ML Assignment 2 – Report File

Abstract

Objective

This study focuses on performing an image classification task using 3 machine learning algorithms namely K- nearest neighbour, Naive bayes and Random Forest. The main objective is to evaluate the performance of these models on the EMNIST handwritten by class training dataset and report a comparative evaluation of the three models.

Methodology

The EMNIST dataset was cleaned and pre-processed by using techniques like normalisation, reshaping and label encoding to make compatible with the chosen model. The pre-processed data was fit through the original models and then the models were fine-tuned by varying the hyperparameters to obtain the best performing model.

Results

Based on the methodology, Random Forest model classified the test data with an accuracy of 0.79875 when the hyperparameters were tuned to 'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200. This was followed by KNN with an accuracy of 0.77335 when the hyperparameters were tuned to 'n_neighbors': 7, 'p': 2, 'weights': 'distance'. The worst performing model was Naive Bayes with an accuracy of only 0.51720 with and without hyperparameter tuning.

Conclusion

In conclusion, Random Forest model was able to classify the test data with the highest accuracy followed by KNN followed by Naive Bayes model. Naive Bayes models seems unsuitable for this dataset and its prediction capability cannot be improved even with hyperparameter tuning.

Introduction

Overview

The aim of this study is to display our understanding of a complete machine learning pipeline with respect to an image classification task using the EMNIST Handwritten Character dataset. We apply our semester long understanding to demonstrate the performance of 3 supervised learning ML models - KNN, Random Forest and Naive Bayes on an image classification problem. A secondary objective of the study is to interpret the results of the models and conduct a comparative evaluation of the models' performance to determine the suitability of these models to our task.

Background

The uptake in the utilization of digital trends and tools has led to an increase in the quantity of unstructured data that is present in the form of images. Recent advancements in the field of Machine Learning have allowed us to capitalise on this unstructured image-based data by deep diving into the world of Image Classification (Gaudenz Boesch, n.d.). Image classification is a sub-sector of the field of computer vision and finds it applications in various fields such as healthcare, automobile industry, E-commerce, Law enforcement, etc. The most used ML models in this domain are Support Vector Machines (SVM), K Nearest Neighbours (KNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Ensemble Methods.

Problem Statement

In this report, we aim to apply our machine learning and data mining skills learnt over the course of the semester to classify handwritten digits using the EMNIST Handwritten character dataset. We aim to demonstrate our understanding of the complete Machine pipeline by performing image classification on the EMNIST Handwritten character dataset using Naïve Bayes, Random Forest and KNN algorithms. Furthermore, we will also demonstrate our understanding of improvisation of models through fine tuning their performance using their respective hyperparameters. Lastly, we shall evaluate each model, before fine-tuning and after fine tuning, based on various evaluation metrics such as accuracy, precision, recall, and F1 score. The main input for these models would be the pre-processed dataset of the EMNIST handwritten character dataset by class. The original dataset has 814,255 characters and 62 unbalanced classes (NIST, 2019). For this assignment, a smaller dataset has been provided and used which has almost 100,000 characters and 62 unbalanced classes. As the output we expect the 3 ML algorithms to classify these characters with acceptable accuracy. Furthermore, we also expect to improve the performance of these models by fine tuning the hyperparameters.

Methodology

Fig 1. ML Pipeline Overview for Assignment 2.

Data Pre-Processing

The data has been presented to us in a pickle file and has a slightly different approach of loading it. While usually we use pandas library to open a csv, excel, or any such file, for a pickle file we use the pickle library alongside the pandas library. And considering that the data is an image file, we use the matplotlib.pyplot library to view the data. This can be seen in the "Unpacking and Exploring the Dataset" section of the Python file submitted alongside this report.

The success of any ML algorithm is dependent on many factors including quality representation of instance data. Presence of noise or unreliable data makes interpretation of ML model results difficult and inaccurate. There are various techniques used for Data Pre-processing such as data cleaning, data transformation, and data reduction (Miller, 2019). We have used these techniques for our data pre-processing as well. The techniques we used were:

- ***Label Encoding***

  Label Encoding is a ML data pre-processing technique used to assign numerical values to categorical data. The main objective of label encoding is to allow numeric based ML algorithms to work on those data frames that contain non-numeric classes. Naive bayes is one such model that requires numeric labels (Geron, 2017).

```
# Useful for the Naive Bayes Classifier to convert categorical variables into numericc
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
Encoded_y_train = label_encoder.fit_transform(y_train)
Encoded_y_test = label_encoder.transform(y_test)
```

Fig 2. Code snippet for Label encoding.

- ***Normalisation***

  Normalisation is a ML data pre-processing technique to ensure the data is standardised and that every variable has an equal weight. It also referred to as feature scaling. The main objective of Normalisation is to reduce redundant data and errors related to data modification specially when dealing with large amounts of data. Normalising the data has benefits of data consistency, decrease in data redundancy and cost, increase in data security (Geron, 2017).

```
# There are a total of 255 features hence dividing by 255 to normalise pixel values be
Train_Images = X_train / 255.0
Test_Images = X_test / 255.0
```

Fig 3. Code snippet for Normalisation.

- ***Reshaping***

  It is crucial to understand the shape of the data to ensure its compatibility with various models. Our data post normalisation and label encoding are still not a usable format for ML models as it is a 3-dimensional array. The current shape of the data frame stands at containing 100,000 values each with a 28* 28-pixel height and width. For us to be able to fit the training data to models such as KNN, we convert this 3-Dimensional array into a 2-D array. Figure 4 shows the code for reshaping in which the original 28*28 2-D image is flattened to a 1-D vector of length 784 thus the data is converted to a usable format.

```
# Reshaping the flattened feature arrays into 2D representation to restore original i
X_Train_Images = Train_Images.reshape(Train_Images.shape[0], -1)
X_Test_Images = Test_Images.reshape(Test_Images.shape[0], -1)
```

Fig 4. Code snippet for Reshaping.

The data post normalisation seems quite straight forward and well-aligned as shown in Figure 5. This eliminates the need for oversimplification which might lead to loss of useful information (Geron, 2017).

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
99996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
99997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
99998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
99999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[100000 rows x 784 columns]

Fig 5. Training data post normalisation/feature scaling and reshaping.

