COMP5318
Assignment 1

SID: 490196302
NAME: Andy Lu

SID: 500672949
NAME: Shiming Guo

Introduction

With the rapid development of the world in recent decades, we have entered the information-rich digital age, with cell phones and computers coming into millions of homes, and our lives have become much easier. We usually search for keywords to get results in traditional search methods and on most platforms, such as Google and Amazon, but with the explosion in the number of images, keyword search has exposed many problems. First, simple keywords can hardly summarise the complete characteristics of an image. Secondly, the primary application platforms prefer to recommend items that have paid for advertising and promotion to customers. Thus, this causes a lot of bias in our search process, resulting in very low efficiency.

In this case, the image-based search method has gained attention, and it can make up for the shortcomings of the traditional keyword search method. The image content-based search method extracts the features of the images through a large number of training sets. The method applies a suitable classifier to measure the similarity and then sorts the results according to the similarity to get the retrieval results. Compared to keyword searches, the efficiency of image searches can be significantly improved.

Generally, before applying classifiers, we need to pre-process the data to reduce our training or prediction time afterwards. Different pre-processing methods will affect our prediction accuracy later. The choice of image classifiers and the choice of parameters for the same classifier will also affect image classification accuracy. Therefore, it is essential to select different pre-processing methods, classifiers and parameters, which have important implications for our image search.

We first apply HOG and PCA to pre-process the data in this experiment. There are 30,000 rows in the dataset on canvas, and each row represents a 28x28 grey image. We first use the first 22,500 rows as the training set to train different classifiers, and after that, we use the last 7,500 rows as the validation set and apply grid search to check the accuracy of various parameters. The performance of different classifiers is compared to find the best parameters and classifiers. Finally, the performance of the different classifiers is compared. Analysis and recommendations are also given. The methods, experimental results and discussion, and summary are presented in the following sections.

Methods:

Pre-processing methods:

Normalisation and standardisation:

Normalisation and standardisation are fundamental tasks in data mining. Different evaluation indicators often have different magnitudes and magnitude units. Such a situation will affect the results of data analysis. In order to eliminate the influence of the magnitude between indicators, data standardisation processing is required. After the original data has been standardised, the indicators are in the same magnitude order and suitable for comprehensive comparative evaluation.

Normalisation means limiting the range of our data values to between [0,1], and we usually use the following formula(Fig.1). Standardisation means making the processed data conform to a standard normal distribution, i.e., a mean of 0 and a standard deviation of 1(Fig.2). μ is the mean of the data, and δ is the standard deviation of the data.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Fig.1. Normalisation

$$x' = \frac{x - \mu}{\delta}$$

Fig.2. Standardisation

Hog feature extraction:

The full name of HOG is Histogram of Oriented Gradient. In each image, there are many pixel points. Each pixel has its gradient size and gradient direction. Consider each pixel point as a two-dimensional function. Each point has x, y two-directional derivatives(Fig.3) and then calculates the gradient size and direction(Fig.4). The gradient value is large means that the difference in value between pixels is large, and it can be judged as an edge.

$$\frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y) = gx$$
$$\frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y) = gy$$

Fig.3. Derivatives in x, y direction

$$\boldsymbol{g} = \sqrt{\boldsymbol{g}_x^2 + \boldsymbol{g}_y^2}$$
$$\boldsymbol{\theta} = \mathbf{arctan}\frac{\boldsymbol{g}_y}{\boldsymbol{g}_x}$$

Fig.4. the gradient size(g) and direction(θ)

Now that you have the magnitude and direction of the gradient for each pixel, the next step is to combine this data into a histogram. First, divide a graph into several small cells, then create an array. Each position of this array represents a gradient direction. Judge the direction and size of each pixel in the cell, fill in the value of the size to the module of the corresponding direction in the array, and then get the gradient histogram of the cell.

First, normalise the adjacent cells, choose n cells as blocks, normalise the blocks to get a histogram, move the blocks by one cell distance and normalise again to get another histogram. So on, we can get features of HOG, i.e. the gradient values and the direction of the gradient in each block after normalisation.

PCA:

The main idea of PCA is to map n-dimensional features to k-dimensions, which are new orthogonal features, also known as principal components, reconstructed from the original n-dimensional features. The choice of new axes is closely related to the data itself. The first new axis is chosen in the direction of the largest variance in the original data, the second axis is selected from the plane orthogonal to the first axis that makes the largest variance, and the third axis is the plane orthogonal to the first and second axis that makes the largest variance. By analogy, n such axis can be obtained. With the new axis obtained in this way, we find that most of the variance is contained in the first k axis and that the latter axis contains almost zero variance. This is equivalent to reducing the dimensionality of the data by keeping only the dimensional features that contain the majority of the variance and ignoring those that contain almost zero variance.

Machine learning methods:

KNN:

The core idea of the KNN algorithm is that if the majority of the K most adjacent samples in the feature space belong to that category, then that sample also belongs to that category and has the characteristics of the samples in that category. The KNN method relates to only a tiny number of neighbouring samples when making category decisions. KNN algorithm has the following two elements:

1)Selection of K value

There is only one hyperparameter k in the KNN algorithm, and the determination of the k-value has a crucial impact on the prediction results of the KNN algorithm.

If the value of k is relatively small, it is equivalent to training our samples to predict instances in a smaller domain. At this point, the approximation error of the algorithm will be smaller, as only training samples that are similar to the input instances will work on the prediction results. However, there are obvious disadvantages: the estimation error of the algorithm is relatively large, and the prediction result is susceptible to the nearest neighbour, i.e. the prediction will be wrong if the nearest neighbour is a noisy point. Therefore, too small a value of k can easily lead to overfitting the KNN algorithm.

