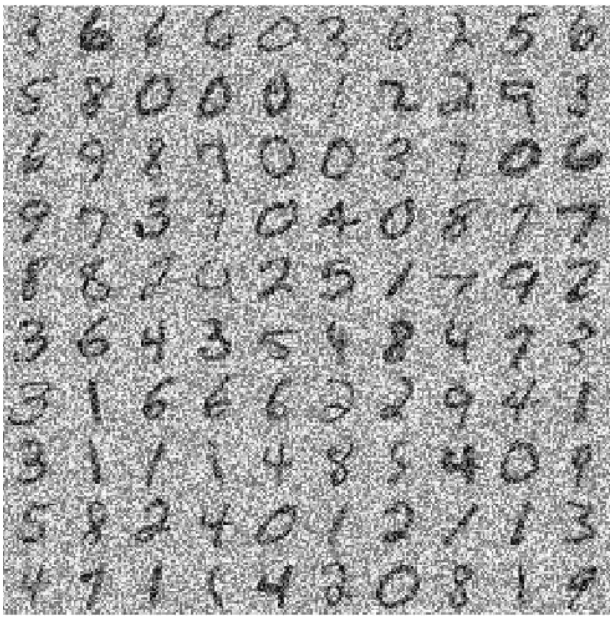
Tam Phi
ECE 612
Project 2

# Introduction

In this project, student will develop an ensemble learn from any 3 models to classify digits from MNIST dataset. Two modified datasets pulled from the MNIST will be provided. The following program will utilize multiple classifiers.

# Design and Implementation

**I. Data Processing**



A quick plotting of the provided dataset (left) and the MNIST dataset (right) reveals that the provided one has a lot of noise. One step could be denoise that dataset to improve performance of classification models. However, within the scope of this project, denoising is not applied but instead **StandardScaler()** is used to process data.

The provided dataset is scalar fit to reduce runtime and increase the performance particularly for SVM model (as detailed in chapter 3: Classification). As discussed later, it will become clear why certain models don't use the scaled dataset.

```
[6]  from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
```

X_train = unpickled("mnist_X_train.pkl)

**II. Grid Search & Model Selection**

Grid Search on a subset (5000) of the dataset is performed on the 3 models: Non-linear SVM, KNN, Logistic Regression.

**1)Non-linear SVM**

Parameter grid:

The following parameters are used in the search

```
[ ]  from sklearn.model_selection import GridSearchCV
     from sklearn.svm import SVC

     param_grid = [{'kernel':["poly"], 'degree':[3,4,5], 'coef0':[1],'C':[2,3,4]}]
     svc_clf = SVC()
     grid_search = GridSearchCV(svc_clf, param_grid, cv=5, verbose=3)
     grid_search.fit(sub_X_scaled, sub_y)
```

**Best score and parameters**

```
{'C': 2, 'coef0': 1, 'degree': 4, 'kernel': 'poly'}
0.8583999999999999
```

The kernel set to “poly” because the provided dataset (subset of MNIST) has many features per instance. The degree and C value are varied to achieve optimal score for the SVM model. A test of degree =3, 4, 5 and C=2,3,4 reveals that the best parameters are C =2 and degree =4. The increase in degree causes the increase in runtime, which will be discussed in subsequent sections. Further more, because the grid search is only on the subset of 5000 instances, the score is not has high as it could be in the full dataset of 60,000 (law of large number).

**2) KNN**

```
[ ]  from sklearn.neighbors import KNeighborsClassifier

     param_grid = [{'weights': ["distance"], 'n_neighbors': [7,10,12]}]
     knn_clf = KNeighborsClassifier()
     grid_search = GridSearchCV(knn_clf, param_grid, cv=5, verbose=3)
     grid_search.fit(sub_X, sub_y)
```

**Best score and parameters**

```
{'n_neighbors': 12, 'weights': 'distance'}
0.8328000000000001
```

The chosen “weight” parameter is “distance” instead of uniform because each digit is not weighted equally, meaning they don’t appear within the set the same number of times. It took many trial to find

the ideal “n_neighbors” for this model. First, this parameter was set to 5,6,7  it resulted in 7 as the best one. Next, 7, 10,12 are plugged in, showing that 12 is the best number of neighbors for this dataset. Similar to the SVM model, the score is ,while good (above average), will perform much better on the actual dataset.

