Analysis of Classification Models on MNIST Dataset

Arpitha Srinivas, Ankita Jaswal, Reg Anastacio, Sarah Yu
San Jose State University
San Jose, U.S.A.

***Abstract*---Handwritten text is difficult to store and access, but converting it to digital data introduces issues in conversion accuracy and speed. This project analyzes three popular classification techniques to get the top performing model: Decision Trees, Random Forest, and k-Nearest Neighbors (KNN). The data comes from a subset of the Modified National Institute of Standards and Technology (MNIST) dataset which contains over 300,000 handwritten English capital letters, A-Z. Each model is benchmarked using accuracy and time efficiency for the same train and test datasets. One issue that we encountered during this research is validating the hyperparameters chosen for each model. From the experiments conducted, Random Forest outperforms KNN and Decision Trees in time performance and accuracy respectively.**

I. INTRODUCTION

Converting handwritten documents into digitized versions is the perfect way to practice data mining techniques that are used widely in machine learning for text recognition. The process of converting handwritten characters into digital format is currently being explored on a frequent basis. This is a result of current advancements made in the field of data storage and data retrieval being easily accessible from any device. One key area of study that our project focuses on is handwritten character recognition using popular Optical Character Recognition (OCR) classification models. A wide variety of models have been developed and proposed over the years and continue to be developed and modified to increase the accuracy of classification problems and improve the efficiency of handwriting character recognition.

In this paper, we compare three classification models based on their accuracy and efficiency in terms of performance measured in time. Decision Tree, Random Forest, and k-Nearest Neighbors (KNN) classifiers are cross-validated, trained, and tested using a Modified National Institute of Standards and Technology dataset. This dataset is a subset from the National Institute of Standards and Technology (NIST) Special Database 19 which contains A-Z handwritten English alphabet images with over 370,000 samples [1]. This dataset was pre-processed and formatted as a comma separated values (csv) file by Sachin Patel on Kaggle for easy application of predictive models to recognize handwritten characters [2]. This can be further used in organizations and companies that store important documents only in written format. It becomes easier and faster to complete the work with such a system available at hand.

II. RELATED WORK

Lavanya K. et al. [3] performed an experiment on Handwritten Digit Recognition using Hoeffding Tree, Decision Tree, and Random Forest. This paper elaborates the approach taken for each one of these algorithms and compares which is the most effective for handwritten digit recognition. They train the system and pre-process the MNIST dataset with a series of steps. The input data images used are converted into grayscale and binarized. These converted images then go through a process of feature extraction to further enhance them. This data is then stored as comma separated values (csv) format for training and testing. After these steps, model fitting is done using cross-validation and results are gathered using a classification report and a confusion matrix. From the results, the authors conclude that Decision Tree performs poorly in comparison to Random Forest and Hoeffding Tree. They stated that both Random Forest and Hoeffding Tree are effective models for this application where Random Forest provides more accurate classification in certain cases and Hoeffding Tree provides faster classification results in terms of time.

Rajni Jindal et al. [4] studied current techniques used in text classification and provided a literature review of their findings. Their work provides various insights into popular techniques in the area of text classification of documents being organized into digital data. From the literature survey done in their research, the authors listed various machine learning methods that are used most frequently in this area. However, the most popular used data mining methods are Support Vector Machine (SVM) and k-Nearest Neighbors (KNN). Zhenyun Deng et al. [5] propose an efficient KNN classification algorithm on big data which includes nine large datasets. One of the datasets used was the MNIST dataset where the accuracy and efficiency is measured for the proposed KNN algorithm in comparison to the original KNN algorithm. The authors analyzed the k-value which is a parameter often optimized to improve the algorithm's accuracy and efficiency [6]. Their experiment concluded that by optimizing the parameters and deriving a new KNN algorithm, the accuracy and efficiency does in-fact improve when classifying big data.

III. METHODOLOGY

A. *Dataset Preprocessing*

When preprocessing the dataset, a few factors had to be considered. Prior to preprocessing, we considered splitting the dataset in train and test datasets. This was done using the sk-learn library using train_test_split with default parameters. Training data consists of 80% of the original MNIST dataset and testing data consists of 20% of the original dataset. Splitting the dataset reduces chances of data snooping. After the split we began preprocessing the training dataset.

Only a few steps were taken to preprocess the training dataset as the original dataset already contained gray-scaled 28x28 pixeled samples stored in a csv format [2]. Also, the dataset consisted of zero null samples. This allows us to easily fit our data to any classification model. Although the training data was now preprocessed, we noticed a small discrepancy. While observing the preprocessed training dataset, we noticed that class distribution was unbalanced which may lead to unintended bias. As shown in Fig.1, we can see that the class distribution is unbalanced with the letter "O" being oversampled and the letter "I" being under-sampled. As a result, a stratified shuffle split was used to prevent more bias in a training and test dataset split [7]. A stratified shuffle split from the sk-learn library is a combination of a shuffle split and a stratified k-fold [8]. Shuffle split introduces randomness when splitting the dataset, while stratified k-fold ensures distributions of splits match the original dataset distribution. This creates splits of training and test dataset with distributions nearly equal to the original dataset.

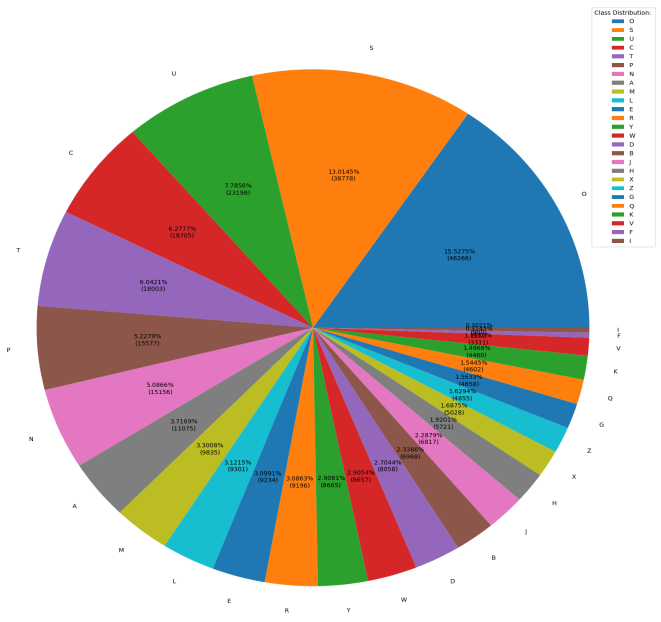


Fig.1. Pie Chart of A-Z letter class distribution.