Similarly, if the k value is chosen to be large, training samples that are farther away can also affect the prediction results of the examples. The model is relatively robust, and individual noise points do not affect the final prediction result. However, the disadvantage is also obvious: the nearest neighbour error of the algorithm will be large, and distant points (which are not similar to the predicted example) will also have an impact on the prediction results, making the prediction results more biased, and the model will be prone to under-fitting. Therefore, in actual practice, we generally use cross-validation to select the k-value.

2） Distance measurement

The distance between two points in the sample space indicates the degree of similarity between the two sample points: the shorter the distance, the higher the degree of similarity; conversely, the lower the degree of similarity. Common distance measures include Minkowski distance, Euclidean distance, Manhattan distance, Chebyshev distance and cosine distance. The Euclidean distance is the most commonly used distance measure(Fig.5).

$$d_{xy} = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

Fig.5. Euclidean distance

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

Fig.6. Sigmoid

Logistic regression:

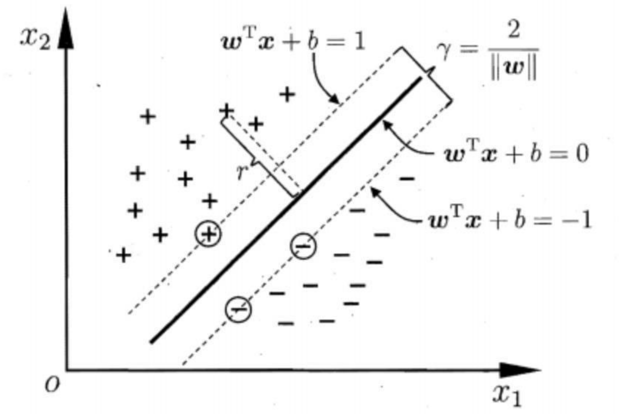
For a binary classification problem, y ∈ {0, 1}, 1 denotes a positive class, and 0 denotes a negative class. Logistic regression is based on a linear function. It outputs the actual value of the prediction and finds a hypothesis function Sigmoid (Fig.7) to map the actual value to between 0 and 1. If the hypothesis function is greater than 0.5, the prediction belongs to the positive class; if the hypothesis function is less than 0.5, the prediction belongs to the negative class.

Naive Bayes:

Naive Bayes is a simple but powerful linear classifier whose main basis is the Bayesian formula, defined as follows(Fig.7). It is only called naive because it assumes that features are independent of each other, but this assumption is mainly untrue in real life. Then even under conditions where the assumption does not hold, it still performs well, especially with small samples. However, if there is a strong correlation between each feature and non-linearity, the classification problem can lead to poor results from the Naive Bayes model.

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Fig.7. Bayes formula



x_2
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 1$
$\gamma = \frac{2}{\|\boldsymbol{w}\|}$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = -1$
r
o
x_1

Fig.8. The principle diagram of the SVM

Support vector machine:

In machine learning, SVM (Support Vector Machine) is a supervised learning model and a binary classification model commonly used for pattern recognition, classification and regression analysis. The purpose of the SVM is to draw a line that best distinguishes between the two types of points (Fig.8) so that if new points become available later, the line will also make a reasonable classification. SVM is suitable for medium to large data samples and non-linear, high-dimensional classification problems.

Random forest:

Random Forest (Fig.9) is an algorithm that integrates multiple trees through the idea of ensemble learning. Its basic unit is the decision tree. Each decision tree is a classifier, then for one input sample, N trees will have N classification results. The random forest integrates all the classification votes and assigns the category with the highest number of votes as the final output. Random forests build multiple decision trees and merge them to obtain more accurate and stable predictions. A major advantage of random forests is that they can be used for classification and regression problems.

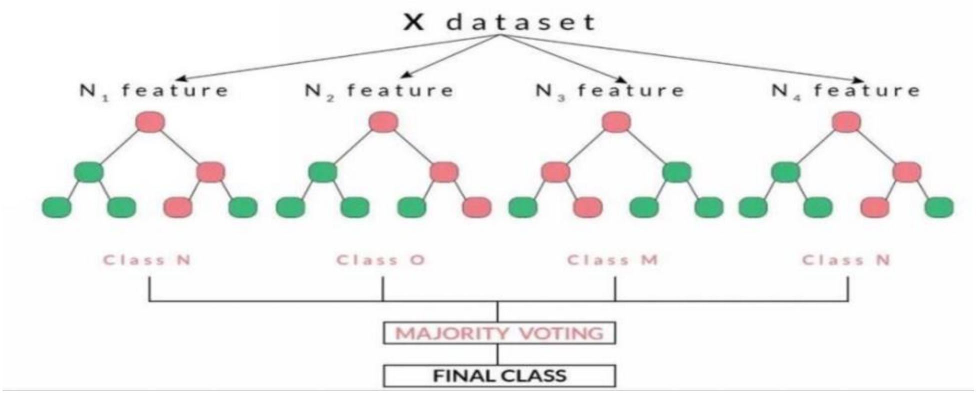
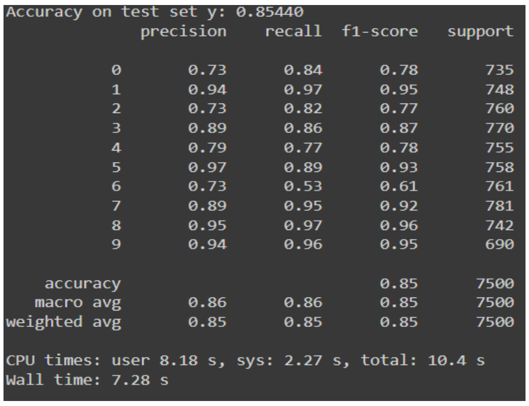