**3) Logistic Regression**

```
[ ]  from sklearn.model_selection import GridSearchCV
     from sklearn.linear_model import LogisticRegression
     scaler = StandardScaler()
     sub_X_scaled = scaler.fit_transform(sub_X.astype(np.float64))
     logreg_grid = [{'multi_class':["multinomial"],'solver':["lbfgs"],'C': [3,4,5],
                     'class_weight':["balanced"], 'random_state': [42],'max_iter':[3000,6000]}]
     logreg_clf = LogisticRegression()
     grid_search = GridSearchCV(logreg_clf, logreg_grid, cv=5, verbose=3)
     grid_search.fit(sub_X, sub_y)
```

**Best score and parameters**

```
{'C': 4, 'class_weight': 'balanced', 'max_iter': 30000, 'multi_class': 'multinomial', 'random_state': 42, 'solver': 'lbfgs'}
0.7516
```

The intial intended model is RandomForest; however, the grid search for this model shows that it needs a tremendous amount of leaves (800) to achieve a score of roughly 0.8. While the

**III. Classification Model**

**The IDE changes from Google Colab to Spyder because of disconnection and runtime problem on the Google virtual machine. Executed codes are identical*

#Pseudo code

X_train_scaled = scalar_fit(X_train)

**1) Nonlinear SVM**

```
55 #%%
56 from sklearn.svm import SVC
57 from sklearn.pipeline import Pipeline
58
59 non_linear_svm_clf = Pipeline([
60 ("scaler", StandardScaler()),
61 ("svm_clf", SVC(kernel="poly",degree=4,coef0=1,C=2))
62 ])
63 non_lin_model = non_linear_svm_clf.fit(X_train_scaled, y_train)
```

```
[0.90236953 0.90314516 0.89843477]
```

Non-linear SVM performance is quite good, in the range of 0.89-0.9. The input X_train is scaled to improve performance of the model.

**2) KNN**

```
#%%
from sklearn.neighbors import KNeighborsClassifier
y_train_large = (y_train >= 6)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]
knn_clf = KNeighborsClassifier(n_neighbors=12,weights ='distance')
knn_clf.fit(X_train, y_multilabel)
```

```
[0.9064   0.9091   0.90675]
```

The label for KNN is split into 2 sets: large y_train (digit >= 6), and odd y_train (odd digit). This model, however, does not use scaled data for classification as I found the model performs better with not-scaled data.

**3) Logistic Regression**

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
softmax_clf = Pipeline([
("scaler",StandardScaler()),
("softmax_clf", LogisticRegression(
  multi_class="multinomial",
  solver="lbfgs", C=5, random_state=42,max_iter=60000))
])
softmax_clf.fit(X_train_scaled,y_train)
```

```
[0.83799534 0.83824754 0.82786202 0.8345     0.83666667 0.8409735
 0.831972   0.82863811 0.83408371 0.83622415]
```

Out of all classifier, the logistic regression performed worst but still results in a good range of 0.83 - 0.83. The scaled data set is used on this model.

**4) Voting Classifier**

The supplied data sets, X_train and y_train, are split into 2 set: test set and train set for the voting classifier. The voting classifier use estimators from the 3 previous classifiers: KNN, SVM, and Logistic Regression

```
#%%
from sklearn.model_selection import train_test_split
X_train_set, X_test_set = train_test_split(X_train, test_size=0.2, random_state=42)
y_train_set, y_test_set = train_test_split(y_train, test_size=0.2, random_state=42)
```

```
105 from sklearn.metrics import accuracy_score
106 for clf in (softmax_clf, knn_clf, non_linear_svm_clf, voting_clf):
107     clf.fit(X_train_set, y_train_set)
108     y_pred = clf.predict(X_test_set)
109     print(clf.__class__.__name__, accuracy_score(y_test_set, y_pred))
```

*Pipeline in line 1 is Logistic Regression. In line 3 is the Nonlinear SVM

```
Pipeline 0.8298333333333333
KNeighborsClassifier 0.8903333333333333
Pipeline 0.90575
VotingClassifier 0.8966666666666666
```

The voting is hard for the this Voting Classifier and result in a performance of 0.896. Performance might be improved via soft voting. However, due to time and resources constraints (soft voting did not finish running over 10hrs), hard voting was the final.

## Result and Discussion

**1) Confusion matrix**

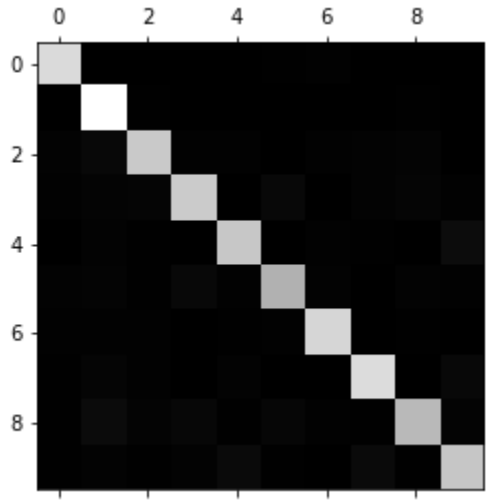
*Code is not shown here as it is simple and identical to the book. Please refer to the submitted code2.py

Confusion matrix of Voting Classifier on the y_train_pred and y_train_set

#Pseudo code

y_train_pred = votingclassifier.crossvalpredict(X_train_set,y_train_set)

