Sahib Athwal
A13450589
COGS 118A: Final
Professor Fleischer

Experiment Overview

This section will run through each of the respective supervised machine learning algorithms that include: Decision Tree, Random Forests, and KNN Models. The results will be displayed for each of the respective datasets we are running our algorithms on. The implementation of each of the supervised machine learning techniques will utilize python libraries stated below in our implementation.

Import Libraries

```
# Here are the respective established python libraries
# we will be utilizing in order to make our supervised
# machine learning algorithms.

import scipy.io as sio
from scipy import stats
import numpy as np
import pandas as pd
from sklearn.svm import SVR
from sklearn import tree
from scipy.stats import ttest_ind
from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import cross_val_score
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
%config InlineBackend.figure_format = 'retina'
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Classifier Functions

Heat Map Visualiztion

This is the helper method that allows us to view each of our respective classifiers as a heat map.

```
global count
count = 0
def draw_heatmap(acc, acc_desc, C_list, character):
    global count
    plt.figure(figsize = (2,4))
    ax = sns.heatmap(acc, annot=True, fmt='.3f', yticklabels=C_list,
                     xticklabels=[])
    ax.collections[0].colorbar.set_label("accuracy")
    ax.set(ylabel='$'  + character + '$')
    plt.title(acc_desc + ' w.r.t $' + character + '$')
    sns.set_style("whitegrid", {'axes.grid' : False})
    plt.show()
    count+=1
```

Linear Support Vector Machine Classifier

This is the helper method that allows us to run the SVM algorithm on our dataset. Initially, we used RBF, but we found it was faster to work with linear in our trials as this method took the longest. Inside the method, we optimize the hyper parameters using the grid search that utilizes K 5 Folds method. Then, it proceeds to go on and train and test with the best parameters.

```
def svm_func():
    #SVM binary classification used linear instead of RBF (Faster runtime)
    classifier = svm.SVC(kernel = 'linear')

    # Different C values to try
    C_list     = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 1]
    parameters = {'C': C_list}

    #Perform a grid Search to identify the best C and to perform K 5 Folds
    clf = GridSearchCV(classifier, parameters, return_train_score = 'true',
                       cv=5)
    #Have to fit the classifier with the training data
    clf.fit(X_train_val, Y_train_val)

    #Extract the training and validation accuracies and plot them as heat maps
```

```
    #to visualize the best C parameter
    train_acc = clf.cv_results_['mean_train_score']
    draw_heatmap(train_acc.reshape(-1,1), 'train accuracy', C_list, 'C')

    val_acc = clf.cv_results_['mean_test_score']
    draw_heatmap(val_acc.reshape(-1,1), 'val accuracy', C_list, 'C')

    #Find the optimal C parameter and use that to redefine the classifier
    optimal_classifier = svm.SVC(kernel = 'linear', C = clf.best_params_['C'] )

    for i,j in enumerate(C_list):
        if j == clf.best_params_['C']:
            best_train_acc = train_acc[i]

    #Find test accuracy
    optimal_classifier.fit(X_train_val, Y_train_val)
    test_acc = optimal_classifier.score(X_test, Y_test)
    return test_acc, best_train_acc, clf.best_params_['C']
```

Decision Tree Classifier

This is the helper method that allows us to run the Decision Tree algorithm on our dataset. Inside the method, we optimize the hyper parameters using the grid search that utilizes K 5 Folds method. Then, it proceeds to go on and train and test with the best parameters.

```
def decision_Tree():
    D_list = np.array([1, 2, 3, 4, 5])
    parameters = {'max_depth':D_list}

    classifier_grid = GridSearchCV(DecisionTreeClassifier(criterion="entropy"),
                                   parameters, cv=5, return_train_score=True)

    #Have to fit the classifier with the training data
    classifier_grid.fit(X_train_val, Y_train_val)

    #Show the heatmaps
    draw_heatmap(classifier_grid.cv_results_['mean_train_score'].reshape(5,1),
                 'DT train accuracy', D_list, 'D')
    draw_heatmap(classifier_grid.cv_results_['mean_test_score'].reshape(5,1),
                 'DT val accuracy', D_list, 'D')

    #Train and Test with best parameters
    D_star = classifier_grid.best_params_['max_depth']
    classifier_test = DecisionTreeClassifier(max_depth=D_star,
                                             criterion="entropy")
    classifier_test.fit(X_train_val, Y_train_val)
    Desicion_test_acc = classifier_test.score(X_test,Y_test_val)

    train_acc = classifier_grid.cv_results_['mean_train_score']
```

```
    for i,j in enumerate(D_list):
        if j == D_star:
            best_train_acc = train_acc[i]

    return Desicion_test_acc, best_train_acc, D_star
```

Random Forest Classifier

This is the helper method that allows us to run the Random Forest algorithm on our dataset. Inside the method, we optimize the hyper parameters using the grid search that utilizes K 5 Folds method. Then, it proceeds to go on and train and test with the best parameters.

```
def rand_Forest():
    D_list = np.array([1, 2, 3, 4, 5])
    parameters = {'max_depth':D_list}

    #Tried various parameters on the docs this got me the fastest result using
    #K5 Folds specified
    classifier_grid = GridSearchCV(RandomForestClassifier(criterion="entropy"),
                                   parameters, cv=5, return_train_score=True)
    classifier_grid.fit(X_train_val, Y_train_val)

    #Show Heatmaps
    draw_heatmap(classifier_grid.cv_results_['mean_train_score'].reshape(5,1),
                 'RF train accuracy', D_list, 'K')
    draw_heatmap(classifier_grid.cv_results_['mean_test_score'].reshape(5,1),
                 'RF val accuracy', D_list, 'K')

    #Train and Test with best parameters
    D_star = classifier_grid.best_params_['max_depth']

    #Entropy worked better than the default gini
    classifier_test1 = RandomForestClassifier(max_depth=D_star,
                                              criterion="entropy")
    classifier_test1.fit(X_train_val, Y_train_val)
    randForest_acc = classifier_test1.score(X_test,Y_test_val)


    train_acc = classifier_grid.cv_results_['mean_train_score']
    for i,j in enumerate(D_list):
        if j == D_star:
            best_train_acc = train_acc[i]

    return randForest_acc, best_train_acc, D_star
```

KNN Classifier

This is the helper method that allows us to run the KNN algorithm on our dataset. Inside the method, we optimize the hyper parameters using the grid search that utilizes K 5 Folds method. Then, it proceeds to go on and train and test with the best parameters.

```
def knn_classifier():
    k_list = np.array([1, 2, 3, 4, 5, 6])
    parameters = {'n_neighbors':k_list}
    classifier_grid = GridSearchCV(KNeighborsClassifier(), parameters, cv=5,
                                   return_train_score=True)
    classifier_grid.fit(X_train_val, Y_train_val)

    #Plot heatmaps for the Training and Testing scores respectively
    draw_heatmap(classifier_grid.cv_results_['mean_train_score'].reshape(6,1),
                 'KNN train accuracy', k_list, 'K')
    draw_heatmap(classifier_grid.cv_results_['mean_test_score'].reshape(6,1),
                 'KNN val accuracy', k_list, 'K')

    #Train and Test with best parameters
    k_star = classifier_grid.best_params_['n_neighbors']
    classifier_test2 = KNeighborsClassifier(n_neighbors=k_star)
    classifier_test2.fit(X_train_val,Y_train_val)
    knn_acc = classifier_test2.score(X_test,Y_test_val)

    train_acc = classifier_grid.cv_results_['mean_train_score']
    for i,j in enumerate(k_list):
        if j == k_star:
            best_train_acc = train_acc[i]

    return knn_acc, best_train_acc, k_star
```

Bankruptcy Dataset

This given dataset provides 96 various attributes that give us the necessary information to predict whether a company will go bankrupt or not. The reasoning behind using this dataset is for a basis for a real world scenario, where a company may exhibit similarities to these features and may need to take decisive action if they are going to become bankrupt.

Cleaning Bankruptcy Dataset

```
bankrupcy_data_preserved = pd.read_csv('Bankrupcy.csv')
bankrupcy_data = bankrupcy_data_preserved #Temp for manipulation
bankrupcy_data #preview of dataset
```

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973
...	...	...	...	...	...	...	...
6814	0	0.493687	0.539468	0.543230	0.604455	0.604462	0.998992
6815	0	0.475162	0.538269	0.524172	0.598308	0.598308	0.998992
6816	0	0.472725	0.533744	0.520638	0.610444	0.610213	0.998984
6817	0	0.506264	0.559911	0.554045	0.607850	0.607850	0.999074
6818	0	0.493053	0.570105	0.549548	0.627409	0.627409	0.998080

6819 rows × 96 columns