Fig.9. The principle diagram of the Random Forest

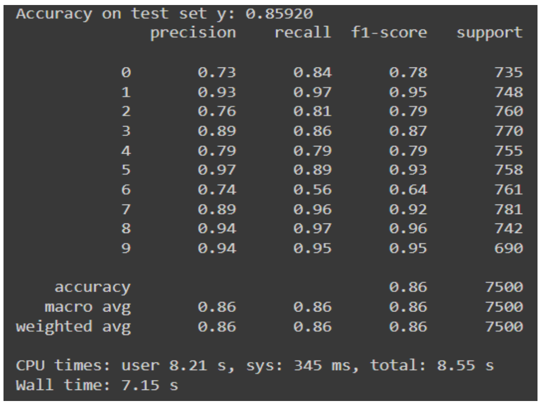
Result and discussion:

KNN:

For the k-nearest neighbour with the default parameters, it obtains an accuracy of 0.8544. After performing with 5 fold cross-validation, we found out that the best hyperparameter for the k-nearest neighbour is to use 7 for the 'n_neighbours' and 2 for the parameter 'p', which stands for the Euclidean distance, with the turning the k-nearest neighbour classifier got an accuracy score of 0.8592 which is slightly better than the default one with an improvement of 0.0048 on accuracy. For the run time, it took a total of 14.43 seconds to run both default and turned classifiers.



```
Accuracy on test set y: 0.85440
              precision    recall  f1-score   support

           0       0.73      0.84      0.78       735
           1       0.94      0.97      0.95       748
           2       0.73      0.82      0.77       760
           3       0.89      0.86      0.87       770
           4       0.79      0.77      0.78       755
           5       0.97      0.89      0.93       758
           6       0.73      0.53      0.61       761
           7       0.89      0.95      0.92       781
           8       0.95      0.97      0.96       742
           9       0.94      0.96      0.95       690

    accuracy                           0.85      7500
   macro avg       0.86      0.86      0.85      7500
weighted avg       0.85      0.85      0.85      7500

CPU times: user 8.18 s, sys: 2.27 s, total: 10.4 s
Wall time: 7.28 s
```

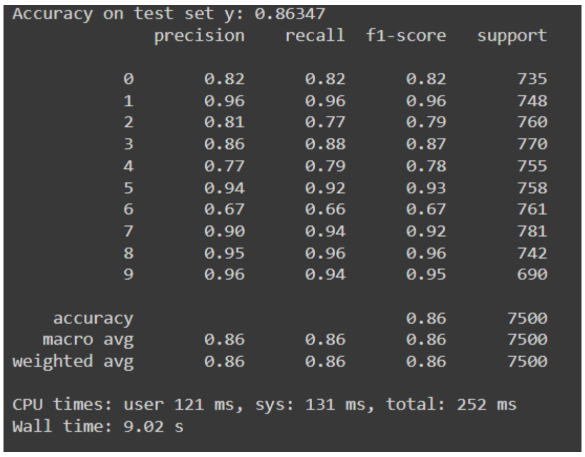


```
Accuracy on test set y: 0.85920
              precision    recall  f1-score   support

           0       0.73      0.84      0.78       735
           1       0.93      0.97      0.95       748
           2       0.76      0.81      0.79       760
           3       0.89      0.86      0.87       770
           4       0.79      0.79      0.79       755
           5       0.97      0.89      0.93       758
           6       0.74      0.56      0.64       761
           7       0.89      0.96      0.92       781
           8       0.94      0.97      0.96       742
           9       0.94      0.95      0.95       690

    accuracy                           0.86      7500
   macro avg       0.86      0.86      0.86      7500
weighted avg       0.86      0.86      0.86      7500

CPU times: user 8.21 s, sys: 345 ms, total: 8.55 s
Wall time: 7.15 s
```

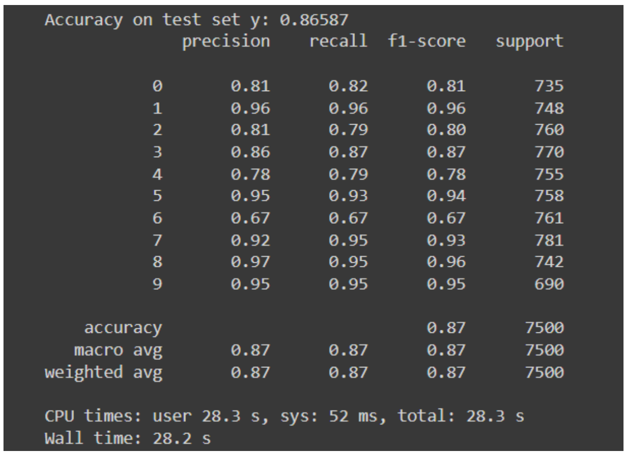
(Figure 10 and 11, on the left is the accuracy score for KNN with default parameters, right is after turning)

Logistic regression:

