COMP5318
Machine Learning and Data Mining
Assignment 2
Group 44

University of Sydney

November 18, 2020

Team members

Alex Augustine [500596748]

Oliver Stefan Nawrot [500680715]

Zirong Guo [500417317]

Contents

1 Introduction

1.1 Abstract

This article shows three classification algorithms, all of which achieve the task of classifying the handwritten data set in the Chars74 data set. Before implementing the classification algorithm, we performed a series of pre-operations on the data set, which were the binarization operation of the image, the de-redundancy operation of the image, and the reshaping operation of the image size, followed by the scaling operation. To reduce the amount of calculation. In the dimensionality reduction part, we used the PCA dimensionality reduction method to reduce the dimensionality of the training set and the validation set. After that, three classification algorithms of Naive Bayes, Logistic Regression and Decision Tree were used to perform 10-fold cross-validation. Finally output the data to a csv file for statistical operations

1.2 Introduction

Image detection is an extensively explored concept and is prevalent in today's society. It is used in one form or another in many modern technological gadgets, from face recognition as a security measure to translation of foreign languages to English characters. While character detection models have been studied from as far back as the 19th century[1], it is still a widely investigated research field. Since then, there have been several breakthroughs in image recognition, mainly in the field of Artificial Intelligence where, for example, the use of convolutional neural networks in object detection have been pioneered by academics like Yann LeCun[2].

This report focuses on a large subset of image detection: character recognition, specifically handwritten character recognition. Mantas[1], in his paper in 1986, describes handwritten classification as one of the most challenging problems to solve in image classification, due to unpredictable writing styles and shapes of handwritten characters. Over the years, there have been significant advances in preprocessing methods such as the use of the Histogram of Oriented Gradients to extract the most important features from an image, denoising and bounding uneven images using contours to eliminate unwanted noise in the image etc. Such methods along with the evolution of computer processing power, have led to interesting and life changing inventions.

The main aim of this paper is to determine an algorithm that can accurately predict characters from handwritten images. In particular, this paper intends to investigate the performance of Random Forest, Logistic Regression and Naive Bayes classifiers on the handwritten dataset

of Chars74k. In carrying out this experiment, comparison of current research on the topic with this paper's results can be made. The significance of this project rests on the fact that image classification is a hot topic in the field of machine learning. The methodology and analysis presented in this report can provide additional meaningful insights into how images can be preprocessed to improve training phases in the machine learning process.

2 Literature Review

There are numerous academic papers which tackle the problem of character recognition using machine learning algorithms. The earliest methods of character recognition used cathode ray tube scanners to read in characters and evaluate results based on the shape of the character[1]. While the general concept behind solving character recognition problems has not changed significantly, improved pre-processing methods and modernised machine learning algorithms have gone a long way in achieving satisfactory results in character recognition. Furthermore, recent developments in detecting characters from documents have been influential in creating commercial character recognition systems such as Adobe Acrobat[3] and ABBYY[4].

However, these software deal mainly with analysing documents with computer generated characters and are not efficient in detecting handwritten characters or characters from natural images[2,5]. For example, to study the effects of complex feature extraction methods on accuracy of models, Sinha[2] applied k-Nearest Neighbours and Random Forest classifiers on the Chars74 dataset and obtained superior results to that of more advanced techniques using discriminative learning. Using Otsu's thresholding for image processing and histogram of oriented gradients for feature extraction, Sinha[2] acquired a maximum accuracy of 71.6% on test data using the Random Forest classifier. Similarly, de Campos[5] implemented k-Nearest Neighbours, Support Vector Machines and Multiple Kernel Learning, and produced results that outperformed industry standard programs such as those mentioned above. However, in both papers, the handwritten characters dataset was not thoroughly explored with their respective machine learning models. Sinha[2] carried out experiments on only the natural images dataset, whereas in this report, the Random Tree classifier will be used on the handwritten dataset, along with other pre-processing methods as outlined in Methodology. de Campos[5] focused on feature extraction methods such as shape context and geometric blur to pre-process the images, which resulted in accuracies ranging from 25%-69% from all of their applied models for the handwritten dataset. As was noted in the paper, such a wide range of accuracies could be attributed to the fact that only 15 images per character were used for training the models.

As part of the training models implemented in this paper, all 55 samples will be used for each character to maximise the efficiency of each model. Furthermore, de Campos[5] came to the conclusion that the character font dataset would be a better option for training models and testing on other natural characters, as it was easier to train and provided the most accurate results. The experimental methodology in this report will use different classifiers to that used by de Campos[5], and evaluate their performance using hyperparameter tuning and 10-fold cross validation.

The pre-processing methods deployed in the aforementioned papers did not include feature dimensionality algorithms such as principal component analysis (PCA) or image modification methods such as normalisation and bounding using character contours. Due to the unique aspect of handwritten characters, for example - characters in cursive writing are significantly different to block letters, different writing styles etc. - such pre-processing methods may prove beneficial in helping algorithms to discern between similarly shaped characters. Thus, the investigation outlined in this report will include a pipeline of image modification steps which will aim to reduce these unique characteristics of handwriting. Moreover, the effects of PCA on model scoring metrics will be thoroughly explored, such as the effect of number of PCA components versus average accuracy of model.

There have not been major studies on the application of Naive Bayes or Logistic Regression classifiers on the Chars74k dataset. The use of probabilistic classifiers such as Naive Bayes is not common for multi class image classification problems, most likely due to its linearity and fundamental assumption of independence - which is not strictly true when applied to features of real world data. However, studies have suggested that even with this assumption, Naive Bayes can capture natural image data with high model accuracy compared to more complicated and expensive algorithms such those implementing neural networks and support vector machines [A7,A8]. Oren[6] proposed the combination of Naive Bayes with the Nearest Neighbour classifier for large image datasets. This resulted in a net performance gain of 12% using the Caltech dataset, compared to other classifiers which were also implementing the nearest neighbour approach. Shilpy[7] applied the Naive Bayes algorithm on 'Gurmukhi' characters of the Punjabi language, which consisted of 35 characters with 70 handwritten samples each. They achieved an accuracy average of approximately 77% after 10 fold cross validation. In their study, they applied pre-processing methods such as PCA, normalisation and zoning of pixels to reduce dimensionality. This report will build on these methods, along with additional pre-processing such as image binarisation and mean value for feature extraction, which will be shown to be useful for the larger Chars74k handwritten dataset.

As mentioned above, handwritten characters do not have the same degree of fixed shapes as printed characters. Processing of aspects such as character thickness, size of individual character attributes (for example: a handwritten zero may perceive to be a capital 'O' depending on how the circle was written) is crucial for obtaining accurate models. Kinjal[8] used logistic regression and preprocessing methods such as character rotation to show that the logistic regression classifier was able to predict characters with comparable geometry at relatively high accuracies. They applied their algorithm on characters from the Bengali language, with 50 characters and 300 handwritten samples for each character. The same analysis has not been applied to the English language, where there is high similarity between certain characters and between upper and lower case counterparts ('n' and 'm','b' and 'd', 'x' and 'X' etc). The strength of logistic regression on classifying handwritten English characters will be examined comprehensively in this report. Additionally, this report will research the effects of options in logit models such as 'one vs rest' strategy and multinomial regression against the overall capability of the model's classification accuracy.