```
bankrupcy_data.dropna(inplace=True)
bankrupcy_data.shape
bankrupcy_data.head()

#Dropped some of the columns deemed as insignificant in comparison
#to those that remain. (Based off of Research of Important Aspects of Companies)

bankrupcy_data = bankrupcy_data.iloc[:,0:20]
bankrupcy_data = bankrupcy_data.drop(labels=[" ROA(C) before interest and depreciation before
bankrupcy_data = bankrupcy_data.drop(labels=[" Pre-tax net Interest Rate"],axis=1)
bankrupcy_data = bankrupcy_data.drop(labels=[" After-tax net Interest Rate"],axis=1)
bankrupcy_data = bankrupcy_data.drop(labels=[" Non-industry income and expenditure/revenue"],
bankrupcy_data = bankrupcy_data.drop(labels=[" Continuous interest rate (after tax)"],axis=1)
bankrupcy_data = bankrupcy_data.drop(labels=[" Interest-bearing debt interest rate"],axis=1)
bankrupcy_data = bankrupcy_data.drop(labels=[" Persistent EPS in the Last Four Seasons"],axis

#Checking what data types we have in the dataset
print(bankrupcy_data.dtypes)
bankrupcy_data.head()#Data preview
#Convert Data to a numpy Array
bankrupcy_data = bankrupcy_data.sample(n=5000).reset_index(drop=True)
```

```
print(bankrupcy_data[0:5])#Preview
bankrupcy_data
```

```
Bankrupt?                                                  int64
 ROA(A) before interest and % after tax                  float64
 ROA(B) before interest and depreciation after tax       float64
 Operating Gross Margin                                  float64
 Realized Sales Gross Margin                             float64
 Operating Profit Rate                                   float64
 Operating Expense Rate                                  float64
 Research and development expense rate                   float64
 Cash flow rate                                          float64
 Tax rate (A)                                            float64
 Net Value Per Share (B)                                 float64
 Net Value Per Share (A)                                 float64
 Net Value Per Share (C)                                 float64
dtype: object
   Bankrupt?  ...   Net Value Per Share (C)
0          0  ...                  0.168387
1          0  ...                  0.190510
2          0  ...                  0.179386
3          0  ...                  0.194176
4          0  ...                  0.185664

[5 rows x 13 columns]
```

	Bankrupt?	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Operating Expense Rate
0	0	0.428314	0.427967	0.607006	0.607006	0.999048	1.358729e-04
1	0	0.584714	0.580491	0.609255	0.608988	0.999015	2.008606e-04
2	0	0.594581	0.574656	0.611749	0.611813	0.999121	1.398045e-04
3	0	0.543393	0.556400	0.595526	0.595526	0.998971	2.860000e+09
4	0	0.534562	0.517105	0.601118	0.601118	0.998997	9.240000e+08
...	...	...	...	...	...	...	...
4995	0	0.524913	0.522619	0.623546	0.623546	0.998944	5.055628e-04
4996	0	0.624291	0.621232	0.615309	0.615309	0.999199	1.179492e-04
4997	0	0.561328	0.556454	0.604016	0.603965	0.999030	1.054605e-04
4998	1	0.400403	0.398094	0.597025	0.597025	0.998804	2.197093e-04
4999	1	0.468437	0.473473	0.587303	0.587303	0.998730	1.378387e-04

5000 rows × 13 columns

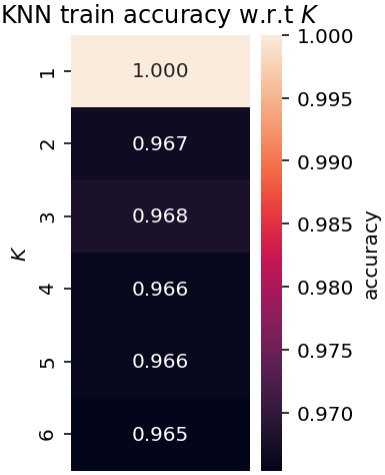
Running Classifiers For Bankrupcy Data

```
#Split Data By 80/20, 50/50, 20/80
partitionVal = [0.8, 0.5, 0.2]
result_table = np.zeros((3,7))
result_table1 = np.zeros((3,7))
result_table2 = np.zeros((3,7))
for i, partition in enumerate(partitionVal):
    print("Partition: ", partition)

    knn_test_acc = []
    rand_forest_test_acc = []
    decision_tree_test_acc = []
    svm_test_acc = []



    NUM_TRIALS = 5
    for trial in range(NUM_TRIALS):
        #Mix up the data
        bankrupcy_data = bankrupcy_data.sample(frac=1).reset_index(drop=True)
        #Find the point where to split the data
        breakNum = int(partition*len(bankrupcy_data))

        X_train_full = bankrupcy_data.loc[0:breakNum]
        X_train_val = X_train_full.drop("Bankrupt?", axis = 1)
        Y_train_val = X_train_full["Bankrupt?"]
        X_test_full = bankrupcy_data.loc[breakNum:]
        X_test= X_test_full.drop("Bankrupt?", axis = 1)
        Y_test_val = X_test_full["Bankrupt?"]

        #Call the svm classifier (Took Too Long For ALL Datasets)
        #test_acc,best_train0,C0 = svm_func()
        #svm_test_acc.append(test_acc)

        #Call the knn classifier
        test_acc,best_train1,C1 = knn_classifier()
        knn_test_acc.append(test_acc)

        #Call the Decision Tree classifier
        test_acc,best_train2,C2 = decision_Tree()
        decision_tree_test_acc.append(test_acc)

        #Call the Random Forest classifier
        test_acc,best_train3,C3 = rand_Forest()
        rand_forest_test_acc.append(test_acc)


    #result_table[i, 0] = sum(svm_test_acc)/NUM_TRIALS
```

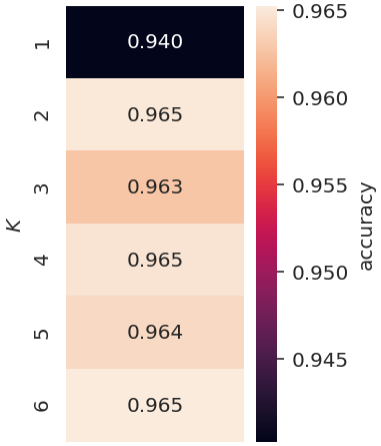
```
result_table[i, 1] = sum(knn_test_acc)/NUM_TRIALS
result_table[i, 2] = sum(decision_tree_test_acc)/NUM_TRIALS
result_table[i, 3] = sum(rand_forest_test_acc)/NUM_TRIALS


#result_table1[i, 0] = best_train0
result_table1[i, 1] = best_train1
result_table1[i, 2] = best_train2
result_table1[i, 3] = best_train3


#result_table2[i, 0] = C0
result_table2[i, 1] = C1
result_table2[i, 2] = C2
result_table2[i, 3] = C3


#Average all test accuracies for all 5 trials
print("Test Accuracy Average for knn = ", sum(knn_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Random Forest = ",
      sum(rand_forest_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Decision Tree = ",
      sum(decision_tree_test_acc)/NUM_TRIALS)

#print("Test Accuracy Average for SVM = ", sum(svm_test_acc)/NUM_TRIALS)

#y-axis: partition
#x-axis: classifier
print(result_table)
print("############################")
print(result_table1)
print("############################")
print(result_table2)
```

