}{Count(all\ samples)} \tag{8}$$

With the introduction of tf-idfs this has been adapted to (equation 8):

$$P(C_j) = \frac{Sum(tf - idf\ of\ samples\ in\ C_j)}{Sum(tf - idf\ of\ all\ samples)} \tag{9}$$

2.2.3 Evidence

When comparing classes, the evidence of a sample within the data is equal and therefore can be ignored.

2.2.4 Posterior probabilities

Many computers are unable to handle numbers with decimal points, often rounding them to zero, so to handle this problem logs of both the prior and conditional probabilities are taken (as log is a monotonically increasing function [14]). The log probabilities are calculated for each class and the maximum is chosen for prediction (equation 9) [14].

$$C_{pred} = \arg\max\left(\log P(c) + \sum_{i=1}^{d} \log P(x_i \mid C\right) \tag{10}$$

2.3 Logistic Regression

Logistic regression is a regression model that can be used for binary classification by estimating probabilities for each class [15]. Linear models such as logistic regression, ridge regression and support vector machines work well for text classification because of the high dimensionality, linear separability and importance of features [16]. This was corroborated during the explorations with 'sklearn'. Logistic regression with L2 regularisation (ridge regression) was selected due to faster run-time and higher accuracy.

The logistic function is:

$$P(C|\boldsymbol{x}) = h_w(\boldsymbol{x}) = \frac{1}{1+e^{-f(x)}} \tag{11}$$

$$f(x) = w_0 + \sum_{i=1}^{d} w_i x_i \tag{12}$$

The weights, **w,** are then discovered by minimising the cost function in equation 12:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_w\left(x^{(i)}\right)\right) - (1-y^{(i)})\log(1- h_w\left(x^{(i)}\right))\right] \tag{13}$$

The regularisation term λ is then added:

$$J(w) = J(w) + \frac{\lambda}{2m}\sum_{i=1}^{d} w_i^2 \tag{14}$$

which through calculus reveals the below gradient for the cost function (equation 14) [17]:

$$\frac{dJ(w)}{dw_j} = \frac{1}{m}\left(\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \lambda w_j\right) \tag{15}$$

To extend to multiclass classification:

The *m x 1* multiclass label, with *o* unique labels, is converted to *m x o* binary labels. A logistic model is trained on each label and the probability that the label belongs to that class recorded in an array. The label with the highest probability is used for prediction

$$C_{pred} = \arg\max(P_o(C\ |x\ )) \tag{16}$$

2.4 Ensemble Model

An ensemble combines multiple models in the attempt to achieve better predictions [18]. Ensembles are often used with non-parametric algorithms, such as Random Forests, by combining a sequence of "weak learners" to create a "strong learner". The models are combined to average bias, reduce variance and minimize the impact of overfitting [19].

When results between "strong learners" are not correlated a weighted voting classifier which selects the majority vote between each classifier [20].

$$C_{pred} = \text{maxcount } (v_{lr}\hat{y}_{lt}, \quad v_{nb}\hat{y}_{nb}, v_{nn}\hat{y}_{nn}) \tag{17}$$

Where v is the number of votes associated with the y predictions of each participating model, logistic regression (lr), naïve bayes (nb) and nearest neighbours (nn).

Chapter 3: Experiments and Results

Chapter 3 discusses in detail the performance, accuracy and value of the developed classification models. Many models were developed, and tested on a range of hyperparameters. As a result, this chapter is split into three main sections. The first is address the best performing of each model, addressing its accuracy and best hyperparameters. The second section is a high-level comparison of the best performing models calling out valuable insights and differences. Finally, the third section summaries the results of this research and the impact it has on the area of study.

1 Model Comparison

Model selection and establishing appropriate benchmarks were immediately important. To do this, sklearn was employed to test a range of different classifiers with a range of hyperparameters to develop expectations and benchmarks. The following criteria were ascertained from the exploration: run time, test accuracy, train accuracy vs test accuracy (overfitting) and model complexity.

The results of the initial investigation into potential classifiers is shown below tables and charts.

Table 3: Classifier analysis and assessment

Model[1]	α	c	e[2]	Accuracy (Training)	Accuracy (Test)	Recall (Training)	Recall (Test)	F-Score (Training)	F-Score (Test)
GNB				0.9171	0.4355	0.9171	0.4355	0.9186	0.4366
LR		0.1		0.6632	0.5934	0.6632	0.5934	0.6479	0.5751
LR		0.5		0.7468	0.6359	0.7468	0.6359	0.7432	0.6309
LR		1		0.7936	0.6481	0.7936	0.6481	0.7917	0.6441
LR		1.5		0.8234	0.6549	0.8234	0.6549	0.8219	0.6514
LR		2.5		0.8640	0.6571	0.8640	0.6571	0.8631	0.6544
LR		3.5		0.8887	0.6561	0.8887	0.6561	0.8881	0.6537
LR		5		0.9137	0.6546	0.9137	0.6546	0.9134	0.6527
LR		10		0.9506	0.6466	0.9506	0.6466	0.9505	0.6462
MNB	0.001			0.8917	0.6088	0.8917	0.6088	0.8915	0.6059
MNB	0.01			0.8696	0.6250	0.8696	0.6250	0.8690	0.6222
MNB	0.05			0.8396	0.6322	0.8396	0.6322	0.8386	0.6284
MNB	0.1			0.8213	0.6319	0.8213	0.6319	0.8198	0.6262
MNB	0.15			0.8100	0.6337	0.8100	0.6337	0.8080	0.6270
MNB	0.2			0.8005	0.6307	0.8005	0.6307	0.7979	0.6233

[1] GNB = Gaussian Naïve Bayes, LR = Logistic Regression, MNB = Multinomial Naïve Bayes, RF = Random Forest, RC = Ridge Classifier, SVM = Linear SVC, NN = Nearest Neighbours, NC = Nearest Centroid

[2] α, c and e are hyperparameters required for the classifiers specified

MNB	0.25			0.7926	0.6287	0.7926	0.6287	0.7897	0.6208
RF			50	0.9975	0.6001	0.9975	0.6001	0.9975	0.5824
RF			100	0.9976	0.6133	0.9976	0.6133	0.9976	0.5967
RF			200	0.9976	0.6252	0.9976	0.6252	0.9976	0.6100
RF			500	0.9976	0.6280	0.9976	0.6280	0.9976	0.6114
RF			1000	0.9976	0.6309	0.9976	0.6309	0.9976	0.6158
RF			2000	0.9976	0.6304	0.9976	0.6304	0.9976	0.6153
RC		0.1		0.9406	0.6250	0.9406	0.6250	0.9404	0.6193
RC		0.5		0.8957	0.6486	0.8957	0.6486	0.8949	0.6407
RC		1		0.8671	0.6531	0.8671	0.6531	0.8655	0.6441
RC		2.5		0.8229	0.6543	0.8229	0.6543	0.8198	0.6437
RC		5		0.7853	0.6535	0.7853	0.6535	0.7804	0.6419

