Project 2 Report:
Machine Learning on handwritten digits

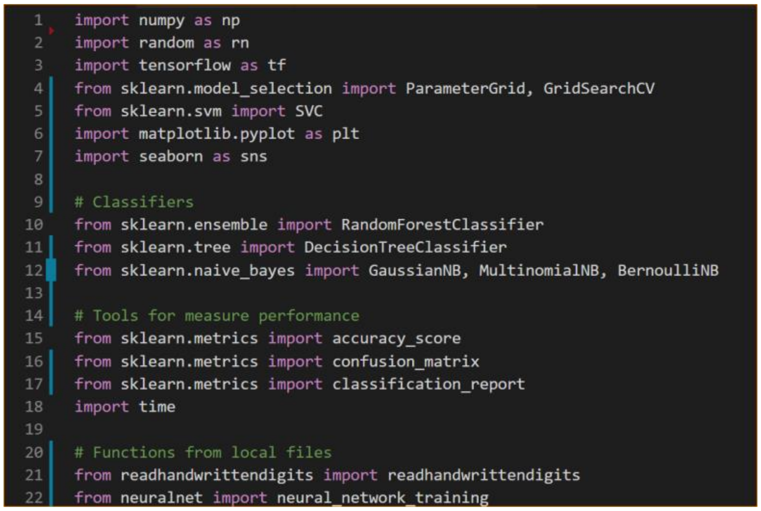
Introduction

This report contains observation and explanation on three different types of classification algorithms to recognizes different handwritten digits, and the goal is to classify handwritten digits correctly based on the training that is done for each classifier.

The code used for this report is given a set of data containing handwritten digits. The data is split into a training set and a test set. The training set is used to train the algorithms. This is necessary process as they will be able to predict a handwritten digit to an actual digit. At the end of the training, the algorithms are going to predict handwritten digits in the test data. This confirms the learning process of the algorithms to see if the algorithms are doing what it is supposed to do.

Preprocessing steps

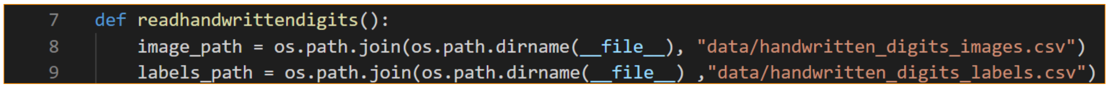
The steps needed to start the project was to install the necessary program, in this case, Python. Necessary libraries and imports must also be installed to use the machine learning algorithms and other features.



```
import numpy as np
import random as rn
import tensorflow as tf
from sklearn.model_selection import ParameterGrid, GridSearchCV
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns

# Classifiers
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB

# Tools for measure performance
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import time

# Functions from local files
from readhandwrittendigits import readhandwrittendigits
from neuralnet import neural_network_training
```

The data that is used on the algorithms are downloaded from MittUiB, which is from the MNIST database. The data contains 70,000 examples which is used to train the algorithm and confirm that they are working.

The handwritten files are saved in a folder ‘data’ and are accessed by python file ‘readhandwrittendigits.py’.
The python file uses relative path to access the data, which is to avoid use of absolute path and having the files in the same folder as the python file.



```python
def readhandwrittendigits():
    image_path = os.path.join(os.path.dirname(__file__), "data/handwritten_digits_images.csv")
    labels_path = os.path.join(os.path.dirname(__file__) ,"data/handwritten_digits_labels.csv")
```

A Python version 3.6 was installed instead of the newest version 3.7, because the version 3.7 were not compatible with the library tensorflow which is used to run neural network algorithm from keras.

Candidate algorithms and choice of candidate

The process behind choosing an algorithm was to pick three types of classification according to their performance.
In this project, the chosen classifications are Naive Bayes, Random Forest and Neural Network, and each of these classifications are very different from each other.

Naive Bayes

Naive Bayes is more lightweight which means that is quicker and faster than the other two. It performs well when we have multiple classes and are working with text classification. It converges quicker if the conditional independence holds, resulting in less training data needed. It is quicker because it requires less model training time relative to the two other algorithms.