B. *Decision Tree*

Decision Tree is a tree shaped supervised learning technique that is popularly used for both classification and regression problems. There are a root node, some decision nodes, and some leaf or terminal nodes in a Decision Tree. How the Decision Tree model works is by learning simple decision rules through asking questions about the features from the training data to predict the class or value of the target variable. The answers to the features split the tree into subsets. The process is repeated until all the target values in the subset at each leaf node are pure, or until it reaches the maximum depth that's preset.

As a popular and widely used supervised learning mode, there are many advantages of the Decision Tree model. It is easy to comprehend and interpret. And not much data preparation is needed, e.g. taking care of blank values. It's not expensive to run the prediction and train the model. It works not only on numerical data, but also on categorical data. It can be validated numerically, etc. There are also disadvantages, like the possibilities of overfitting. The result is not always stable. If the training dataset is not balanced, biased trees can be produced, etc [9].

C. *Random Forest*

Random Forest is a supervised learning algorithm that has applications in both classification and regression. In our project, we are focusing on the classification aspect of this model. Random Forest is a solution to the overfitting problem that is commonly occurring in Decision Trees[10]. Random Forest algorithms have a few hyperparameters, for our classifier we are using sk-learn's RandomForestClassifier which contains n_estimators as the main hyperparameter [9] .

The Random Forest algorithm is made up of a collection of Decision Trees and each tree contains a part of the data from a training dataset. Since we are only tuning n_estimator, the rest of hyperparameters are set to their default values. One of these default parameters is known as bootstrap which by default is True and uses the entire training dataset for each decision tree since no max_samples are set [9]. Bootstrap, which randomly selects subsets of samples from the training dataset, introduces diversity in the trees[10]. To predict a class using Random Forest, a majority vote is taken from the Decision Trees[10].

D. *KNN*

K Nearest Neighbors (KNN) is a supervised learning algorithm which has applications in both classification and regression. In the case of this project's application, we will be using the classification approach. The KNN algorithm considers its nearest neighbors to predict the class/label of the output object [12]. It performs the following steps to predict the class:

1. User sets the number of K neighbors.
2. Calculates the distance of training data points for K number of neighbors.

3. Takes the K nearest neighbors from the result of calculated distance into consideration.
4. Using the K neighbors in step 3, counts the number of data points in each class.
5. Assigns the class of test data points depending on the mode of class from the K neighbors.

KNN can also use distance as a scoring system for deciding which label to classify the data with. In sk-learn, the inverse of the distances is added for the same labels and the label with the highest score within k neighbors is the label for the test data. Closer neighbors are then considered more important.

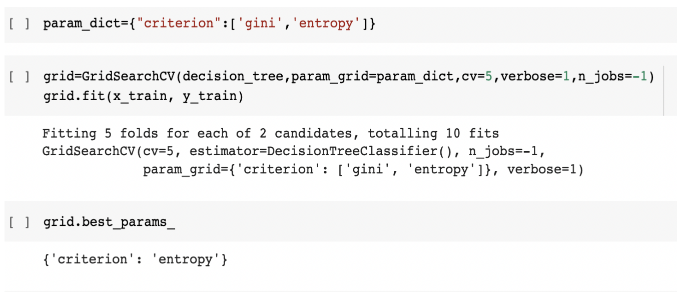
E. Model Performance

Each model for this research will be timed as another way of measuring performance aside from their accuracy. This is an important metric as classifying letters should not only be accurate, but it should also be quick as it may need to work on large datasets. Since each model handles fitting and predictions in different time scales, separate timings will be done for both fit and scoring functions.

IV. EXPERIMENTS

A. Decision Tree Approach and Results

In order to optimize our hyperparameters, we used Sklearn's GridSearchCV with 5 cross validations. One of the parameters being "entropy" or "gini". Entropy is a measure of how random the information being processed is. The higher the entropy is, the more random the information is and the harder it is to come up with any conclusions. "Gini" can be regarded as a cost function to be used to estimate the splits. Fig.2. shows the code snippet and result for applying GridSearchCV with 5 cross validations on applying "entropy" and "gini". As shown from the figure, "entropy" outperforms "gini", and that is the parameter we will choose for the Decision Tree model in this project. And the classification report is attached below.



```
[ ] param_dict={"criterion":['gini','entropy']}
```

```
[ ] grid=GridSearchCV(decision_tree,param_grid=param_dict,cv=5,verbose=1,n_jobs=-1)
    grid.fit(x_train, y_train)

    Fitting 5 folds for each of 2 candidates, totalling 10 fits
    GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                 param_grid={'criterion': ['gini', 'entropy']}, verbose=1)
```

```
[ ] grid.best_params_

    {'criterion': 'entropy'}
```

Fig.2. GridSearchCV implementation for Decision Tree hyperparameter tuning

TABLE I
Classification Report Decision Tree

CLASS	Precision	Recall	F1-score	Support
A	0.95	0.94	0.95	2774
B	0.92	0.91	0.92	1734
C	0.96	0.97	0.97	4682
D	0.92	0.93	0.93	2027
E	0.94	0.95	0.95	2288
F	0.91	0.84	0.87	233
G	0.91	0.91	0.91	1152
H	0.91	0.91	0.91	1444
I	0.94	0.85	0.89	224
J	0.93	0.92	0.93	1699
K	0.92	0.90	0.91	1121
L	0.97	0.97	0.97	2317
M	0.95	0.89	0.92	2467
N	0.95	0.97	0.96	3802
O	0.98	0.99	0.98	11565
P	0.97	0.98	0.97	3868
Q	0.91	0.89	0.90	1162
R	0.93	0.93	0.93	2313
S	0.98	0.98	0.98	9684
T	0.98	0.98	0.98	4499
U	0.97	0.98	0.97	5801
V	0.96	0.94	0.95	836
W	0.95	0.95	0.95	2157
X	0.93	0.95	0.94	1254
Y	0.94	0.95	0.94	2172
Z	0.95	0.94	0.94	1215
Accuracy			0.96	74490
Macro avg	0.94	0.94	0.94	74490
Weighted avg	0.96	0.96	0.96	74490