```
Partition:  0.8
```

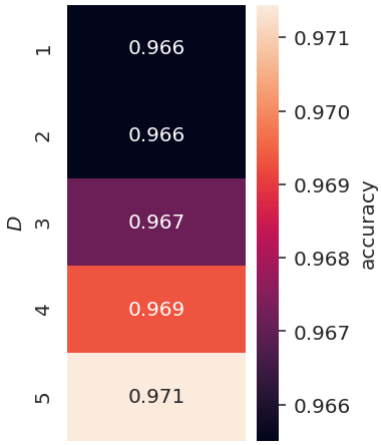
KNN train accuracy w.r.t *K*



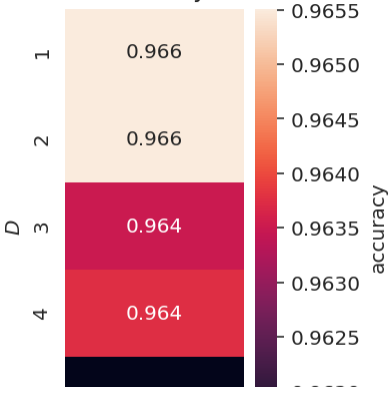
KNN val accuracy w.r.t *K*



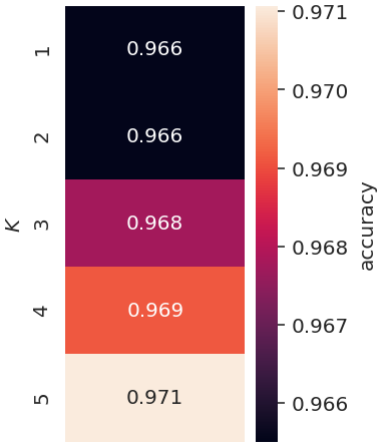
DT train accuracy w.r.t *D*



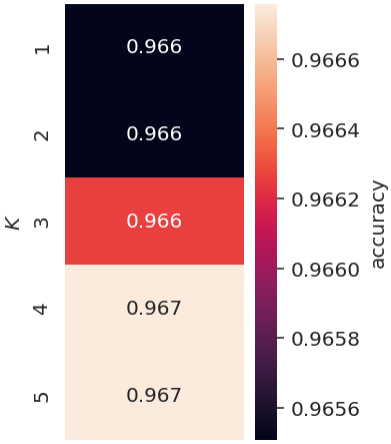
DT val accuracy w.r.t *D*



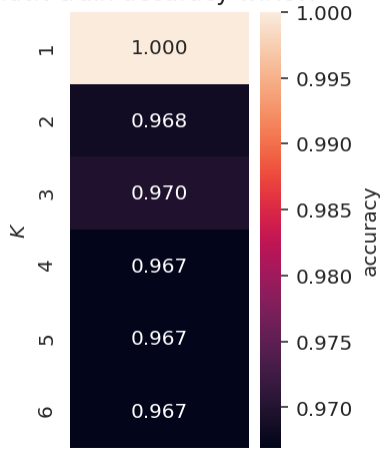
RF train accuracy w.r.t *K*



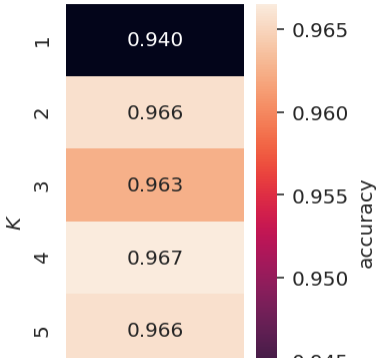
RF val accuracy w.r.t *K*

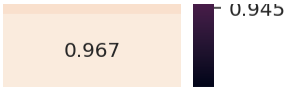


KNN train accuracy w.r.t *K*

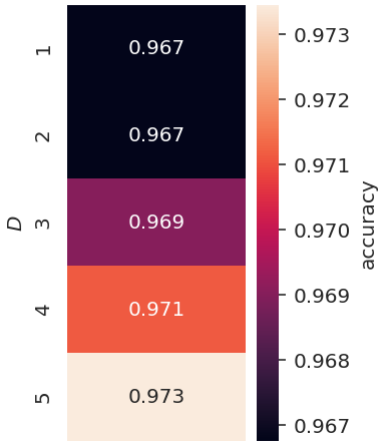


KNN val accuracy w.r.t *K*

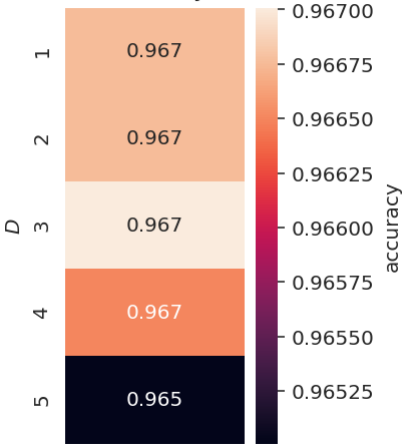




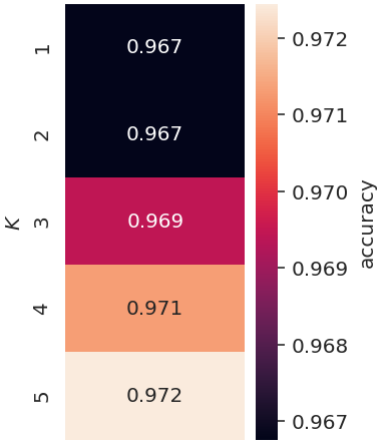
DT train accuracy w.r.t *D*



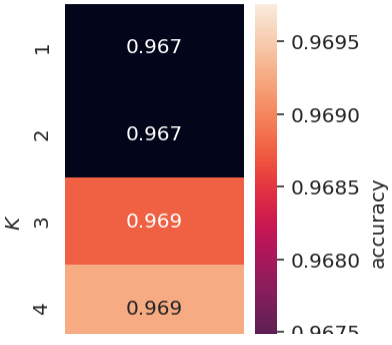
DT val accuracy w.r.t *D*



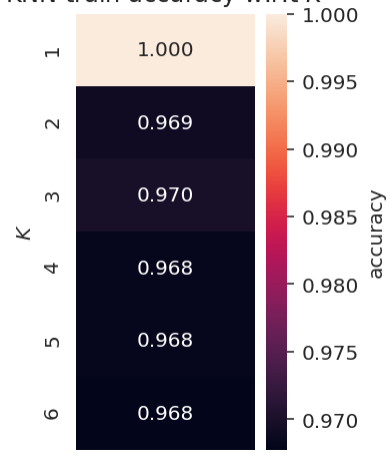
RF train accuracy w.r.t *K*



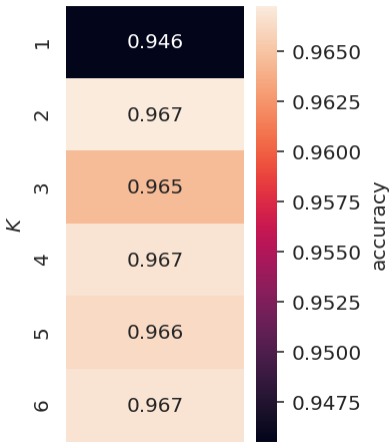
RF val accuracy w.r.t *K*



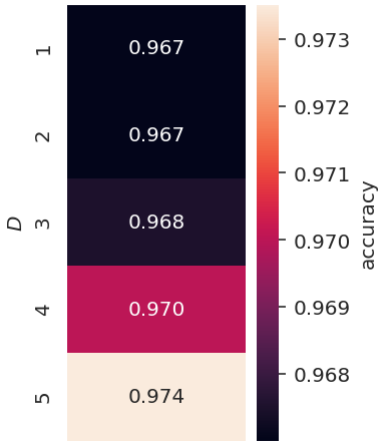
KNN train accuracy w.r.t *K*



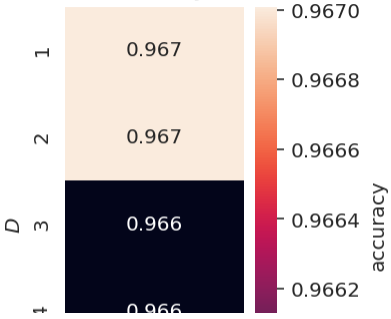
KNN val accuracy w.r.t *K*

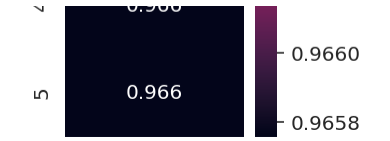


DT train accuracy w.r.t *D*

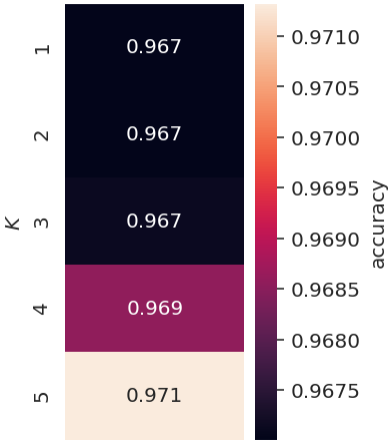