Methods

KNN

K-Nearest neighbour (KNN) is a type of supervised ML algorithm. It is commonly used for both classification and regression problems. It is intuitive by nature and does not make any assumptions about the distribution of the data. It makes predictions on the test/unseen data by computing the distance between the test data point and all the other existing data points. The distance is calculated by Euclidean or Manhattan formula and can be varied by the user to

optimise performance (IBM, n.d.). The major merits of KNN are that it does not learn anything during the training period and learns only at the time of making predictions. This makes KNN faster compared to any other training-based algorithms such as SVM or LR. Addition of new data is easy as the model itself does not undergo any changes and is simple to fine tune with only 2 major hyperparameters: distance and number of neighbours. The main demerit of the KNN algorithm is that is it suitable only for 2-dimensional data and introduction of more dimensions affects the computation power of the model. As the size of the model increases, the computational time also increases as the model must calculate larger distances. This is seen by the amount of time taken for our models to run (i2 tutorials, n.d.).

Nevertheless, we chose KNN as it can be used as both a classification and regression algorithm and owing to these merits of easy interpretability and ease of fine tuning the hyperparameters. KNN was one of the initial algorithms studied in the semester and its suitability for such task aided in the decision making of incorporating this algorithm in the analysis.

Random Forest

Random Forest is based off the Decision Tree model where multiple decision trees are trained on different subsets of the training dataset. Each tree is allowed to grow and make predictions and the results of these individual tree predictions are then used for either classification (by considering majority vote) or regression(by considering the average result of all decision trees). The main objective of using multiple trees and creating an ensemble is to overcome limitations such as high variance, overfitting etc. which are often faced by singular decision trees. By introducing randomness in the tree building process, the random forest model is more robust and generalisable (Breiman, 2001). It has various merits such as being robust and avoids overfitting. They are well equipped to handle noise and outliers. It has high accuracy in prediction and can handle large datasets with relatively less computational time. It also has some demerits as well such as the interpretation of the results of Random forests are difficult to understand as compared to decision trees. Or that the training time of the model increases with the number of trees it has. A major demerit of the model is that as it stores and learns multiple trees, it requires higher memory space than other models which is a major issue in case of large datasets such as the EMNIST dataset (Rebellion Research, 2023).

Despite the demerits, we choose Random Forest as it is a powerful classification model. Using a decision tree could lead to over fitting of the dataset whereas using a Random Forest model could help us eliminate this problem. Furthermore, as the dataset is multi-dimensional and may contain outliers, we considered this as one of the more suitable algorithms to apply for this dataset. We believe, based on the inherent ability of the Random Forest model, it would be able to capture the non-linear relationships between the various features that could help in more robust classification.

Naïve Bayes

Naive bayes is also a supervised ML model which finds its primary application in classification problems such as spam email filtering and text classification. Naive Bayes is thus named as it assumes that the features of the data have no conditional dependence on each other (which would be highly unlikely in real world). Using the training data, it memorises the probability distribution of each of the feature and uses this to predict the probability of a new instance based on Bayes' Theorem. Despite this base assumption, Naive Bayes performs well for classification and depending on the type of data, either Gaussian, Multinomial or Bernoulli Naive Bayes model is used. It has merits such as its great suitability for multi-class predictions

and binary class predictions, if the assumption of feature independence holds true, then the model can also be used in use cases where less training data is available. Another advantage of using a Naïve Bayes algorithm is its simplicity to implement and fast run time. As there are advantages, there are disadvantages as well. One disadvantage is that smoothing is often required in cases where test data contains a new categorical class to avoid the zero-frequency phenomenon. Another demerit is that in real life, the features of a dataset often show some dependence and thus negates the base assumption of the algorithm which leads to inaccuracy in the results (MLNerds, 2021).

Considering its pros and cons, Naive Bayes is one of the baseline models used for classification tasks. While Naive Bayes is commonly used for text classification, we wanted to understand its performance in an image classification problem setting. The most suitable one for our problem statement is multinomial naive bayes as our data neither has a gaussian distribution (Gaussian Naive Bayes) nor is binary (Bernoulli Naive Bayes). NB was also chosen as one of the models for its ease of implementation and less computational time and power.

Implementation

As per instructions, the data has already been divided into a testing dataset and a training dataset. Hence the data was directly called into the testing and training variables.

```
#Splitting the data as provided.
X_train = Train_Data["data"]
y_train = Train_Data["labels"]
X_test = Test_Data["data"]
y_test = Test_Data["labels"]
```

Fig 6. Code snippet of data split.

Furthermore, data pre-processing was further carried out on both, the train and test dataset simultaneously so that the implementation and testing of the datasets could be easier. To eliminate python notebook related warnings, we import warnings module as follows:

```
#The following code is to filter all types of warnings to make the code look cleaner.
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from sklearn.exceptions import UndefinedMetricWarning
warnings.filterwarnings("ignore", category=UndefinedMetricWarning)
```

Fig 7. Code snippet of ignore warnings.

Considering we used 3 different Machine Learning Algorithms, the code snippets with a brief explanation for fitting the training data to each of the original models:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_re

naive_bayes = MultinomialNB()
naive_bayes.fit(Scaled_Train_Images, Encoded_y_train)
y_pred_nb = naive_bayes.predict(Scaled_Test_Images)
```

Fig 8. Fitting the training data to Naïve Bayes Algorithm.

For the Naïve Bayes Algorithm, we imported the Multinomial Naïve Bayes package from the sklearn.naive_bayes package. We called the function of Multinomial Naïve Bayes function in the variable as shown above and then fit the training data into the function.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix,

random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(Scaled_Train_Images, Encoded_y_train)

y_pred_rf = random_forest.predict(Scaled_Test_Images)
```

Fig 9. Fitting the training data to Random Forest Classifier Algorithm.

For the Random Forest Classifier, we imported the Random Forest Classifier package from the sklearn.ensemble package. We called the Random Forest Classifier function with the parameters as seen above in the variable "random_forest".