3 Methodology

3.1 Dataset

The dataset used in this report is the 'Chars74k' dataset[9], consisting of a collection of characters from the English and Kannada language taken from natural photos. The term 74k in its naming stands for the number of English characters (including from natural images, handwritten and computer generated) in the entire dataset. For this report, it was decided to focus specifically on the handwritten dataset as the original paper on the dataset did not produce satisfactory results in classification accuracy[2]. Moreover, de Campos[?] discovered that the fonts (computer generated) dataset performed much better than handwritten or natural image characters, while natural image characters returned the lowest accuracy across different models. In addition to its low accuracy, the natural images dataset had varying numbers of samples for each character (some characters had only 25 samples while others had greater than 100 samples). Since this report employs the use of machine learning algorithms that heavily favour big datasets (i.e. more samples = better accuracy), it was determined that the natural images dataset may not be useful in this scenario. In comparison, the handwritten dataset was more structured and had an equal number of sample images for each character. Therefore, for maximum insight into how images processing can affect the classification accuracy of different

algorithms, the handwritten dataset was chosen for this experiment. The handwritten dataset consisted of 62 characters - numerical characters from 0-9 and alphabetical characters from a-z and A-z. Each character contained 55 different handwritten image samples. A sample set of characters from this dataset are shown in Figure [AAF1]. As can be seen from Figure [AAF1],

Figure 1: Some images taken from character 'A' and 'a'. Note the inclusion of cursive handwritten characters in (d), which increases the difficulty of classifying characters accurately.

characters are not always centered on the image and in some cases are significantly smaller in size compared to other characters. For this reason, it was decided that preprocessing was a crucial step in preparing the dataset for training with machine learning algorithms. The preprocessing methods used to address these issues are described in the next section.

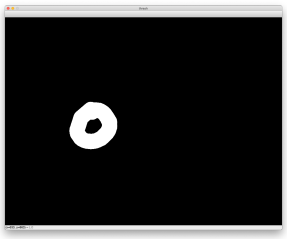
3.2 Preprocessing

The following steps are used to preprocess the images in the handwritten Chars74k dataset (also shown in Figure 2).

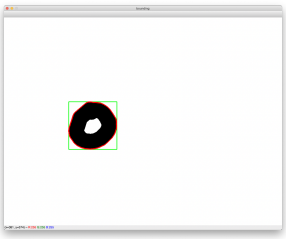
1. Transform color to binary

   First we use the cvt_color() function in opencv lib to convert the original image to a grayscale image which means each pixel only contains one number from 0 to 255. Then we use functions threshold() to convert the grayscale image to a binary image which means each pixel contains either 1 or 0. Because of the particularity of the dataset we use, that is, all the original images are handwritten characters with black characters on white background and there is almost no noise, so it is not necessary to deal with Gaussian blur. The reason for this step is to reduce the redundant information in each pixel.

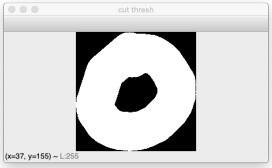
2. Bound, cut and resize The threshold() will return the binary image, then we use findContours() to get the contours of the characteristic and use boundingRec() to get the minimum boundary of characters and delete pixels outside the boundary. It should be mentioned that this minimum boundary refers to the minimum boundary obtained without rotation operation. The purpose of this step is to delete other useless pixels, which



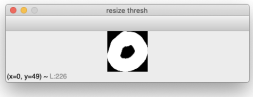
(a) Image size: original 900*1200)



(b) Bounding by rectangle using contours (displayed in green)



(c) Cut characteristic from original image



(d) Resize the image to the size required

Figure 2: Preprocessing steps applied to images

will reduce the noise of the training model and improve the accuracy of validation and testing.

3. Then we resize the data and the purpose of resizing is to make all samples have the same number of features, so that subsequent training and calculations can be carried out.

4. Scaling and PCA dimension reduction We use sklearn.preprocessing to standardize the data and sklearn.PCA to reduce dimension.

3.3 Learning, Validation and Testing

In line with good data science practice the implementation in this paper splits the dataset into subsets with the following functions[10]:

- Training set, which contains:
  - Learning set: training the algorithm in preparation for validation
  - Validation set: assessing the performance of the trained algorithm, which is then used in hyperparameter and preprocessing parameter optimisation. This step helps in reducing overfitting.
- Testing set: optimised algorithm from the validation step is used in final testing, the results of which are used in benchmarking. This step increases the fairness of the final

test by preventing apriori knowledge being used in parameter optimisation.

3.4 Majority Voting Classification

As the learning and validation data differ for each of the 10 folds, there are essentially 10 predictors that are available to classify the test set of data. This means that 10 predictions of the test data will be produced, whereas ultimately only one prediction of the test class is required. Majority voting is used in this paper, which means that the test point will be assigned to the class that is most predicted by the folds. This has the effect of increasing the accuracy of the algorithm[11].

3.5 Scoring Metrics

As the research problem to be solved was one of classification, scoring metrics most suited for classification problems were used. For ease of presentation, the common classification scoring metrics for binary classification are summarised below:

$$\text{Accuracy: } \frac{TP + TN}{TP + FP + TN + FN}$$

$$\text{Precision: } \frac{TP}{TP + FP}$$

$$\text{Recall: } \frac{TP}{TP + TN}$$

$$\text{F1-score: } 2\text{x}\frac{\text{precision x recall}}{\text{precision + recall}}$$

The use of the sklearn library allowed these scoring metrics to be applied to the multiclass classification problem explored in this paper.

3.6 Logistic Regression

The reason for choosing Logistic Regression is that firstly logistic regression is a linear model, it is simple to implement and very efficient, and secondly there are many ways to solve overfitting, such as l1 and l2 regularization[R2].

The core idea of logistic regression is to do MLE (Maximum likelihood estimation) on the input training examples of class 0 and class 1 by using sigmoid function, here is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

Derivation by formula, here is P(y|x),

$$p_1 = P(y = 1|x) = \sigma(w^T x) = \frac{1}{1 - e^{-w^T x}}$$
$$p_0 = P(y = 0|x) = 1 - P(y = 1|x) = \frac{e^{-w^T x}}{1 + e^{-w^T x}}$$
$$P(y|x) = p_1^y p_0^{1-y}$$

then do MLE with data with X and Y to get the weights W

$$W = argmax\_w\Big(\prod_{i=1}^{N} P(y_i|x_i)\Big) \tag{2}$$

Here the MLE can be equivalently viewed as maximizing the CLL (conditional log likelihood)

$$W = argmax\_w\Big(\prod_{i=1}^{N} ln[P(y_i|x_i)]\Big) \tag{3}$$

For multi classification, each class will have an individual classification sigmoid function to identify whether the test or validation sample is consistent with its own class.

3.7 Naive Bayes

Naive Bayes is a generative machine learning model that relies on an assumed probability distribution to predict the class of an unknown point. The probability distribution assumed for each class feature is the gaussian distribution with the mean and standard deviation set to equal that of the feature for the given class. The algorithm is referred to as naive as it assumes independence between variables, an assumption that is not generally true in a strict sense but still functions well as an approximation[12].

During the algorithm run, the Naive Bayes classifier will determine the probability that each of the features belongs to each of the classes, based on the aforementioned gaussian distributions. The application of Bayes Theorem and performing log-likelihood allows the calculation of an overall probability-in-class metric, with the highest probability class being assigned to the unknown point.