DT val accuracy w.r.t *D*

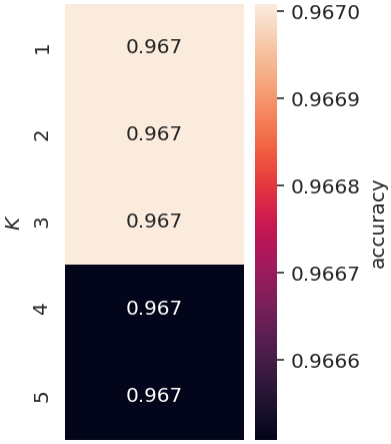




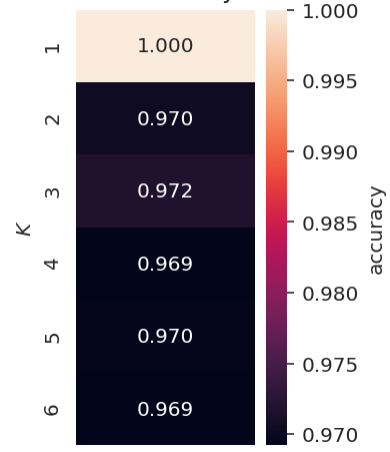
RF train accuracy w.r.t *K*



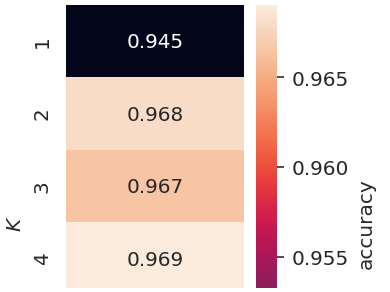
RF val accuracy w.r.t *K*

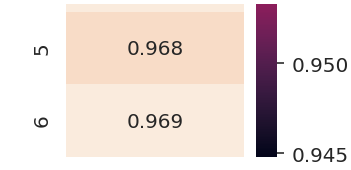


KNN train accuracy w.r.t *K*

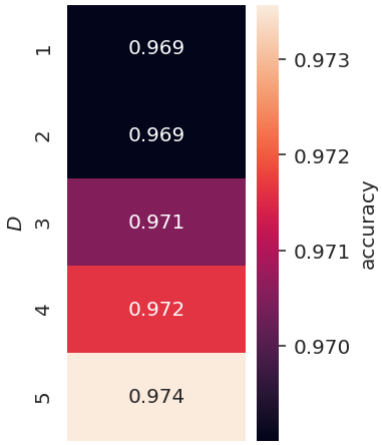


KNN val accuracy w.r.t *K*

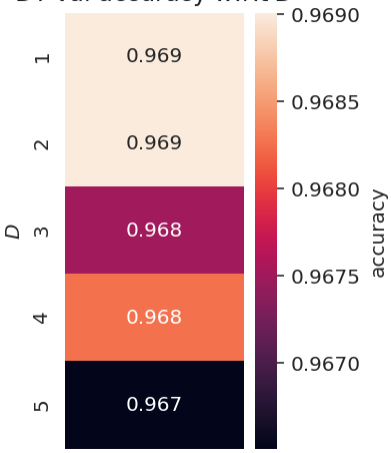




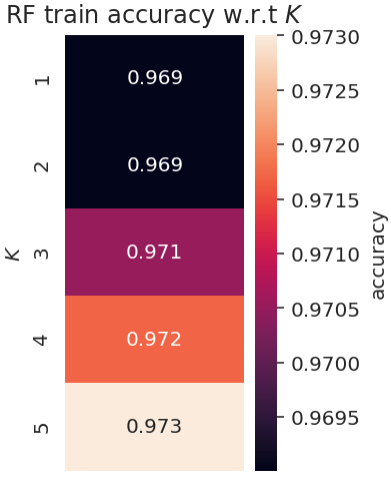
DT train accuracy w.r.t *D*



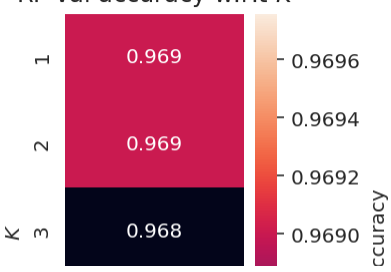
DT val accuracy w.r.t *D*

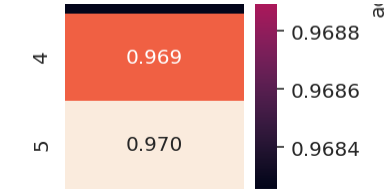


RF train accuracy w.r.t *K*

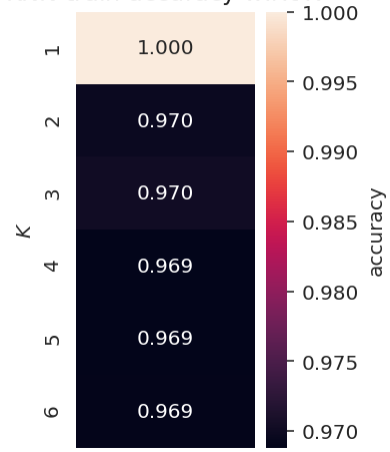


RF val accuracy w.r.t *K*

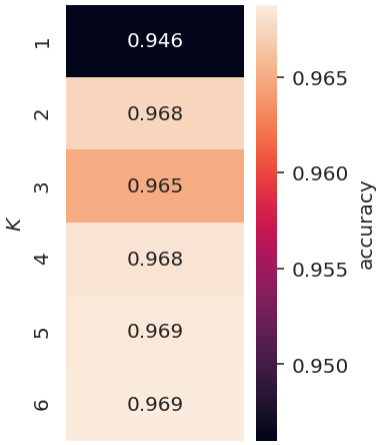




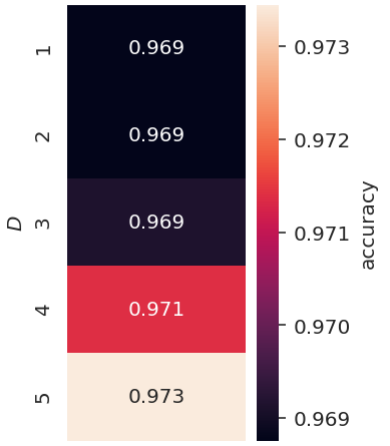
KNN train accuracy w.r.t *K*



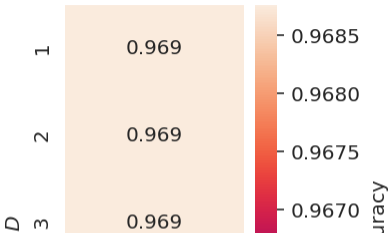
KNN val accuracy w.r.t *K*

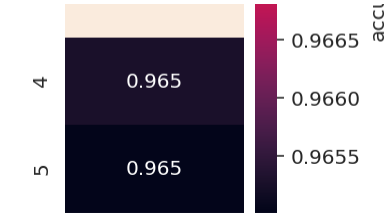


DT train accuracy w.r.t *D*

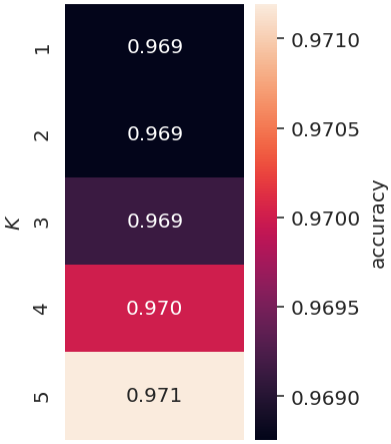


DT val accuracy w.r.t *D*

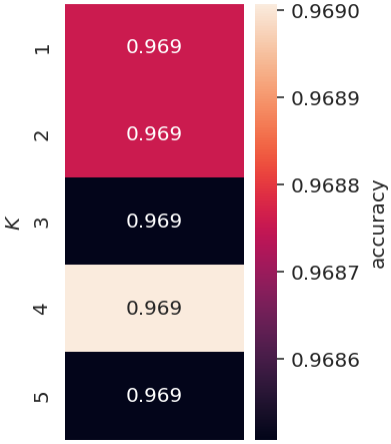




RF train accuracy w.r.t *K*

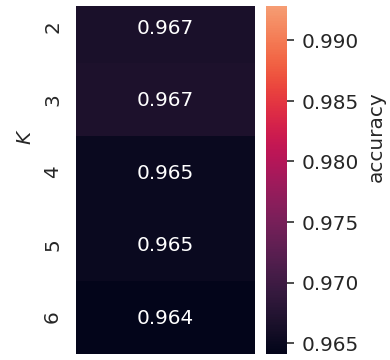


RF val accuracy w.r.t *K*

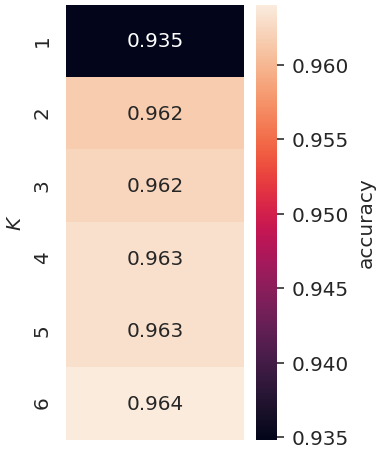