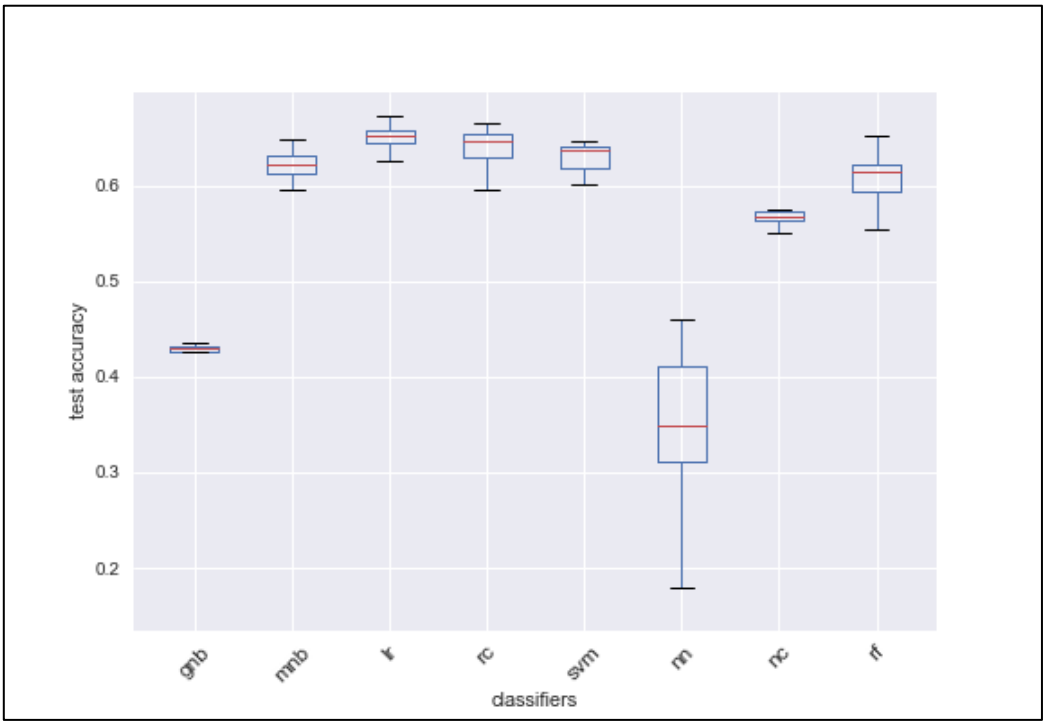

Figure 3: Classifier comparison, test accuracy box plot

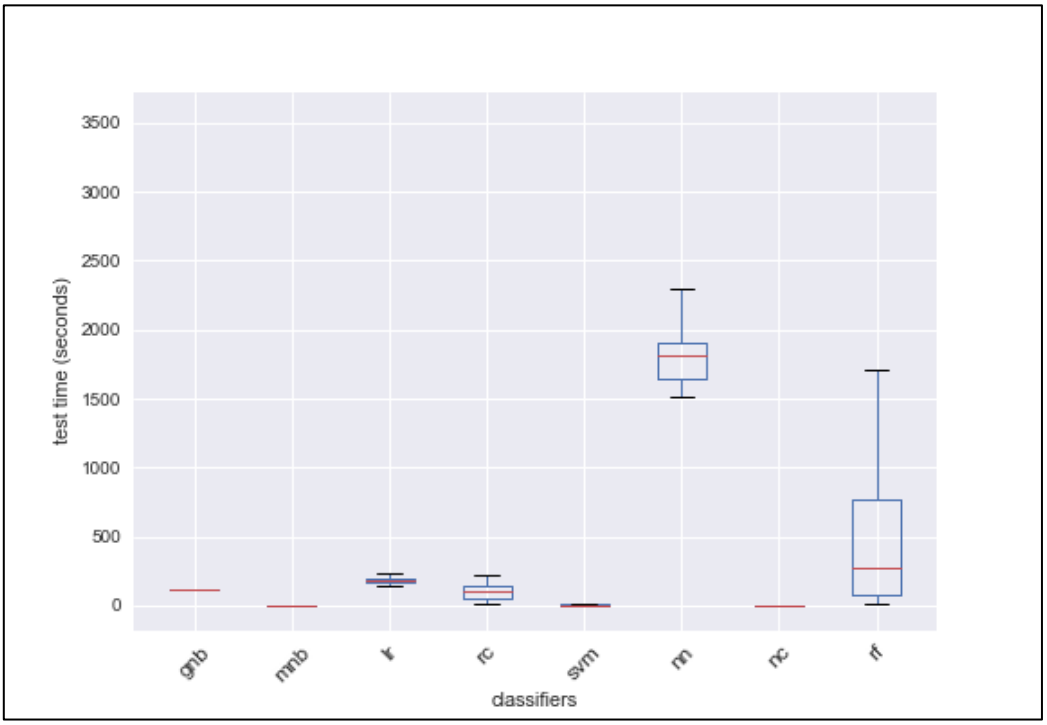

Figure 4: Classifier comparison, test time box plot

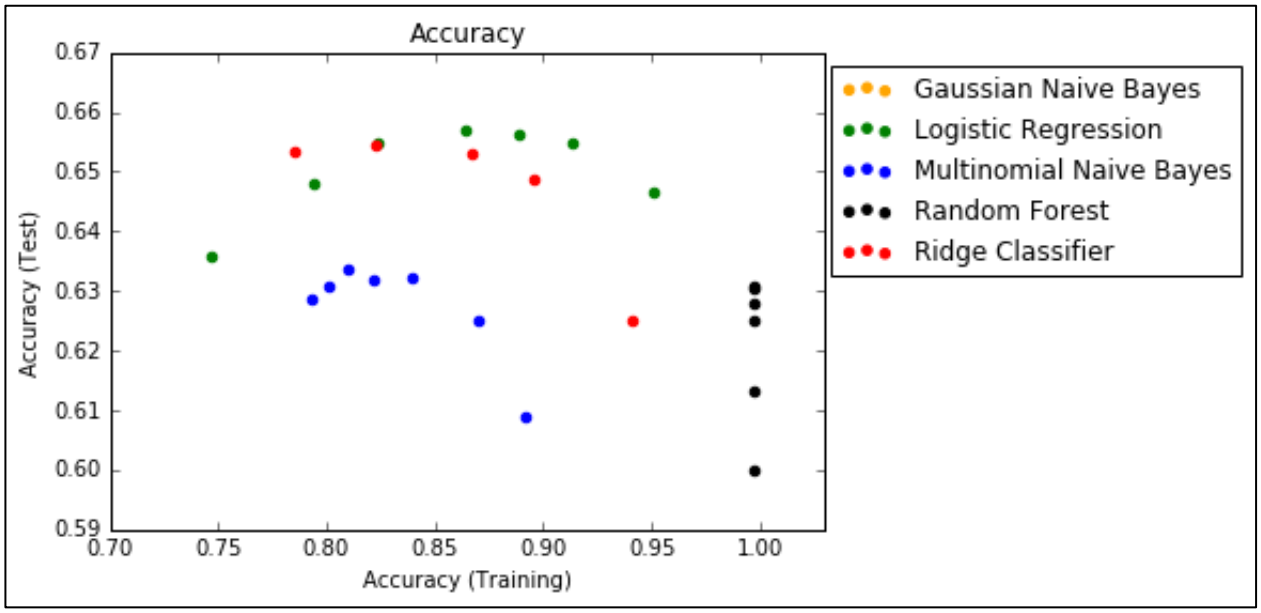

Figure 5: Classifier comparison, training accuracy against test accuracy

The charts (Figure 3, Figure 4, Figure 5) demonstrate the following:

- Of the Bayesian models, Multinomial posteriors are more predictive for this problem than Gaussian. The multinomial naïve bayes model also had low standard deviation, low runtime and low complexity to implement.

- Of the Linear models, Logistic Regression outperformed sklearn's Ridge Classifier and Linear SVC with hinge (approximates SVM). Logistic regression also had lower standard deviation than the other models, with a small sacrifice in runtime and less complexity to implement than SVC.

- Of the Lazy Learning models, Nearest Neighbours was significantly the worst model based on runtime, accuracy but due to its ease to implement it was included for its potential benefit within an Ensemble.