With the default parameter, the logistic regression classifier gives a result of the accuracy of 0.8635. There is the inverse of regularisation strength 'C' and the solver to turn for the logistic regression. After the 5 fold cross-validation, we got the 'C' to be 5 and the solver to be 'sag'. The turning brings a 0.0024 improvement to the classifier, and the accuracy score is 0.8659. The run time for the classifier is 9.02 seconds for the default parameter and 28.2 seconds for the turned one, since we allow the turned classifier iteration more by setting the 'max_iter' to 800, so it will most likely diverge.



```
Accuracy on test set y: 0.86347
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       735
           1       0.96      0.96      0.96       748
           2       0.81      0.77      0.79       760
           3       0.86      0.88      0.87       770
           4       0.77      0.79      0.78       755
           5       0.94      0.92      0.93       758
           6       0.67      0.66      0.67       761
           7       0.90      0.94      0.92       781
           8       0.95      0.96      0.96       742
           9       0.96      0.94      0.95       690

    accuracy                           0.86      7500
   macro avg       0.86      0.86      0.86      7500
weighted avg       0.86      0.86      0.86      7500

CPU times: user 121 ms, sys: 131 ms, total: 252 ms
Wall time: 9.02 s
```

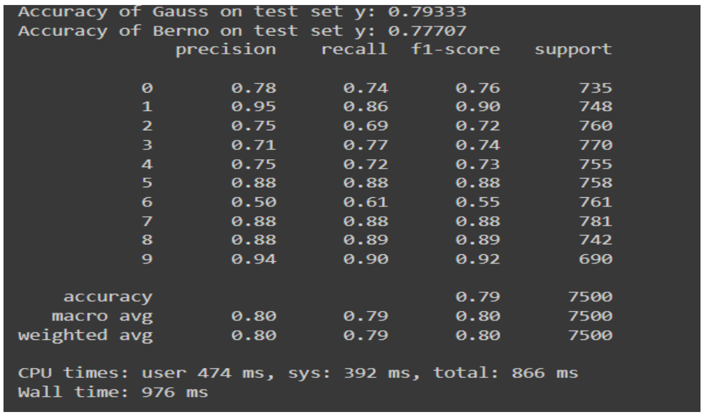


```
Accuracy on test set y: 0.86587
              precision    recall  f1-score   support

           0       0.81      0.82      0.81       735
           1       0.96      0.96      0.96       748
           2       0.81      0.79      0.80       760
           3       0.86      0.87      0.87       770
           4       0.78      0.79      0.78       755
           5       0.95      0.93      0.94       758
           6       0.67      0.67      0.67       761
           7       0.92      0.95      0.93       781
           8       0.97      0.95      0.96       742
           9       0.95      0.95      0.95       690

    accuracy                           0.87      7500
   macro avg       0.87      0.87      0.87      7500
weighted avg       0.87      0.87      0.87      7500

CPU times: user 28.3 s, sys: 52 ms, total: 28.3 s
Wall time: 28.2 s
```

(Figure 12 and 13, on the left is the accuracy score for logistic regression with default parameters, the right is after turning)

Naive Bayes:

Since Naive Bayes there are no hyperparameters to turn. Therefore, we use two different naive Bayes methods and compare the results. Gaussian and Bernoulli are chosen since these two are suitable for multidimensional numerical value datasets. The results of Gaussian have an accuracy score of 0.7933 and Bernoulli scores of 0.7771. These two classifiers run very fast with a total runtime of 976ms.

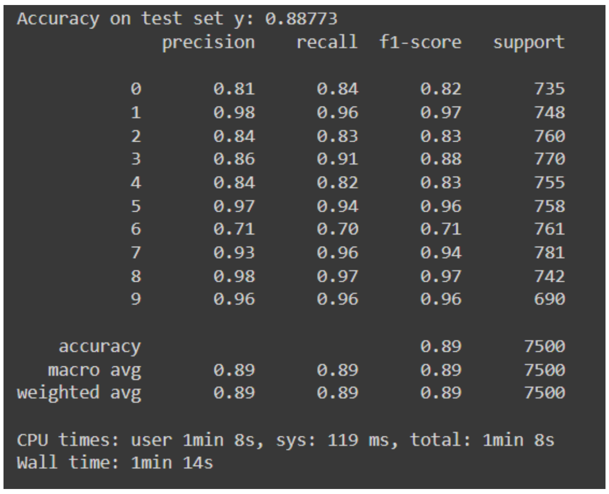


```
Accuracy of Gauss on test set y: 0.79333
Accuracy of Berno on test set y: 0.77707
              precision    recall  f1-score   support

           0       0.78      0.74      0.76       735
           1       0.95      0.86      0.90       748
           2       0.75      0.69      0.72       760
           3       0.71      0.77      0.74       770
           4       0.75      0.72      0.73       755
           5       0.88      0.88      0.88       758
           6       0.50      0.61      0.55       761
           7       0.88      0.88      0.88       781
           8       0.88      0.89      0.89       742
           9       0.94      0.90      0.92       690

    accuracy                           0.79      7500
   macro avg       0.80      0.79      0.80      7500
weighted avg       0.80      0.79      0.80      7500

CPU times: user 474 ms, sys: 392 ms, total: 866 ms
Wall time: 976 ms
```

