Digit Recognizer

1.1 Data

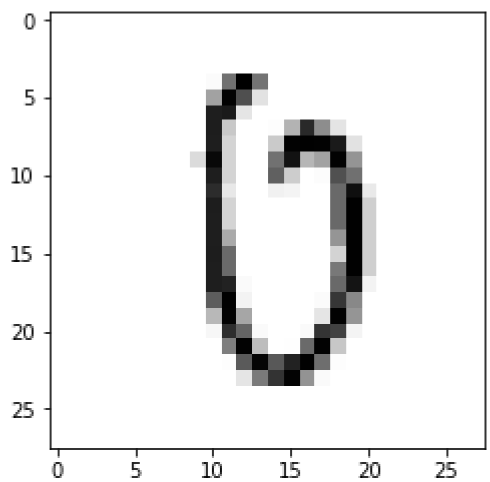
For reviewing the data, I conducted some necessary tests; for instance, the following information shows that the data is balanced concerning the classes.

```
Class Labels:
 [0 1 2 3 4 5 6 7 8 9]

Num of samples for each class respectively:
 [6903 7877 6990 7141 6824 6313 6876 7293 6825 6958]

Ratio of samples for each class respectively:
 [0.09861429 0.11252857 0.09985714 0.10201429 0.09748571 0.09018571
 0.09822857 0.10418571 0.0975     0.0994    ]
```

Visualization of one of the samples:



1.2 Code

The chosen algorithms are “**decision tree**,” “**random forest**,” and “**multi-layer perceptron**,” and the **objective is better classification performance**. There must be a model selection between the three chosen learning algorithms. However, before the selection between these three algorithms, we need to select the best parameters for each of these algorithms.

For the decision tree, we look for the best maximum depth for the tree. For random forest we need to find the best number trees through model selection, and, finally, for our multi-layer perceptron algorithm, we need to look for the best number of hidden layers, activation functions, number of epochs, and batch size. The possible values for the parameters can be changed in the code.

The general process is provided in Figure 1.

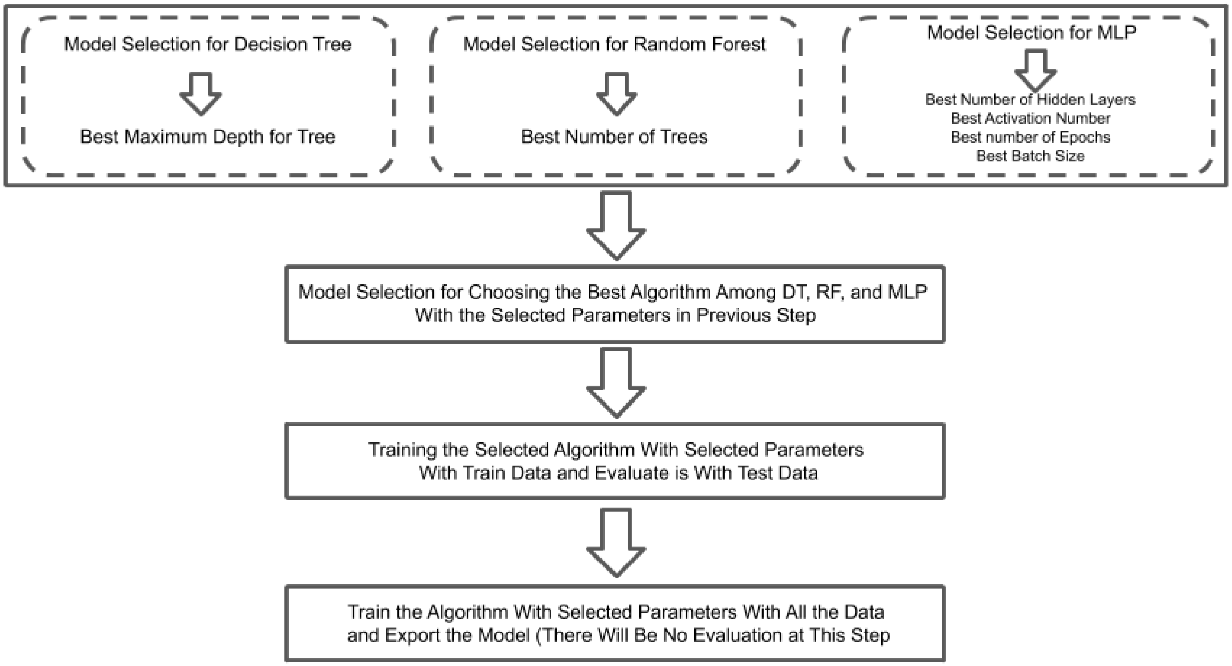


Figure 1 the process for model selection, evaluation, and final model generation.