- Of the Non-linear models, Random Forest and XGBoost both performed worse than Linear models, took much longer to run and suffered from overfitting on the training data. For these reasons, Random Forest was not chosen to be developed.

1.1 Sklearn Verse Hand-coded Models

The next stage in the model selection process was to code the classifiers of choice and compare their performance to sklearn's equivalent implementations. It was desired to ensure that run times of the classifiers developed were within acceptable ranges.

The results of these comparisons are given below.

1.1.1 K-Nearest Neighbours

- Processor Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 3201 Mhz, 4 Cores
- 16GB RAM
- Data split: 90% training / 10% test

Table 4: KNN Sklearn vs Hand-coded

sklearn model	Hand-coded model	sklearn run time (s)	Hand-coded run time (s)
k-nearest neighbours $k = 10$	**k-nearest neighbours** $k = 10$	75.1	275.5
k-nearest neighbours $k = 50$	**k-nearest neighbours** $k = 50$	76.3	244.2

1.1.2 Naïve Bayes

- Processor Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 3201 Mhz, 4 Cores
- 16GB RAM
- Data split: 90% training / 10% test

Table 5: NB Sklearn vs Hand-coded

sklearn model	Hand-coded model	sklearn run time (s)	Hand-coded run time (s)
Multinomial Naïve Bayes $\alpha = 0.1$	**Multinomial Naïve Bayes** $\alpha = 0.1$	0.7	12.4
Multinomial Naïve Bayes $\alpha = 0.2$	**Multinomial Naïve Bayes** $\alpha = 0.2$	0.8	12.4

1.1.3 Logistic Regression

Processor Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 3201 Mhz, 4 Cores

16GB RAM

Data split: 90% training / 10% test

Table 6: LR Sklearn vs Hand-coded

sklearn model	Hand-coded model	sklearn run time (s)	Hand-coded run time (s)
Logistic Regression $c = 0.5$	**Logistic Regression** $c = 0.5$	85.8	280.0
Logistic Regression $c = 2.5$	**Logistic Regression** $c = 2.5$	96.3	262.1

All hand-coded run times, while longer than their sklearn equivalents, are still acceptably fast.

1.2 Hyperparameter Optimisation

Finally, a grid search was run within each classifier to identify the optimal hyperparameters to use.

1.2.1 K-Nearest Neighbours

Table 7: KNN Hyperparameter comparison

k	Accuracy	Precision	Recall	F1 Score
10	0.5311	0.5947	0.5311	0.5463
20	0.5569	0.5884	0.5569	0.5550
35	0.5525	0.5822	0.5525	0.5455
50	0.5420	0.5879	0.5420	0.5364

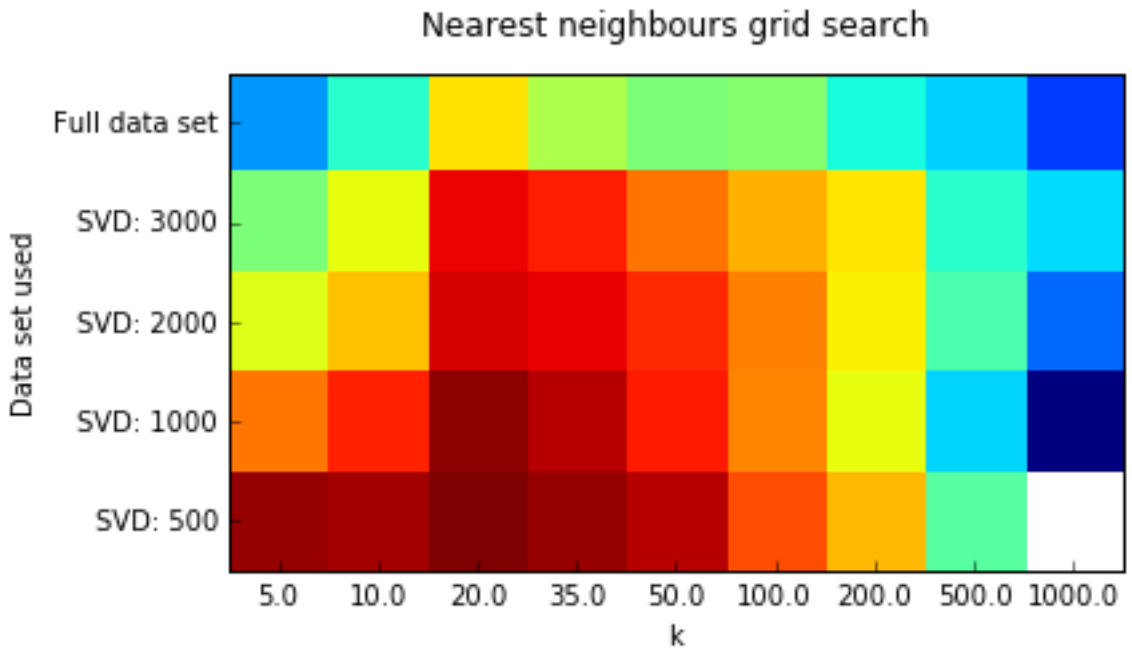

Figure 6: KNN Hyperparameter Heatmap, where darker red is higher accuracy

The optimal number of neighbours to search for appears to lie somewhere between 20 and 35. In addition, using a small number of principal components (between 500 and 1000) appears to improve the accuracy of the classifier.

1.2.2 Naïve Bayes

Table 8: NB Hyperparameter comparison

α	Accuracy	Precision	Recall	F1 Score
0	0.6183	0.6280	0.6183	0.6088
0.1	0.6154	0.6293	0.6154	0.6054
0.2	0.6116	0.6294	0.6116	0.6008
0.3	0.6072	0.6289	0.6072	0.5950
0.4	0.6030	0.6282	0.6030	0.5892
0.5	0.5984	0.6271	0.5984	0.5833
0.6	0.5939	0.6263	0.5939	0.5777
0.7	0.5893	0.6244	0.5893	0.5720
0.8	0.5847	0.6208	0.5847	0.5665
0.9	0.5797	0.6183	0.5797	0.5608
1	0.5734	0.6130	0.5734	0.5536

The ideal α here is likely to be zero – the accuracy, recall and f-score all fall as α is increased.

1.2.3 Logistic Regression

Table 9: LR Hyperparameter comparison

c	Accuracy	Precision	Recall	F1 Score
0.1	0.6126	0.6209	0.6126	0.5967
0.5	0.6385	0.6459	0.6385	0.6335
1	0.6450	0.6524	0.6450	0.6423
2.5	0.6425	0.6493	0.6425	0.6419
5	0.6375	0.6438	0.6375	0.6372
10	0.6266	0.6321	0.6266	0.6266