```
[[4529    8   13   16    9   35   49    7   20    6]
 [   0 5299   34   13    5   15    6    6   35    7]
 [  64  176 4173   51   61   18   53   69   83   20]
 [  42   92  109 4221    8  170   20   66  114   51]
 [  15   74   32    6 4133   10   45   27   18  267]
 [  70   94   39  185   48 3682   77   14   67   51]
 [  57   61   44    5   40   61 4437    2   21    0]
 [  23  111   58   16   81   17    3 4565   11  181]
 [  38  234   83  147   38  137   51   27 3839   89]
 [  33   70   24   70  210   37    2  216   28 4106]]
```




The confusion matrix seems relatively good. The main diagonal is bright for most numbers. 5 and7 being the darkest means that they don't match as well as the actual. The precision and recall computation will give a clearer look at the performance of the Voting  Classifier

**2) Computed Precision and Recall of the Confusion Matrix**

```
130 import numpy as np
131 tp_arr = [0]*10
132 fp_arr = [0]*10
133 fn_arr = [0]*10
134
135 recall_arr = [0]*10
136 precision_arr = [0]*10
137 for i in range(10):
138     rowsum =0
139     colsum =[0]*10
140     for j in range(10):
141         rowsum = sum(conf_matrix[i])
142         colsum = np.sum(conf_matrix,axis=0)
143         if (i == j):
144             fp_arr[i] = rowsum - conf_matrix[i][j]
145             tp_arr[i] = conf_matrix[i][j]
146             fn_arr[i] = colsum[i] - conf_matrix[i][j]
147
148 print("N   Precision            Recall")
149 for i in range(10):
150     recall_arr[i] = tp_arr[i] / (tp_arr[i]+fn_arr[i])
151     precision_arr[i] = tp_arr[i] / (tp_arr[i]+fp_arr[i])
152     print(i," ",precision_arr[i]," ",recall_arr[i])
```

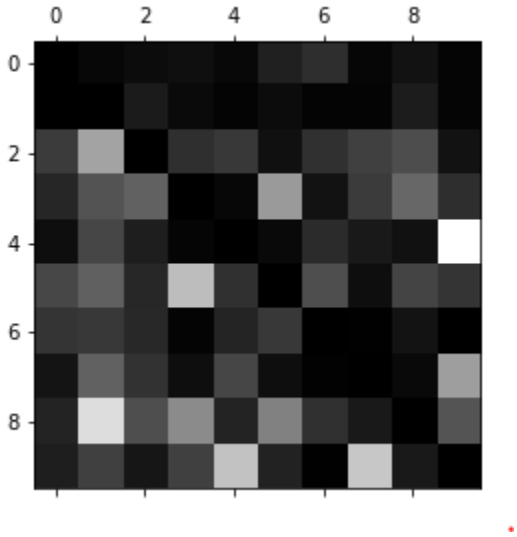
*N stands for digit

| N | Precision | Recall |
|---|---|---|
| 0 | 0.9652600170502984 | 0.9297885444467255 |
| 1 | 0.9776752767527676 | 0.8520662485930214 |
| 2 | 0.8752097315436241 | 0.9054024734215665 |
| 3 | 0.8626609442060086 | 0.8923890063424947 |
| 4 | 0.8932353576831641 | 0.8920785668033672 |
| 5 | 0.8509359833602959 | 0.880439980870397 |
| 6 | 0.9384517766497462 | 0.9354838709677419 |
| 7 | 0.9011054086063955 | 0.9131826365273055 |
| 8 | 0.8197736493700619 | 0.9062795089707271 |
| 9 | 0.8561301084236864 | 0.8593553788195898 |

The precision and recall values are quite good, ranging from 0.85 - 0.97, and 0.85 - 0.93, respectively. A high value for precision means that many digits that were positive (identified as some digit) turned out

to be actually positive (actually that some digit). Similarly, for recall, it indicates how many positive cases (correctly identified digit) that the model was able to identify.

**3) Error Analysis**




The column for digit 1 reveals that many digits were mistakenly identified as 1. While 2 and 1, 3 and 5, were mistaken for each other both ways. The most common digit identified as 10 is 4.

# Conclusion

In this project, we explore the different models for classifications. Most well performed ones are KNN, SVM, and Voting Classifier (0.89 -0.9). Though, the Voting Classifier would have performed better had it did soft voting. However, that begs the question of resources and performance, which is relevant to the topic of Machine Learning. For the scope of this project, it seems that at that level performance, it may satisfy the goal without foregoing too much resources.