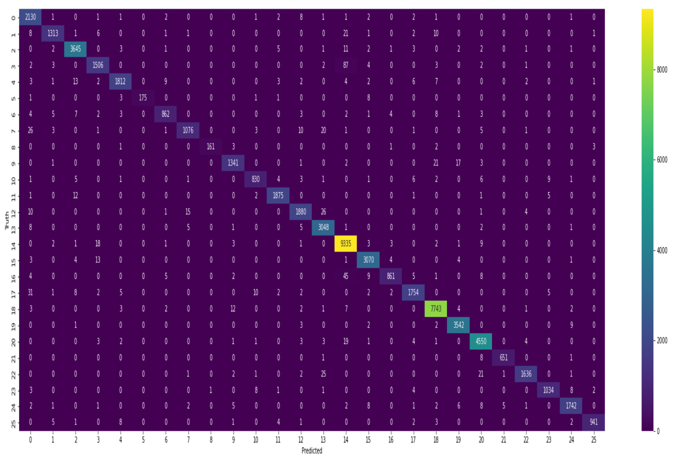


Fig.3. Confusion matrix Decision Tree

B. Random Forest Approach and Result

In the Random Forest Model, the n_estimators stand for the number of trees we build before taking the maximum voting or averages of predictions. The higher the n_estimators value is, the more accurate it is, but the longer time it takes to run the model. [13]

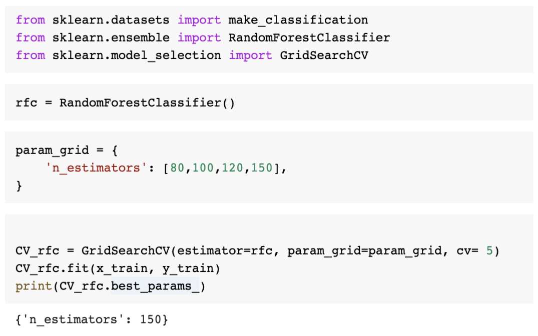
We applied GridSearchCV using n_estimators as the hyperparameter, the code snippet and result is shown in

Fig.4. Table II presents the classification report result for each letter with three different types of scores. As mentioned before, to avoid unintentional bias due to an imbalance in the original dataset, we will focus on the F1-Score. The F1-Score yields a weighted average of 96% in terms of accuracy.

From the Random Forest Classifier predicted result, we generated a confusion matrix as shown in Fig.5 to view the performance of this classifier.

TABLE II
Classification Report using Random Forest

CLASS	Precision	Recall	F1-score	Support
A	0.98	0.99	0.98	2795
B	0.99	0.97	0.98	1700
C	0.99	0.99	0.99	4704
D	0.97	0.96	0.97	2076
E	0.99	0.98	0.99	2206
F	1.00	0.93	0.97	226
G	0.99	0.97	0.98	1104
H	0.99	0.96	0.97	1497
I	1.00	0.94	0.97	220
J	0.99	0.98	0.99	1676
K	0.97	0.97	0.97	1143
L	0.99	0.99	0.99	2285
M	0.98	0.98	0.98	2501
N	0.98	0.99	0.98	3854
O	0.99	1.00	0.99	11559
P	0.99	1.00	0.99	3764
Q	0.98	0.96	0.97	1210
R	0.99	0.97	0.98	2370
S	0.99	1.00	0.99	9641
T	0.99	1.00	1.00	4492
U	0.99	0.99	0.99	5810
V	1.00	0.99	0.99	871
W	0.99	0.97	0.98	2127
X	0.98	0.97	0.98	1244
Y	0.99	0.99	0.99	2194
Z	0.99	0.97	0.98	1221
Accuracy			0.96	74490
Macro avg	0.99	0.98	0.94	74490
Weighted avg	0.99	0.99	0.96	74490



```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

```
rfc = RandomForestClassifier()
```

```
param_grid = {
    'n_estimators': [80,100,120,150],
}
```

```
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(x_train, y_train)
print(CV_rfc.best_params_)
```