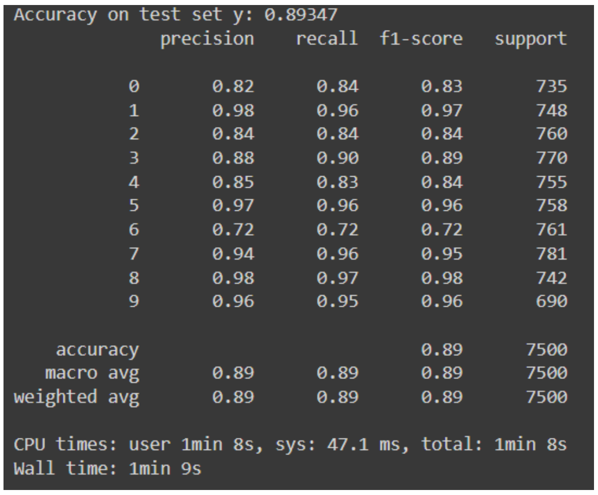
(Figure 14, Naive Bayes accuracy score for Gaussian and Bernoulli)

Support vector machine:

The support vector machine gives us the best result of all the classifiers with default parameters. It has an accuracy score of 0.8877, which is even higher than other turned classifiers. After we evaluate the performance by running a 5 fold cross-validation, it gives us the best value of 3 for turning the parameter 'C' for the regularisation. We also found that the gamma parameter with a value of 1.2 always brings us the best result in this dataset. Therefore, turning the support vector machine gives us an accuracy of 0.8935, which is the best among all classifiers we used. Thus we decided to use this classifier to predict the testing data set. After making the prediction and uploading it to Kaggle, we receive an accuracy score of 0.9095. The Support vector machine took a relatively long time to predict the result, and each time it took around 1 minute and 10 seconds to run.



```
Accuracy on test set y: 0.88773
              precision    recall  f1-score   support

           0       0.81      0.84      0.82       735
           1       0.98      0.96      0.97       748
           2       0.84      0.83      0.83       760
           3       0.86      0.91      0.88       770
           4       0.84      0.82      0.83       755
           5       0.97      0.94      0.96       758
           6       0.71      0.70      0.71       761
           7       0.93      0.96      0.94       781
           8       0.98      0.97      0.97       742
           9       0.96      0.96      0.96       690

    accuracy                           0.89      7500
   macro avg       0.89      0.89      0.89      7500
weighted avg       0.89      0.89      0.89      7500

CPU times: user 1min 8s, sys: 119 ms, total: 1min 8s
Wall time: 1min 14s
```



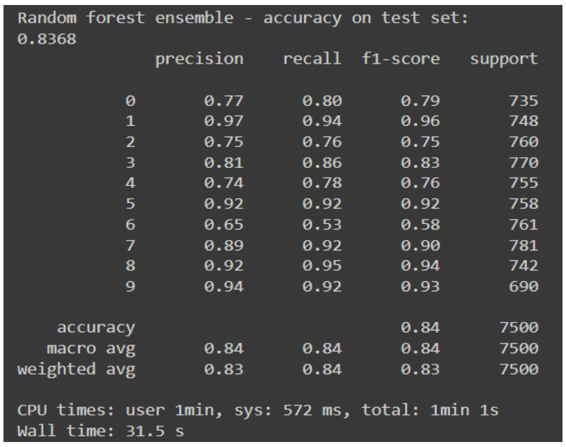
```
Accuracy on test set y: 0.89347
              precision    recall  f1-score   support

           0       0.82      0.84      0.83       735
           1       0.98      0.96      0.97       748
           2       0.84      0.84      0.84       760
           3       0.88      0.90      0.89       770
           4       0.85      0.83      0.84       755
           5       0.97      0.96      0.96       758
           6       0.72      0.72      0.72       761
           7       0.94      0.96      0.95       781
           8       0.98      0.97      0.98       742
           9       0.96      0.95      0.96       690

    accuracy                           0.89      7500
   macro avg       0.89      0.89      0.89      7500
weighted avg       0.89      0.89      0.89      7500

CPU times: user 1min 8s, sys: 47.1 ms, total: 1min 8s
Wall time: 1min 9s
```

(Figure 15 and 16, on the left is the accuracy score for SVM with a default parameter, the right is after turning)

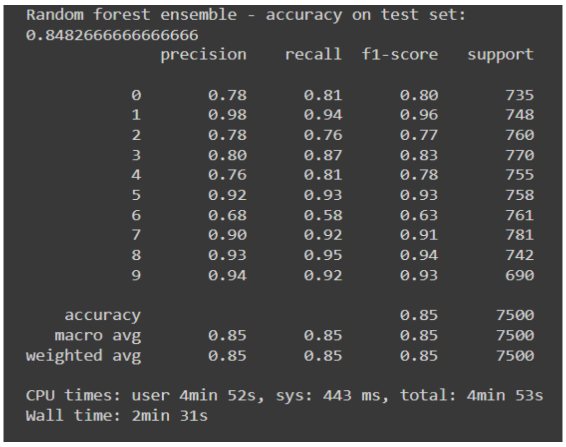
Random forest:

The random forest classifier with default parameters has a relatively low accuracy score of 0.8368. After running the 5 fold cross-validation, we get that the more estimators we have in the classifier, the better the model performs. Since there is a limit in our computation power, we choose 500 as the number of estimators, which gives us a satisfactory improvement in accuracy score to 0.8483 and can still run within 3 minutes with an average of 2 minutes and 30 seconds to finish.