Naive Bayes also doesn't need hyperparameters because it doesn't need lot of data to perform well. It only needs enough data to understand the probabilistic relationship of each attribute with the output variable.

There are different types of Naive Bayes models that uses different formulas to calculate the likelihood, and since Naive Bayes are fast the program uses three types of Naive Bayes algorithms: Gaussian, Multinomial and Bernoulli.

Gaussian Naïve Bayes

Gaussian is the easiest algorithm to work with because you only need to estimate the mean and the standard deviation from the given training data.

The formula used in Gaussian Naive Bayes to find the likelihood of the features:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Multinomial Naive Bayes

Multinomial Naive Bayes are used in text classification and estimate each parameter with a smoothed version of maximum likelihood:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Bernoulli Naive Bayes

There may be multiple features but each one is assumed to be binary-valued variable, according to Bernoulli Boolean. Therefore, the algorithm requires samples to be represented as binary-valued features vectors, which is handled by the BernoulliNB classifier.

The decision rule for this classifier is based on:

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Random Forest

Random Forest is a flexible algorithm. It doesn't require much hyper-parameter tuning to output a good result. By the name, the Random Forest is essentially a collection of Decision Trees and was therefore chosen instead of Decision Tree (see more about Decision Tree under). The Random Forest is a form for Bagging method, because the algorithm takes *n* Decision Trees and trains each of them separately. At the end, the algorithm then takes the average of the predictions to get a result. With Random Forest there is less chance for overfitting, because bagging reduces variance which helps to avoid overfitting.

Neural Network

Neural network is a machine learning algorithm used to model complex patterns in datasets using layers such as input, hidden and output layers and activation functions. Each layer has many neurons where the neurons are use for feature-detecting. If we have enough number of layers and neurons, the network can perform very well as image recognition. The training in neuron network uses labeled dataset of inputs that are tagged with their intended output response. In each layer, the data is calculated with a weight and an activation function.

Neural networks are good model with images and is reliable for tasks involving many features because of the splitting it does to layered network.

Neural network will perform better when given a large dataset, making it more accurate when predicting data, and once a network is trained, it can predict data very fast.

Other classifications

Support Vector Machine

There are other classifications that were considered, like Support Vector Machine and Decision Tree Classifier. The SVM is are great machine learning algorithm for image recognition. SVM is fast and accurate with a small dataset and few classifications but performs poorly when given a large dataset. SVM are also poor for multiclass classification which we have in our case with ten classifiers.
My observation with SVM is that it took too long time to finish with the given dataset. It never finished training the model and therefore was not chosen as my candidate.

Decision Tree Classifier

Decision Trees are easy to interpret and visualize and can handle both numerical and categorical data. Decisions Trees can also perform well on large datasets while being fast. With that said, Decision Trees tend to overfit, especially when the data is large because the tree may become very deep. That is why Random Forest is chosen over Decision Tree, because Random Forest reduces the chance for overfitting.

Choice of hyperparameters

The choice of hyperparameters was a bit of random. In Random Forest Classification, ‘*n_estimators’* and ‘*max_features*’ are the hyperparameters in this algorithm. The values for these parameters are ascending, starting with a low value to a high value.

The same thing is done for Neural Network, where the hyperparameters for this is ‘*epoch*’ and ‘*batch_size’*.

The advantage with Naive Bayes is it does not require hyperparameters to perform well.

Conclusion

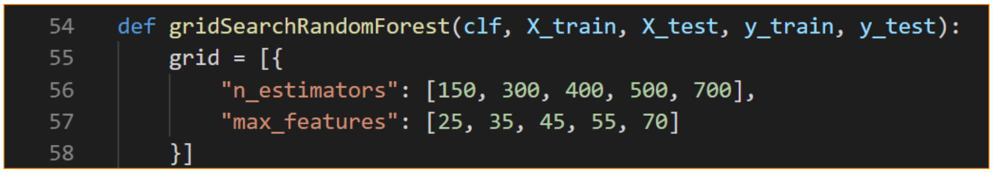
The three candidates for this project are Naive Bayes, Random Forrest and Neural Network. Naive Bayes algorithms are picked as a candidate, because it is fast and doesn’t need to build a new model each time it receives new data. Since the Naive Bayes is fast, the we try out three different types of Naive Bayes Classes; Gaussian, Multinomial and Bernoulli.
While Decision Tree being a powerful classification, Random Forrest was picked as a candidate because it uses bagging method to reduce the chance for overfitting.
Neural network can become very accurate when given lot of training data and are also easy to experiment with (layers, neurons and activation function).