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix,

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(Scaled_Train_Images, Encoded_y_train)
y_pred_knn = knn.predict(Scaled_Test_Images)
```

Fig 10. Fitting the training data to KNN Classifier Algorithm.

For the K-Nearest Neighbours, we imported the KNeighborsClassifier package from the sklearn.neighbors' package. We called the KNeighborsClassifier function with the neighbours set to 5.

To evaluate the performance of our models, we use the evaluation metrics such as the accuracy, precision score, F1 score, etc. that were imported from the sklearn.metrics library. The "matplotlib.pyplot" and "seaborn" libraries were used for visualizations such as the line charts and the heatmaps.

Experiments and Discussion

Experimental Setting

This section of the report is used to describe the setting of the experiments including the details of the dataset, model architecture, hardware, and software specifications of the computer used for the performance evaluations.

Dataset Description

The EMNIST dataset used, comprises of handwritten character digits. The Extended Modified NIST (EMNIST) dataset here used is a subset of the NIST special database 19. Owing to the extensive nature of NIST and its large computational time and power, EMNIST is a smaller specific subset of NIST that thoroughly specifies the nature of the classification task and structure of the database to allow for better and easier comparison among evaluation results. The EMNIST dataset consists of 814255 characters inclusive of uppercase, lowercase characters and digits ranging from 0-9. These characters are divided into 62 unbalanced classes and each image is in the format of a 28 X 28-pixel image. The datatype of the features is dtype=uint8 which is integers between 0 to 255. The main aim of this dataset is to help train various Machine learning models in image classification and is thus designed to place emphasis on character recognition (NIST, 2019).

Hardware Specifications

Processor	12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
Installed RAM	16.0 GB (15.7 GB usable)
System Type	64-bit operating system, x64-based processor

Table 1. Hardware specifications on which the code was run.

Software Specifications

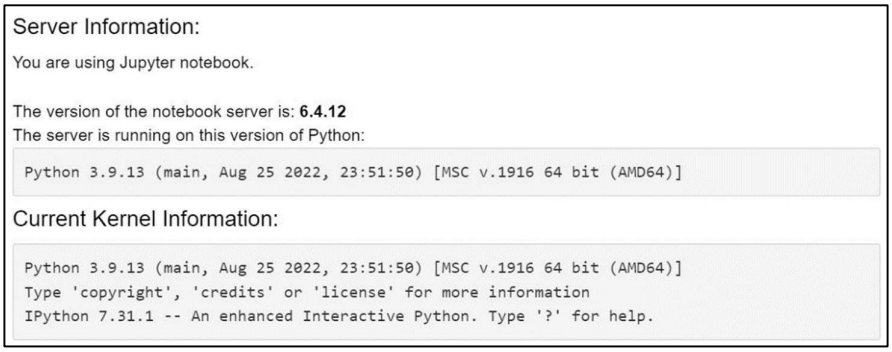

Fig 11. Software specification of the software the code was run on.

Model Architecture

Naïve Bayes Algorithm

A base model is first created in which a Multinomial Naive Bayes classifier is instantiated with MultinomialNB(), and then the fit function is called to train the classifier using the training data (Scaled_Train_Images and Encoded_y_train). After training, the predict function is used to make predictions on the test data (Scaled_Test_Images), and the predictions are stored in y_pred_nb.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c

naive_bayes = MultinomialNB()
naive_bayes.fit(Scaled_Train_Images, Encoded_y_train)
y_pred_nb = naive_bayes.predict(Scaled_Test_Images)
```

Fig 12. Base model of Naïve Bayes Algorithm.

Cross validation was then introduced to improve the performance of the model. A parameter grid was created to introduce different options for the parameters and a grid search was conducted to identify the best performing parameters. Based on these, the best performing parameters were used to generate the best estimating model that was used for prediction.

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,

param_grid = {
    'alpha': [0.1, 0.5, 1.0],
    'fit_prior': [True, False]
}

grid_search = GridSearchCV(naive_bayes, param_grid, cv=Cross_Validation_Fold)
grid_search.fit(Scaled_Train_Images, Encoded_y_train)

best_params_nb1 = grid_search.best_params_
best_model = grid_search.best_estimator_

y_best_pred_nb1 = best_model.predict(Scaled_Test_Images)
```

Fig 13. Fine-tuned model of Naïve Bayes Algorithm.

KNN Algorithm

A base model is first created in which a KNN classifier is instantiated with KNeighborsClassifier(n_neighbors=5), indicating that it will consider the 5 nearest neighbours for classification. The fit function is then used to train the KNN classifier using the training data (Scaled_Train_Images and Encoded_y_train). After training, the predict function is used to make predictions on the test data (Scaled_Test_Images), and the predictions are stored in y_pred_nb.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(Scaled_Train_Images, Encoded_y_train)
y_pred_knn = knn.predict(Scaled_Test_Images)
```

Fig 14. Base model of KNN Algorithm.

Cross validation was then introduced to improve the performance of the model. A parameter grid was created to introduce different options for the parameters and a grid search was conducted to identify the best performing parameters. Based on these, the best performing parameters were used to generate the best estimating model that was used for prediction.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,

knn = KNeighborsClassifier()

param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

grid_search = GridSearchCV(knn, param_grid, cv=Cross_Validation_Fold)
grid_search.fit(Scaled_Train_Images, Encoded_y_train)
best_params_knn1 = grid_search.best_params_
best_model = grid_search.best_estimator_

y_best_pred_knn1 = best_model.predict(Scaled_Test_Images)
```

Fig 15. Fine tuned model of KNN Algorithm.

Random Forest Algorithm

A base model is first created in which a Random Forest Classifier is instantiated with RandomForestClassifier(n_estimators=100, random_state=42). The parameter n_estimators=100 specifies that the random forest will consist of 100 decision trees. The random_state=42 parameter sets the random seed for reproducibility. The fit function is then called to train the random forest classifier using the training data (Scaled_Train_Images and Encoded_y_train). After training, the predict function is used to make predictions on the test data (Scaled_Test_Images), and the predictions are stored in y_pred_rf.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c

random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(Scaled_Train_Images, Encoded_y_train)

y_pred_rf = random_forest.predict(Scaled_Test_Images)
```

Fig 16. Base model of Random Forest Classifier Algorithm.

Cross validation was then introduced to improve the performance of the model. A parameter grid was created to introduce different options for the parameters and a grid search was conducted to identify the best performing parameters. Based on these, the best performing parameters were used to generate the best estimating model that was used for prediction.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,

random_forest = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(random_forest, param_grid, cv=Cross_Validation_Fold)
grid_search.fit(Scaled_Train_Images, Encoded_y_train)
best_params_rf1 = grid_search.best_params_
best_model = grid_search.best_estimator_

y_best_pred_rf1 = best_model.predict(Scaled_Test_Images)
```

Fig 17. Fine tuned model of Random Forest Classifier Algorithm.

Each of the models (base and fine-tuned) computes the accuracy using accuracy_score, comparing the predicted labels (y_pred_(model_name)) with the true labels (Encoded_y_test). The precision, recall, and F1 score are calculated using precision_score, recall_score, and f1_score, respectively. These metrics are computed by comparing the predicted labels with the true labels, considering their weighted average. Additionally, the code generates a classification report using classification_report, providing precision, recall, F1 score, and support for each class in the target variable. It also computes the confusion matrix using confusion_matrix, representing the count of true positive, false positive, true negative, and false negative predictions for each class.

For the fine-tuned models stratified K Fold method of cross validation was used where k was set as 10 (n_splits = 10) as this is the standard value. Random state was set to 0 to ensure that the same train and test split was generated at every run of the model, thus ensuring reproducibility. The code snippet for the same is provided below:

```
from sklearn.model_selection import StratifiedKFold
#Setting the 10 fold stratified cross validation
Cross_Validation_Fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
```

Fig 18. Code for the Cross Validation Fold Method.

Experimental Results

The total time taken for the execution of the code file was 83,822.53 seconds.

Naïve Bayes Algorithm

The fine-tuned model of Naïve Bayes takes about 5-7 minutes to run and gives the following results.