```
{'n_estimators': 150}
```

Fig.4. GridSearchCV implementation for Random Forest hyperparameter tuning

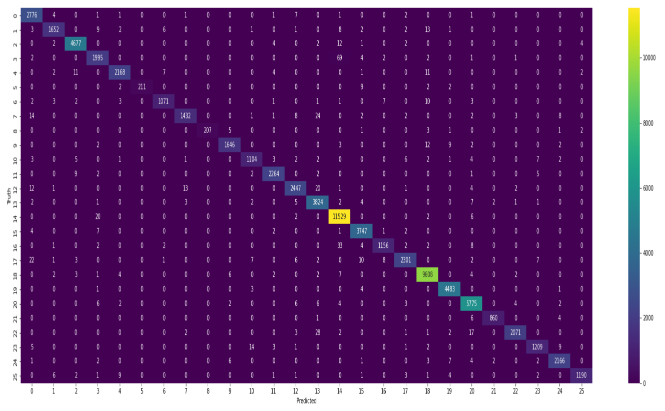


Fig.5. Heatmap of Confusion Matrix using Random Forest

Finally, the heatmap was normalized and the result is as shown in Fig.6 below.

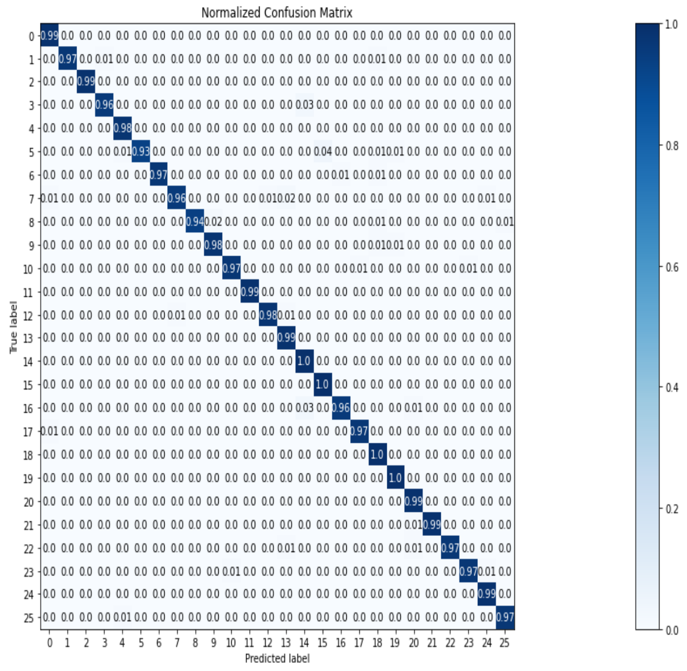


Fig.6. Normalized Confusion Matrix using Random Forest

We reapplied the above operation to the stratified dataset. Table III is the classification report of the stratified dataset. Fig.7 and Fig.8 show the stratified classification report and heat map.

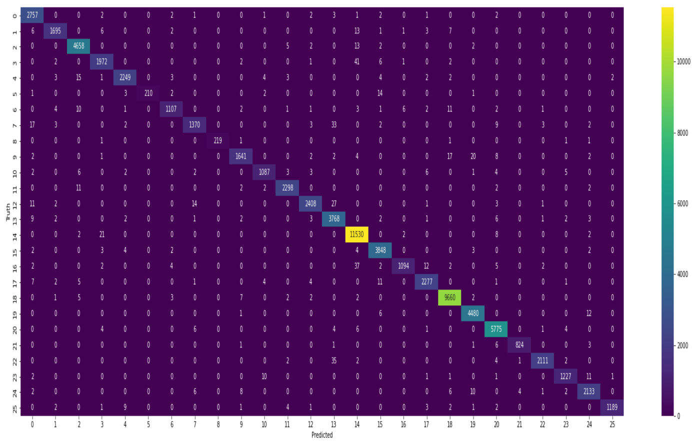


Fig.7. Heatmap of Confusion Matrix using Random Forest on Stratified Dataset

TABLE III
Classification Report using Random Forest on Stratified Dataset

CLASS	Precision	Recall	F1-score	Support
A	0.97	0.99	0.98	2775
B	0.99	0.97	0.98	1734
C	0.99	0.99	0.99	4682
D	0.98	0.98	0.98	2027
E	0.99	0.99	0.99	2288
F	1.00	0.90	0.95	233
G	0.99	0.96	0.97	1152
H	0.98	0.95	0.97	1444
I	1.00	0.97	0.99	224
J	0.98	0.97	0.97	1699
K	0.98	0.97	0.97	1121
L	0.99	0.99	0.99	2317
M	0.99	0.97	0.98	2467
N	0.98	0.99	0.98	3802
O	0.99	1.00	0.99	11565
P	0.98	0.99	0.99	3868
Q	0.99	0.94	0.96	1162
R	0.98	0.98	0.98	2313
S	0.99	1.00	1.00	9684
T	0.99	1.00	0.99	4499
U	0.99	0.99	0.99	5801
V	0.99	0.98	0.99	836
W	1.00	0.98	0.99	2157
X	0.99	0.98	0.98	1254
Y	0.98	0.98	0.98	2172
Z	0.99	0.98	0.99	1215
Accuracy			0.96	74490
Macro avg	0.99	0.98	0.94	74490
Weighted avg	0.99	0.99	0.96	74490

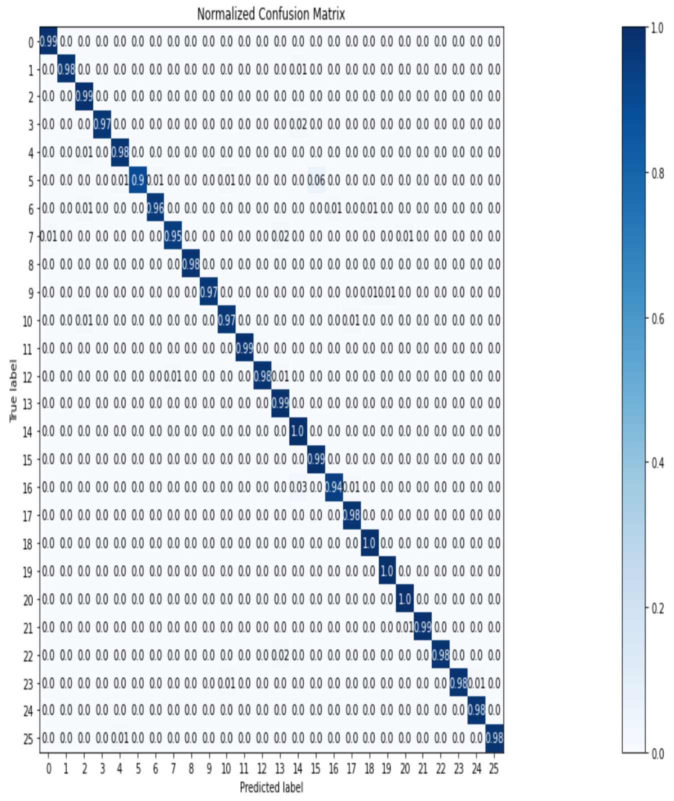


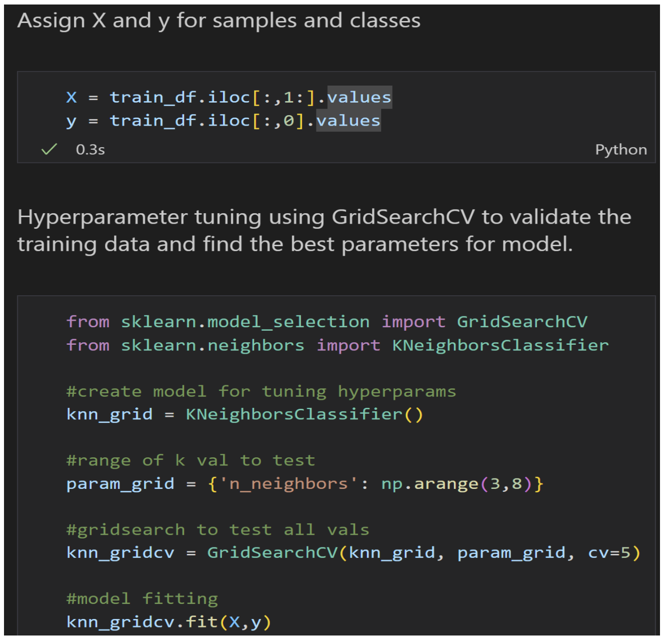
Fig.8. Normalized Confusion Matrix using Random Forest on Stratified Dataset

C. *KNN Approach and Results*

As defined in the data preprocessing section A, using the train and test split datasets, KNN model was fitted. In the case of our project, we used sklearn's neighbors KNeighborsClassifier. This classifier implementation provides a computation from a majority vote of k number of nearest neighbors of each point in the training data. The test data is then assigned a class which has the majority vote within the k nearest neighbors' distance from the test data [14].

To classify the MNIST A-Z data with the best possible classification, we optimized our hyperparameters using Sklearn's GridSearchCV with 5 cross validations. This implementation does the following: "parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid" [15]. One of the parameters being the n_neighbors, the number of neighbors defined by the user. Fig.9 shows our implementation of GridSearchCV using default hyperparameters. Fig.10 shows the results for this implementation's best fit n_neighbor hyperparameter after cross validation.

We attempted to use a range of {3,8} to find an optimal k-value that yields the highest accuracy at a minimal error rate for model fitting. After this observation, we noted that the optimal k value to use for model fitting with default hyperparameters is 5, which is also the default parameter for n_neighbors. As shown in Fig.11, the optimal k value is 5. In comparison to other k values this yields the highest accuracy score at 96%. As shown in Fig.12, the least error rate propagated is when k value is set to 5.



Assign X and y for samples and classes

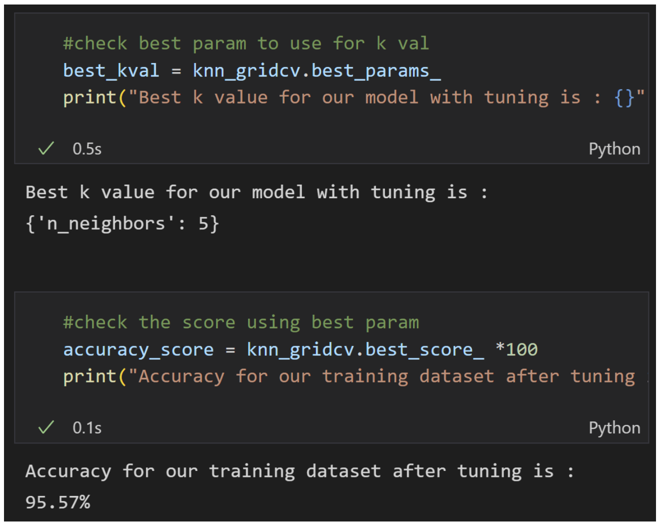
```
X = train_df.iloc[:,1:].values
y = train_df.iloc[:,0].values
```

✓ 0.3s Python

Hyperparameter tuning using GridSearchCV to validate the training data and find the best parameters for model.

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

#create model for tuning hyperparams
knn_grid = KNeighborsClassifier()

#range of k val to test
param_grid = {'n_neighbors': np.arange(3,8)}

#gridsearch to test all vals
knn_gridcv = GridSearchCV(knn_grid, param_grid, cv=5)

#model fitting
knn_gridcv.fit(X,y)
```