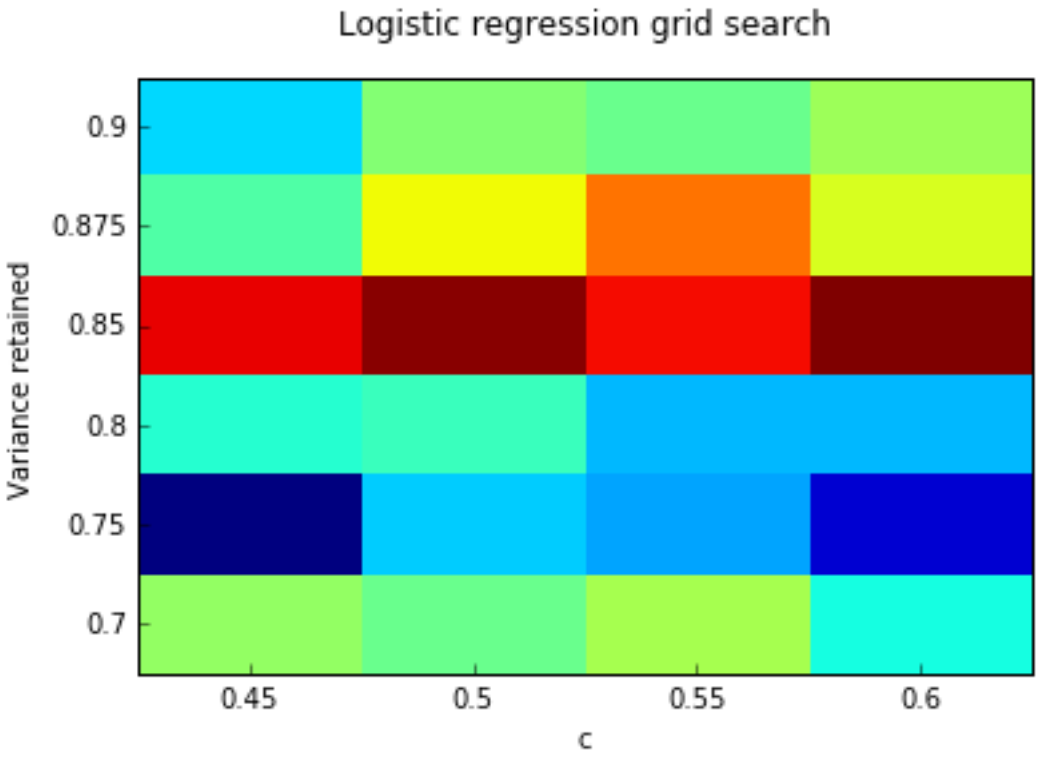

Figure 7: LR Hyperparameter Heatmap, where darker red is higher accuracy

The grid search for logistic regression parameters resulted in minimal deviations around the parameters chosen, which is clear from the inconsistent variation in colour. However, a c-value of 0.55 with approximately 85%-90% of total variance retained from the SVD pre-processing stage still appears to be optimal.

Testing was also performed on a variety of methods of convergence used by scipy's optimize. 'L-BFGS-B' was selected as it was 2x faster than 'BFGS-B' and others, such as 'TNC' (Truncated Newton) failing to converge in less than 2 hours.

2 Model Analysis

The following analysis is a detailed review of the best performing predictions for the developed logistic regression, naïve bayes, nearest neighbour and ensemble model classifiers. The hyperparameters for these models were found with a grid search approach of the data discussed in the previous section. The results are from a computer with the following specifications:

Table 10: System Model analysis was run on

Processor	RAM	System Type	Operating System
Intel(® Core™ i702600 CPU @ 3.40GHz	8.00GB	64-Bit x64-based processor	Windows 10 Home

For each model the key metrics have been calculated as well as the confusion matrix. This evaluation gives a clear indication of the performance for each model and ultimately which set up has been implemented in the final model.

2.1 K-Nearest Neighbours – K = 20

2.1.1 Metrics

Table 11: Metrics for KNN

Input Data	Lift	Accuracy	Precision	Recall	F Measure	Lines of Code	Time		
							Pre-process	Train (s)	Predict (s)
SVD, 500 Components	16.62	0.554	0.558	0.592	0.555	36	8074.235	158.497	0.031

2.1.2 Confusion Matrix

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	55	2	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	5	0	2	0	3	0	0	1	0
1	3	38	0	0	0	1	0	4	9	1	2	5	0	1	0	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0
2	12	0	48	0	4	4	0	0	10	2	0	0	0	0	1	0	2	0	1	0	1	1	0	0	0	0	1	0	0	0
3	1	1	0	31	0	0	0	2	1	0	5	0	1	3	3	0	1	1	0	0	6	0	1	3	0	0	0	3	1	0
4	4	1	2	1	62	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	19	0	13	0	4	25	2	0	7	2	0	1	0	0	0	0	4	0	0	3	0	0	0	0	0	1	0	1	0	0
6	8	3	0	0	0	2	15	2	0	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	4	0	0	0	34	0	1	1	2	0	0	0	0	0	0	3	0	3	0	1	6	0	0	5	4	0	0
8	6	5	5	1	1	2	0	2	28	1	1	1	2	0	0	1	0	0	1	1	0	1	0	0	0	0	1	1	1	0
9	13	0	1	2	0	1	0	4	2	18	0	4	0	1	7	1	6	1	3	6	1	1	1	3	1	1	0	0	0	2
10	1	1	0	4	0	0	0	0	0	0	63	1	0	0	0	0	0	0	0	0	1	0	1	2	0	0	0	0	0	0
11	1	1	0	1	0	1	1	1	2	0	0	40	0	1	0	6	1	0	1	0	1	2	1	0	0	0	2	2	0	0
12	1	5	1	3	1	0	0	1	0	1	1	0	17	0	1	0	0	0	2	3	3	0	0	0	0	1	4	1	3	0
13	4	9	1	2	2	4	0	2	3	1	1	7	0	12	1	0	2	1	3	6	4	0	4	2	0	0	2	3	5	0
14	4	0	0	2	0	1	0	0	1	5	1	0	1	0	40	0	9	1	2	7	3	0	0	0	1	0	0	0	0	0
15	1	4	0	0	0	0	0	1	2	0	0	7	0	0	0	48	1	0	0	1	1	0	0	0	0	0	0	0	1	0
16	3	3	0	1	0	0	0	1	2	0	0	4	0	0	2	0	60	1	0	1	2	0	0	0	1	0	0	0	1	0
17	2	2	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0	44	0	0	0	1	0	0	0	0	2	1	1	0
18	1	0	0	0	0	0	0	4	0	0	0	0	0	1	0	0	3	1	40	2	3	0	0	0	0	0	1	0	0	0
19	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	4	48	0	1	0	1	0	0	1	1	0	0
20	4	1	0	8	0	0	2	8	0	0	0	1	0	0	2	0	1	0	5	2	20	0	1	1	0	0	12	3	1	0
21	10	0	3	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	3	0	0	48	0	0	0	1	0	1	0	0
22	0	0	0	3	1	0	0	0	0	0	2	0	0	1	0	1	0	0	0	1	0	1	53	1	0	0	3	2	3	0
23	2	5	0	2	0	1	0	8	0	0	0	1	0	2	0	0	2	2	3	3	1	0	0	34	0	0	0	2	0	0
24	3	0	0	1	0	0	0	0	1	0	0	2	0	0	1	0	3	2	4	0	1	2	0	1	32	5	0	1	2	0
25	19	0	3	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	3	0	0	2	20	0	0	0	0
26	2	1	0	7	1	0	0	9	1	0	2	0	0	2	4	0	0	0	7	0	6	0	0	0	0	0	28	2	1	1
27	0	1	0	4	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	45	16	0
28	1	1	0	2	0	0	1	2	0	0	0	2	0	2	0	0	0	1	0	0	1	0	0	1	0	0	0	22	36	0
29	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0	0	0	1	0	0	0	3	3	33

Figure 8: K Nearest Neighbours Confusion Matrix Class Count

2.2 Naïve Bayes – Prior Weight = 0

2.2.1 Metrics

Table 12: Metrics for Naïve Bayes

Input Data	Lift	Accuracy	Precision	Recall	F Measure	Lines of Code	Time	
							Train (s)	Predict (s)
Full data set, unaltered	19.05	0.635	0.636	0.659	0.628	72	16.191	0.668