The Naive Bayes classifier was chosen for this project mainly due to its relatively simple but accurate prediction characteristics. Furthermore, the fact that the images are converted to greyscale and binarised will greatly benefit the algorithm as it can help uphold the independence assumption in most cases.

A possible modification to Naive Bayes is to alter the shape of the probability distribution. The gaussian distribution is the standard default distribution for use in Naive Bayes. Performing thorough statistical analyses and minimum likelihood estimations to better characterise the

feature distribution shape and parameters is considered to be outside of the scope of this paper and was not investigated[13].

3.8 Random Forest Ensemble

subsubsectionDecision Trees Decision tree classifiers form a tree-like structure by repeatedly partitioning the samples. The base form of the algorithm means that repeatedly splitting the data will lead to each datapoint being the sole member of a final group. The algorithm starters with the complete data set, known as the root node, and continual splits lead to multiple branches and branch nodes. The finally single data-point branch of the tree is referred to as a leaf node. The branching performed on the root node or the branch nodes, is done based on the feature data. A single feature will be used as a metric in deciding which samples will be put onto which branch. An example of this is:

$$\text{if } x < 0, \text{ assign datapoint to branch A}$$
$$\text{if } x \geq 0, \text{ assign datapoint to branch B}$$

An important part of the algorithm is deciding which feature to base the branch on and what value. The decision tree decides this by considering how to achieve a split that best segregates the data for machine learning purposes. A common version of this is entropy, where the tree makes a split so that the entropy on the branch nodes is minimised. Another popular alternative is gini, where the split is made so as to minimise the value of the gini index.

Decision trees are considered to be a competitive algorithm amongst the fundamental machine learning algorithms, but are prone to overfitting. subsubsectionRandom Forest Random forest is an ensemble algorithm that runs the decision tree multiple times to construct a 'forest' of decision trees. This not only improves the accuracy of the classification through an inherent majority voting function, but also reduces the effect of overfitting to the training data.

The parameters that explored in this paper were[14]:

- Max depth: specifies the maximum depth of the branches
- Gini or entropy split type: specifies the basis on which the data will be split
- Minimum leaf: the minimum number of samples allowed at a leaf node

3.9 Grid Search Optimisation

Grid search algorithms are commonly used to optimise the parameters of an algorithm. For this paper however, not only were algorithm parameters explored, but also preprocessing

parameters. To achieve this functionality, a custom grid search code was programmed to test a multitude of settings and combinations. This process allows not just independent parameters to be tested, but also allows testing of the interdependence of parameters. A basic type pseudo code representing the process is as follows:

```
for parameter setting 1 in [a1,b1,c1,d1,e1,...]:
    for parameter setting 2 in [a2, b2, c2, d2, e2,...]:
        ...
            record result of algorithm run with settings (setting 1, setting 2...)
```

‘

In the writing of this paper, over 1000 combinations over different algorithms and settings were run. Although the process saw a significant improvement in time relative to a manual method, there are likely to be further opportunities to integrate the grid search with other methods. Some possible ideas are gradient descent, Newton's method or even evolutionary algorithms with mutation to avoid local minima.

4 Results

4.1 Algorithm Comparisons

In Figure 3, from the comparison of the confusion matrix, it can be deduced that all three models were able to accurately predict values in the middle of the matrix (i.e. characters mainly in the uppercase region). This could be due to the fact that the larger characters, even after bounding the images, take up more pixels than when compared to smaller characters. This is also backed up by the number characters - we can see that all three classifiers were able to accurately classify 0-9, with the Random Forest algorithm producing more white pixels in that region than any other classifier.

However, as we go down the diagonal of all three matrices, we can see that there is a general trend of white to red, i.e. the models have difficulty in predicted labels in the lowercase characters. This could again be attributed to the thickness of the larger characters taking up more pixel data than the smaller characters. Furthermore, in cursive handwriting, lower characters have more variation in style than upper case characters, which tend to be more in block style. As can be seen from Figure 4(a), the accuracies of all three models follow the same

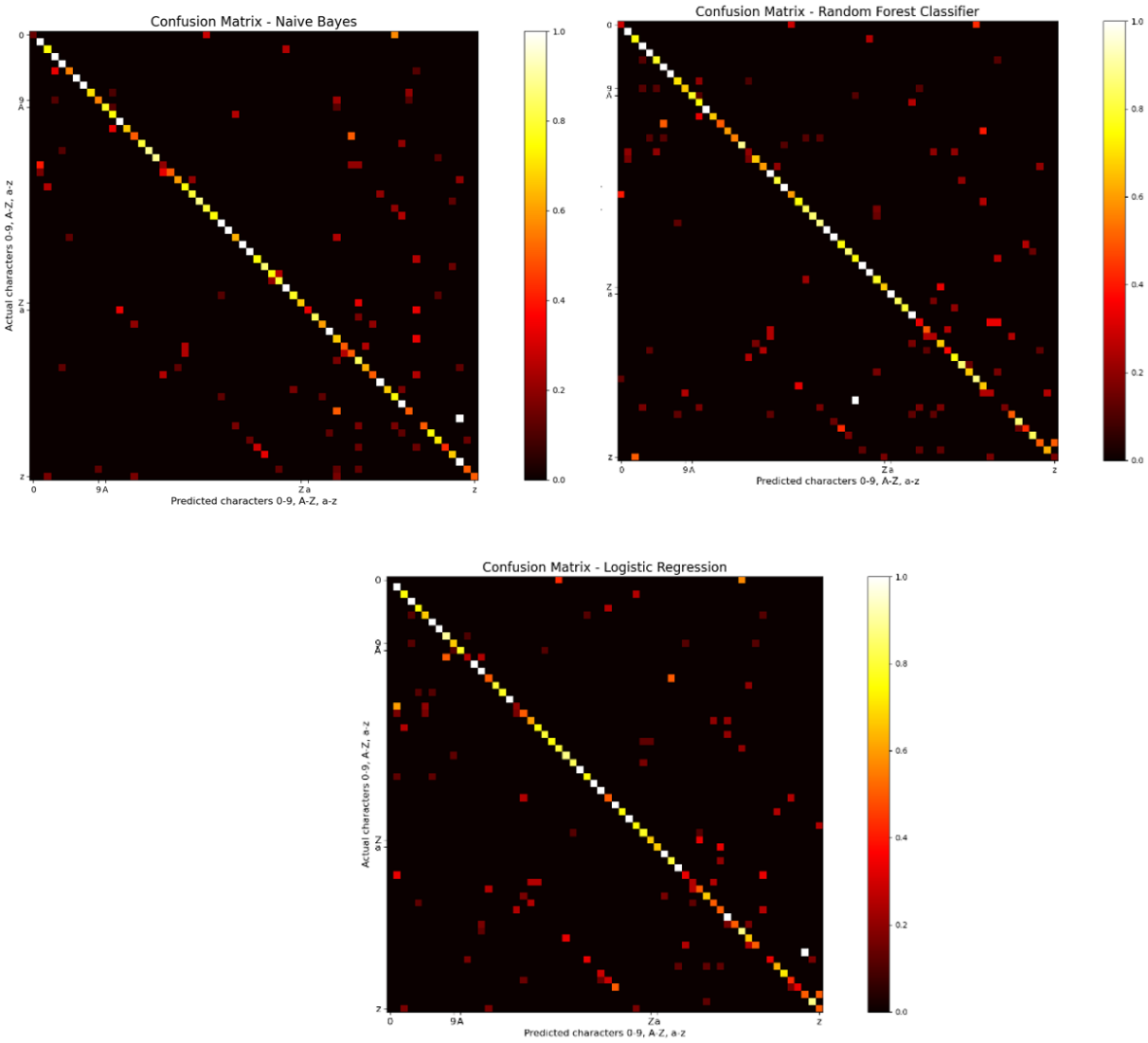
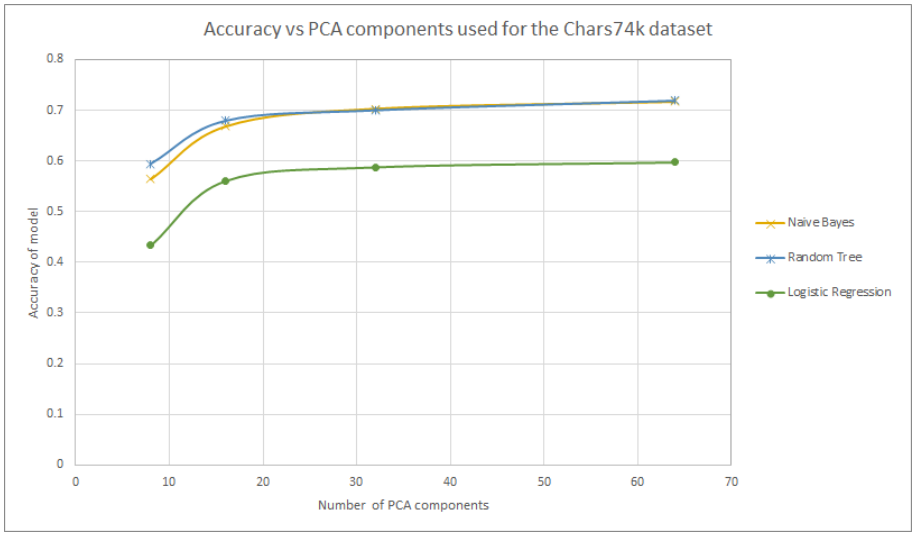