```
Accuracy: 0.5172
Precision: 0.5986342354574439
Recall: 0.5172
F1 Score: 0.5368922551159929
```

```
Best parameters: {'alpha': 1.0, 'fit_prior': True}
Accuracy: 0.5172
Precision: 0.5986342354574439
Recall: 0.5172
F1 Score: 0.5368922551159929
```

Fig 19. Evaluation results of the base model (L) and the fine-tuned model (R).

NB has the worst accuracy of 51.72% and shows no difference after hyperparameter tuning. The best parameters were found to be {'alpha':1.0, 'fit_prior':True} indicating that the model performs best when the prediction is made with the help of Laplace smoothing to avoid zero probability error and fits classes with prior probabilities from the training data.

Based on Figure 27 and Figure 28 in the appendix, Naive Bayes model has poor ability to classify positive and negative predictions, and this is irrespective of the support value - indicating that this model is not suitable for this dataset. This could be because of the inherent feature independence assumption of NB model. This concludes that as the features have some measure of dependence in the EMNIST dataset, Naive bayes is not an appropriate classifier.

Fig 20. Graphical representation of the evaluation metrics before and after fine-tuning the model.

Random Forest Classifier

The fine-tuned model of Random Forest Classifier takes about 4 hours to run and gives the following results.

```
Accuracy: 0.7947
Precision: 0.7770284307029045
Recall: 0.7947
F1 Score: 0.7711061231900168
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}
Accuracy: 0.79875
Precision: 0.7793703154114677
Recall: 0.79875
F1 Score: 0.7750150539449473
```

Fig 21. Evaluation results of the base model (L) and the fine-tuned model (R).

RF has the best accuracy at 79.87% and performs best when the prediction is made using 200 decision trees where each node in the tree has at least 2 instances and the tree is allowed to grow until the node reaches a purity or cannot split further.

Based on Figure 29 and Figure 30 in the appendix, Random Forest model has a good ability to classify positive and negative predictions. Furthermore, most classes that have a low precision, recall and F-1 value are characterised by a low support value - indicating that balancing the training classes or including more instances of these classes could lead to better results. The graphical representation of the evaluation metrics before and after fine-tuning is given below:

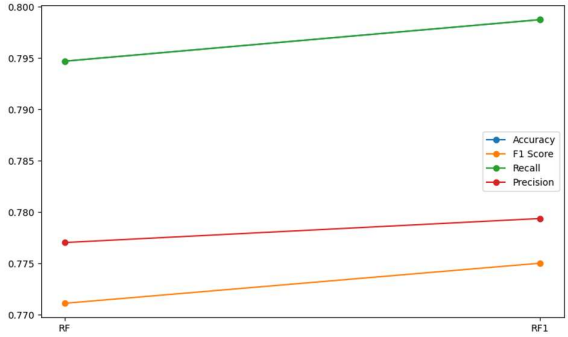

Fig 22. Graphical representation of the evaluation metrics before and after fine-tuning the model.

K Nearest Neighbours Algorithm

The fine-tuned model of KNN algorithm takes almost 19 hours to run and gives the following results.

```
Accuracy: 0.7692
Precision: 0.7593865808347185
Recall: 0.7692
F1 Score: 0.7532891920125383
```