Fig.9. GridSearchCV implementation for KNN hyperparameter tuning



```
#check best param to use for k val
best_kval = knn_gridcv.best_params_
print("Best k value for our model with tuning is : {}"
```

✓ 0.5s Python

```
Best k value for our model with tuning is :
{'n_neighbors': 5}
```

```
#check the score using best param
accuracy_score = knn_gridcv.best_score_ *100
print("Accuracy for our training dataset after tuning
```

✓ 0.1s Python

```
Accuracy for our training dataset after tuning is :
95.57%
```

Fig.10. GridSearchCV results after tuning

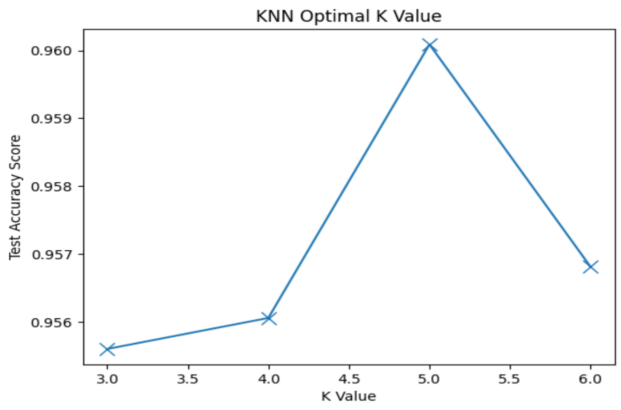


Fig.11. Plot of Optimal K Values in range(3,6).