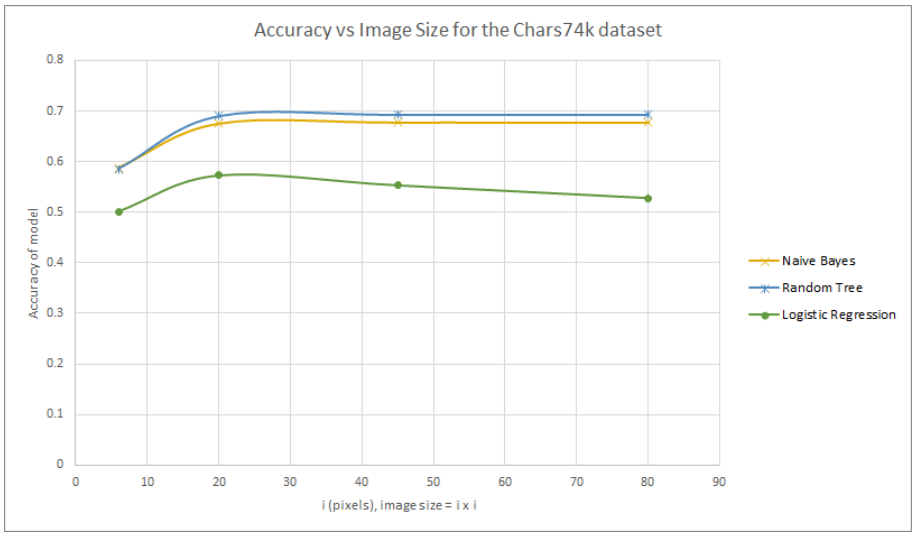
Figure 3: Heat map plot of the confusion matrix for each algorithm, after running with the best parameters. White pixels mean high accuracy and red pixels show a lower accuracy.

pattern. At approximately 8 retained PCA components, all three classifiers return the lowest accuracy. The Logistic Regression classifier performs the worst out of the three models. This could be attributed to the fact that the logit model assumes that there is a linear relationship between the independent and dependent features of a dataset. This dependence can be affected by the number of samples available per class. If a class has a relatively low number of samples, it can affect the performance of the model.

In Figure Figure 4(b), there is a similar trend to that in Figure 4(a). The Naive Bayes and Random Tree classifiers are more capable of predicting the correct labels with a larger number of features. The performance of the logit method is expected because, as mentioned above, the regression model tends to overfit when the number of features is greater than the number of samples in the dataset. For example, the Chars74k dataset has 3410 samples and using 20



(a) Average accuracy vs PCA



(b) Average accuracy vs image size.

Figure 4: Comparison of model performance with the number of pixels (features) used and retention of PCA components.

Algorithm	**Image size (pixels)**	**Extra feature**	**Number of Principal Components**	**Other parameters**
Logistic regression	400	Off	32	Regularisation: L1 Weight: balanced Solver: saga Type: Multinomial
Naive bayes	2025	Off	64	None
Random forest	285	Off	34	Max depth: 35 Criterion: gini Min samples: 1
Note: where a parameter is not specified, default sklearn parameter was used				

Figure 5: Settings used for optimal algorithms

Algorithm	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.716	0.712	0.714	0.694
Naive Bayes	0.72	0.725	0.716	0.697
Random Forest	0.734	0.736	0.737	0.713

Table 1: best performance using optimal settings of each algorithm

PCA components will result in 400 features. If we have a pixel size of 58x58, the number of features will overcome the number of samples in the dataset. At this point in the graph, in fact - starting at a size of around 30x30, it can be seen that the logit model starts to decrease in accuracy. This decrease in performance is not seen with the other two models, which stabilise after the 30 PCA components mark.

In the topic of image classification, this is a significant result because most natural images have a large number of features. Even with feature extraction methods, if the image is large enough, the number of features may still be greater than the number of samples. Therefore, for high quality images, the logit model will not fare as well as other similar algorithms.

4.1.1 Best Performance

Table 1 shows the settings used for the optimal run of each algorithm with table x showing the final scores of the algorithms. Random forest was the best performing algorithm, followed

Algorithm	Image Load Time (s)	Algorithm time (s)	Total Time (s)
Logistic Regression	56	330	386
Naive Bayes	52	14	66
Random Forest	69	30	129

Table 2: best performance using optimal settings of each algorithm

by naive bayes and logistic regression.

Random forest ensemble classifiers are known to be good at achieving reasonable accuracies and avoiding overfitting. It is important to note that the choice of this algorithm as the optimum may be dependent on the way in which the experiment is done. An important feature of this paper is that a validation step was done along with the learning and testing set. This particularly allowed algorithms suffering overfitting to be penalised before getting to the testing step. If this step were not done, it is possible that the order of algorithms could change. Nonetheless, the use of learning, validation and testing datasets is in line with normal practice and is therefore justified.

4.1.2 Speed Tests

Hardware and software specifications can be found in Appendix A3.

Table 2 shows the running times of each setting of optimal parameters for the optimal algorithm. Load times are not an inherent part of the algorithm, but because an optimum algorithm requires a certain set of image load parameters, this timing is seen as an important component of the total running time.