All three classifiers works very well with large datasets, which useful for this task.

Model selection schemes

The models from each classifier was created with sklearn’s libraries, and each classifier has method .fit() which is used to train the models on the training set.
The models are then given a testset it is going to predict on. The result we are interested in from the models is how accurate it is when predicting.
It is difficult to determine a best classifier based on single run and nothing more.
That is why the program uses GridSearchCrossValidation to run each model

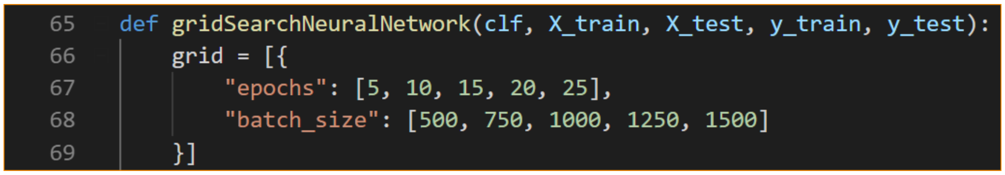
several times. It is running several times because the program test different values of hyperparameters.



```
def gridSearchRandomForest(clf, X_train, X_test, y_train, y_test):
    grid = [{
        "n_estimators": [150, 300, 400, 500, 700],
        "max_features": [25, 35, 45, 55, 70]
    }]
```

For RandomForestClassifier, the hyperparameters *n_estimators* and *max_features* are given different values, in this case [150, 300, 400, 500, 700] and [25, 35, 45, 55, 70] respectively.

The observation concludes that the best parameter for *n_estimator* is 45 and for *max_features* is 400. These parameters for Random Forest Classifier give the highest accuracy.



```
def gridSearchNeuralNetwork(clf, X_train, X_test, y_train, y_test):
    grid = [{
        "epochs": [5, 10, 15, 20, 25],
        "batch_size": [500, 750, 1000, 1250, 1500]
    }]
```

For Neural Network Classifier, *epochs* and *batch_size* is given [5, 10, 15, 20, 25] and [500, 750, 1000, 1250, 1500] respectively.
When running these parameter values, the GridSearchCV gives 20 for *epochs* and 1500 for *batch_size*.

How many layers, inputs and what activation function to use was more of trial-and-error, because it is difficult to find the optimal features for Neural Network. But trying different number of hidden layers and activation function gave more or less the same accuracy.

Performance measure

The performance of these chosen algorithms was based on time, accuracy, recall and specificity.
For each model, we train the classification model with .fit() on dataset X containing the handwritten images and dataset y containing the classifier.
Afterward, we use .predict() on the testset X to get the predicted digits. At the

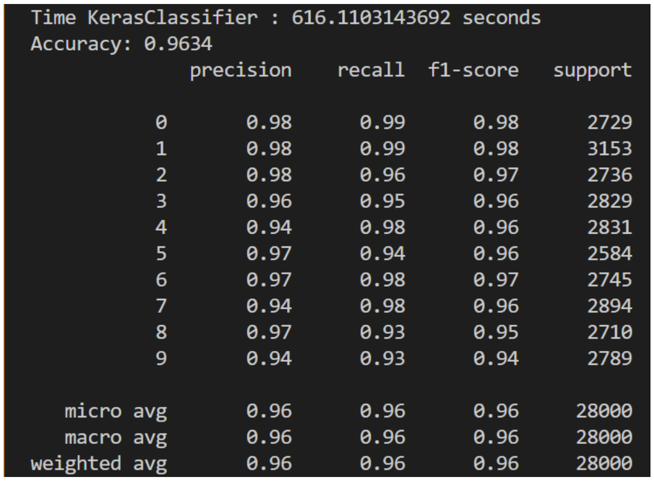
end, we calculate the accuracy score by seeing how many predicted digits from .predict() were predicted correctly with regard to the testset y.