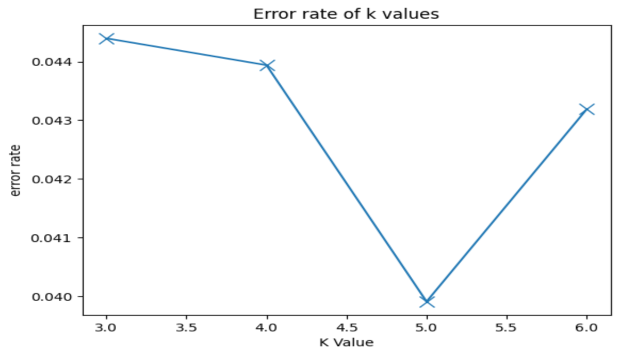


Fig.12. Plot of Error Rate for K Values in range(3,6).

After choosing 5 as the optimal k value or n_neighbors for KNN model fitting, the classification results were observed using the test dataset which are visualized using a confusion matrix and a classification report. Fig.13 displays the confusion matrix and Table IV entails the details of different scoring using the classification report. As seen in the confusion matrix, some test data was misclassified as a different class. For instance, the letter “D” being classified as the letter “O” 159 times; the letter “Q” being classified as the letter “O” 191 times; the letter “R” being classified as the letter “P” 190 times. This could be a result of class distribution being imbalanced where the letter “O” is oversampled in the original dataset. However, to compensate for this issue we have used a stratified shuffle split and tuned our hyperparameters while training our model to deal with the bias that comes from overfitting.

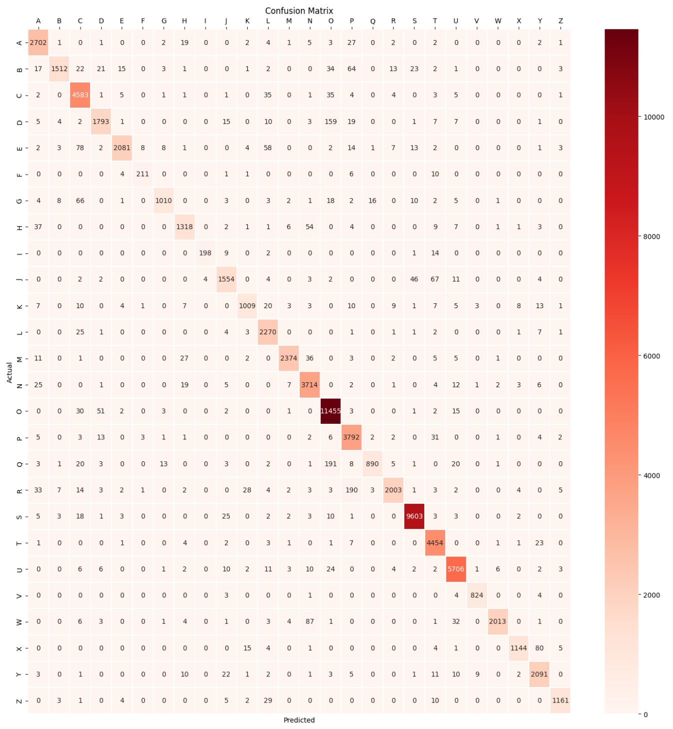


Fig.13. Confusion Matrix of KNN Predicted Labels vs. Actual Labels

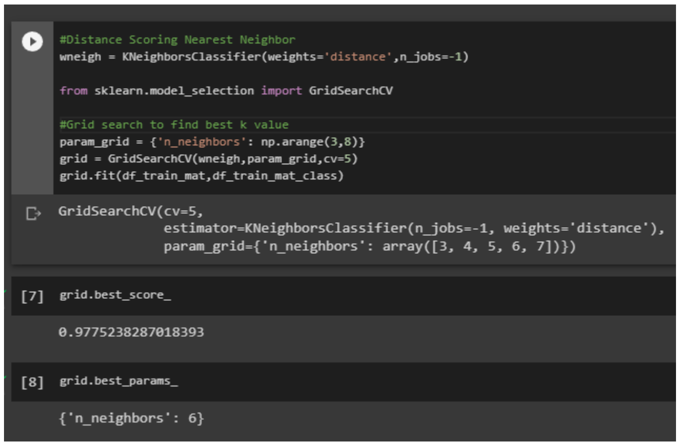
TABLE IV
Classification Report of KNN with different scoring for each class

CLASS	Precision	Recall	F1-score	Support
A	0.95	0.97	0.96	2795
B	0.97	0.88	0.93	1700
C	0.95	0.98	0.96	4704
D	0.94	0.88	0.90	2076
E	0.98	0.91	0.94	2206
F	0.94	0.92	0.93	226
G	0.96	0.88	0.92	1104
H	0.92	0.92	0.92	1497
I	0.98	0.90	0.93	220
J	0.94	0.92	0.93	1676
K	0.93	0.91	0.92	1143
L	0.92	0.98	0.95	2285
M	0.99	0.97	0.98	2501
N	0.95	0.98	0.97	3854
O	0.96	0.99	0.97	11559
P	0.91	0.98	0.94	3764
Q	0.98	0.77	0.86	1210
R	0.97	0.86	0.91	2370
S	0.99	0.99	0.99	9641
T	0.96	0.99	0.98	4492
U	0.97	0.98	0.98	5810
V	0.98	0.99	0.98	871
W	0.99	0.93	0.96	2127
X	0.98	0.92	0.95	1244
Y	0.93	0.97	0.95	2194
Z	0.98	0.94	0.96	1221
Accuracy			0.96	74490
Macro avg	0.96	0.93	0.95	74490
Weighted avg	0.96	0.96	0.96	74490

The classification report contains the precision score, the recall score, the f1-score, and the support. The support tells us the number of samples for each class that were used for scoring. Given the three types of scores, we will be using f1-score as the measure of correct accuracy. This combines both precision and recall then takes the average of both, in other words provides a mean accuracy value for all classes. This accounts for any imbalance present in the dataset and avoids unnecessary bias because it treats all samples in the dataset equally and accounts for under sampled classes.

The same testing is repeated for GridSearchCV on KNN with distance-based scoring to find the best k value to use on the test dataset as shown in Fig.14. It was given the same set of k values to run the grid search on to keep testing consistent. The search resulted in using a k value of 6 and a slightly higher accuracy in the classification report compared to the previous KNN method used: see Table IV and Table V. Typically, KNN should avoid using an even number because there is a chance of having a tie within the neighbors. Distance based scoring doesn't have this issue because each neighbor has its own calculated score which is more unlikely to produce a tie.