Model Selection Decision Tree:

```
# define a function for selecting best number of trees for our decision tree alg.
def DT_select(X, Y, possible_depths=[10,20]):
    acc_list = []
    for d in possible_depths:
        print("training for depth = %d"%d)
        clf = DecisionTreeClassifier(random_state=0, max_depth=d)
        scores = cross_val_score(clf, X, Y, scoring='f1_macro', cv=5)
        acc_list.append(np.mean(scores))
        print("for depth = %d scores are :"%d,scores)
        print("=========================================")
    Best_depth = possible_depths[np.argmax(acc_list)]
    return Best_depth
```

Model Selection Random Forest:

```
# define a function for selecting best number of trees for our random forest alg.
def RF_select(X, Y, possible_num_of_trees=[100, 300, 500, 1000]):
    acc_list = []
    for num in possible_num_of_trees:
        print("training for %d trees"%num)
        clf = RandomForestClassifier(n_estimators=num)
        scores = cross_val_score(clf, X, Y, scoring='f1_macro', cv=5)
        acc_list.append(np.mean(scores))
        print("for %d trees scores are :"%num,scores)
        print("=========================================")
    Best_num_of_trees = possible_num_of_trees[np.argmax(acc_list)]
    return Best_num_of_trees
```

Model Selection Multi-Layer Perceptron:

```python
def MLP_select(X, Y, possible_depths=[1,2,3,4,5],
               possible_activation_functions=["sigmoid", "relu"],
               batch_sizes=[50,100,200,500],
               num_epochs=[50,100,200,500]):
    acc_list = [[[[0 for b in range(len(batch_sizes))]
                  for e in range(len(num_epochs))]
                 for a in range(len(possible_activation_functions))]
                for d in range(len(possible_depths))]
    input_shape = (X.shape[1], )
    num_classes = 10
    Y = to_categorical(Y, num_classes)
    num_folds = 5
    X_train_folds, X_val_folds, Y_train_folds, Y_val_folds = KFold_split(X, Y, num_folds, seed)

    depth_index = 0
    for depth in possible_depths:
        activation_function_index = 0
        for activation_function in possible_activation_functions:
            epoch_index = 0
            for epoch in num_epochs:
                batch_size_index = 0
                for batch_size in batch_sizes:
                    model = KerasClassifier(build_fn=build_MLP, input_shape=input_shape, num_classes=num_classes,
                         activation_type=activation_function, network_depth=depth, epochs=epoch, batch_size=batch_size, verbose=0)

                    print("training for Number of Hidden Layers:{}, Activation Function:{}, Num of Epocs:{}, Batch Size:{}"
                         .format(depth, activation_function,epoch,batch_size))

                    for X_train_fold, X_val_fold, Y_train_fold, Y_val_fold in zip(X_train_folds, X_val_folds, Y_train_folds, Y_val_folds):
                        model.fit(X_train_fold, Y_train_fold)
                        prediction = model.predict(X_val_fold)
                        acc_list[depth_index][activation_function_index][epoch_index][batch_size_index] +=\
                            f1_score(np.argmax(Y_val_fold,axis=1), prediction, average='macro')/num_folds
                    print("for Depth:{}, Activation Function:{}, Num of Epocs:{}, Batch Size:{} mean score is :"
                         .format(depth, activation_function,epoch,batch_size),
                         acc_list[depth_index][activation_function_index][epoch_index][batch_size_index])
                    print("=========================================")
                    batch_size_index = batch_size_index + 1
                epoch_index = epoch_index + 1
            activation_function_index = activation_function_index + 1
        depth_index = depth_index + 1

    acc_list = np.array(acc_list)
    best_depth_i, best_activation_function_i, best_num_epoch_i, best_batch_size_i = \
        np.unravel_index(acc_list.argmax(), acc_list.shape)
    best_depth, best_activation_function, best_num_epoch, best_batch_size = \
        possible_depths[best_depth_i],\
        possible_activation_functions[best_activation_function_i],\
        num_epochs[best_num_epoch_i],\
        batch_sizes[best_batch_size_i]
    return best_depth, best_activation_function, best_num_epoch, best_batch_size
```