Each algorithm produces a classification report which contains different metrics, e.g., precision and recall.
Precision is the accuracy of positive predictions while recall is the fraction of positives that were correctly identified, while recall is a measure of how truly relevant results are returned.

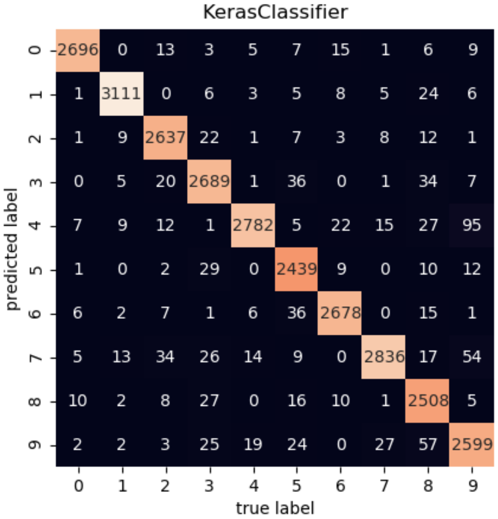
In other words, we want high accuracy and high recall, because high precision relates to a low false positive rate, and high recall relates to a low false negative rate.

Performance from each classifier

Neural Network



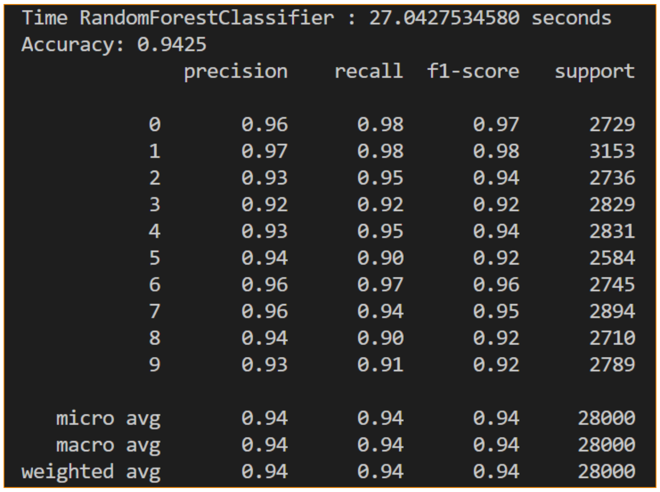
```
Time KerasClassifier : 616.1103143692 seconds
Accuracy: 0.9634
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2729
           1       0.98      0.99      0.98      3153
           2       0.98      0.96      0.97      2736
           3       0.96      0.95      0.96      2829
           4       0.94      0.98      0.96      2831
           5       0.97      0.94      0.96      2584
           6       0.97      0.98      0.97      2745
           7       0.94      0.98      0.96      2894
           8       0.97      0.93      0.95      2710
           9       0.94      0.93      0.94      2789

   micro avg       0.96      0.96      0.96     28000
   macro avg       0.96      0.96      0.96     28000
weighted avg       0.96      0.96      0.96     28000
```