Using the best parameter given by the grid search for KNN with distance-based scoring, the average score is around 98%. It is higher than a regular KNN classification. This cross-validated result remains consistent with the same hyperparameters run on the test dataset.



```
#Distance Scoring Nearest Neighbor
wneigh = KNeighborsClassifier(weights='distance',n_jobs=-1)

from sklearn.model_selection import GridSearchCV

#Grid search to find best k value
param_grid = {'n_neighbors': np.arange(3,8)}
grid = GridSearchCV(wneigh,param_grid,cv=5)
grid.fit(df_train_mat,df_train_mat_class)

GridSearchCV(cv=5,
             estimator=KNeighborsClassifier(n_jobs=-1, weights='distance'),
             param_grid={'n_neighbors': array([3, 4, 5, 6, 7])})

[7] grid.best_score_

0.9775238287018393

[8] grid.best_params_

{'n_neighbors': 6}
```

Fig.14. GridSearchCV implementation for KNN-Distance Weight hyperparameter tuning

TABLE V
Classification Report of KNN with different scoring for each class

CLASS	Precision	Recall	F1-score	Support
A	0.98	0.99	0.99	2774
B	1.00	0.95	0.97	1734
C	0.98	0.99	0.98	4682
D	0.98	0.97	0.97	2027
E	1.00	0.97	0.98	2288
F	0.97	0.90	0.94	233
G	0.99	0.96	0.98	1152
H	0.98	0.97	0.97	1444
I	1.00	0.90	0.95	224
J	0.97	0.96	0.97	1699
K	0.98	0.96	0.97	1121
L	0.97	0.99	0.98	2317
M	1.00	0.97	0.98	2467
N	0.97	0.99	0.98	3802
O	0.99	1.00	0.99	11565
P	0.96	1.00	0.98	3868
Q	0.99	0.91	0.85	1162
R	0.99	0.96	0.97	2313
S	1.00	1.00	1.00	9684
T	0.98	1.00	0.99	4499
U	0.99	1.00	0.99	5801
V	0.99	0.99	0.99	836
W	1.00	0.98	0.99	2157
X	0.99	0.97	0.98	1254
Y	0.98	0.99	0.98	2172
Z	1.00	0.99	0.99	1215
Accuracy			0.98	74490
Macro avg	0.99	0.97	0.98	74490
Weighted avg	0.98	0.98	0.98	74490

D. *Performance Approach and Results*

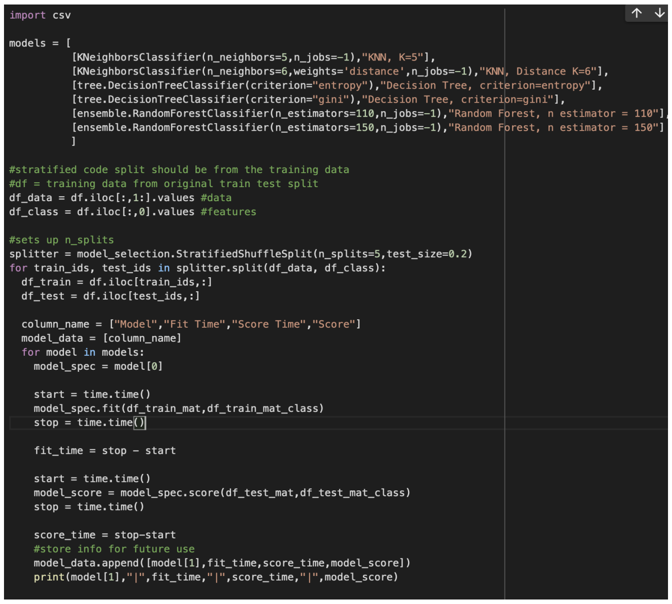
To time each model efficiently and with consistency, all tests were performed on the same machine. Since, sk-learn abstracts the methods for fit and scoring for each model a for loop is used to run each model and time them as shown on Fig.15.

The results for the tree classifiers and neighbor classifiers, taken separately, were similar for their own classifier groups as shown on Table VI. Both KNN classifiers had short fit time and long scoring time. Both tree classifiers had the opposite, long fit time, and short scoring time. When the two classifier groups are compared to each other, the results are drastically different.

KNN classifiers didn't do any fitting as evident by the relatively short fitting time. This is due to the high dimensionality of the dataset, which means that the models are using brute force to calculate nearest neighbors instead of trying to fit the data into a tree-like data structure for easier access. The time it would take to do this is exponentially longer than just using brute force [14]. KNN classifiers had a longer scoring time due to brute force calculations using Euclidean distance to calculate the closest neighbors.

The tree classifiers, on the other hand, had a longer fitting time due to the need to create their tree structures. Once the fitting is done, scoring is orders of magnitude faster than KNN due to less calculations done. Random Forest is slightly slower in scoring as it has a bigger tree than Decision Trees.

Overall, tree classifiers performed better than KNN classifiers when considering both fit and scoring time. Scoring time is especially useful in determining if a model can be used well in real time applications.



```
import csv

models = [
          [KNeighborsClassifier(n_neighbors=5,n_jobs=-1),"KNN, K=5"],
          [KNeighborsClassifier(n_neighbors=6,weights='distance',n_jobs=-1),"KNN, Distance K=6"],
          [tree.DecisionTreeClassifier(criterion="entropy"),"Decision Tree, criterion=entropy"],
          [tree.DecisionTreeClassifier(criterion="gini"),"Decision Tree, criterion=gini"],
          [ensemble.RandomForestClassifier(n_estimators=110,n_jobs=-1),"Random Forest, n estimator = 110"],
          [ensemble.RandomForestClassifier(n_estimators=150,n_jobs=-1),"Random Forest, n estimator = 150"]
          ]

#stratified code split should be from the training data
#df = training data from original train test split
df_data = df.iloc[:,1:].values #data
df_class = df.iloc[:,0].values #features

#sets up n_splits
splitter = model_selection.StratifiedShuffleSplit(n_splits=5,test_size=0.2)
for train_ids, test_ids in splitter.split(df_data, df_class):
  df_train = df.iloc[train_ids,:]
  df_test = df.iloc[test_ids,:]

  column_name = ["Model","Fit Time","Score Time","Score"]
  model_data = [column_name]
  for model in models:
    model_spec = model[0]

    start = time.time()
    model_spec.fit(df_train_mat,df_train_mat_class)
    stop = time.time()

    fit_time = stop - start

    start = time.time()
    model_score = model_spec.score(df_test_mat,df_test_mat_class)
    stop = time.time()

    score_time = stop-start
    #store info for future use
    model_data.append([model[1],fit_time,score_time,model_score])
    print(model[1],"|",fit_time,"|",score_time,"|",model_score)
```

