HW01

1) Classifiers used:

- KNN-

    A. Function: $f_{KNN}(x) = arg\ max_{y \in \Upsilon} \sum_{i \in N_K(x)} I[y_i = y]$
    B. Hyperparameters adjusted here: Nearest neighbor(K-value) and p-value for Minkowski's distance formula
    C. Why KNN: KNN is specifically chosen to be trained on USPS-Digits data. It showed least generalization error compared to other classifier models for this data. The data set is the second largest, so it is appropriate for a low bias-high variance model like KNN. It is a simple, but highly accurate classifier. Moreover, this data set has the most classes, which is suitable for training KNN

- Decision Tree-

    A. Function: Individual leaves in a decision tree contains rules (x_d<t) or (x_d=t) which compares dimension d with value t, based on which the case is passed on to the left or right child.
    B. Hyperparameters adjusted here: Maximum-depth and minimum sample leaf.
    C. Why decision tree: This classifier was primarily chosen for the data set of Occupancy_detection, for which it yielded lowest generalization error. It has the least dimensionality, hence maximum depth can be kept low to avoid overfitting and achieve lower variance for this large data set. Since the tree can be shallow, it can be trained faster.

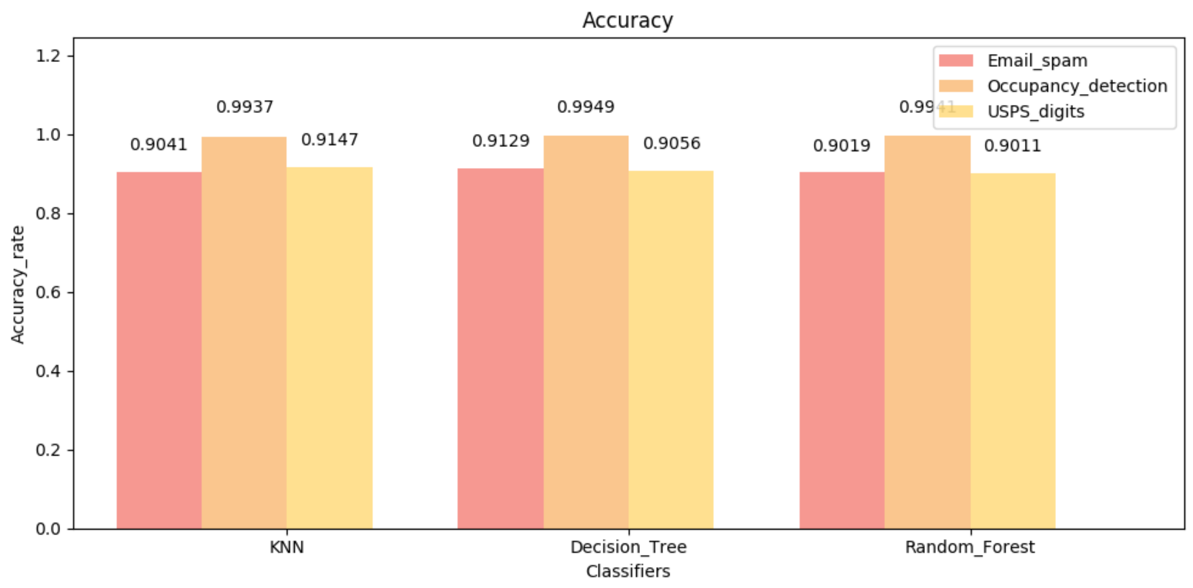
- Random Forest-

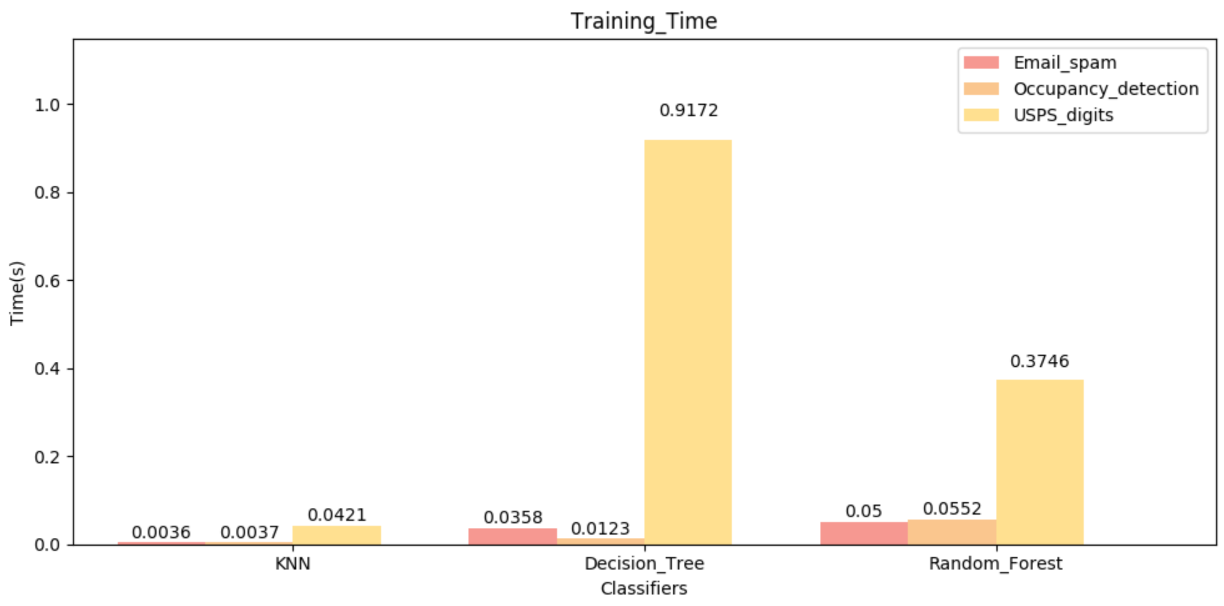
    A. Function: It creates multiple decision trees, from which it selects the better-performing decision tree based on hyperparameters
    B. Hyperparameters adjusted here: n-estimators, maximum features and minimum sample leaf.
    C. Why Random Forest-It performed best for Email_spam dataset, which had the smallest dataset, leading to initial high generalization error. Random forest can be fine tuned using aforementioned hyperparameters for it to minimize overfitting.

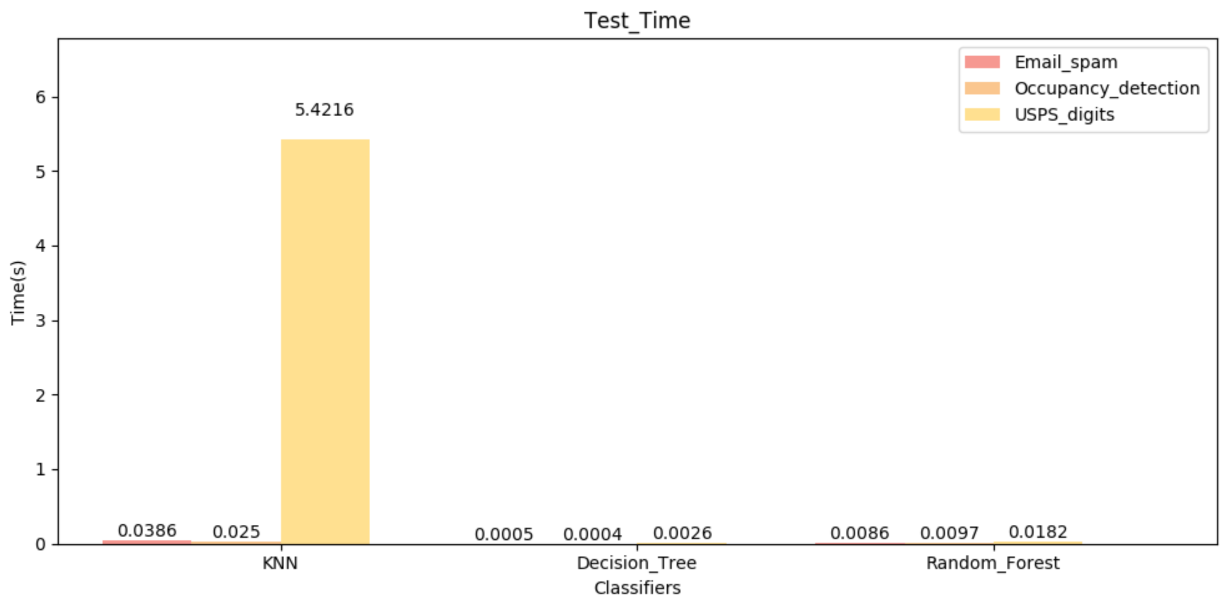
2) Random Forest uses an 'ensemble' of decision trees to train them with the given dataset and selects the one with the best performance. Random Forest does have decision boundary geometry similar to that of KNN, as both make predictions for x based on its closest neighbors.