In the next step, model selection between DT, RF, and MLP with the best parameters:

```
#DT
clf = DecisionTreeClassifier(random_state=0, max_depth=Best_depth)
scores = cross_val_score(clf, X, Y, scoring='f1_macro', cv=5)
acc_list[0] = np.mean(scores)
print("Decision Tree Classification Performance: ",acc_list[0])
print("=========================================")

# RF
clf = RandomForestClassifier(n_estimators=Best_num_of_trees)
scores = cross_val_score(clf, X, Y, scoring='f1_macro', cv=5)
acc_list[1] = np.mean(scores)
print("Random Forest Classifier Classification Performance: ",acc_list[1])
print("=========================================")

#MLP
input_shape = (X.shape[1], )
num_classes = 10
Y = to_categorical(Y, num_classes)
num_folds = 5
clf = KerasClassifier(build_fn=build_MLP, input_shape=input_shape, num_classes=num_classes,
        activation_type=Best_activation_function, network_depth=Best_num_of_hidden_layers,
                      epochs=best_num_epoch, batch_size=best_batch_size, verbose=0)
X_train_folds, X_val_folds, Y_train_folds, Y_val_folds = KFold_split(X, Y, num_folds, seed)
for X_train_fold, X_val_fold, Y_train_fold, Y_val_fold in zip(X_train_folds, X_val_folds, Y_train_folds, Y_val_folds):
    clf.fit(X_train_fold, Y_train_fold)
    prediction = clf.predict(X_val_fold)
    acc_list [2] += f1_score(np.argmax(Y_val_fold,axis=1), prediction, average='macro')/num_folds
print("MLP Classifier Classification Performance: ",acc_list[2])
print("=========================================")

model_index = np.argmax(acc_list)
```

Next step: evaluation of the final selected model:

```
# Model Evaluation

if model_index==0: #DT
    model.fit(X_train, Y_train)
    pred_Y = model.predict(X_test)
    Classification_Performance = f1_score(Y_test, pred_Y, average='macro')
elif model_index==1: #RF
    model.fit(X_train, Y_train)
    pred_Y = model.predict(X_test)
    Classification_Performance = f1_score(Y_test, pred_Y, average='macro')
else: #MLP
    num_classes = 10
    Y_train_cat = to_categorical(Y_train, num_classes)
    Y_test_cat = to_categorical(Y_test, num_classes)
    model.fit(X_train, Y_train_cat)
    pred_Y = model.predict(Y_test_cat)
    Classification_Performance = f1_score(Y_test, pred_Y, average='macro')

print("The Classification Performance for test set is: ", Classification_Performance)
```

Final step: training the final selected model by all the data and generating a model for the final application:

```
# Train the selected model on the whole data (there will be no evaluation)
if model_index==0: #DT
    model.fit(data, labels)
elif model_index==1: #RF
    model.fit(data, labels)
else: #MLP
    num_classes = 10
    labels_cat = to_categorical(labels, num_classes)
    model.fit(data, labels_cat)

print("==========================================")
print("THE FINAL MODEL IS GENERATED!")
print("==========================================")
```

Classification Performance Measure:

The measure chosen for evaluating the classification performance is F1 macro; F1 can be a reliable measure as it is the combination of precision and recall measures, and the macro version of it was used in the code since the problem is a multi-class classification.