```
Best parameters: {'n_neighbors': 7, 'p': 2, 'weights': 'distance'}
Accuracy: 0.77335
Precision: 0.7633499823272847
Recall: 0.77335
F1 Score: 0.7578332118995986
```

Fig 23. Evaluation results of the base model (L) and the fine-tuned model (R).

KNN algorithm has a good accuracy at 77.33% after hyperparameter tuning indicating that the model performs the best with 7 neighbours and Euclidean measure of distance.

Based on Figure 31 and Figure 32 in the appendix, KNN model has a good ability to classify positive predictions. Furthermore, most classes that have a low precision, recall and F-1 value are characterised by a low support value - indicating that balancing the training classes or including more instances of these classes could lead to better results. The graphical representation of the evaluation metrics before and after fine-tuning is given below:

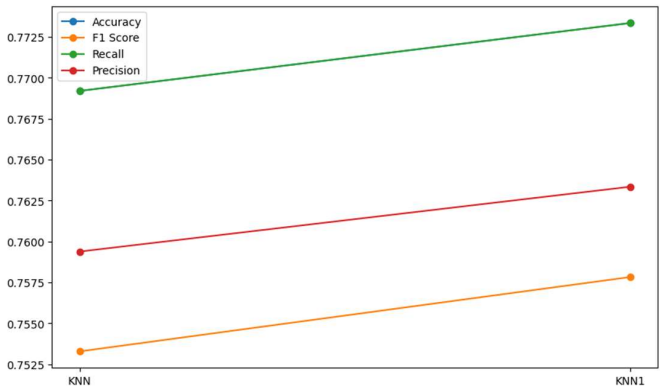

Fig 24. Graphical representation of the evaluation metrics before and after fine-tuning the model.

Comparison

	Model	Accuracy before tuning	Accuracy after tuning
0	Naive Bayes	0.5172	0.51720
1	Random Forest	0.7947	0.79875
2	KNN	0.7692	0.77335

	Precision before tuning	Precision after tuning	Recall before tuning
0	0.598634	0.598634	0.5172
1	0.777028	0.779370	0.7947
2	0.759387	0.763350	0.7692

	Recall after tuning	F1 before tuning	F1 after tuning
0	0.51720	0.536892	0.536892
1	0.79875	0.771106	0.775015
2	0.77335	0.753289	0.757833

Fig 25. Tabular representation of evaluation metric results of all models before and after hyper-parameter tuning with red indicating worst performing and yellow indicating best performing metrics.

With the same set of input variables, there is a stark difference in the prediction capabilities of the models as indicated above. Fine Tuned Random Forest Model is the best performing model with the highest accuracy followed by KNN and Naive Bayes. Naive bayes

has comparatively very poor performance. Though RF has better overall prediction, KNN has slightly better prediction.

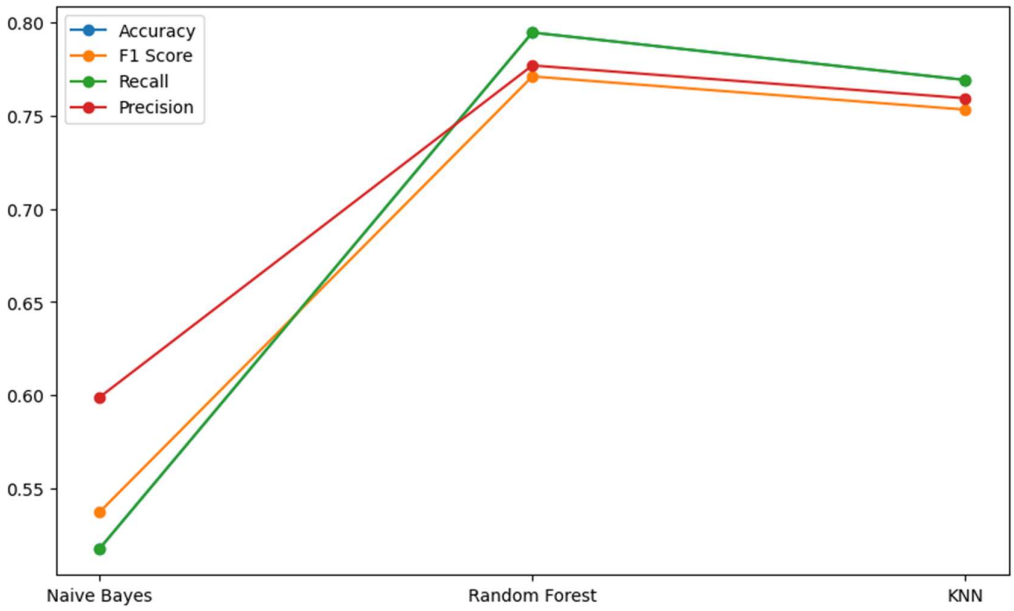

Fig 26. Comparative evaluation of performance of all NB, RF, and KNN.

Reflection

The different models were chosen for various reasons. KNN was chosen due to its ease of implementation and interpretation. Being one of the initial models studied in the semester, we were confident in our ability to execute it. Random Forest was chosen because of its compatibility with the dataset. We believed decision tree to be simple of an algorithm to capture the intricacies of the dataset and thus decided to proceed with Random Forest. Random forest model can capture the 3-D nature of the dataset as well. As Naive Bayes is commonly used for text classification tasks, we assumed to be the most suitable model for such a dataset. However, our results prove otherwise. All three models were chosen for their low computation power. These three models were first run as standard models and then were optimised by conducting a grid search and fitting the best parameter to the training set. This led to a slight improvement in the performance metrics of the models.