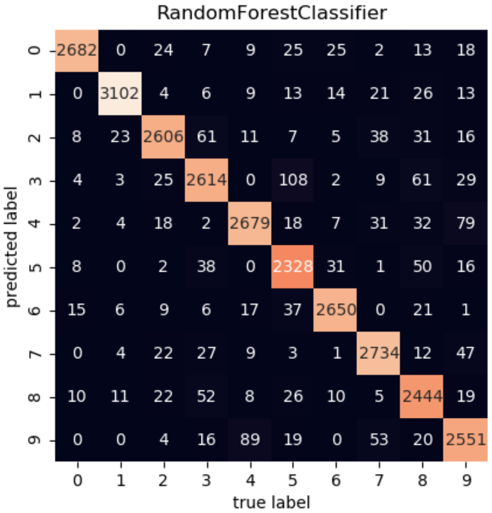
	0	1	2	3	4	5	6	7	8	9
0	2696	0	13	3	5	7	15	1	6	9
1	1	3111	0	6	3	5	8	5	24	6
2	1	9	2637	22	1	7	3	8	12	1
3	0	5	20	2689	1	36	0	1	34	7
4	7	9	12	1	2782	5	22	15	27	95
5	1	0	2	29	0	2439	9	0	10	12
6	6	2	7	1	6	36	2678	0	15	1
7	5	13	34	26	14	9	0	2836	17	54
8	10	2	8	27	0	16	10	1	2508	5
9	2	2	3	25	19	24	0	27	57	2599

Running the Neural Network classifier takes approximately 10 minutes. It takes a long time to train, but the tradeoff is that the algorithm gets a high accuracy, which is around 96%. We can see on the confusion matrix that this accuracy is true, because it almost got all prediction right. Some noticeable observation is that it predicted ‘4’ as ‘9’ 95 times. ‘9’ and ‘4’ is very similar in handwriting, and that it got only 95 wrong is very good. Giving the Neural Network more training, we can get near perfect with the prediction.

Random Forest



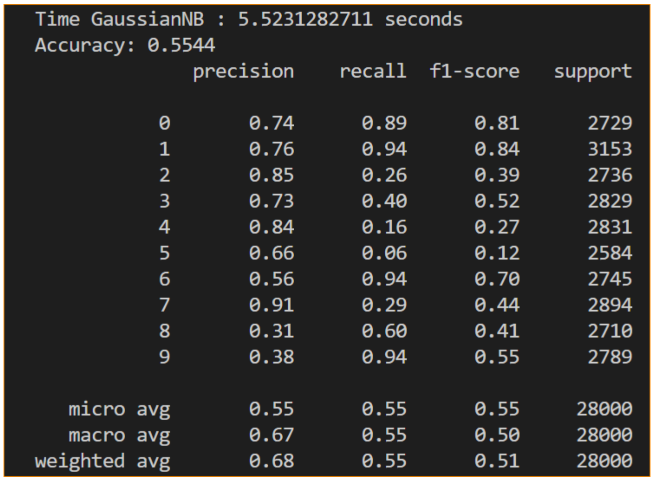
```
Time RandomForestClassifier : 27.0427534580 seconds
Accuracy: 0.9425
                 precision    recall  f1-score   support

           0       0.96      0.98      0.97      2729
           1       0.97      0.98      0.98      3153
           2       0.93      0.95      0.94      2736
           3       0.92      0.92      0.92      2829
           4       0.93      0.95      0.94      2831
           5       0.94      0.90      0.92      2584
           6       0.96      0.97      0.96      2745
           7       0.96      0.94      0.95      2894
           8       0.94      0.90      0.92      2710
           9       0.93      0.91      0.92      2789

   micro avg       0.94      0.94      0.94     28000
   macro avg       0.94      0.94      0.94     28000
weighted avg       0.94      0.94      0.94     28000
```



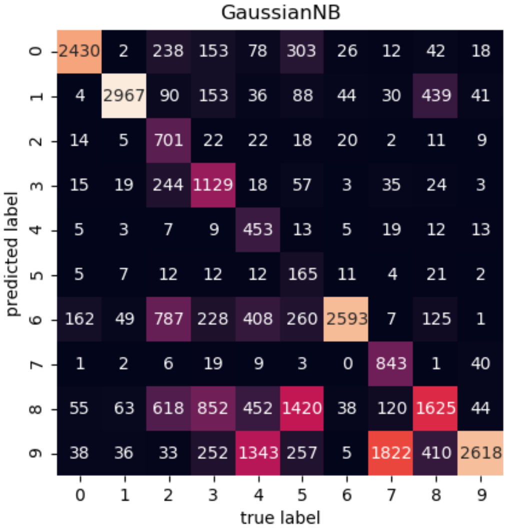
	0	1	2	3	4	5	6	7	8	9
0	2682	0	24	7	9	25	25	2	13	18
1	0	3102	4	6	9	13	14	21	26	13
2	8	23	2606	61	11	7	5	38	31	16
3	4	3	25	2614	0	108	2	9	61	29
4	2	4	18	2	2679	18	7	31	32	79
5	8	0	2	38	0	2328	31	1	50	16
6	15	6	9	6	17	37	2650	0	21	1
7	0	4	22	27	9	3	1	2734	12	47
8	10	11	22	52	8	26	10	5	2444	19
9	0	0	4	16	89	19	0	53	20	2551