(Figure 17 and 18, on the left is the accuracy score for the random forest with default parameters, the right is after turning)



```
Random forest ensemble - accuracy on test set:
0.8368
              precision    recall  f1-score   support

           0       0.77      0.80      0.79       735
           1       0.97      0.94      0.96       748
           2       0.75      0.76      0.75       760
           3       0.81      0.86      0.83       770
           4       0.74      0.78      0.76       755
           5       0.92      0.92      0.92       758
           6       0.65      0.53      0.58       761
           7       0.89      0.92      0.90       781
           8       0.92      0.95      0.94       742
           9       0.94      0.92      0.93       690

    accuracy                           0.84      7500
   macro avg       0.84      0.84      0.84      7500
weighted avg       0.83      0.84      0.83      7500

CPU times: user 1min, sys: 572 ms, total: 1min 1s
Wall time: 31.5 s
```



```
Random forest ensemble - accuracy on test set:
0.8482666666666666
              precision    recall  f1-score   support

           0       0.78      0.81      0.80       735
           1       0.98      0.94      0.96       748
           2       0.78      0.76      0.77       760
           3       0.80      0.87      0.83       770
           4       0.76      0.81      0.78       755
           5       0.92      0.93      0.93       758
           6       0.68      0.58      0.63       761
           7       0.90      0.92      0.91       781
           8       0.93      0.95      0.94       742
           9       0.94      0.92      0.93       690

    accuracy                           0.85      7500
   macro avg       0.85      0.85      0.85      7500
weighted avg       0.85      0.85      0.85      7500

CPU times: user 4min 52s, sys: 443 ms, total: 4min 53s
Wall time: 2min 31s
```

Comparison between each machine:

We compute each model's accuracy score in a hold-out valid set to compare each machine learning model. The visualised results indicate that support vector machines perform the best within the five models we use, with an accuracy score of 0.8931. The naive Bayes performs the worst with an accuracy score of 0.7933. The rest of the models perform relatively similarly around the accuracy of 0.85.

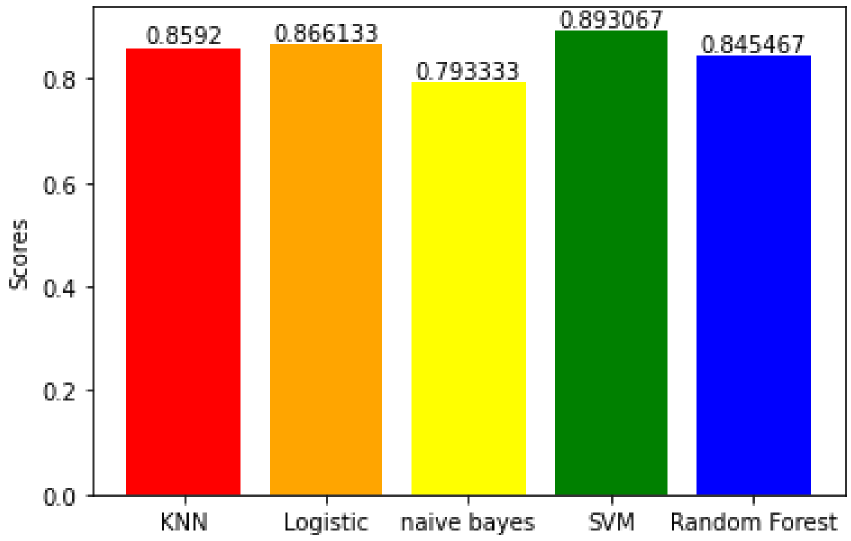


Figure 19. Accuracy is the different classifier

For the accuracy analysis, these results generally hold for our expectation, with support vector machines performing the best since the support vector machines generally perform well on image and high dimension data. Our project's data are images data with a high dimension space. Although the data is multi-class, the support vector machines can still use kernel tricks to perform a good classification. On the other hand, the naive Bayes performs the worst in this dataset for a few reasons. The naive Bayes assumes that all features are independent, which can cause some feature pixels in the picture data to have a higher bias than we want and hinder its performance compared to other models. The logistic regression gives us the second-best performance. It performs the linear search in the entire image space. Therefore it is suitable for this dataset. It does suffer from complex relations that could cause it not to perform as well as SVM. KNN has a middle score on all the models we tried, it works well with a small-sized dataset, but our image dataset is quite large. Thus KNN struggles to make a good prediction for new data entries. Surprisingly Random Forest does not perform well in this case. We observe a few limitations that might cause it not to perform well. The number of estimators we used during the training is only 500 due to the limitation of the computational power. However, if we can have more estimators in the classifier, it might perform better. The other reason is that Random Forest is an ensemble of decision trees. It also fails to determine the significance of each feature in our image dataset.

For time complexity analysis, for our operations, Naive Bayes has the fastest running time with O(n*d) training time complexity and O(c*d) running time complexity, followed by KNN with time complexity of O(knd) and logistic regression with time complexity of O(nd), these two both finished within one minute. The support vector machine has a training time complexity of $O(n^2)$ and a runtime complexity of O(k*d). Random forest with training time complexity of O(n*log(n)*d*k) and runtime complexity of O(depth of tree* k) are two relatively time-consuming algorithms because support vector machines take a long time to train with a large dataset since it has to perform kernel function. The Random forest needs to compute the k amount of decision trees. These two hold for our operations that SVM took around 1 minute and 10 seconds and Random forest took 2 minutes and 30 seconds for a 500 estimator classifier.

For the space complexity, KNN has O(nd), "Testing takes longer because you have to compare every test instance to the whole training data."(Paritosh Kumar, 2019). Logistic regression has a space complexity of O(d), support vector machine has a space complexity of O(k*d) and random forest have a space complexity of O(depth of tree *k).