2.2.2 Confusion Matrix

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	53	0	2	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	2	1	0	8	0	0	0	2	0	1	0	0
1	0	36	0	2	0	0	0	2	6	0	2	4	0	7	0	1	1	1	0	0	1	0	0	3	0	0	2	0	2	0
2	4	0	62	0	3	4	0	0	8	1	0	0	0	0	0	0	2	0	0	1	0	1	0	0	0	0	1	0	0	0
3	0	0	0	32	0	0	0	3	1	0	5	0	0	1	3	0	1	1	0	0	7	0	7	2	0	0	0	0	1	0
4	1	0	1	0	67	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
5	12	0	12	0	4	36	1	0	2	2	0	1	0	0	0	1	2	0	1	3	0	2	0	3	0	0	0	0	0	0
6	6	1	0	0	0	0	10	1	0	1	0	1	0	0	0	0	0	0	7	6	0	0	0	1	0	0	1	0	0	0
7	0	0	1	2	0	0	0	42	0	0	0	1	1	0	1	0	0	0	4	0	3	0	2	4	0	0	1	1	1	0
8	0	0	3	1	0	3	0	1	41	1	1	0	0	0	1	1	0	0	1	2	0	0	0	0	1	0	2	1	2	0
9	1	0	0	0	1	7	0	5	1	19	0	3	0	2	12	0	4	2	1	11	0	0	0	6	4	1	0	0	0	0
10	0	0	0	2	0	0	0	0	0	0	69	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
11	0	1	0	0	0	0	0	0	0	0	1	49	0	1	0	4	2	0	0	1	0	0	0	0	1	0	2	2	1	0
12	1	5	0	1	0	0	0	1	0	1	1	1	25	0	1	0	0	0	3	3	3	0	0	0	0	0	1	0	2	0
13	1	11	1	2	0	1	0	3	0	1	1	4	0	24	0	0	3	0	1	6	3	0	7	4	1	0	3	1	2	1
14	0	0	0	2	0	1	0	2	1	3	0	0	1	0	47	0	8	1	2	7	1	0	0	0	0	0	1	1	0	0
15	0	0	0	0	0	0	0	1	1	0	0	9	0	1	0	53	1	0	0	1	0	0	0	0	0	0	0	0	0	0
16	1	3	0	0	0	1	0	0	1	0	0	1	0	2	3	0	60	3	1	0	1	0	0	1	1	0	2	0	1	0
17	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	50	0	0	0	0	1	1	0	0	0	1	0	1
18	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0	1	1	46	2	1	0	0	0	0	0	2	0	0	0
19	1	0	0	0	0	1	0	1	0	0	0	0	0	0	2	0	0	0	2	47	0	1	0	3	0	0	2	0	0	0
20	0	0	0	4	0	0	0	9	1	0	1	3	1	0	3	0	2	0	6	0	20	0	1	3	0	0	16	1	1	0
21	10	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	51	0	0	0	0	0	0	0	0
22	0	0	0	4	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	1	59	1	0	0	1	1	1	0
23	0	1	0	0	1	1	0	8	0	0	0	1	0	1	1	1	2	3	2	2	1	0	0	43	0	0	0	0	0	0
24	1	0	0	0	2	0	0	0	0	0	0	2	0	0	2	0	1	1	2	2	0	0	0	1	43	0	1	1	2	0
25	4	0	2	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	9	0	0	6	21	0	0	0	0
26	1	1	0	2	0	0	0	9	1	0	3	0	0	1	6	0	0	0	8	2	4	0	1	0	0	0	34	0	1	0
27	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	2	1	0	0	2	53	8	0
28	0	2	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	2	16	46	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	0	0	0	0	1	0	0	1	0	0	39

Figure 9: Naive Bayes Confusion Matrix Class Count

2.3 Logistic Regression – Alpha = 0.55

2.3.1 Metrics

Table 13: Metrics for Linear Regression

Input Data	Lift	Accuracy	Precision	Recall	F Measure	Lines of Code	Time	
							Train (s)	Predict (s)
Full data set, unaltered	19.35	0.645	0.650	0.658	0.645	121	1052.260	0.003

2.3.2 Confusion Matrix

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	49	1	2	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	7	0	0	0	4	0	1	0	0
1	0	39	0	0	0	0	0	2	5	0	2	4	0	7	0	1	1	1	0	2	0	0	0	3	0	0	1	0	2	0
2	5	0	64	1	3	3	0	0	5	2	0	0	0	0	0	0	2	0	0	0	1	0	0	0	0	1	0	0	0	0
3	0	0	0	41	0	0	0	2	0	0	3	0	0	1	3	0	1	1	0	0	7	0	3	1	0	0	0	0	1	0
4	0	0	1	0	67	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
5	10	1	12	0	3	35	1	0	5	1	1	1	0	0	0	1	3	0	1	2	0	1	0	2	0	1	0	1	0	0
6	5	2	0	0	0	0	15	0	0	1	0	0	0	1	0	0	1	0	7	1	0	0	0	1	0	0	1	0	0	0
7	0	1	0	2	0	0	0	42	0	0	0	0	1	0	1	1	0	0	2	0	2	0	2	4	0	0	3	1	2	0
8	0	1	5	1	0	2	0	0	41	2	0	0	3	0	3	0	0	0	1	1	0	0	0	0	0	0	1	0	1	0
9	1	1	0	1	1	7	0	3	1	25	0	1	1	3	11	0	5	1	1	7	0	0	0	6	3	1	0	0	0	0
10	0	0	0	3	0	0	0	0	0	0	67	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0
11	0	2	0	0	0	0	0	1	0	0	0	44	0	1	0	8	1	0	0	0	1	2	0	0	1	0	2	1	1	0
12	1	3	0	1	0	0	0	1	0	1	0	0	28	2	1	0	1	0	1	2	2	0	0	1	0	0	2	0	1	1
13	0	10	0	1	1	2	0	4	1	1	0	4	1	24	0	1	3	0	0	2	5	0	5	5	1	0	5	1	2	2
14	0	0	0	1	0	0	0	1	0	7	0	0	2	0	49	0	8	1	1	5	2	0	0	0	1	0	0	0	0	0
15	0	1	0	0	0	0	0	1	2	0	0	8	0	1	0	52	0	0	0	1	0	0	0	1	0	0	0	0	0	0
16	1	1	0	0	0	1	0	2	2	0	0	2	0	3	4	0	59	1	1	0	0	0	0	0	1	0	3	0	1	0
17	0	0	0	2	0	1	0	0	1	0	0	0	0	1	0	0	0	49	0	0	0	0	0	1	0	0	0	2	0	0
18	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	1	1	43	2	1	1	0	0	0	0	3	0	0	0
19	0	0	0	0	0	1	0	0	0	1	0	0	0	2	0	0	0	0	3	47	0	1	0	3	0	0	2	0	0	0
20	0	0	0	4	0	0	0	9	1	0	1	3	0	0	3	0	2	0	5	0	23	0	1	2	0	0	16	1	1	0
21	9	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	48	0	0	0	3	0	1	0	0
22	0	0	0	5	1	0	0	1	0	0	2	0	0	0	0	0	0	0	0	1	0	0	57	1	0	0	1	2	1	0
23	0	2	0	0	1	1	0	7	2	1	0	0	0	2	0	0	1	3	2	1	1	0	0	43	0	0	0	0	1	0
24	1	0	0	0	0	0	0	0	0	0	0	2	0	1	2	1	1	0	2	0	0	0	0	1	44	1	3	0	2	0
25	8	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	3	0	0	7	28	0	0	0	0
26	1	2	0	2	0	0	0	9	0	1	2	0	0	1	4	0	0	0	6	1	6	0	0	0	0	0	38	0	1	0
27	0	0	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	1	47	16	0
28	0	0	0	0	0	0	1	1	0	0	0	0	0	3	0	0	0	0	0	0	0	0	2	0	0	0	2	14	49	0
29	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	1	0	0	0	0	0	41