The Random Forest classifier is also very good, almost same accuracy as Neural Network. The same case applies for Random Forest where it also predicts ‘4’ and ‘9’ wrong, but it also predicts ‘8’ as ‘3’ and vice versa. Despite having lower accuracy than Neural Network, it is much faster than the former classifier. It only takes 27 seconds to train it got 96% in accuracy.

Naive Bayes

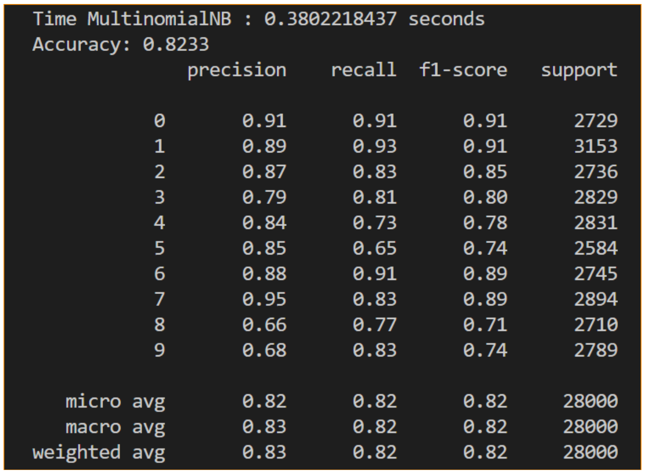


```
Time GaussianNB : 5.5231282711 seconds
Accuracy: 0.5544
                 precision    recall  f1-score   support

           0       0.74      0.89      0.81      2729
           1       0.76      0.94      0.84      3153
           2       0.85      0.26      0.39      2736
           3       0.73      0.40      0.52      2829
           4       0.84      0.16      0.27      2831
           5       0.66      0.06      0.12      2584
           6       0.56      0.94      0.70      2745
           7       0.91      0.29      0.44      2894
           8       0.31      0.60      0.41      2710
           9       0.38      0.94      0.55      2789

   micro avg       0.55      0.55      0.55     28000
   macro avg       0.67      0.55      0.50     28000
weighted avg       0.68      0.55      0.51     28000
```

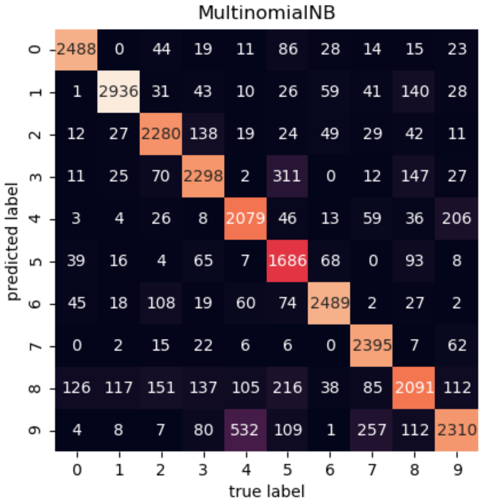


	0	1	2	3	4	5	6	7	8	9
0	2430	2	238	153	78	303	26	12	42	18
1	4	2967	90	153	36	88	44	30	439	41
2	14	5	701	22	22	18	20	2	11	9
3	15	19	244	1129	18	57	3	35	24	3
4	5	3	7	9	453	13	5	19	12	13
5	5	7	12	12	12	165	11	4	21	2
6	162	49	787	228	408	260	2593	7	125	1
7	1	2	6	19	9	3	0	843	1	40
8	55	63	618	852	452	1420	38	120	1625	44
9	38	36	33	252	1343	257	5	1822	410	2618

The Gaussian Naive Bayes is not good at all as a classifier for this dataset based on the observation. The precisions for ‘8’ and ‘9’ are very low, which means the classifier predicted these digits almost wrong all the time.