Conclusion

Summary

Random forest is the best performing model with an accuracy of 79.85% . The fine-tuned model with selected parameters of 'max_depth' : None, 'min_samples_split' : 2 and 'n_estimators' : 200 performs better than the base model by a margin of 0.5%. KNN model follows Random Forest and has an accuracy of 77.35%. The fine-tuned model with hyperparameters of 'n_neighbors' : 7, 'p' : 2 and 'weights' : 'distance' outperforms the original model by 1.5%.KNN model has a better ability to classify positive predictions that the other two models. However, due to the size of the dataset, the KNN model is time expensive with a run time of 23 hours. This reduces the desirability of KNN as a model for performing image classification. Naive Bayes model is the worst performing model among the three models with an accuracy of 51%. There is no effect of hyperparameter tuning on Naive Bayes model as reflected in its lack of difference in accuracy. This is indicative of the fact that though Naive

Bayes is generally considered as a suitable model for classification techniques, it is not a suitable model for this dataset.

This analysis tells us that there are several factors to be considered while performing an analysis and that only performance metrics cannot be considered. The computation power available and the run time of the models are also an important factor to consider for model implementation. Thus, while KNN is easy to interpret, needs low computational power and is suitable for large datasets, it is extremely time consuming making it a subpar model for this dataset.

Future Work

Future works for this dataset could include alternative models for prediction such as Neural networks (CNN, DNN) etc., instead of Naive Bayes, that seem to have better prediction capability for the model. We could also try other possibilities on this dataset such as implement the Random Forest model without altering the 3-D shape of the dataset to see how it compares with the current results.

References

Breiman, L. (2001). Random Forests. *Springer*, 5-32.

Gaudenz Boesch. (n.d.). *A Complete Guide to Image Classification in 2023*. Retrieved from viso.ai: https://viso.ai/computer-vision/image-classification/

Geron, A. (2017). *Hands-On Machine Learning with Scikit Learn.* O'Reilly Media, Inc.

i2 tutorials. (n.d.). *What are the Advantages and Disadvantages of KNN Classifier?* Retrieved from i2 tutorials: https://www.i2tutorials.com/advantages-and-disadvantages-of-knn-classifier/

IBM. (n.d.). *K-Nearest Neighbors Algorithm*. Retrieved from IBM: https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point.

Miller, R. (2019, December 13). *Data Preprocessing: what is it and why is important*. Retrieved from CEOWORLD Magazine: https://ceoworld.biz/2019/12/13/data-preprocessing-what-is-it-and-why-is-important/