Figure 10: Logistic Regression Confusion Matrix Class Count

2.4 Ensemble – LR 4 Votes, NB 2 Votes, KNN 3 Votes

2.4.1 Metrics

Table 14: Metrics for Ensemble

Input Data	Lift	Accuracy	Precision	Recall	F Measure	Lines of Code	Time
							Total Time
SVD 500 Components	19.5	0.650	0.655	0.666	0.650	252	9353.057

2.4.2 Confusion Matrix

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	51	1	1	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	7	0	0	0	4	0	1	0	0
1	0	41	0	0	0	0	0	2	5	0	2	4	0	5	0	1	1	1	0	2	0	0	0	3	0	0	1	0	2	0
2	5	0	63	1	3	3	0	0	7	2	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	39	0	0	0	2	1	0	4	0	0	1	3	0	1	1	0	0	7	0	3	1	0	0	0	0	1	0
4	1	0	1	0	67	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
5	12	0	12	0	4	34	1	0	5	1	0	1	0	0	0	1	3	0	1	2	0	1	0	2	0	1	0	1	0	0
6	6	2	0	0	0	0	15	0	0	1	0	0	0	0	0	0	1	0	7	1	0	0	0	1	0	0	1	0	0	0
7	0	1	0	2	0	0	0	42	0	0	0	1	1	0	1	0	0	0	2	0	3	0	2	5	0	0	2	1	1	0
8	0	1	5	1	0	2	0	1	41	1	0	0	2	0	2	0	0	0	1	2	0	0	0	0	0	0	1	0	2	0
9	1	1	0	0	1	7	0	4	1	24	0	1	1	3	11	0	5	2	1	7	0	0	0	6	3	1	0	0	0	0
10	0	0	0	3	0	0	0	0	0	0	68	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
11	0	2	0	0	0	0	0	1	0	0	0	46	0	1	0	6	1	0	0	0	1	2	0	0	1	0	2	1	1	0
12	1	4	0	1	0	0	0	1	0	1	0	0	28	1	1	0	0	0	2	2	3	0	0	0	0	0	1	0	2	1
13	1	9	0	1	1	2	0	4	1	1	1	4	1	24	0	1	1	0	1	4	5	0	5	5	1	0	3	1	2	2
14	0	0	0	1	0	1	0	1	1	5	0	0	2	0	47	0	8	1	2	6	2	0	0	0	1	0	0	0	0	0
15	0	0	0	0	0	0	0	1	1	0	0	8	0	1	0	55	0	0	0	1	0	0	0	0	0	0	0	0	0	0
16	1	1	0	0	0	1	0	2	2	0	0	2	0	2	3	0	61	1	1	0	0	0	0	0	1	0	3	0	1	0
17	1	0	0	2	0	1	0	0	1	0	0	0	0	0	0	0	0	49	0	0	0	0	0	1	0	0	0	2	0	0
18	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	1	1	44	2	1	0	0	0	0	0	3	0	0	0
19	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	3	48	0	1	0	2	0	0	2	0	0	0
20	0	0	0	5	0	0	0	9	1	0	1	3	0	0	3	0	2	0	5	0	22	0	1	2	0	0	16	1	1	0
21	9	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	50	0	0	0	2	0	0	0	0
22	0	0	0	5	1	0	0	1	0	0	2	0	0	0	0	0	0	0	0	1	0	1	57	1	0	0	1	1	1	0
23	0	2	0	0	1	1	0	8	2	0	0	0	0	2	0	0	2	2	2	2	1	0	0	43	0	0	0	0	0	0
24	1	0	0	0	0	0	0	0	0	0	0	2	0	1	2	0	1	0	3	0	0	0	0	1	44	1	2	1	2	0
25	8	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	3	0	0	6	29	0	0	0	0
26	1	2	0	2	0	0	0	9	0	0	2	0	0	1	4	0	0	0	7	1	6	0	0	0	0	0	38	0	1	0
27	0	0	0	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	1	51	12	0
28	0	0	0	0	0	0	1	1	0	0	0	0	0	3	0	0	0	0	0	0	0	0	2	0	0	0	2	16	47	0
29	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0	0	0	1	0	0	0	0	0	40

Figure 11: Ensemble Confusion Matrix Class Count

2.5 Discussion

The above results clearly indicate the variation between models. With some analysis, the benefits and drawbacks of each model can be perceived. The first call out is that all models performed well considering it is 30 category classification problem. Each had a lift above 16.5, indicating any one of these models is at least 16 times better than a random guess. Processing wise there is a large disparity between run times, this is due to the impact and requirement for SVD feature selection.

Table 15 is a summary of the data in section 2.4, colour has been added to indicate the best performing for each category. A first look indicated that the Ensemble is the best performing based on classification, but computationally it is the least efficient. KNN is the simplest to implement but computationally heavy. Naïve Bayes performed well and was drastically quicker than any other model. Finally, Logistic regression was a strong middle ground in all categories, relatively complex, mid-range run time, however it achieved almost the highest accuracy.

Table 15: Comparison of the best models

Model	Accuracy	Precision	Recall	Lines of Code (Complexity)	Time (s)
					Total
KNN	0.554	0.558	0.592	36	8175.877
Naïve Bayes	0.635	0.636	0.659	72	16.859
Logistic Regression	0.645	0.650	0.658	121	1052.263
Ensemble	0.650	0.655	0.666	252	9353.050

A deep analysis into each model's confusion matrix, precision and recall values gives insight into false negative/positive performance as well as giving an indication of why the Ensemble (the amalgamation of all models) is the best performer. When looking at the confusion matrix it appears KNN has fewer false negatives and more false positives, whereas for NB and LR the reverse is true. This is supported by the model's precision and recall values. Table 16 depicts the change in each metric compared to accuracy. From this the huge improvement in Recall for KNN is outlined, almost double that of any other model having almost 7% difference. Whereas for precision it is lower. The inverse relationship to the other models is depicted through the difference in colour for the two metrics. Logistic Regression, alternatively, performs in the reverse manner having greater relative precision performance. This inverse performance is smoothed out when the multiple models are combined in the Ensemble, allowing for improved performance.

Table 16: Precision and Recall comparison

Model	Percent Change	
	Precision	Recall
KNN	0.722%	6.859%
Naïve Bayes	0.157%	3.780%
Logistic Regression	0.775%	2.016%
Ensemble	0.769%	2.462%

Due to the complexity of the problem there is no clear model that is the best performer, if accuracy is essential and time is not the Ensemble will be the best method, alternatively if time is a concern the Naïve Bayes method should be considered. To meet both time and accuracy requirements Logistic Regression has been used. This decision process is outlined in the implemented model, allowing the user to select the most appropriate model for their situation.

3 Summary of Results

Model selection and development comprised of three main steps;

1. Model investigation and comparison; understanding what is currently available, addressing the performance of hand-written code to that of premade functions

2. Model tuning and optimisation; finding the best hyperparameters

3. Model performance; analysing the best of the best

Each step was an intricate part of this process to achieve high accuracy and low run time. The initial analysis of existing solutions provided insight into the value of which methods would be appropriate and a great benchmark to aim for. Following this the three models NB, LR, and KNN were clear winners. The hand-coded implementations performed close to that of their optimised Sklearn counterparts. Thus, indicating a success in their development.

Selecting the hyperparameters was the next step in the process and ultimately boosted the result of the models as they were tuned to their highest performing range.

The final analysis on the best performing models evaluated their success, drawbacks and how to prevent overfitting.

The research concluded the highest accuracy was achieved at 0.686 with an Ensemble of the three models. Yet if time is a constraint it is best to use a Logistic Regression achieving a slightly lower accuracy in a fraction of the time.