```
Test Accuracy Average for knn =  0.9658
Test Accuracy Average for Random Forest =  0.9663999999999999
Test Accuracy Average for Decision Tree =  0.9635999999999999
[[0.     0.9658 0.9636 0.9664 0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.    ]]
##############################
[[0.         0.96894525 0.96913271 0.9700075  0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 3. 4. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.5
```

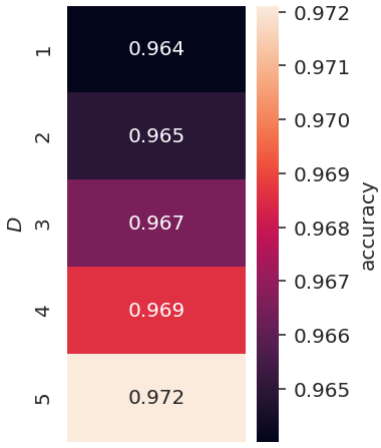
KNN train accuracy w.r.t *K*



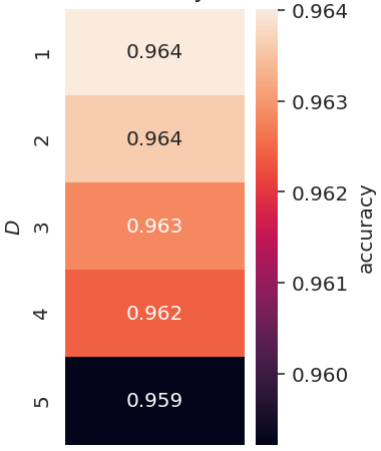
KNN val accuracy w.r.t *K*



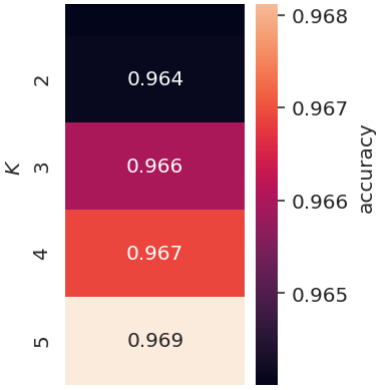
DT train accuracy w.r.t *D*



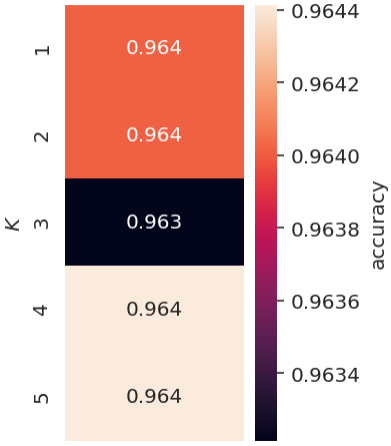
DT val accuracy w.r.t *D*



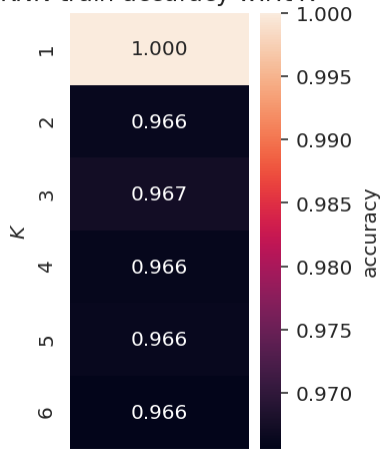
RF train accuracy w.r.t *K*



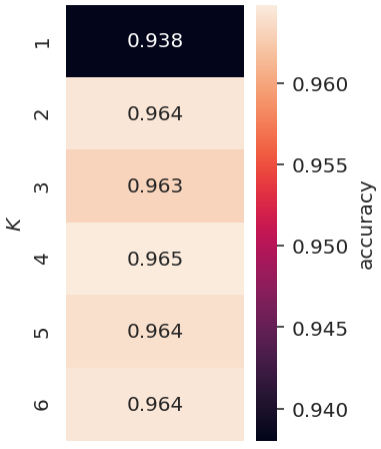
RF val accuracy w.r.t *K*



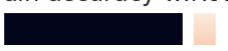
KNN train accuracy w.r.t *K*

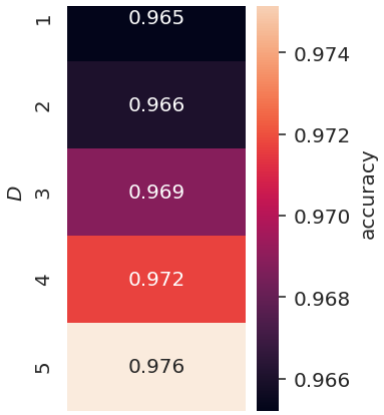


KNN val accuracy w.r.t *K*

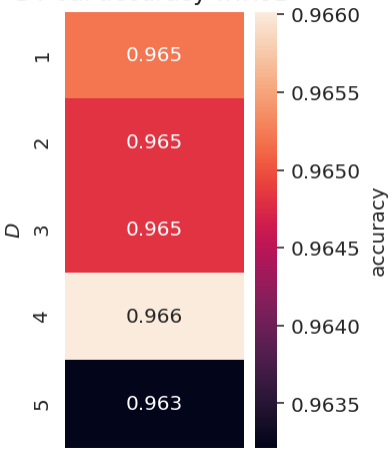


DT train accuracy w.r.t *D*

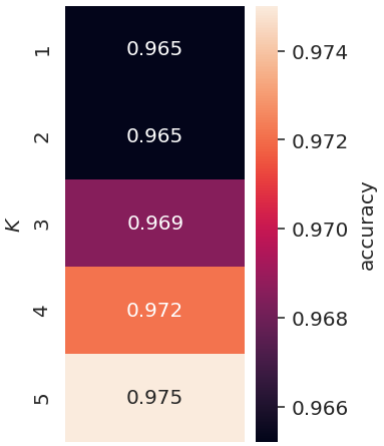




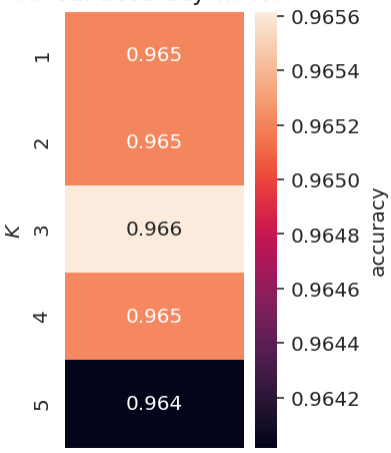
DT val accuracy w.r.t *D*



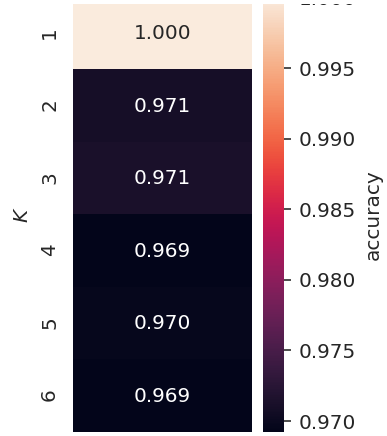
RF train accuracy w.r.t *K*



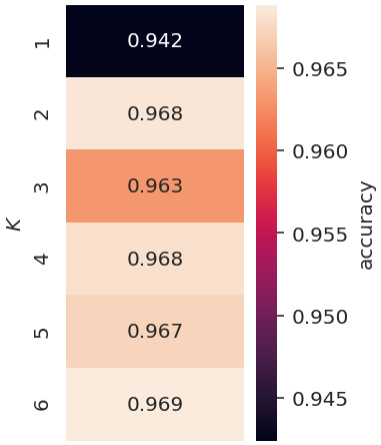