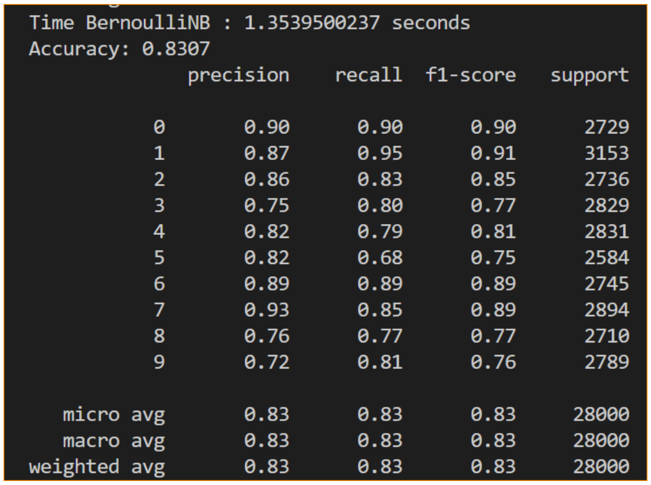


```
Time MultinomialNB : 0.3802218437 seconds
Accuracy: 0.8233
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      2729
           1       0.89      0.93      0.91      3153
           2       0.87      0.83      0.85      2736
           3       0.79      0.81      0.80      2829
           4       0.84      0.73      0.78      2831
           5       0.85      0.65      0.74      2584
           6       0.88      0.91      0.89      2745
           7       0.95      0.83      0.89      2894
           8       0.66      0.77      0.71      2710
           9       0.68      0.83      0.74      2789

   micro avg       0.82      0.82      0.82     28000
   macro avg       0.83      0.82      0.82     28000
weighted avg       0.83      0.82      0.82     28000
```

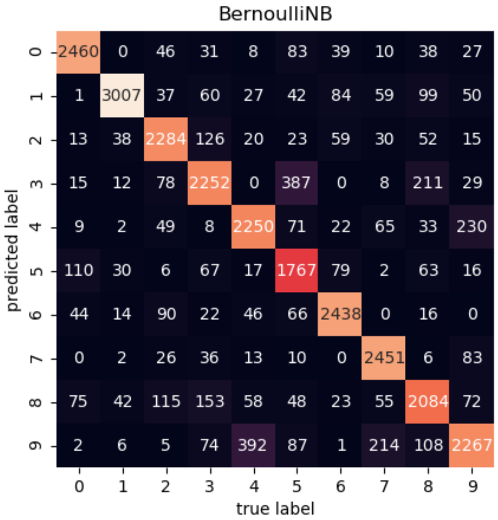


	0	1	2	3	4	5	6	7	8	9
0	2488	0	44	19	11	86	28	14	15	23
1	1	2936	31	43	10	26	59	41	140	28
2	12	27	2280	138	19	24	49	29	42	11
3	11	25	70	2298	2	311	0	12	147	27
4	3	4	26	8	2079	46	13	59	36	206
5	39	16	4	65	7	1686	68	0	93	8
6	45	18	108	19	60	74	2489	2	27	2
7	0	2	15	22	6	6	0	2395	7	62
8	126	117	151	137	105	216	38	85	2091	112
9	4	8	7	80	532	109	1	257	112	2310