MLNerds. (2021, July 30). *Naive Bayes Classifier : Advantages and Disadvantages*. Retrieved from Machine Learning Interviews: https://machinelearninginterview.com/topics/machine-learning/naive-bayes-classifier-advantages-and-disadvantages/#Advantages%20of%20Using%20Naive%20Bayes%20Classifier

NIST. (2019, March 28). *The EMNIST Dataset*. Retrieved from NIST: https://www.nist.gov/itl/products-and-services/emnist-dataset

Rebellion Research. (2023, February 19). *What Are The Advantages And Disadvantages Of Random Forest?* Retrieved from Rebellion Research:

https://www.rebellionresearch.com/what-are-the-advantages-and-disadvantages-of-random-forest

Appendix

How to run the code

1. Make sure to install the packages pandas, NumPy, seaborn, matplotlib.pyplot, pickle, warnings, and sklearn.
2. Use the reduced data provided in the google drive as the base dataset.
3. Make sure that the name of the training dataset is “emnist_train.pkl”.
4. Make sure that the name of the testing dataset is “emnist_test.pkl”.
5. Make sure that the data files and the code file are in the same working environment.
6. Run the code as it is.

Confusion Matrix for each model

Naïve Bayes Algorithm after hyper parameter tuning

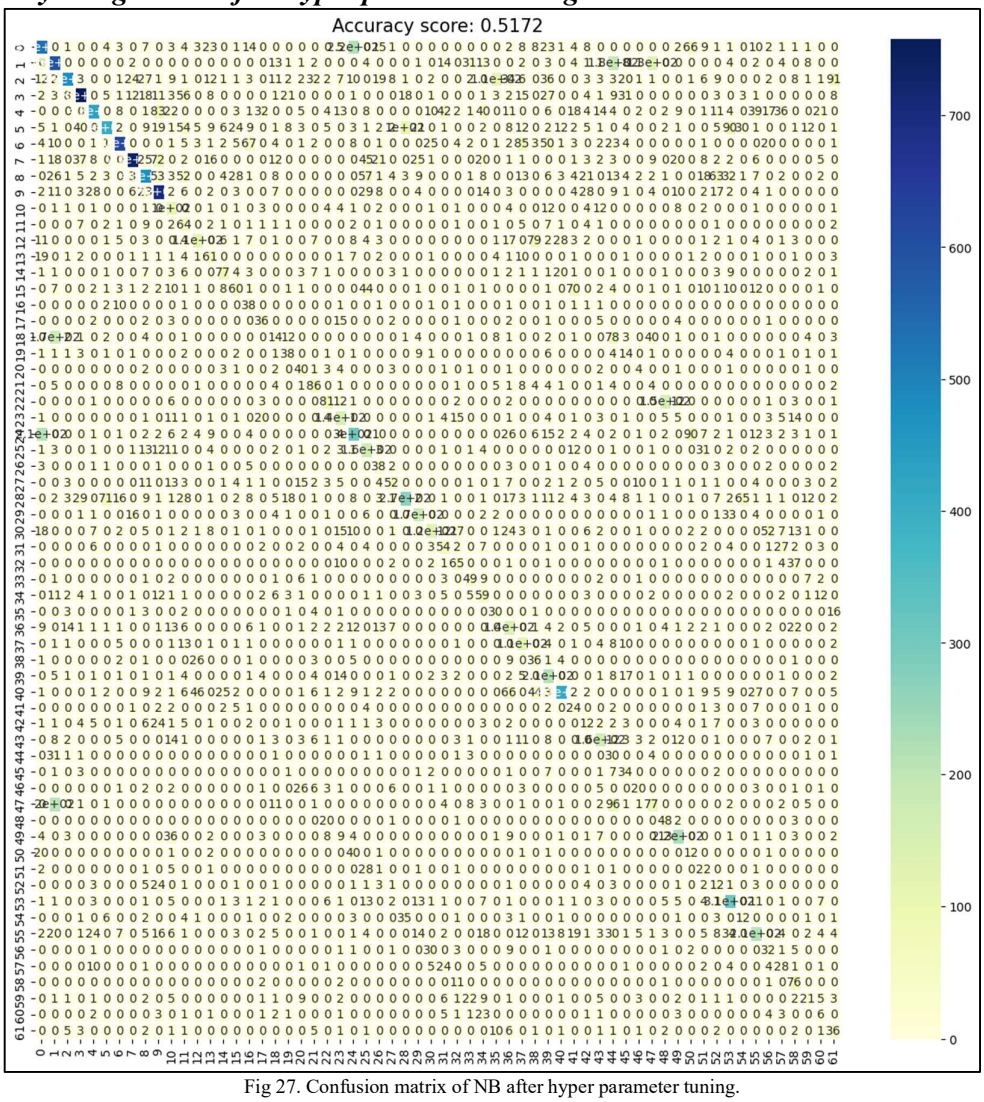

Fig 27. Confusion matrix of NB after hyper parameter tuning.

	precision	recall	f1-score	support		precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.61	0.54	0.58	976										
1	0.52	0.58	0.55	1023	21	0.48	0.61	0.54	141	41	0.13	0.42	0.20	57
2	0.90	0.47	0.62	1003	22	0.53	0.30	0.38	273	42	0.09	0.12	0.10	97
3	0.83	0.73	0.78	1035	23	0.50	0.58	0.54	249	43	0.62	0.58	0.60	269
4	0.79	0.48	0.59	903	24	0.46	0.41	0.44	741	44	0.05	0.39	0.10	76
5	0.79	0.43	0.56	928	25	0.39	0.59	0.47	277	45	0.20	0.57	0.30	60
6	0.88	0.64	0.74	959	26	0.24	0.57	0.34	67	46	0.34	0.24	0.28	83
7	0.91	0.67	0.77	1098	27	0.51	0.33	0.40	160	47	0.28	0.18	0.22	421
8	0.68	0.50	0.58	941	28	0.62	0.43	0.51	624	48	0.20	0.64	0.30	75
9	0.67	0.75	0.71	929	29	0.61	0.68	0.65	249	49	0.68	0.67	0.67	332
10	0.33	0.61	0.43	170	30	0.58	0.37	0.45	342	50	0.07	0.15	0.09	79
11	0.19	0.54	0.28	118	31	0.29	0.43	0.35	126	51	0.14	0.34	0.20	65
12	0.54	0.35	0.42	316	32	0.47	0.52	0.49	125	52	0.07	0.18	0.10	68
13	0.37	0.48	0.41	128	33	0.38	0.58	0.46	85	53	0.55	0.75	0.63	411
14	0.56	0.48	0.52	162	34	0.24	0.42	0.30	141	54	0.11	0.15	0.13	79
15	0.38	0.23	0.29	261	35	0.17	0.48	0.25	62	55	0.56	0.42	0.48	502
16	0.21	0.63	0.32	60	36	0.32	0.49	0.39	278	56	0.20	0.36	0.26	89
17	0.30	0.49	0.37	74	37	0.35	0.66	0.46	170	57	0.21	0.32	0.25	88
18	0.11	0.04	0.06	350	38	0.19	0.40	0.26	90	58	0.36	0.82	0.50	93
19	0.34	0.40	0.37	95	39	0.44	0.72	0.55	297	59	0.24	0.20	0.22	107
20	0.33	0.56	0.41	71	40	0.82	0.58	0.68	709	60	0.08	0.10	0.09	60
										61	0.18	0.43	0.26	83

Fig 28. Classification report of NB after hyper parameter tuning.

Random Forest Classifier after hyper parameter tuning

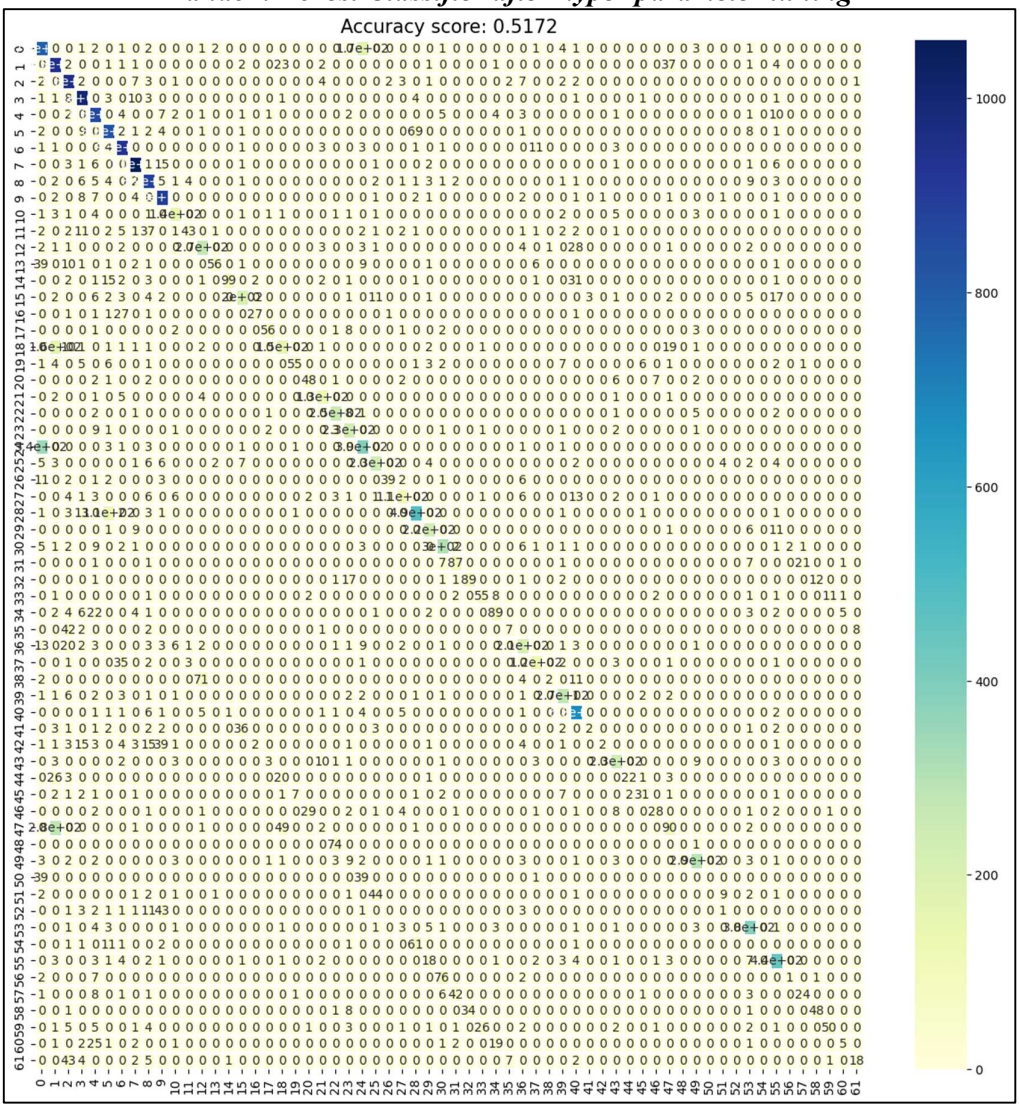

Fig 29. Confusion matrix of RF after hyper parameter tuning.

	precision	recall	f1-score	support		precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.62	0.81	0.70	976										