RF val accuracy w.r.t *K*



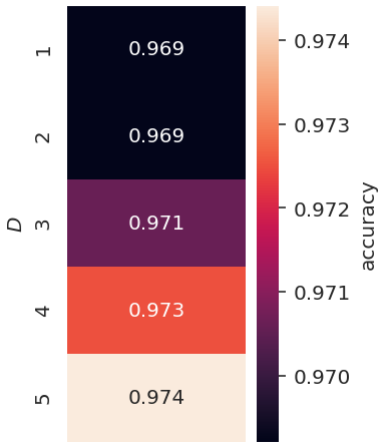
KNN train accuracy w.r.t *K*



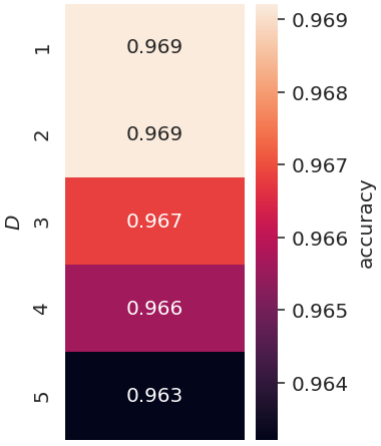
KNN val accuracy w.r.t *K*



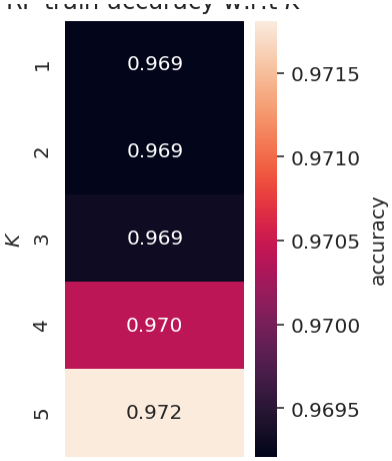
DT train accuracy w.r.t *D*



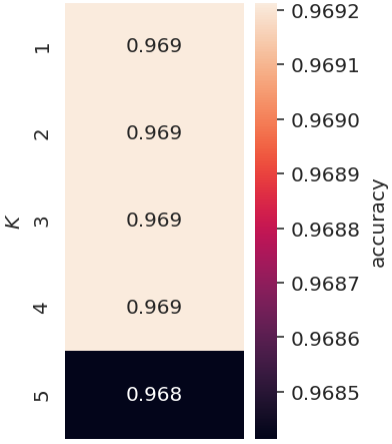
DT val accuracy w.r.t *D*



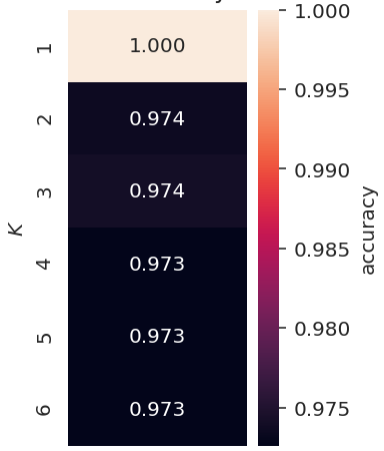
RF train accuracy w.r.t *K*



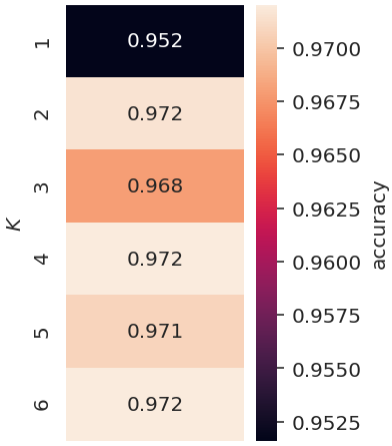
RF val accuracy w.r.t *K*



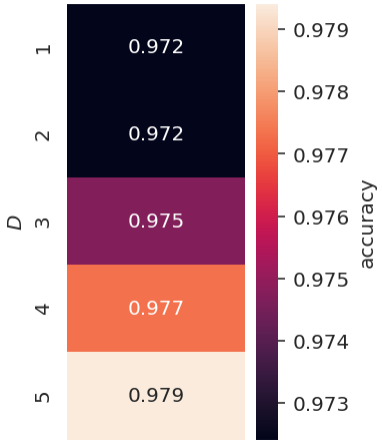
KNN train accuracy w.r.t *K*



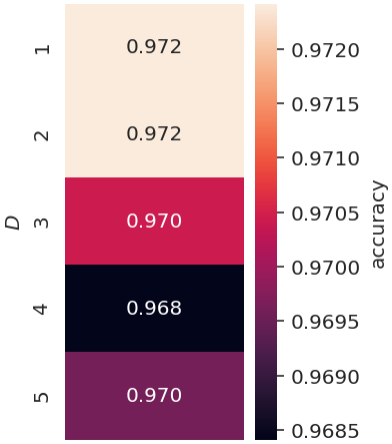
KNN val accuracy w.r.t *K*



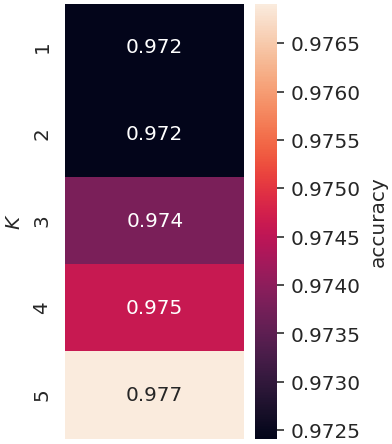
DT train accuracy w.r.t *D*



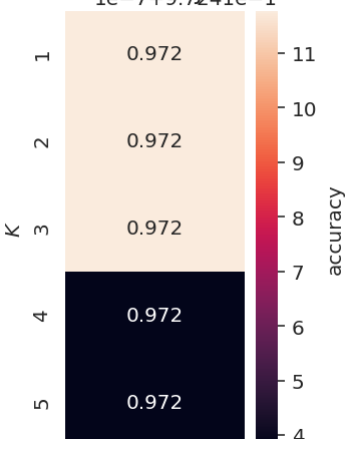
DT val accuracy w.r.t *D*



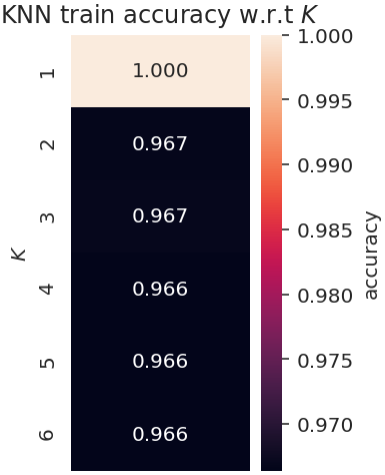
RF train accuracy w.r.t *K*



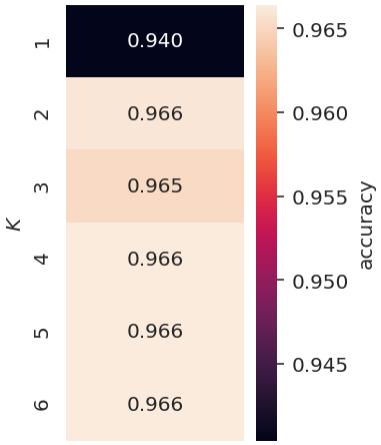
RF val accuracy w.r.t *K*



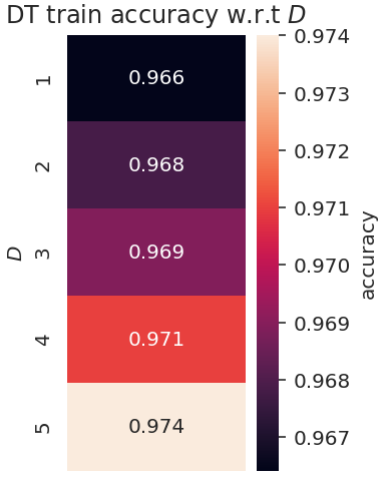
KNN train accuracy w.r.t *K*



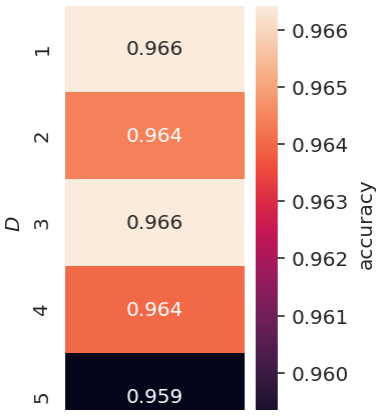
KNN val accuracy w.r.t *K*



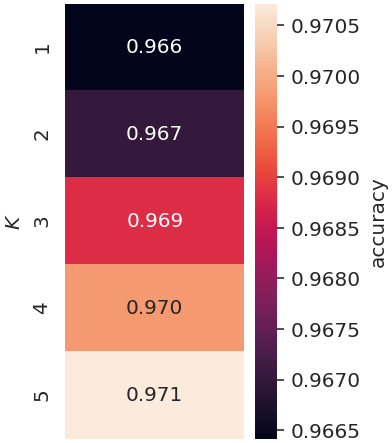
DT train accuracy w.r.t *D*



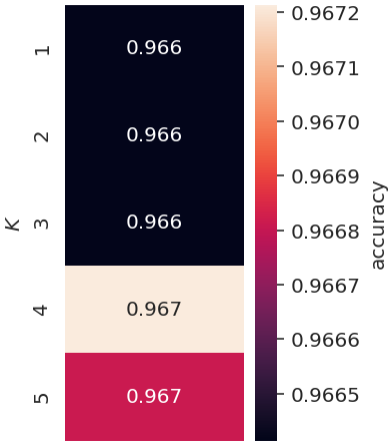
DT val accuracy w.r.t *D*



RF train accuracy w.r.t *K*

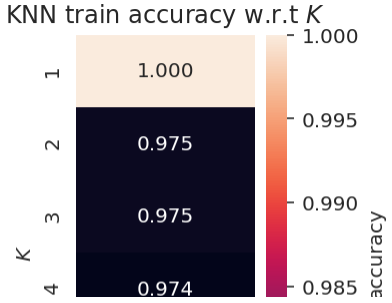


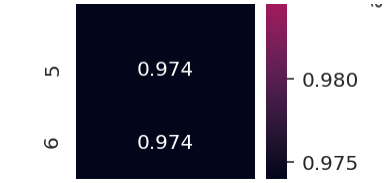
RF val accuracy w.r.t *K*



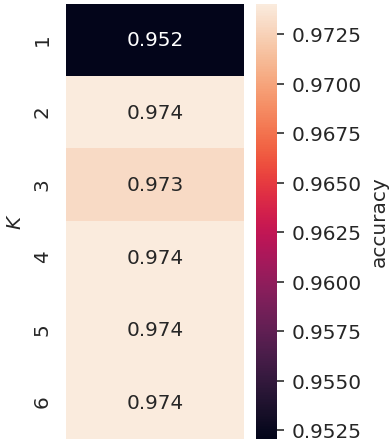