Fig.15. Timing implementation for all models with fit() and score()

TABLE VI
Time results for all models with fit() and score()

Model	Fit Time(s)	Score Time(s)
KNN, K=5	0.0165	587.376
KNN, Distance K=6	0.016005	579.88
Decision Tree, criterion=entropy	66.15101	0.130492
Decision Tree, criterion=gini	72.7325	0.125499
Random Forest, n estimator = 110	36.7605	1.087499
Random Forest, n estimator = 150	49.19368	1.444497

V. CONCLUSION

The project proposed in this paper tries to implement three different supervised learning algorithms: Decision Trees, Random Forest, and K Nearest Neighbors, to find the best performing model for the application of OCR (Optical Character Recognition). For measuring performance, a preprocessed MNIST dataset of handwritten English Characters from Kaggle was used. This dataset was split into training and testing datasets to avoid data snooping. Each model uses cross-validation to validate the best-fit hyperparameters then each model is trained using the training dataset. At last, using the test dataset, each model is compared for their performance with time for model fitting, time for scoring, and accuracy score. From the results gathered in this project, both accuracy and speed, we can conclude that Random Forest is the best model for the application of OCR.

VI. CONTRIBUTIONS

Arpitha Srinivas - Random Forest
- Github - Link

Ankita Jaswal - KNN
- Github - Link

Reg Anastacio - KNN Distance and benchmarks
- Github - Link

Sarah Yu - Decision Tree and Random Forest
- Github - Link

All - Report and presentation

REFERENCES

[1] "Special Database 19." *NIST*, 27 Apr. 2019, www.nist.gov/srd/nist-special-database-19.

[2] "A-Z Handwritten Alphabets in .Csv Format." *Kaggle*, 16 Feb. 2018, www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format.

[3] K. Lavanya, S. Bajaj, P. Tank and S. Jain, "Handwritten digit recognition using hoeffding tree, decision tree and random forests — A comparative approach," 2017 International Conference on Computational Intelligence in Data Science(ICCIDS), 2017, pp. 1-6, doi: 10.1109/ICCIDS.2017.8272641.

[4] Jindal, Rajni, et al. "Techniques for Text Classification: Literature Review and Current Trends." *Webology*, 26 Dec. 2015, www.webology.org/2015/v12n2/a139.pdf. Accessed 12 Dec. 2022.

[5] Deng, Zhenyun, et al. "Efficient kNN Classification Algorithm for Big Data." *Neurocomputing*, vol. 195, Elsevier BV, June 2016, pp. 143–48. https://doi.org/10.1016/j.neucom.2015.08.112.

[6] S. Zhang, X. Li, M. Zong, X. Zhu and R. Wang, "Efficient kNN Classification With Different Numbers of Nearest Neighbors," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 5, pp. 1774-1785, May 2018, doi: 10.1109/TNNLS.2017.2673241.

[7] "Pros and cons of different sampling techniques." *International Journal of Applied Research*, 24 June 2017, https://d1wqtxts1xzle7.cloudfront.net/58765080/Pros_and_cons_of_sampling-with-cover-page-v2.pdf?Expires=1668401838&Signature=gMxaSnmwHOMvqG0rB33rjZxI3igTvUrflacsFhki2rH56lhGQW-nlNSpoUAsHdsJiYS0d8WGGGBNFLtUShy3mvQWi92YEnyY1Pwlt9RPZ9-oWRs4rrw0FPkj33-4UQxflJKQaNRwnrsa2J~SgWDpM6ah9c09niModGzbUIgM6H87isXO5lxbRHQS56vbDmMGmyrvfi6B~9GbTo-5qY7Rrc02ad6GB3Y8aRFTXPkGcUlv70MMYyvQ1DbA24YXobYrUEody3xhsBmOR1DlJev7bm8Xp4nALkLK8gqNMd2AWmVCLPy3ZxQmPGZm1yHcEDa3EWkngnIkZ90yqKevYr-tbg__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA. Accessed 13 November 2022.

[8] "3.1. Cross-validation: Evaluating Estimator Performance." *Scikit-learn*, scikit-learn.org/stable/modules/cross_validation.html.

[9] "Decision Trees", Scikit-learn, scikit-learn.org/stable/modules/tree.html

[10] Yiu, Tony. "Understanding Random Forest - Towards Data Science." Medium, 10 Dec. 2021, towardsdatascience.com/understanding-random-forest-58381e0602d2.

[11] h1ros. "Interpretability of Random Forest Prediction for MNIST Classification." *Step-by-Step Data Science*, 9 July 2019, https://h1ros.github.io/posts/interpretability-of-random-forest-prediction-for-mnist-classification-using-lime/.

[12] *Background of k-Nearest Neighbors (KNN)*. www.ibm.com/docs/en/ias?topic=knn-background.

[13] "Tuning the parameters of your Random Forest model" https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/#:~:text=n_estimators%20%3A,but%20makes%20your%20code%20slower.

[14] "1.6. Nearest Neighbors." *Scikit-learn*, scikit-learn.org/stable/modules/neighbors.html.

[15] "Sklearn.Model_Selection.GridSearchCV." *Scikit-learn*, scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.