The Multinomial Naive Bayes classifier is much better than the previous one, also it much faster with only 0.3 seconds. Getting an accuracy on 82% is very good for having only 0.3 seconds of training.



```
Time BernoulliNB : 1.3539500237 seconds
Accuracy: 0.8307
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      2729
           1       0.87      0.95      0.91      3153
           2       0.86      0.83      0.85      2736
           3       0.75      0.80      0.77      2829
           4       0.82      0.79      0.81      2831
           5       0.82      0.68      0.75      2584
           6       0.89      0.89      0.89      2745
           7       0.93      0.85      0.89      2894
           8       0.76      0.77      0.77      2710
           9       0.72      0.81      0.76      2789

   micro avg       0.83      0.83      0.83     28000
   macro avg       0.83      0.83      0.83     28000
weighted avg       0.83      0.83      0.83     28000
```



	0	1	2	3	4	5	6	7	8	9
0	2460	0	46	31	8	83	39	10	38	27
1	1	3007	37	60	27	42	84	59	99	50
2	13	38	2284	126	20	23	59	30	52	15
3	15	12	78	2252	0	387	0	8	211	29
4	9	2	49	8	2250	71	22	65	33	230
5	110	30	6	67	17	1767	79	2	63	16
6	44	14	90	22	46	66	2438	0	16	0
7	0	2	26	36	13	10	0	2451	6	83
8	75	42	115	153	58	48	23	55	2084	72
9	2	6	5	74	392	87	1	214	108	2267

Bernouilli Naive Bayes is almost similar as Multinomial in accuracy and time. It takes a little bit more time, but the accuracy is the same.

Conclusion

From the confusion matrix, we can see some digits get predicted wrong in all classifiers, specifics ‘3’ for ‘8’, and ‘4’ for ‘9’ and vice versa for both. This is because the digits are very similar in written form, and having the classifiers predicting wrong is understandable.

The prediction made by these classifiers is overall very good. Naive Bayes classifiers have lower accuracy than Neural Network and Random Forest, but their running time is incredible fast, with some of them having under six seconds and even one. The two Naive Bayes classifier are very good, and with hyperparameter tuning, they might get even better score overall, but this observation was made without having hyperparameters on the Naive Bayes classifiers, because they perform very well without them.

Random Forest and Neural Network are very similar in accuracy, and the confusion matrixes are almost the same. The only thing that is different is the training time. Random Forest is much faster than Neural Network, and if time and resource are important factors for deciding a classifier, Random Forest would have been chosen. Since time and resource was available for this project, the Neural Network was chosen over Random Forest. In practical, the Neural Network would be more beneficial because it doesn't need to build the learning process all over again with new data, versus the Random Forest which need to create new Decision Trees again when receiving new data.

Final classifier

The deciding classifier for this project was Neural Network. The reason for this classifier is that it got the highest accuracy of the candidates despite having the longest training time in training.

All three Naive Bayes Classifiers did not perform very well. This may be because the classifiers make a strong assumption on the shape of the data distribution. The result can be potentially very bad, therefore "naive" classifier.

While Random Forest Classifier got relatively high accuracy, even sometimes more than Neural Network given some specific hyperparameters, it is easier to use Neural Network. When more training data is observed, the Random Forest must build a new model from scratch versus Neural Network doesn't need to do this. Since we aim to use our machine learning algorithm on a real-life task, Neural Network has an advantage here over Random Forest.

Expected performance in production

The expectation for Neural Network is that it is going to perform well in production. The reason for this is the accuracy it got from the training and testset was relatively high, or at least enough to be used in real life tasks.

An advantage with Neural Network is there more training data it gets, the more accurate it becomes. If there comes any new data, the Neural Network doesn't need to be built again from the scratch, but rather built on the existing training. Even though the classifier does not get 100% accuracy, which is ideal, an accuracy on 97% is good enough. This means that it has 3% of predicting wrong, but we must also take human error in consideration. If we allow human error, then we must also allow the machine learning algorithms to do at least 3% of the prediction wrong.

Improvement

With more time and/or computing resources, the neural network classifier will be more accurate if the number of epochs and the size of batch are increased. The computing time will get affected heavily, but the accuracy will at least increase which is a trade we can afford. If 97% accuracy is good enough, and with more time and resources, the machine learning algorithm may become more efficient than a human doing the same job.