KNN is non-parametric, hence it needs to store all training data in order to make predictions, requiring large amounts of memory. As a result, despite training time being small, it is quite slow at making predictions. Ability to change p for Minkowski distance for different cases make it flexible. On the other hand, unlike KNN, decision trees are parametric, acquiring rules from data set rather than storing the data set entirely. Hence, memory consumption is very low. It has high flexibility owing to interpretability of said rules. Test time decreases with decreasing depth, whereas training time can be high, as it uses greedy algorithms. Random forests are highly accurate as it trains less powerful classifiers with portions of data. The fact that it can be fine tuned owing to the plethora of available hyperparameters makes it more flexible compared to the other classifiers. But it can take a long time to train. Since it is non-parametric like KNN, it consumes more memory.

3) a)





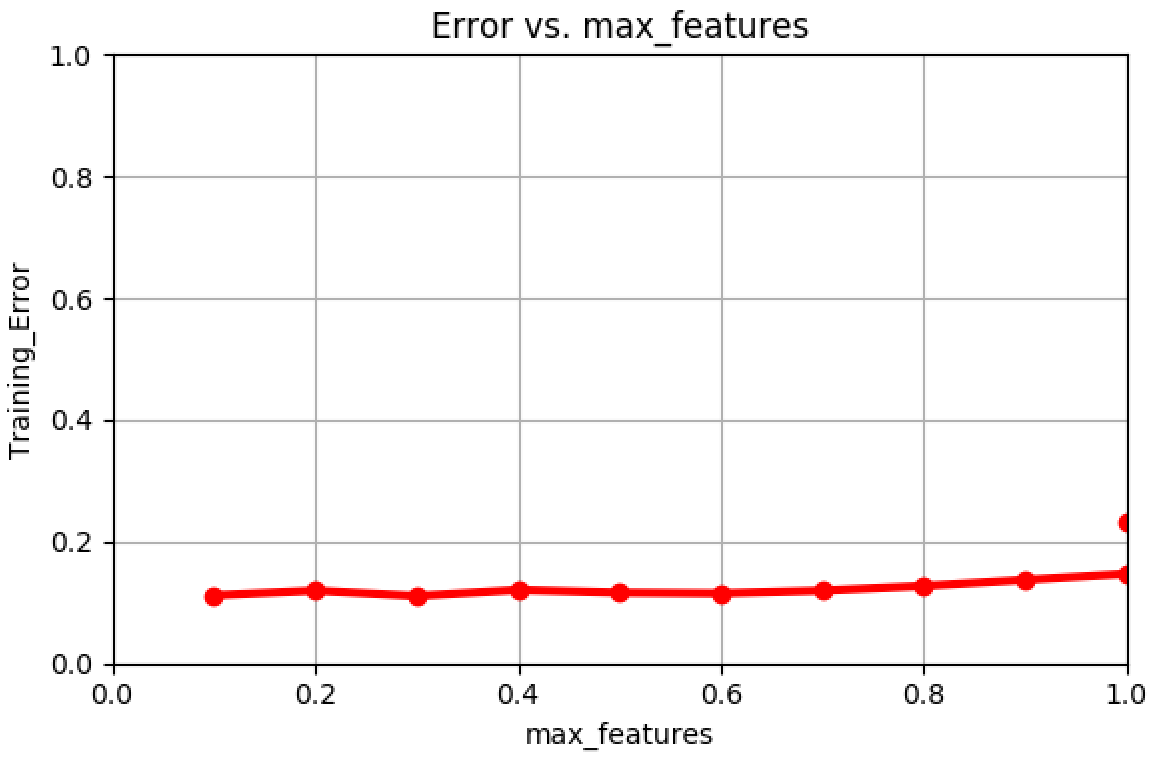


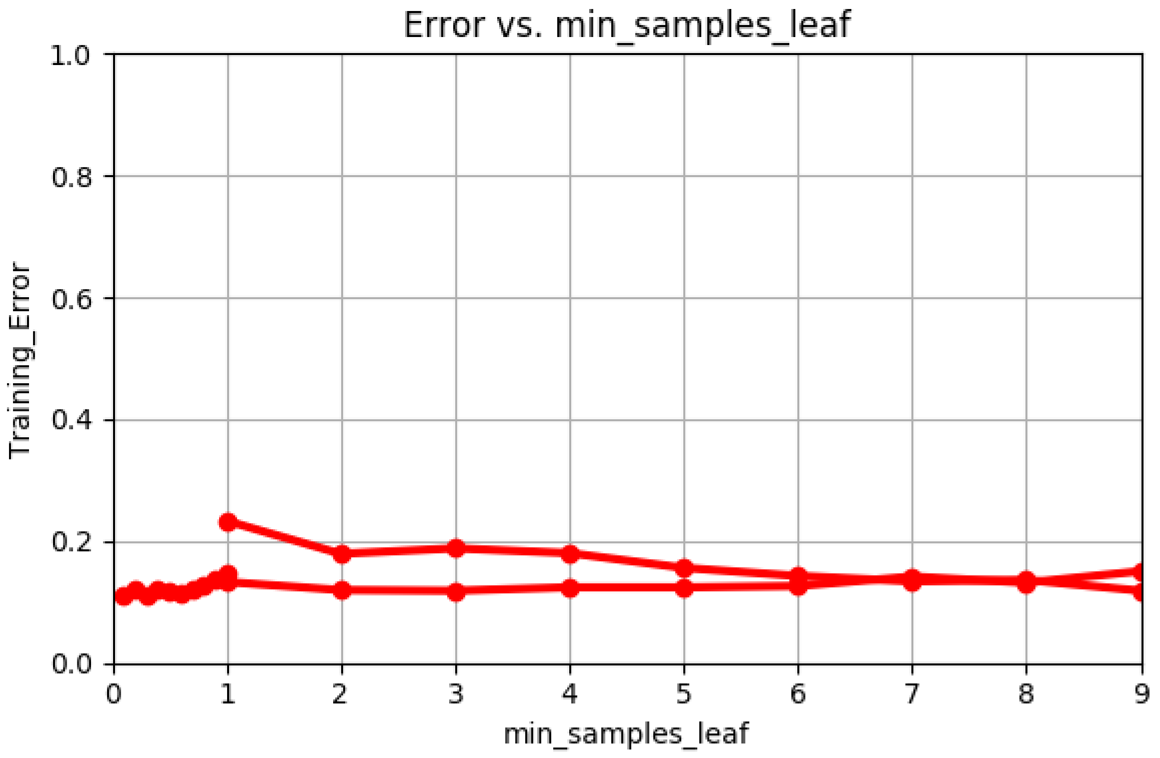
b) In terms of training time, KNN proved to be the fastest, but trailed behind the others when it came to testing, especially for USPS_digits database. On the other hand, decision trees took the least time to test, but took moderate amount of time for training. However, it took the most time to be trained with USPS_digits data. Decision trees, on average, seemed to be most accurate. Hence, it can be said that decision trees have a better speed-accuracy tradeoff compared to the other two classifiers.