The fastest algorithm is the naive bayes, followed by random forest and logistic regression. Seeing that logistic regression also has a lower accuracy than the other two algorithms, it can easily set aside as a sub-optimal algorithm.

The choice of whether to use naive bayes or random forest comes down to whether accuracy or running time is more important: the decision depends on required application. A large dataset that requires the algorithm to run many times may be a better match with naive bayes as the total time is about half of random forest. However, a dataset which requires fewer times to run would likely be better matched with the higher accuracy, random forest.

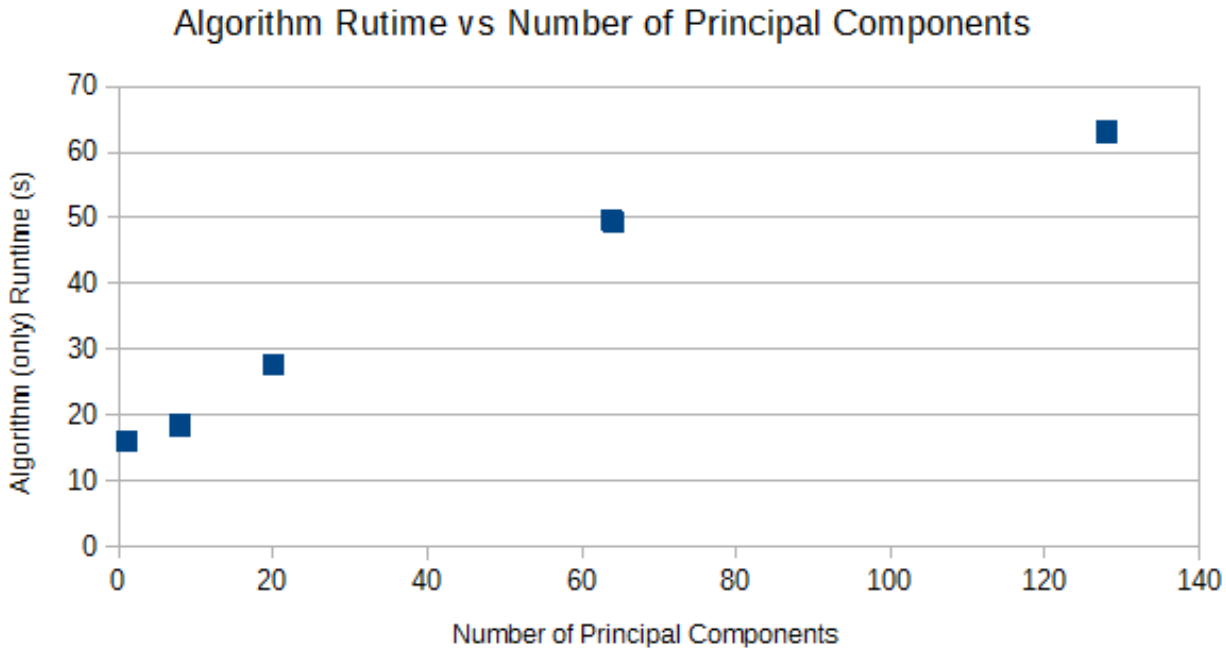


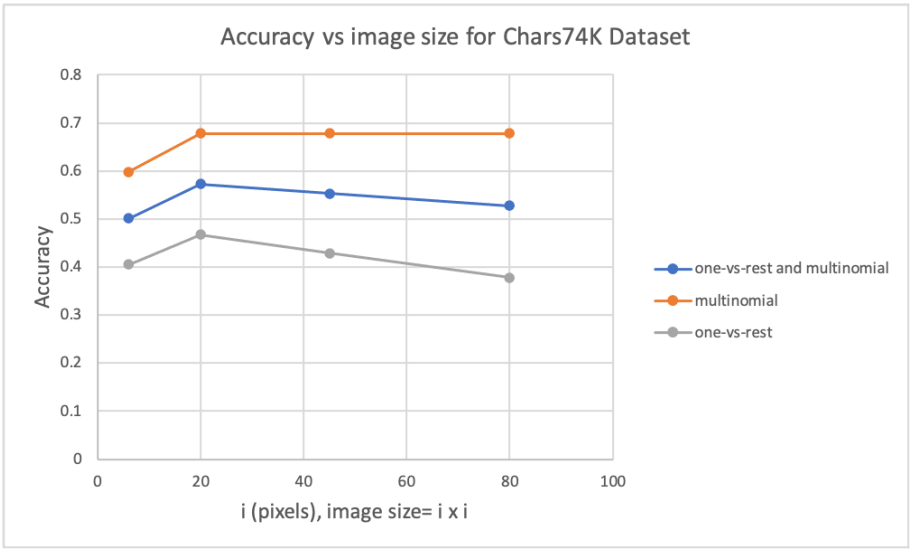
Figure 6: Arbitrary random forest algorithm runtime as a function of number of principal

4.1.3 Preprocessing Sensitivities

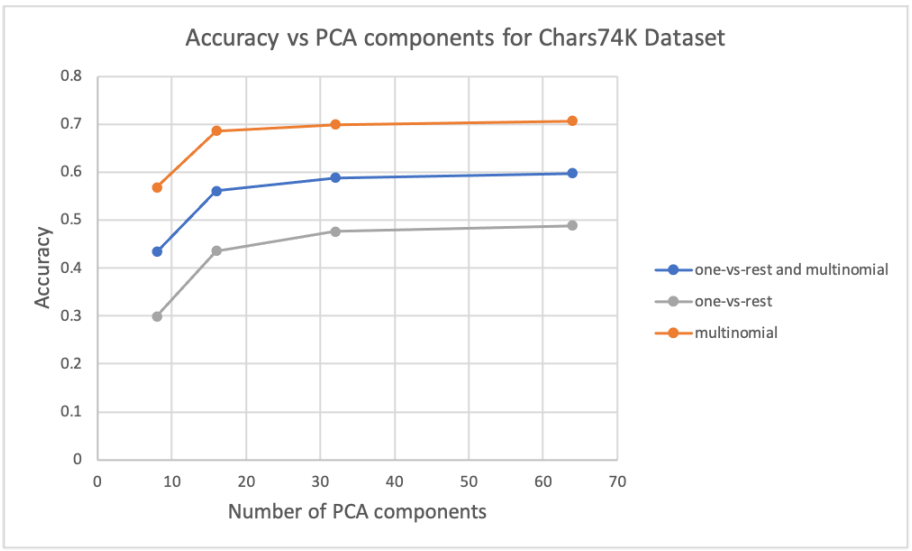
Figure x demonstrates an arbitrary random forest run to explore how the number of principal components affects the algorithm runtime. It is emphasised that the load time is not included here as all of these runs used the same image size of 900 pixels and was thus a constant. As expected, an increase in the number of principal components leads to higher runtimes, but the relationship is sublinear if the first observation (1 component) is overlooked: each additional principal component will increase the speed by progressively less. The figure demonstrates an arbitrary random forest run to explore how the number of principal components affects the algorithm runtime. It is emphasised that the load time is not included here as all of these runs used the same image size of 900 pixels and was thus a constant. As expected, an increase in the number of principal components leads to higher runtimes, but the relationship is sublinear if the first observation (1 component) is overlooked: each additional principal component will increase the speed by progressively less.