```
Test Accuracy Average for knn =  0.9666399999999999
Test Accuracy Average for Random Forest =  0.9671199999999999
Test Accuracy Average for Decision Tree =  0.96624
[[0.      0.9658  0.9636  0.9664  0.      0.      0.     ]
 [0.      0.96664 0.96624 0.96712 0.      0.      0.     ]
 [0.      0.      0.      0.      0.      0.      0.     ]]
##############################
[[0.         0.96894525 0.96913271 0.9700075  0.         0.
  0.        ]
 [0.         0.96641344 0.96641344 0.96981189 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 3. 4. 0. 0. 0.]
 [0. 4. 1. 4. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.2
```

KNN train accuracy w.r.t *K*

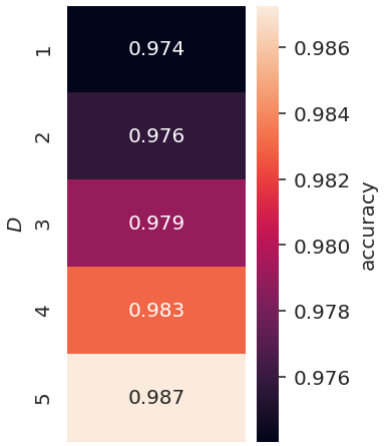




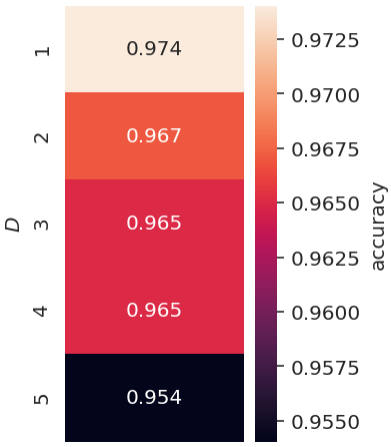
KNN val accuracy w.r.t *K*



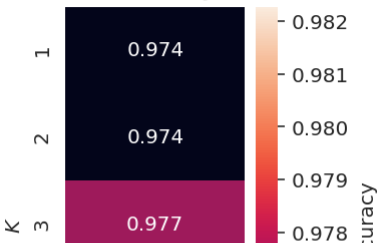
DT train accuracy w.r.t *D*

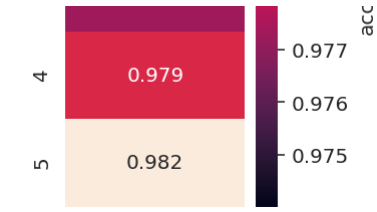


DT val accuracy w.r.t *D*

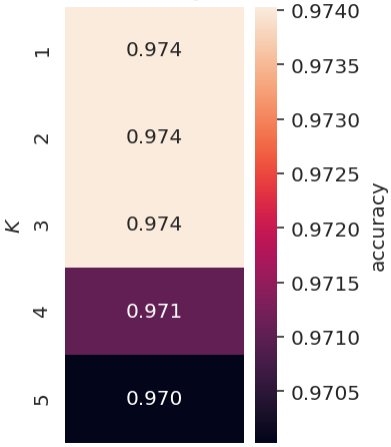


RF train accuracy w.r.t *K*

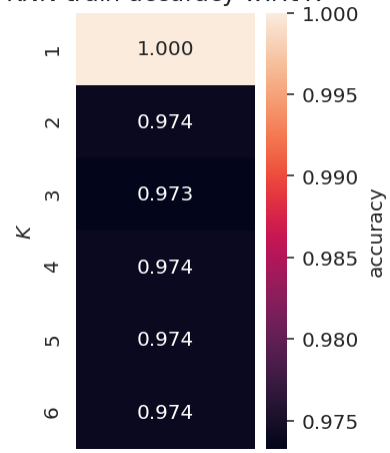




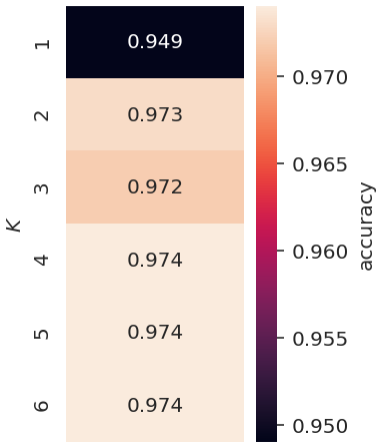
RF val accuracy w.r.t *K*



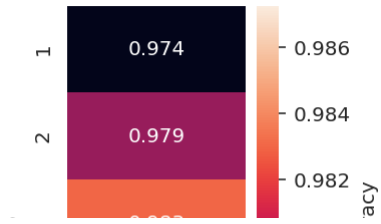
KNN train accuracy w.r.t *K*

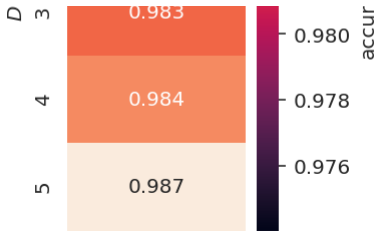


KNN val accuracy w.r.t *K*

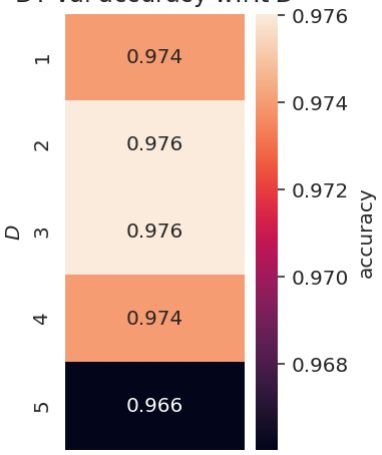


DT train accuracy w.r.t *D*

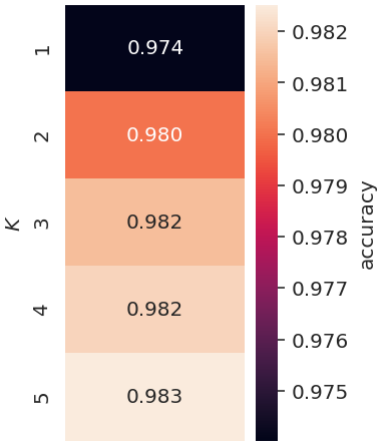