F1-score = 2 × (precision × recall)/(precision + recall)

Macro-F1 = (Val #1 + Val #2 + Val #3) / 3 = Val

Class	F1-score (%)
Cat. #1	Val #1
Cat. #2	Val #2
Cat. #3	Val #3

Train Test split:

I used 70% of data for training and 30% for testing the learned selected model.

```
# Devide data to train and test
X_train, X_test, Y_train, Y_test = train_test_split(data, labels, test_size = 0.3, rand
om_state = 0)
```

Validation Folds:

The number of folds for all cross-validations is set to 5.

```
scores = cross_val_score(clf, X, Y, scoring='f1_macro', cv=5)
```

```
num_folds = 5
X_train_folds, X_val_folds, Y_train_folds, Y_val_folds = KFold_split(X, Y, num_folds, s
eed)
```

More Information:

My first choice of algorithms where KNN, RF, and MLP; however, since KNN needs the distance between all the samples and the volume of data it was extremely slow, so I changed my choice to the decision tree. In regard to dimensionality reduction, I implemented a model selection for autoencoder for different dimensionalities, but after conducting some test it turned out that the more reasonable result comes from dimensionalities almost close the original dimensionality; still the accuracy was degraded to considerable extent (for dimensionality reduction from 784 to 500

by nearly 15 percent of reduction in f1_macro score). To make sure that the algorithm does not overfit, I used 30% of the dataset for the final evaluation (test set).
There can be more choices for values of the parameters considered for model selection, but because of the limitations regarding the computational load and time, I suffice to values provided in the source code. Moreover, the parameters itself can be different, for instance number of units in the input and hidden layers of the MLP would be reasonable parameters to be selected; however, due to the limitations regarding the computational load I ignored it for model selection and chose 30 for the input layer and 60 for hidden layers.

Results:

Model Selection for DT (possible_depths=[8, 15]):

```
################____DT____#################
training for depth = 8
for depth = 8 scores are : [0.80025981 0.80975031 0.79846313 0.80527481
0.80632002]
=========================================
training for depth = 15
for depth = 15 scores are : [0.86058851 0.86452152 0.85933953 0.86518803
0.85727826]
=========================================
The best depth is:  15
```

Model Selection for RF (possible_num_of_trees=[100, 500]):

```
training for 100 trees
for 100 trees scores are : [0.96571259 0.9635131  0.96440118 0.96285871
0.96681695]
=========================================
training for 500 trees
for 500 trees scores are : [0.9668079  0.96708214 0.96672265 0.96455033
0.9696471 ]
=========================================
The best num of trees is:  500
```