4.2 Logistic Regression

Additionally, the regularization will improve numerical stability. Theoretically, if the number of features is more than the number of samples, it will cause overfitting. However, if using 'l2' is still overfitting then 'l1' might be a good choice. After the experiment, we found that all



(a) Average accuracy vs image size

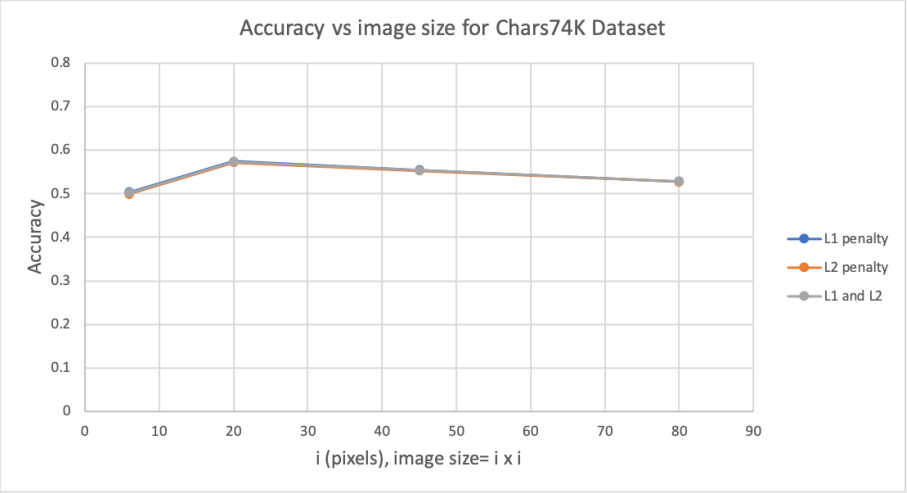


(b) Average accuracy vs PCA

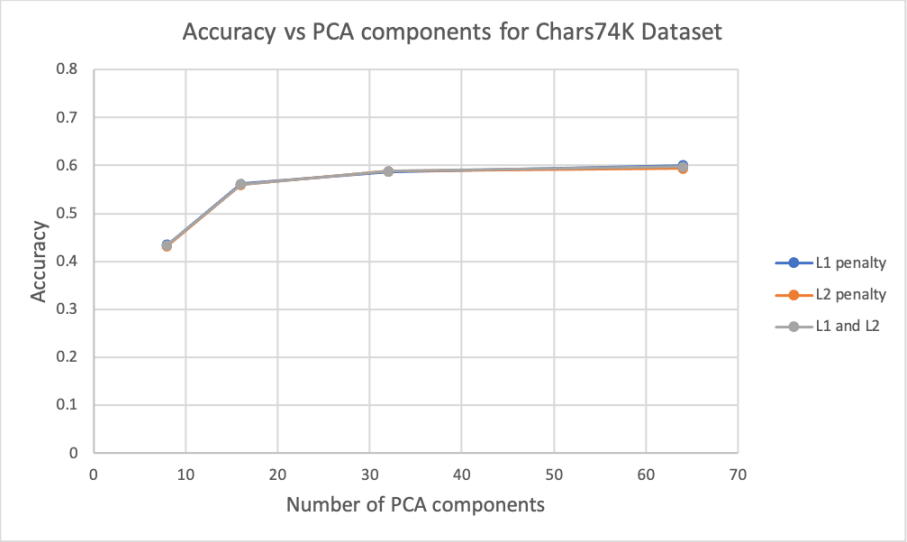
Figure 7: Comparison of model performance on Chars74k dataset

the results of l1 and l2 are almost the same as the results of only l1 or l2 alone (Figure 8). The LR function has a parameter named n_jobs, which can set the number of CPU cores used. If set to -1, all cores in the computer will be used. The ratio of time consumed when not using CPU acceleration to using CPU acceleration is:

- Multinomial = 1.01
- One vs Rest = 4.53



(a) Average accuracy vs image size



(b) Average accuracy vs PCA

Figure 8: Comparison of model performance on Chars74k dataset

4.3 Naive Bayes

For the Naive Bayes classifier, it was decided to explore the relationship of how the model behaves as the number of features are reduced using PCA. Since it is already a quick algorithm, the effect of increasing image sizes to half the original image size was investigated. As can be seen from Figure 9, increasing the number of PCA components does not result in an increase in the accuracy of the model. The best results are obtained when the number of retained PCA components is at approximately 40-45. After this point, the accuracy does not vary significantly, in fact after retaining roughly 260 PCA components, the accuracy of the model does not increase. Therefore, the relationship between the number of PCA components and accuracy is not linearly proportional. This goes to show the significance of applying PCA on

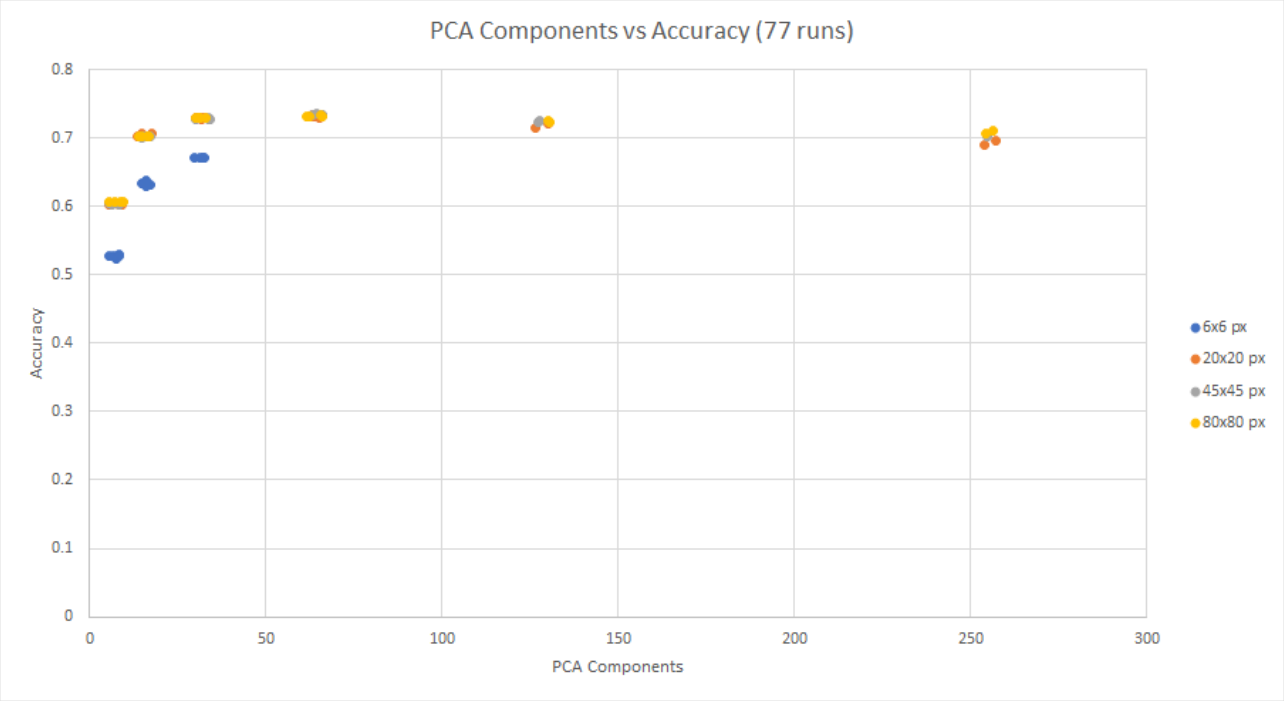


Figure 9: Plot of number of PCA components vs the accuracy of Naive Bayes model.

image datasets. At the start of the model, the original image size was 1200x900 pixels, which corresponded to a matrix of size (3410,5625). For the best accuracy results, after applying PCA, the number of features drastically reduced to 45 (or approximately 75%-80% of the max variance in the data. As such, PCA is an extremely important tool in image classification, it can significantly reduce big data and at the same time, maintain the accuracy of the model.