Conclusion:

This experiment focuses on classifying several classifiers on several 28*28 grey images, comparing their performance, and finding the best classifier. First, we pre-process the data using HOG, normalisation, standardisation and PCA. HOG constitutes features by computing and counting the histogram of gradient directions in local regions of the image. The HOG-processed data performs better than the original and improves our accuracy. Standardisation and normalisation put all the data between [0, 1], which also satisfies the normal distribution simultaneously, and PCA reduces the dimensionality of the data, both of which will substantially reduce the time required to train the model.

The accuracy of logistic regression is slightly higher than that of KNN, and KNN is slightly higher than random forest, but their accuracy is roughly around 0.85. The accuracy of Naive Bayes is much lower than that of the above three models, approximately about 0.8, while SVM has the highest accuracy, reaching around 0.9. This is because SVM usually performs well on image data and high-dimensional data. In contrast, Naive Bayes assumes that attributes are independent of each other and therefore does not classify well when the number of attributes is relatively large or when the correlation between attributes is large.

The computation time of Naive Bayes was the fastest, followed by KNN and logistic regression, while SVM and random forest took longer. This is because Naive Bayes converges faster than logistic regression, KNN has low training time complexity and high computational complexity compared to Naive Bayes. SVM is a time-consuming algorithm because it needs to run kernel functions, and in the random forest, we need to compute 500 classifiers, which also takes a lot of time.

In conclusion, Naive Bayes is more suitable for small and independent datasets, while the other four algorithms are more suitable for large and complex datasets, especially SVM, which achieved the highest accuracy in this experiment. We will improve the current experiments and enhance the pre-processing work in future research. In addition, different dimension reduction methods and finding optimal parameters can be tried.

REFERENCE:

Mrinal Tyagi. (2021). HOG (Histogram of Oriented Gradients): An Overview, Retrieved from https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f

Zakaria Jaadi. (2021). A Step-by-Step Explanation of Principal Component Analysis (PCA), Retrieved from https://builtin.com/data-science/step-step-explanation-principal-component-analysis

Aniruddha. (2020). Feature Scaling for Machine Learning: Understanding the Difference Between Normalisation vs. Standardisation, Retrieved from https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

Onel Harrison. (2018). Machine Learning Basics with the K-Nearest Neighbors Algorithm, Retrieved from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Saishruthi Swaminathan. (2018). Logistic Regression — Detailed Overview, Retrieved fromhttps://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Sunil. (2017). 6 Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R, Retrieved from https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

Bruno Stecanella. (2017). Support Vector Machines (SVM) Algorithm Explained, Retrieved from https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/

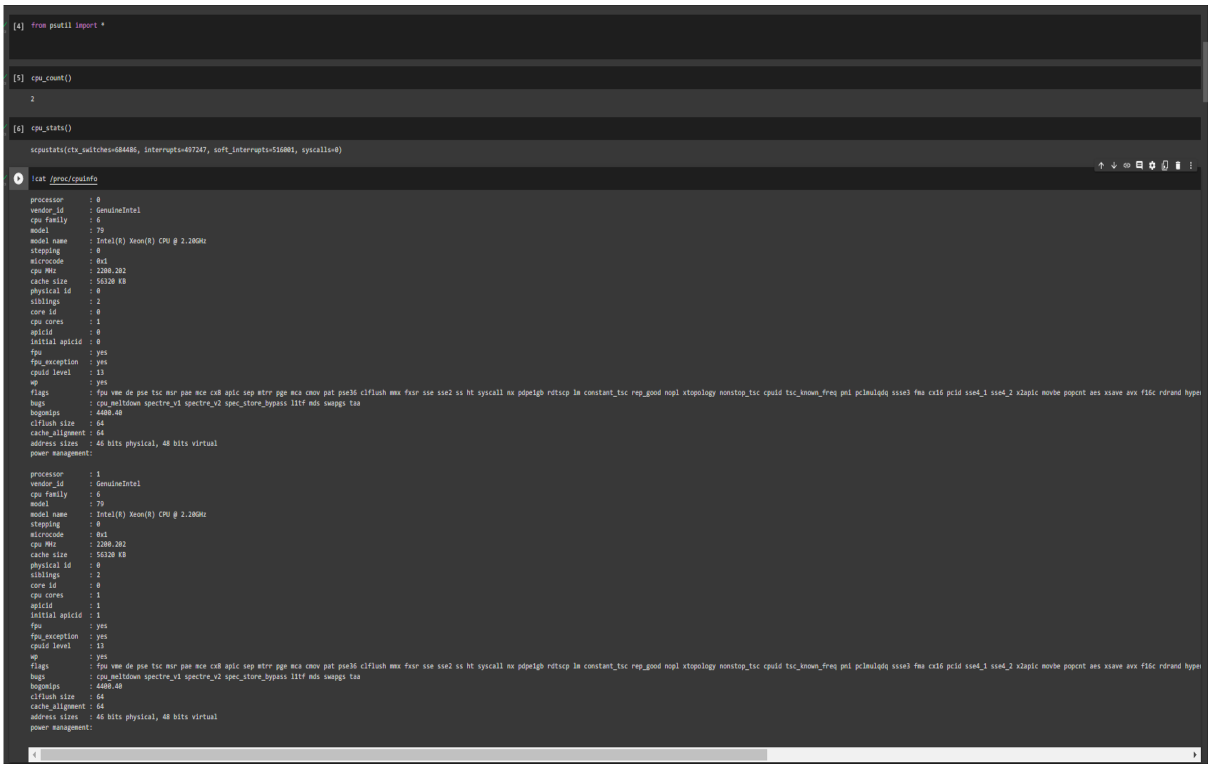
Onesmus Mbaabu. (2020). Introduction to Random Forest in Machine Learning, Retrieved from https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/
Paritosh Kumar. (2019). Computational Complexity of ML Models, Retrieved from https://medium.com/analytics-vidhya/time-complexity-of-ml-models-4ec39fad2770