Chapter 4: Discussion

Due to the complexity, work requirement and time constraints the work load of this assignment was split between the three team members; therefore, each has a different perspective and value gained from undertaking their task. In this chapter, each member discusses the meaningful and personal reflection of their work and the value it has provided them.

1 Reflection – Cameron Wasilewsky

Undertaking the model comparison and research led to an insight around the best structure and mathematical approach to text classification problems [21]. Both the research and practical application of models indicated higher performance in accuracy and computing time for linear models such as Logistic Regression and Support Vector Machines (SVM) with linear kernels. It is believed this comes about because non-linear models tend to overfit on high dimensional data. Linear models attempt to simplify and reduce the impact of dimensionality issues with lower weightings and simpler parameter estimation. This understanding will be valuable when applied to similar problems going forward. Rather than applying a Deep Learning technique straight away other methods will be considered

2 Reflection - Kristopher Lopez

Whilst building a range of models the 'Curse of Dimensionality' [9] (COD) became evident. Prior to undertaking this research this concept was simply theoretical, however when implementing a KNN the practical impact was clear. COD is an issue associated with high dimensional problems such as this one (13626). By increasing the feature space, there is a reduced number of options for several samples with each combination. This effect impacts KNN directly as it becomes difficult to reject candidates by using their differences. The computational cost of comparing so many data points to one another also indicated the drastic impact of such a large data set. Whereas methods that utilised general linear algebra resulted in better performance and accuracy.

3 Reflection - Arjun Sathasivam

When analysing and comparing the results of the different models the 'Bias-Variance trade off' greatly impacted on the results [22]. Bias is missing relevant data and getting errors from assumptions, whereas variance is error from sensitivity having little flexibility and misclassifying. When doing the comparison of models and hyperparameters it became clear that both ends of the spectrum were present and it was vital to find the right hyperparameters to control this issue. The impact of the two can be seen in Figure 12, emphasising why it is so important to find the hyperparameter 'sweet spot'. Going forward in industry this issue will need to be considered and methods of finding the best hyperparameters will be essential.

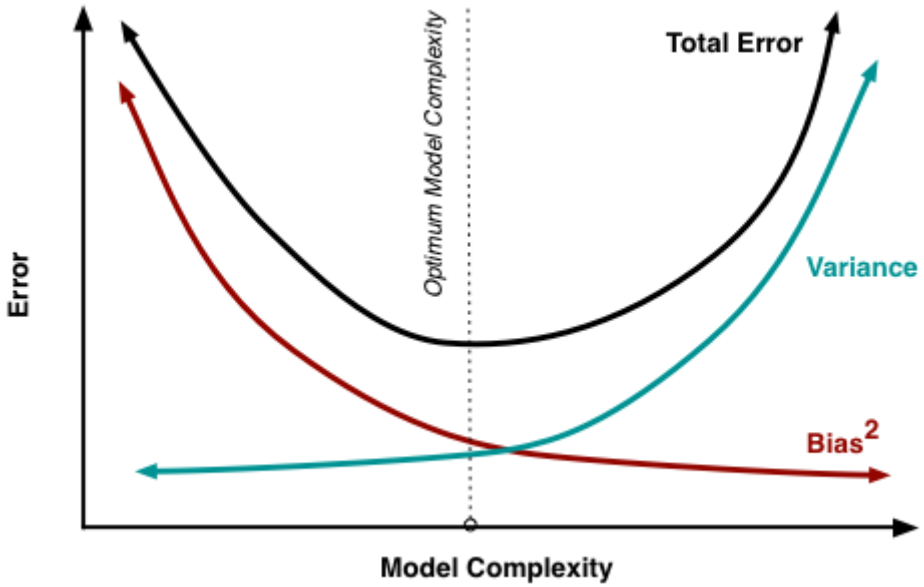

Figure 12: Bias and Variance, impact on error

Chapter 5: Conclusion and Future Work

1 Conclusion

After carrying out the classification task, a maximum accuracy of 0.686 was achieved. This was a high result, surpassing all tested prebuilt functions in 'sklearn'. This result is impressive, especially considering the fact there were 30 classes. The project has also been extremely instructive in many common and widely used machine learning techniques, providing the authors with an understanding of complex mathematical models and evaluation techniques.

Whether the results achieved in this project are enough to pursue practically (for example, by designing an automated app/category matching engine) is debatable. Ideally, with more time and a deeper knowledge of the subject, the predictive power generated from this task could be increased. The practical implications would then become more significant, which ultimately is the reason this work is undertaken.

2 Future Work

Due to limited time and resources this body of research was only able to focus on well-known and developed Machine Learning (ML) classification models. There was little time put into improved feature selection, developing new mathematical structures or considering Deep Learning solutions, future work could focus on these two elements.

2.1 Improved Feature Selection

As only common feature reduction methods were evaluated in this body of research, further improvements could be made to reduce the size of the data set and extracting the relevant and useful features. However, for the best result the term frequencies would need to be provided. Given that, data feature selection techniques such as mutual information, chi-square and n-gram could be utilised.

2.2 Support Vector Machine (SVM) Implementation

Within the initial stages of this research 'sklearn' models were benchmarked and tested to set expectations and aspirations. From this testing, SVC.Linear a SVM model was the best performer. Due to both technical skills and time limitations implementing this model from scratch was not possible. However, it likely would have been the best performer.

2.3 Deep Learning Classifiers

With the improvements in computational processing power, research and applications of Deep Learning (DL) technologies such as Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, Boltzmann Machines or Autoencoders has grown exponentially [23]. These models have begun to show huge promise in categorisation, surpassing most machine learning (ML) techniques. Thus, given more time future work should be put towards applying DL methods to the problem and evaluating their performance, comparing to ML.

2.4 New Mathematical Models

This body of research was only able to focus on building models that already exist and have for a couple of decades, given more time and resources it would be both valuable and educational to attempt to develop a new model, a mathematical classification system that has yet to exist. This would be by no means an easy feat, however there is a potential for developing something that will greatly benefit the research community, and potentially society, depending on its accuracy and application.

2.5 Alternate Ensemble Methods

Simply model weighting was used in this body of work, however going forward improvements could be achieved by implementing bootstrapping and bagging could be used to build an ensemble of linear models

Bibliography

[1] R. Fabio, "COMP5318: Machine Learning and Data Mining - Course Outline," The University of Sydney, Sydney, 2017.

[2] P. Haffner, "What is Machine Learning - and Why is it Important?," Interactions, 7 July 2016. [Online]. Available: https://www.interactions.com/machine-learning-important/. [Accessed 29 April 2017].

[3] Wikipedia, "tf–idf," Wikipedia, 19 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Tf%E2%80%93idf. [Accessed 29 April 2017].

[4] J. Shlens, "A Tutorial On Principal Component Analysis," pp. 1-16, 25 March 2003.

[5] scikitlearn, "Decomposing signals in components (matrix factorization problems)," scikit-learn developers, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/decomposition.html. [Accessed 1 May 2017].

[6] B. Scholkopf, A. Smola and K.-R. Muller, "Kernel Principal Component Analysis," *International Conference on Artificial Neural Networks*, pp. 583-588, October 1997.

[7] S. Raschka, "Kernel tricks and nonlinear dimensionality reduction via RBF kernel PCA," SebastianRaschka, 14 September 2014. [Online]. Available: http://sebastianraschka.com/Articles/2014_kernel_pca.html. [Accessed 1 May 2017].

[8] S. Raschka, "Why is Nearest Neighbor a Lazy Algorithm?," SebastianRaschka, 2017. [Online]. Available: https://sebastianraschka.com/faq/docs/lazy-knn.html. [Accessed 1 May 2017].