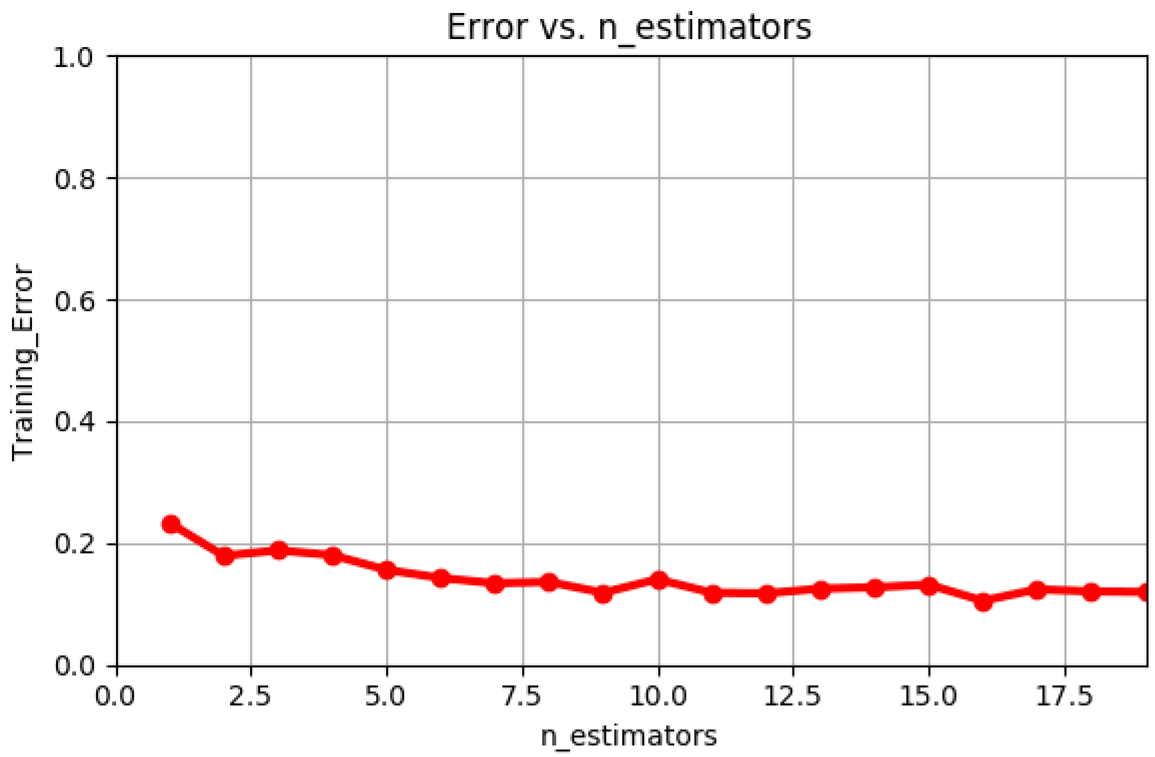
4) a) A crossvalidation-crossvalidation technique is being implemented here to optimize hyperparameters, where number of splits for outer loop can be stated. Assuming the number of splits for outer loop is n, 1/n of input test data is reserved for test to calculate generalization error. The outer loop iterates through the n-1 blocks and passes each block onto desired classifier method. This method contains the inner loop, which performs a split of n-1. 1/(n-1) of data is reserved for validation. The remaining n-2 blocks are iterated through to calculate validation error An outer loop within the classifier method varies hyperparameter value to obtain mean validation error for hyperparameter to be optimized. Based on lowest mean validation error, optimal hyperparameter value is chosen. This is carried out for all hyperparameters to be optimized. All optimal hyperparameter values are used to train the data on n-1 blocks passed into the classifier method. Later, the initially partitioned test data(n-1) is used to calculate test error, which are then in turn used to calculate generalization error outside outer loop of crossvalidation-crossvalidation method.

b) Available datasets are quite small, so in order to avoid overfitting, high bias and high variance, crossvalidation-crossvalidation was used. Crossvalidation-test method was avoided owing to absence of larger data sets and it showed higher generalization error compared to the previous one.

5) a)


Error vs. max_features
Training_Error
max_features
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Error vs. min_samples_leaf
Training_Error
min_samples_leaf
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9



b) Test accuracy obtained from Kaggle is 0.93598. Mean training accuracy obtained is 0.875342465753 for n_estimators=14, max_features=0.1, min_samples_leaf=3.

c) Accuracy obtained using default parameters for Random forest was 0.9107, which is a bit less than test score of 0.93598 when crossvalidation-crossvalidation was used.

6) For email_spam dataset, an attempt of feature selection was made, which involved removing features with variance less than 80%. Mean training accuracy obtained was around 0.92, which is pretty close to accuracy obtained from Kaggle(0.92715). Crossvalidation-Crossvalidation has been implemented for other datasets too. For USPS-digits, training accuracy showed to be 0.919322033898, whereas testing accuracy on Kaggle was 0.9667. On the other hand, training accuracy for Occupancy_detection showed to be 0.991052631579, whereas testing accuracy on Kaggle was 0.99032