1	0.65	0.93	0.77	1023	21	0.80	0.90	0.85	141	41	0.40	0.04	0.06	57
2	0.84	0.96	0.89	1003	22	0.73	0.92	0.82	273	42	0.67	0.02	0.04	97
3	0.91	0.97	0.94	1035	23	0.77	0.91	0.83	249	43	0.85	0.86	0.85	269
4	0.85	0.95	0.90	903	24	0.61	0.52	0.56	741	44	0.92	0.29	0.44	76
5	0.82	0.89	0.86	928	25	0.77	0.83	0.80	277	45	0.72	0.52	0.60	60
6	0.89	0.97	0.93	959	26	0.91	0.58	0.71	67	46	0.70	0.34	0.46	83
7	0.95	0.97	0.96	1098	27	0.81	0.68	0.74	160	47	0.55	0.21	0.31	421
8	0.86	0.94	0.90	941	28	0.77	0.78	0.78	624	48	0.00	0.00	0.00	75
9	0.87	0.97	0.92	929	29	0.82	0.89	0.85	249	49	0.90	0.88	0.89	332
10	0.82	0.84	0.83	170	30	0.73	0.89	0.80	342	50	0.00	0.00	0.00	79
11	0.84	0.36	0.51	118	31	0.62	0.69	0.65	126	51	0.60	0.14	0.22	65
12	0.75	0.84	0.79	316	32	0.72	0.71	0.72	125	52	0.00	0.00	0.00	68
13	0.90	0.44	0.59	128	33	0.65	0.65	0.65	85	53	0.85	0.93	0.89	411
14	0.98	0.61	0.75	162	34	0.71	0.63	0.67	141	54	0.00	0.00	0.00	79
15	0.79	0.77	0.78	261	35	0.44	0.11	0.18	62	55	0.84	0.88	0.86	502
16	0.87	0.45	0.59	60	36	0.78	0.74	0.76	278	56	0.33	0.01	0.02	89
17	0.86	0.76	0.81	74	37	0.81	0.71	0.76	170	57	0.51	0.27	0.36	88
18	0.60	0.43	0.50	350	38	0.67	0.02	0.04	90	58	0.79	0.52	0.62	93
19	0.85	0.58	0.69	95	39	0.87	0.91	0.89	297	59	0.79	0.47	0.59	107
20	0.59	0.68	0.63	71	40	0.86	0.96	0.91	709	60	0.42	0.08	0.14	60
										61	0.67	0.22	0.33	83

Fig 30. Classification Report of RF after hyper parameter tuning.

K Nearest Neighbour after hyper parameter tuning

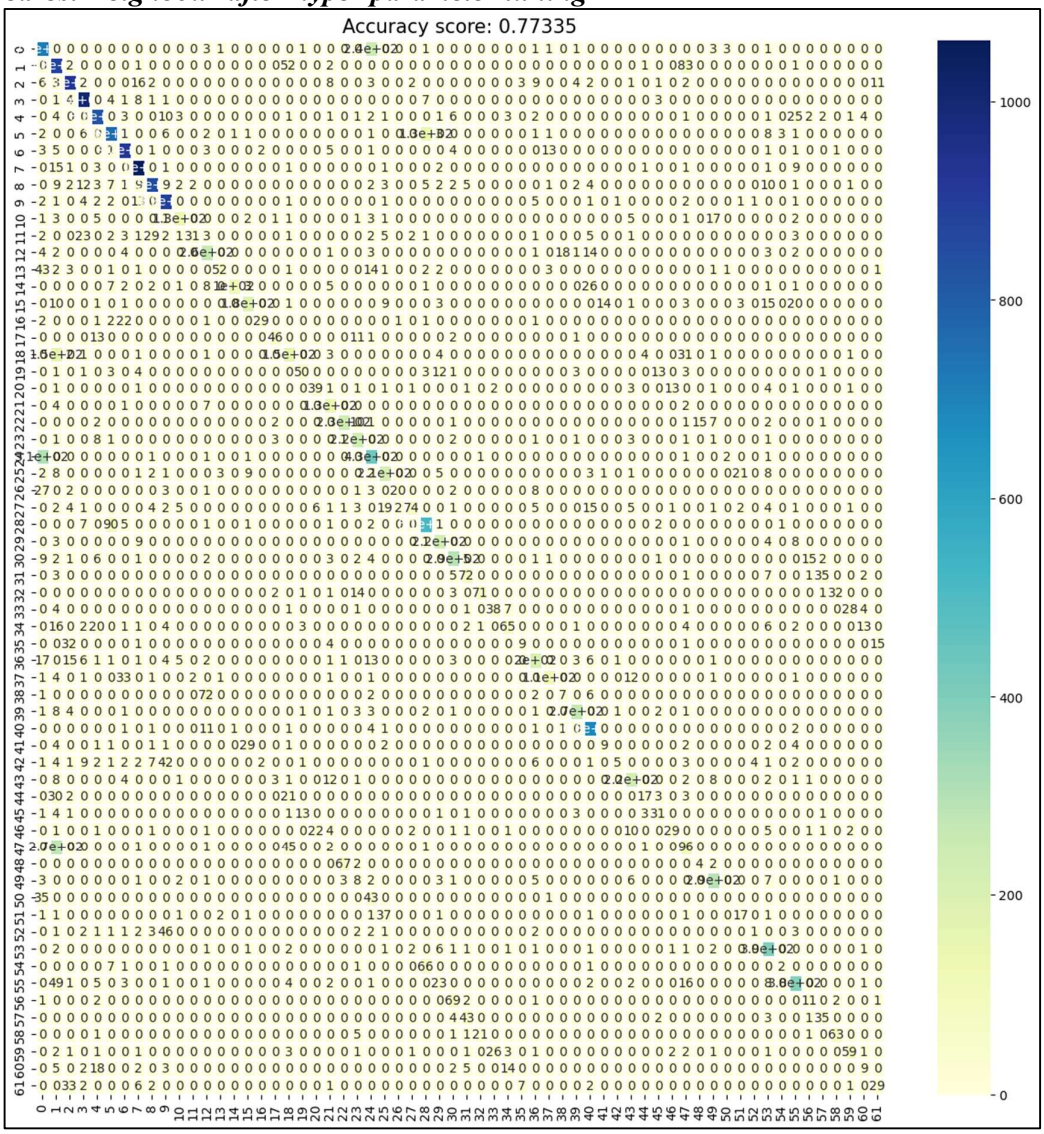

Fig 31. Confusion matrix of KNN after hyper parameter tuning.

	precision	recall	f1-score	support		precision	recall	f1-score	support		precision	recall	f1-score	support
0	0.60	0.73	0.66	976	21	0.68	0.90	0.77	141	41	0.38	0.16	0.22	57
1	0.58	0.86	0.69	1023	22	0.76	0.85	0.80	273	42	0.62	0.05	0.10	97
2	0.89	0.93	0.91	1003	23	0.77	0.90	0.83	249	43	0.82	0.84	0.83	269
3	0.93	0.97	0.95	1035	24	0.54	0.58	0.56	741	44	0.63	0.22	0.33	76
4	0.90	0.92	0.91	903	25	0.71	0.76	0.74	277	45	0.53	0.52	0.53	60
5	0.85	0.82	0.84	928	26	0.87	0.30	0.44	67	46	0.63	0.35	0.45	83
6	0.91	0.96	0.93	959	27	0.88	0.46	0.61	160	47	0.36	0.23	0.28	421
7	0.93	0.97	0.95	1098	28	0.70	0.82	0.75	624	48	0.21	0.05	0.09	75
8	0.94	0.90	0.92	941	29	0.76	0.90	0.82	249	49	0.86	0.87	0.87	332
9	0.87	0.96	0.91	929	30	0.72	0.84	0.77	342	50	0.00	0.00	0.00	79
10	0.86	0.74	0.79	170	31	0.51	0.57	0.54	126	51	0.39	0.26	0.31	65
11	0.89	0.26	0.41	118	32	0.76	0.57	0.65	125	52	0.17	0.01	0.03	68
12	0.68	0.83	0.75	316	33	0.58	0.45	0.50	85	53	0.78	0.94	0.85	411
13	0.88	0.41	0.56	128	34	0.69	0.46	0.55	141	54	0.33	0.03	0.05	79
14	0.98	0.65	0.78	162	35	0.47	0.15	0.22	62	55	0.81	0.76	0.78	502
15	0.79	0.69	0.74	261	36	0.78	0.71	0.74	278	56	0.33	0.12	0.18	89
16	0.88	0.48	0.62	60	37	0.84	0.65	0.74	170	57	0.44	0.40	0.42	88
17	0.81	0.62	0.70	74	38	0.27	0.08	0.12	90	58	0.64	0.68	0.66	93
18	0.52	0.43	0.47	350	39	0.93	0.90	0.91	297	59	0.62	0.55	0.58	107
19	0.71	0.53	0.61	95	40	0.88	0.97	0.92	709	60	0.26	0.15	0.19	60
20	0.58	0.55	0.57	71						61	0.51	0.35	0.41	83

Fig 32. Classification report of KNN after hyper parameter tuning.