[9] V. Spruyt, "The Curse of Dimensionality in classification," VisionDummy, 21 April 2014. [Online]. Available: http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/. [Accessed 1 May 2017].

[10] J. Liber, "k-NN computational complexity," StackExchange, 19 June 2016. [Online]. Available: https://stats.stackexchange.com/questions/219655/k-nn-computational-complexity. [Accessed 1 May 2017].

[11] V. Vryniotis, "Machine Learning Tutorial: The Naive Bayes Text Classifier," DatumBox, 13 October 2013. [Online]. Available: http://blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/. [Accessed 1 May 2017].

[12] S. Raschka, "Introduction and Theory," SebastianRaschka, 4 October 2014. [Online]. Available: http://sebastianraschka.com/Articles/2014_naive_bayes_1.html. [Accessed 1 May 2017].

[13] J. Brownlee, "Naive Bayes Tutorial for Machine Learning," Machine Learning Algorithms, 13 April 2016. [Online]. Available: http://machinelearningmastery.com/naive-bayes-tutorial-for-machine-learning/. [Accessed 1 May 2017].

[14] "Monotonically Increasing and Decreasing Functions: an Algebraic Approach," OpenCurriculum, 2016. [Online]. Available: https://opencurriculum.org/5512/monotonically-increasing-and-decreasing-functions-an-algebraic-approach/. [Accessed 1 May 2017].

[15] Wikipedia, "Logistic regression," Wikipedia, 27 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression. [Accessed 1 May 2017].

[16] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," 2015. [Online]. Available: http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf. [Accessed 1 May 2017].

[17] M. Caraciolo, "Machine Learning with Python - Logistic Regression," AI Motion, 6 October 2011. [Online]. Available: http://aimotion.blogspot.com.au/2011/11/machine-learning-with-python-logistic.html. [Accessed 1 May 2017].

[18] "Ensemble learning," Wikipedia, 14 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Ensemble_learning. [Accessed 29 April 2017].

[19] W. Chen, "How do ensemble methods work and why are they superior to individual models?," Quora, 13 November 2014. [Online]. Available: https://www.quora.com/How-do-ensemble-methods-work-and-why-are-they-superior-to-individual-models. [Accessed 1 May 2017].

[20] S. Raschka, "Implementing a Weighted Majority Rule Ensemble Classifier," SebastianRaschka, 11 January 2015. [Online]. Available: http://sebastianraschka.com/Articles/2014_ensemble_classifier.html. [Accessed 1 May 2017].

[21] D. Marsupial, "Why does ridge regression classifier work quite well for text classification?," StackExchange, 29 October 2011. [Online]. Available: https://stats.stackexchange.com/questions/17711/why-does-ridge-regression-classifier-work-quite-well-for-text-classification. [Accessed 1 May 2017].

[22] S. Fortmann-Roe, "Understanding the Bias-Variance Tradeoff," CSS from Substance.io, June 2012. [Online]. Available: http://scott.fortmann-roe.com/docs/BiasVariance.html. [Accessed 1 May 2017].

[23] K. Eremenko, Composer, *SDS 047: An expert overview of the deep learning models, supervised and unsupervised.* [Sound Recording]. SuperDataScience. 2017.

[24] Scikit-Learn Developers, "Nearest Neighbors," Scikit-Learn Developers, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html. [Accessed 29 April 2017].

[25] Scikit-Learn Developers, "Naive Bayes," Scikit-Learn Developers, 2016. [Online]. Available: http://scikit-learn.org/stable/modules/naive_bayes.html. [Accessed 29 April 2017].

[26] Wikipedia, "Multinomial logistic regression," Wikipedia, 15 April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Multinomial_logistic_regression. [Accessed 29 April 2017].

[27] B. Boltzmann, "How to write a confusion matrix in Python?," StackOverflow, 27 January 2010. [Online]. Available: http://stackoverflow.com/questions/2148543/how-to-write-a-confusion-matrix-in-python. [Accessed 1 May 2017].

[28] B. Yang, “How to compute precision/recall for multiclass-multilabel classification?,” CrossValidated, 29 October 2014. [Online]. Available: https://stats.stackexchange.com/questions/21551/how-to-compute-precision-recall-for-multiclass-multilabel-classification. [Accessed 1 May 2017].

Appendix

1 Instructions on how to Run Code

1.1 Required Packages

- Numpy
- Scipy
- MatplotLib
- Time
- CSV
- Xlsxwriter
- Sys
- OS

1.2 Recommend Computer Specifications

Table 17: Computer specifications

Processor	RAM	System Type	Operating System	Python Version
Intel(® Core™ i702600 CPU @ 3.40GHz	16.00GB	64-Bit x64-based processor	Windows 10 Home	3.5

1.3 Steps

1. Unzip provided file
2. Copy ‘training_data.csv’ & ‘test_data.csv’ to *‘…\COMP5318_Assignment_1_Group_34\code\input’*
3. Navigate to ‘…\COMP5318_Assignment_1_Group_34\code\algorithm’
4. Hold ‘shift’ and right click in the folder directory
5. Select “Open command window here”
6. Enter ‘python classipyer.py’ into the command window and press enter
7. You will be prompted to select which model you wish to run. The options are outlined in Table 18
8. Once the code has run expected outputs will be in *‘…\COMP5318_Assignment_1_Group_34\code\output’*

Table 18: Model options

Option	Model Name	Description	Classifiers	Expected Accuracy	Expected Run Time
1	Quick Training Model	**Model**: Uses one classifier **Data**: 90% of training data and tests on 10% **Output**: Metrics	LR	0.64	10 (m)
2	Long Training Model	**Model**: Uses Ensemble of three classifiers **Data**: 90% of training data and tests on 10% **Output**: Metrics	• NB • KNN • LR Ensembled together	0.66	30 (m)
3	Quick Testing Model	**Model**: Uses one classifier **Data**: 100% of training data and tests on 100% of test data **Output**: Test Predictions in CSV	LR	N/A	10 (m)
4	Long Training Model	**Model**: Uses Ensemble of three classifiers **Data**: 100% of training data and tests on 100% of test data **Output**: Test Predictions in CSV	• NB • KNN • LR Ensembled together	N/A	30 (m)

2 Percentage Confusion Matrix

2.1 KNN

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
1	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	1%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	1%	0%	1%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	1%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
12	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
13	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
14	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
15	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
16	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
17	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
18	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
19	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	1%	0%	0%	0%
21	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%
22	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%
23	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%
24	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%
25	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%
26	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%
27	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	1%	0%
28	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	2%	0%
29	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%

2.2 NB

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
1	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	1%	0%	1%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
12	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
13	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
14	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
15	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
16	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
17	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
18	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
19	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	1%	0%	0%	0%
21	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%
22	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%
23	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%
24	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%
25	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%
26	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%
27	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%
28	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	2%	0%
29	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%

2.3 LR

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
1	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	0%	0%	1%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
12	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
13	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
14	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
15	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
16	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
17	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
18	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
19	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%	0%	0%
21	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%
22	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%
23	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%
24	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%
25	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%
26	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%
27	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	1%	0%
28	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	2%	0%
29	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%

2.4 Ensemble

True Category \ Predicted Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
1	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	1%	0%	1%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
11	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
12	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
13	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
14	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
15	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
16	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
17	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
18	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
19	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
20	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	1%	0%	0%	0%
21	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%	0%	0%
22	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%
23	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%	0%
24	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%	0%	0%
25	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%
26	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0%	0%	0%
27	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	1%	0%
28	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	2%	0%
29	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%