Model Selection for MLP (possible_depths = [0,1,7], possible_activation_functions = ["sigmoid", "relu"], batch_sizes = [15, 30], num_epochs = [25, 50]):

```
################____MLP____#################
training for Depth:0, Activation Function:sigmoid, Num of Epocs:25, Batc
h Size:15
for Depth:0, Activation Function:sigmoid, Num of Epocs:25, Batch Size:15
mean score is : 0.8724698280246124
=========================================
training for Depth:0, Activation Function:sigmoid, Num of Epocs:25, Batc
h Size:30
for Depth:0, Activation Function:sigmoid, Num of Epocs:25, Batch Size:30
mean score is : 0.8890157139693442
=========================================
training for Depth:0, Activation Function:sigmoid, Num of Epocs:50, Batc
h Size:15
for Depth:0, Activation Function:sigmoid, Num of Epocs:50, Batch Size:15
mean score is : 0.8897884182645952
```

```
=========================================
training for Depth:0, Activation Function:sigmoid, Num of Epocs:50, Batc
h Size:30
for Depth:0, Activation Function:sigmoid, Num of Epocs:50, Batch Size:30
mean score is : 0.9045428449727475
=========================================
training for Depth:0, Activation Function:relu, Num of Epocs:25, Batch S
ize:15
for Depth:0, Activation Function:relu, Num of Epocs:25, Batch Size:15 me
an score is : 0.035476904557302555
=========================================
training for Depth:0, Activation Function:relu, Num of Epocs:25, Batch S
ize:30
for Depth:0, Activation Function:relu, Num of Epocs:25, Batch Size:30 me
an score is : 0.06841115253417357
=========================================
training for Depth:0, Activation Function:relu, Num of Epocs:50, Batch S
ize:15
for Depth:0, Activation Function:relu, Num of Epocs:50, Batch Size:15 me
an score is : 0.020338109356345273
=========================================
training for Depth:0, Activation Function:relu, Num of Epocs:50, Batch S
ize:30
for Depth:0, Activation Function:relu, Num of Epocs:50, Batch Size:30 me
an score is : 0.0524736941947774
=========================================
training for Depth:1, Activation Function:sigmoid, Num of Epocs:25, Batc
h Size:15
for Depth:1, Activation Function:sigmoid, Num of Epocs:25, Batch Size:15
mean score is : 0.8715675628990321
=========================================
training for Depth:1, Activation Function:sigmoid, Num of Epocs:25, Batc
h Size:30
for Depth:1, Activation Function:sigmoid, Num of Epocs:25, Batch Size:30
mean score is : 0.8926324526799325
=========================================
training for Depth:1, Activation Function:sigmoid, Num of Epocs:50, Batc
h Size:15
for Depth:1, Activation Function:sigmoid, Num of Epocs:50, Batch Size:15
mean score is : 0.8889677007145924
=========================================
training for Depth:1, Activation Function:sigmoid, Num of Epocs:50, Batc
h Size:30
for Depth:1, Activation Function:sigmoid, Num of Epocs:50, Batch Size:30
mean score is : 0.9053822053071819
=========================================
training for Depth:1, Activation Function:relu, Num of Epocs:25, Batch S
ize:15
for Depth:1, Activation Function:relu, Num of Epocs:25, Batch Size:15 me
an score is : 0.020340685958551576
=========================================
training for Depth:1, Activation Function:relu, Num of Epocs:25, Batch S
ize:30
```

```
for Depth:1, Activation Function:relu, Num of Epocs:25, Batch Size:30 me
an score is : 0.05232298644933704
========================================
training for Depth:1, Activation Function:relu, Num of Epocs:50, Batch S
ize:15
for Depth:1, Activation Function:relu, Num of Epocs:50, Batch Size:15 me
an score is : 0.020537617103082793
========================================
training for Depth:1, Activation Function:relu, Num of Epocs:50, Batch S
ize:30
for Depth:1, Activation Function:relu, Num of Epocs:50, Batch Size:30 me
an score is : 0.02070756580821505
========================================
training for Depth:7, Activation Function:sigmoid, Num of Epocs:25, Batc
h Size:15
for Depth:7, Activation Function:sigmoid, Num of Epocs:25, Batch Size:15
mean score is : 0.020154212292129655
========================================
training for Depth:7, Activation Function:sigmoid, Num of Epocs:25, Batc
h Size:30
for Depth:7, Activation Function:sigmoid, Num of Epocs:25, Batch Size:30
mean score is : 0.01941162375632829
========================================
training for Depth:7, Activation Function:sigmoid, Num of Epocs:50, Batc
h Size:15
for Depth:7, Activation Function:sigmoid, Num of Epocs:50, Batch Size:15
mean score is : 0.01921255805558271
========================================
training for Depth:7, Activation Function:sigmoid, Num of Epocs:50, Batc
h Size:30
for Depth:7, Activation Function:sigmoid, Num of Epocs:50, Batch Size:30
mean score is : 0.020081765165077897
========================================
training for Depth:7, Activation Function:relu, Num of Epocs:25, Batch S
ize:15
for Depth:7, Activation Function:relu, Num of Epocs:25, Batch Size:15 me
an score is : 0.9507402516341135
========================================
training for Depth:7, Activation Function:relu, Num of Epocs:25, Batch S
ize:30
for Depth:7, Activation Function:relu, Num of Epocs:25, Batch Size:30 me
an score is : 0.9485595035923156
========================================
training for Depth:7, Activation Function:relu, Num of Epocs:50, Batch S
ize:15
for Depth:7, Activation Function:relu, Num of Epocs:50, Batch Size:15 me
an score is : 0.9527279235008697
========================================
training for Depth:7, Activation Function:relu, Num of Epocs:50, Batch S
ize:30
for Depth:7, Activation Function:relu, Num of Epocs:50, Batch Size:30 me
an score is : 0.9519217424356275
========================================
The best number of hidden layers is:  7
```

```
The best activation function is:  relu
The best number of epocs is:  50
The best batchsize is:  15
```

Evaluation of the Best Model:

```
=======================Selection Between DT, RF, and MLP===============
===========
Decision Tree Classification Performance:  0.8613831716103608
=========================================
Random Forest Classifier Classification Performance:  0.9665362143782851
=========================================
MLP Classifier Classification Performance:  0.93859088418216
=========================================
The best model is learnt by Random Forest Classifier!
```

Generation of the Final Model:

```
=========================================
THE FINAL MODEL IS GENERATED!
=========================================
```

Please the notebook at the end of this document for more information.