4.4 Random Forest

Table 10 shows the accuracy of individual runs with different parameter settings, with the pixel count parameter emphasised on the x axis. As can be seen, a higher pixel count tends to yield higher accuracy. For an individual pixel count, the different grouping is due to differing numbers of principal components, with the intra-grouping accuracy differences due to other run parameter settings. Table 11 shows the accuracy of individual runs with different parameter settings, with the number of principal components emphasised on the x axis. In line with previous observations, use of more principal components tends to yield better accuracy, with an eventual levelling off. Particularly interesting is that the three higher pixel parameters tend to give similar results, with the lower image size setting underperforming by a significant margin. This is because the levelling off of the accuracy for image size tends to occur somewhere between 36 and 400 pixels.

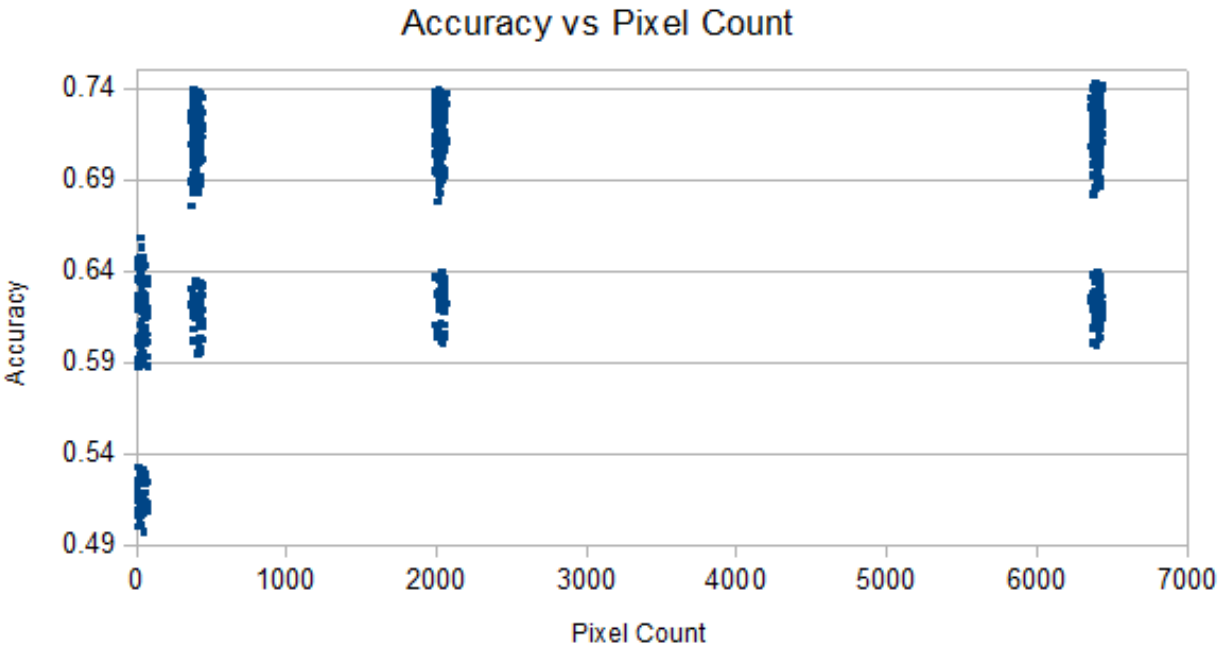


Figure 10: accuracy of individual random forest runs pixel count

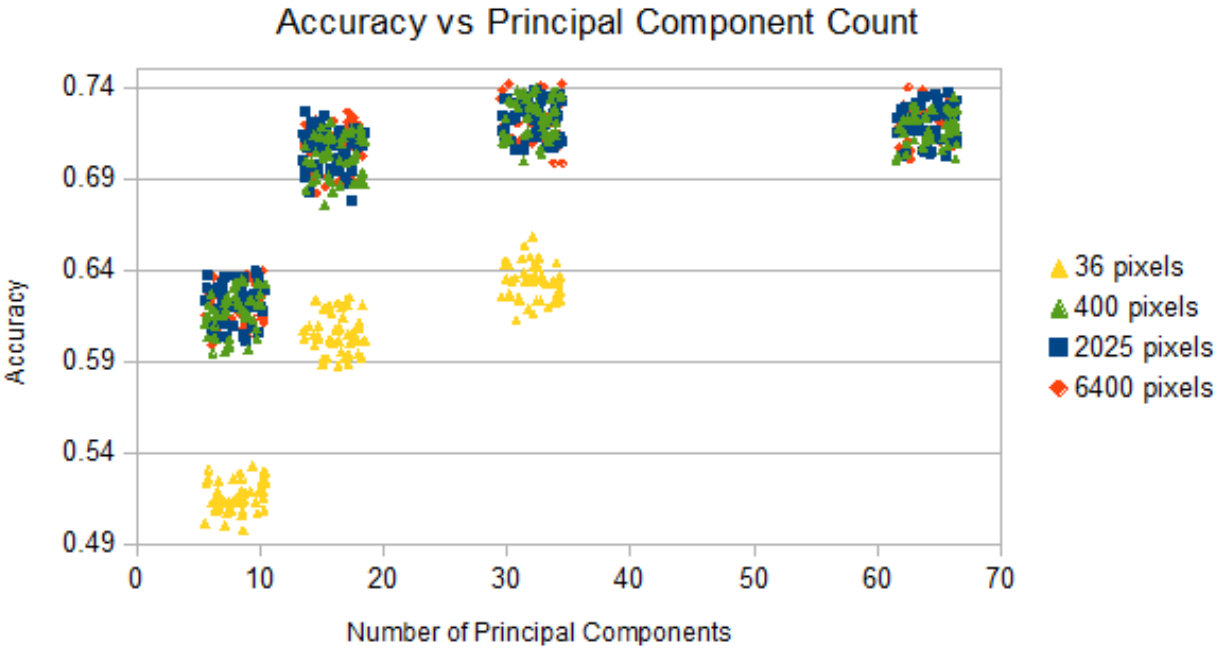


Figure 11: accuracy of individual random forest runs pixel count

5 Conclusion

Three algorithms were explored in this paper, which were logistic regression, naive bayes and random forest. The algorithm with the best accuracy was random forest, followed by naive bayes and finally logistic regression. Thus, for this particular problem, where accuracy is key, the best algorithm to use is random forest.