APPENDIX:

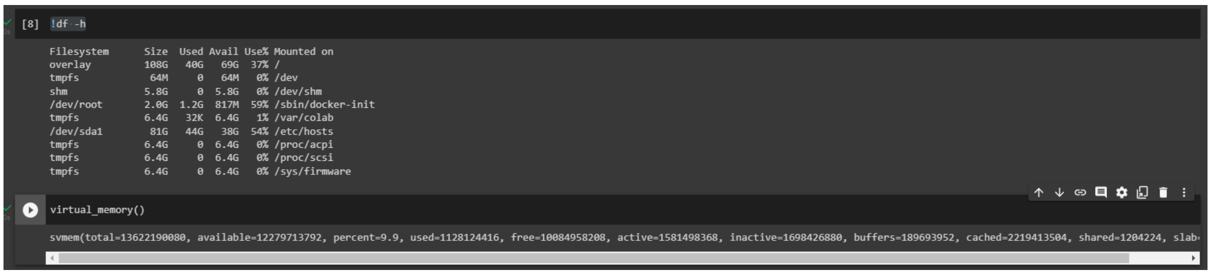
The file input is not attached to any google drive, so please create two Folders that have the name "Input" and "Output" at the same level of the coding file and upload the "train.csv" and "test.csv" provided in "Input" folder and the best result of our model's output will be placed in "Output" folder.
Our code can be run simply one by one cell. The logic for running the code is as follows.

1. If this file, please make sure to use the code section in 3.1 to check the version of the matplotlib. If it is below version 3.4.3, you can use the following code section to update the matplotlib, restart the runtime, and then re-run the file with the new version of matplotlib since there are functions for plotting that require a more recent version of matplotlib to function correctly.
2. Pre-process the data using HOG, PCA, Normalisation and standardisation.
3. Split the 30,000 rows of data into a training set and a validation set.
4. The KNN, logistic regression, Naive Bayes, SVM and random forest were shown. For each classifier, we offer two results, with the parameters as defaults and with the parameters we chose. SVM algorithm is our most accurate algorithm. We have output the test data labels after the SVM algorithm and uploaded it to Kaggle for comparison.
5. Implementing and training machine learning classification algorithms for KNN, logistic regression, SVM and random forest respectively (building models and finding the best parameters using grid search)
6. A barplot was plotted, with the x-axis representing the different classifiers and the y-axis representing the accuracy rates.

We are using google colab to run our file and get the results, and our runtime type is CPUs. The following picture is the CPU and memory specs.



```
[4]  from psutil import *

[5]  cpu_count()

     2

[6]  cpu_stats()

     scpustats(ctx_switches=684486, interrupts=497247, soft_interrupts=516001, syscalls=0)

     !cat /proc/cpuinfo

     processor       : 0
     vendor_id       : GenuineIntel
     cpu family      : 6
     model           : 79
     model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
     stepping        : 0
     microcode       : 0x1
     cpu MHz         : 2200.202
     cache size      : 56320 KB
     physical id     : 0
     siblings        : 2
     core id         : 0
     cpu cores       : 1
     apicid          : 0
     initial apicid  : 0
     fpu             : yes
     fpu_exception   : yes
     cpuid level     : 13
     wp              : yes
     flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hype
     bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs taa
     bogomips        : 4400.40
     clflush size    : 64
     cache_alignment : 64
     address sizes   : 46 bits physical, 48 bits virtual
     power management:

     processor       : 1
     vendor_id       : GenuineIntel
     cpu family      : 6
     model           : 79
     model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
     stepping        : 0
     microcode       : 0x1
     cpu MHz         : 2200.202
     cache size      : 56320 KB
     physical id     : 0
     siblings        : 2
     core id         : 0
     cpu cores       : 1
     apicid          : 1
     initial apicid  : 1
     fpu             : yes
     fpu_exception   : yes
     cpuid level     : 13
     wp              : yes
     flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hype
     bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs taa
     bogomips        : 4400.40
     clflush size    : 64
     cache_alignment : 64
     address sizes   : 46 bits physical, 48 bits virtual
     power management:
```



```
[8] !df -h

    Filesystem      Size  Used Avail Use% Mounted on
    overlay         108G   40G   69G  37% /
    tmpfs            64M     0   64M   0% /dev
    shm             5.8G     0  5.8G   0% /dev/shm
    /dev/root       2.0G  1.2G  817M  59% /sbin/docker-init
    tmpfs           6.4G   32K  6.4G   1% /var/colab
    /dev/sda1        81G   44G   38G  54% /etc/hosts
    tmpfs           6.4G     0  6.4G   0% /proc/acpi
    tmpfs           6.4G     0  6.4G   0% /proc/scsi
    tmpfs           6.4G     0  6.4G   0% /sys/firmware
```

```
virtual_memory()

svmem(total=13622190080, available=12279713792, percent=9.9, used=1128124416, free=10084958208, active=1581498368, inactive=1698426880, buffers=189693952, cached=2219413504, shared=1204224, slab=
```

The python version we used is `3.7.13`

```
!python --version

Python 3.7.13
```