Naive bayes had the second highest accuracy, but was the fastest algorithm to run. For datasets where speed is most important, naive bayes may be the better choice. However, these results are specific to the dataset used in this paper as well as the particular implementation. Thus, the observation that naive bayes performs reasonably well in terms of accuracy and is the fastest algorithm may or may not generalise to other problems.

An important observation of this paper is that the biggest improvements in accuracy were a result of preprocessing techniques, rather than algorithm specific tuning parameters. The implementation of converting to greyscale cut down the number of features and tended to filter out information that was superfluous to the algorithm. Bounding the image to the contours of the character also saw significant accuracy increases.

Increases in algorithm run speed were highly dependent on number of principal components and also the image size. Generally, the performance improves with a larger image size and more principal components, but this performance levels off rapidly meaning that setting these at higher numbers leads to immaterial improvements in accuracy but significantly slow the algorithm.

This paper has studied fundamental algorithm implementation (logistic regression, naive bayes and decision tree) with some enhancements (random forest, image preprocessing techniques, grid search and parameter tuning). Promising areas for further research are:

- Implementing more advanced algorithms, such as neural networks, which are known to be good at image classification problems, although some caution is required as the dataset is relatively small compared to normal neural network datasets.

- Implementing more advanced feature engineering on preprocessing. The features representing the pixels in this implementation are largely independent; by adding features that better reflect dependency between features and their context within the overall image may provide increases in performance.

- Applying the techniques in this paper into noisier datasets, where the overfitting reduction methods implemented in this paper would become even more important.

References

[1] Mantas J. An overview of character recognition methodologies. *Pattern Recognition*, 19: 425–430, 1986.

[2] de Campos TE, Babu BR, and Varma M. Proceedings of the fourth international conference on computer vision theory and applications. *Pattern Recognition*, 2:273–280, 2009.

[3] Adobe. Adobe Acrobat Pro DC. `https://acrobat.adobe.com/au/en/acrobat/acrobat-pro.html`. [Internet, cited 14 Nov 2020].

[4] ABBYY. ABBYY fineReader PDF. `https://pdf.abbyy.com/`. [Internet, cited 14 Nov 2020].

[5] Sinha Y, Jain P, and Kasliwal N. Comparative study of preprocessing and classification methods in character recognition of natural scene images. *Machine Intelligence and Signal Preprocessing, Advances in Intelligent Systems and Computing*, 390:119–129, 2016.

[6] Oren B, Eli S, and Michal I. In defence of nearest-neighbor based image classification. *IEEE conference on computer vision and pattern recognition*, 2008.

[7] Shilpy B, Mamta G, and Munish K. A technique for offline handwritten character recognitions. *IJCAT International Journal of Computing and Technology*, 1:210–215, 2014.

[8] Kinjal B, Radhika N, and Umapada P. Recognition of similar shaped handwritten characters using logistic regression. *10th IAPR International Workshop on Document Analysis Systems*, 2010.

[9] de Campos TE, Babu BR, Varma M. The chars74k dataset: character recognition in natural images. `http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/`. [Internet, cited 28 Oct 2020].

[10] Towards Data Science. About Train, Validation and Test Sets in Machine Learning. `https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7`.

[11] Machine Learning Mastery. How to Develop Voting Ensembles with Python. `https://machinelearningmastery.com/voting-ensembles-with-python/`, .

[12] Machine Learning Mastery. Naive Bayes for Machine Learning. `https://machinelearningmastery.com/naive-bayes-for-machine-learning/`, .

[13] Stanford University. Maximum Likelihood Estimation Lecture. http://statweb.stanford.edu/~susan/courses/s200/lectures/lect11.pdf.

[14] Scikit Learn. Random Forest Classifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

A Appendix

A.1 Running code in Jupyter Notebook

Please follow the steps outlined below to run the submitted .ipynb files successfully on Jupyter Notebook. Two Jupyter notebook files have been submitted: *group44_best_algorithm.ipynb* and *group44_other_algorithm.ipynb*. Both files follow the same general structure and unless otherwise stated, must follow the same steps shown below.

1. The dataset used for this project is the 'EnglishHnd.tgz' file. Available to download from http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/. Please do not download 'EnglishImg.tgz' or 'EnglishFnt.tgz', or any of the Kannada datasets as our models were only trained and tested on the 'EnglishHnd' dataset.

2. **IMPORTANT:** once downloaded, uncompress the folder and navigate to './English/', which is the main folder. You should now see two subfolders 'Hnd' and 'Img'. Please set this as the working directory, i.e. run the .ipynb files in this 'English' folder.
   If the working directory needs to be changed, please replace the working directory in the variable 'path_find_images' (in the first cell of notebook) to the appropriate directory.

3. Execute Cell 1. It contains all required libraries for the model. The module cv2 may need to be installed separately if it has not been used on the host computer before. In that case, please type the command

   ```
   pip install opencv-python
   ```

   on Jupyter Notebook or Anaconda Command Prompt (Terminal) to install the module. If command is run on Jupyter, the kernel may need to be restarted for the package to be implemented in the program.

4. The following cells must be executed in a top to bottom order - each preceding cell sets up the calculations for the next cell.

5. Please note that computationally expensive code has been commented out for the marker's convenience, which includes the hyperparameter tuning using grid search.

6. The best parameters and hyperparameters have already been applied on both .ipynb files.

A.2 Group Member Contribution

The table below shows the contribution of each group member. The group agrees each member contributed equally.

Alex Augustine

- Wrote up Abstract, Introduction, Literature review
- Came up with code for confusion matrix plot
- Included contribution of each team member in appendices
- Included instructions on how to run code in Appendix
- Applied Naive Bayes algorithm to Chars74k dataset
- Wrote up results and discussion for Naive Bayes algorithm
- Compared all three algorithms in Discussion and Results
- Merged preliminary code from team members to form base code
- Wrote up Methodology for Naive Bayes
- Organised and set up team meetings on Zoom
- Contributed to other parts of the report

Oliver Stefan Nawrot

- Wrote up Conclusion
- Ran tests on all algorithms to record timings
- Made table of model statistics (f1,recall,precision,accuracy) for each algorithm
- Applied Random Forest classifier to Chars74k dataset
- Wrote up results and discussion for Random Forest algorithm
- Came up with code for grid search
- Implemented idea of adding extra feature of representing mean value of pixel data
- Contributed to designing code to read in image data as numpy arrays, and to designing code for image modifications

- Compared all three algorithms in Discussion and Results
- Wrote up Methodology for Random Forest
- Organised and set up team meetings on Zoom
- Contributed to other parts of the report

Zirong Guo

- Wrote up preprocessing methods
- Applied Logistic Regression classifier to Chars74k dataset
- Wrote up results and discussion for Logistic Regression algorithm
- Compared all three algorithms in Discussion and Results
- Contributed to designing code to read in image data as numpy arrays, and to designing code for image modifications
- Wrote up Methodology for Logistic Regression
- Organised and set up team meetings on Zoom
- Contributed to other parts of the report

A.3 System Specifications

A.3.1 Hardware Specifications

The optimal algorithm speed tests were run on the following machine specifications

- CPU: Intel® Core(TM) i7- 3537U CPU @ 2.00GHz
- RAM: 8.0 GB DDR3

A.3.2 Software Specifications

- Windows version: 8.1, 64 bit
- Python version: 3.8.3
- IDE: Jupyter Notebook (v number)
- sklearn v number
- open cv v number