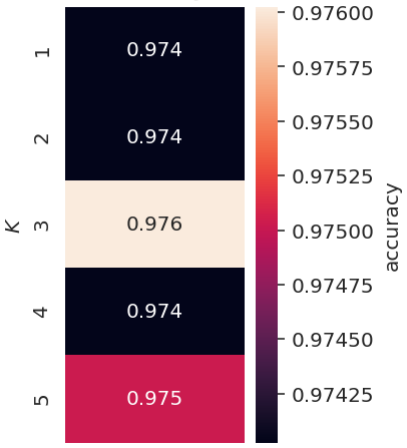
DT val accuracy w.r.t *D*



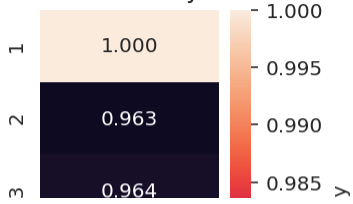
RF train accuracy w.r.t *K*

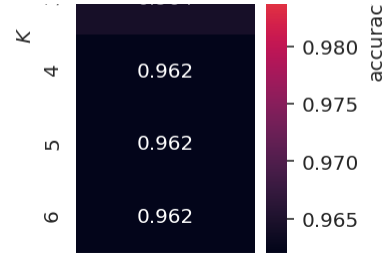


RF val accuracy w.r.t *K*

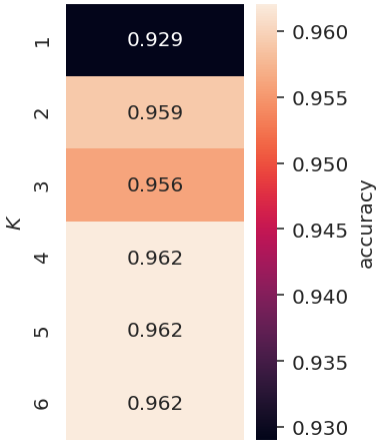


KNN train accuracy w.r.t *K*

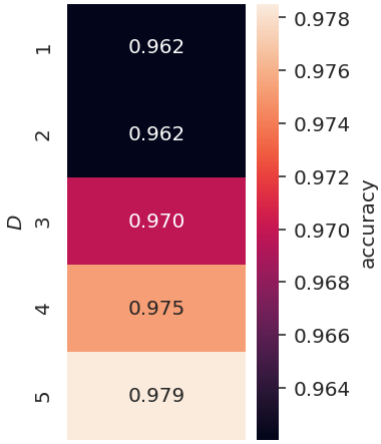




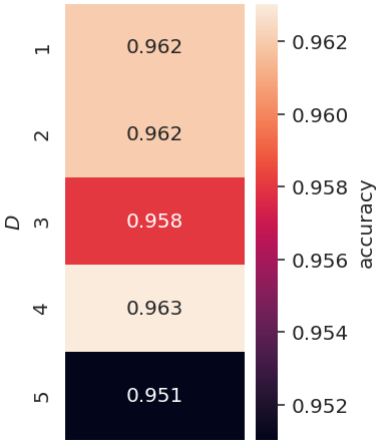
KNN val accuracy w.r.t *K*



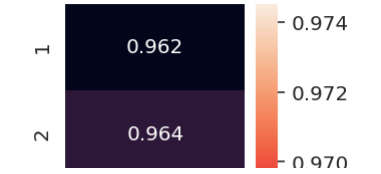
DT train accuracy w.r.t *D*

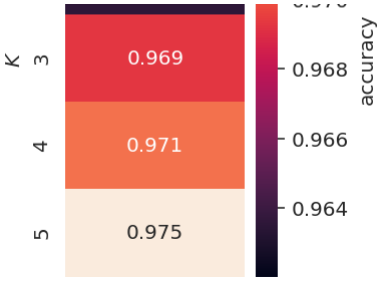


DT val accuracy w.r.t *D*

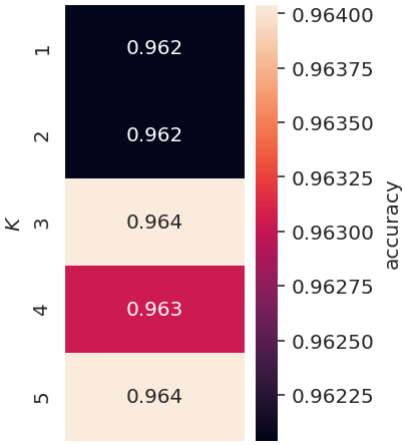


RF train accuracy w.r.t *K*

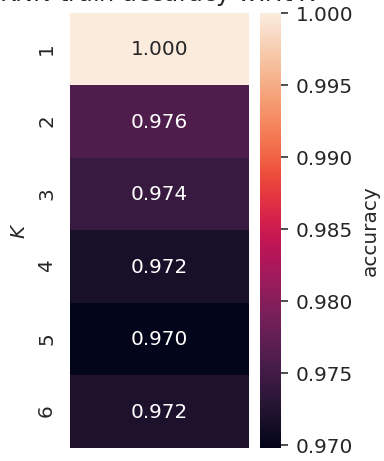




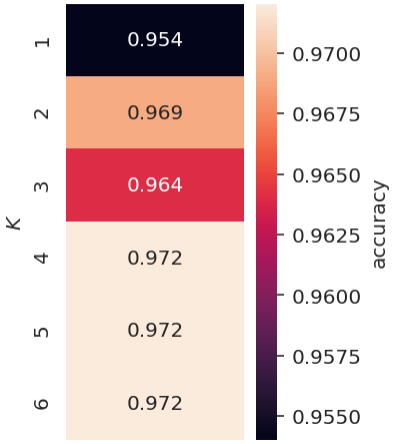
RF val accuracy w.r.t *K*



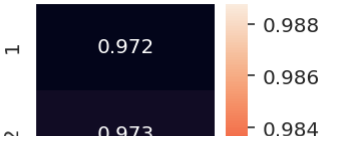
KNN train accuracy w.r.t *K*

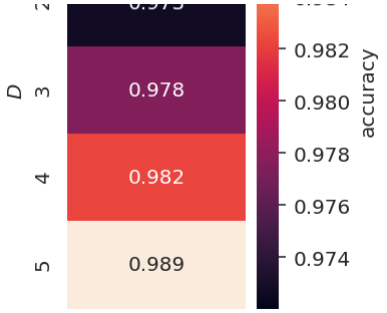


KNN val accuracy w.r.t *K*

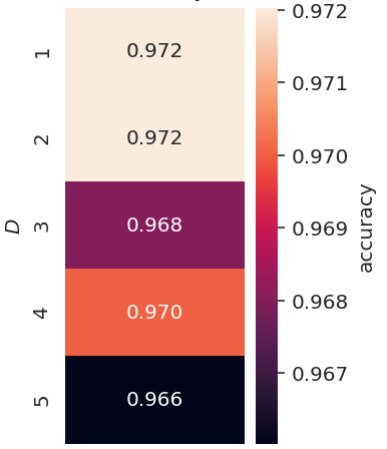


DT train accuracy w.r.t *D*

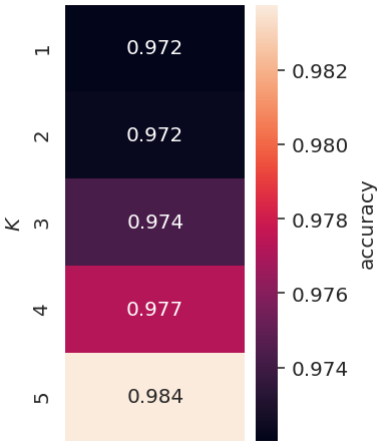




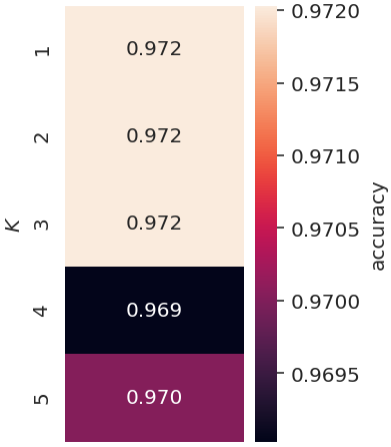
DT val accuracy w.r.t *D*



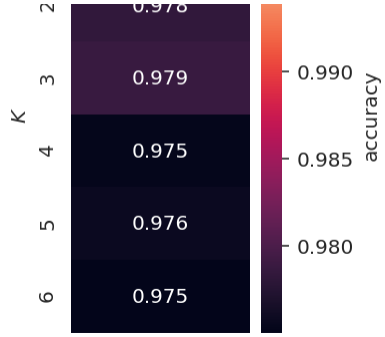
RF train accuracy w.r.t *K*



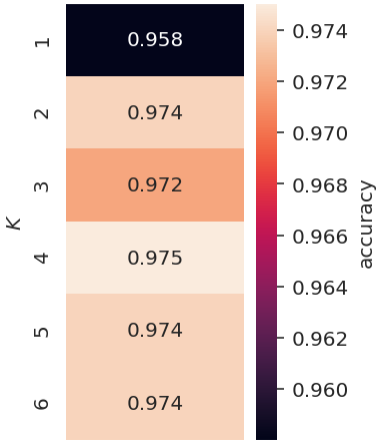
RF val accuracy w.r.t *K*



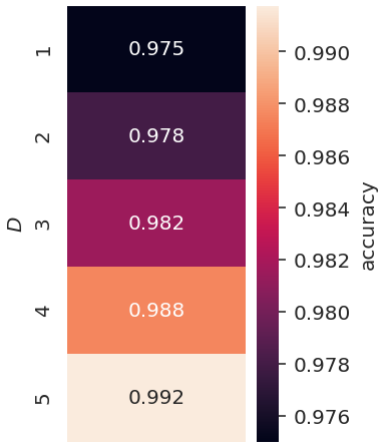
KNN train accuracy w.r.t *K*



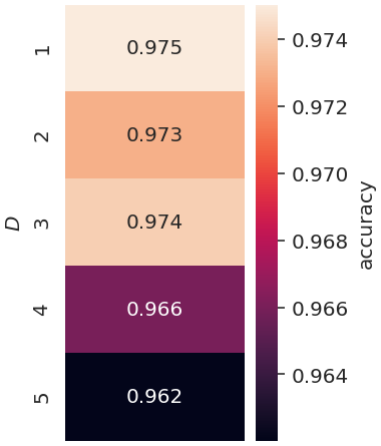
KNN val accuracy w.r.t *K*



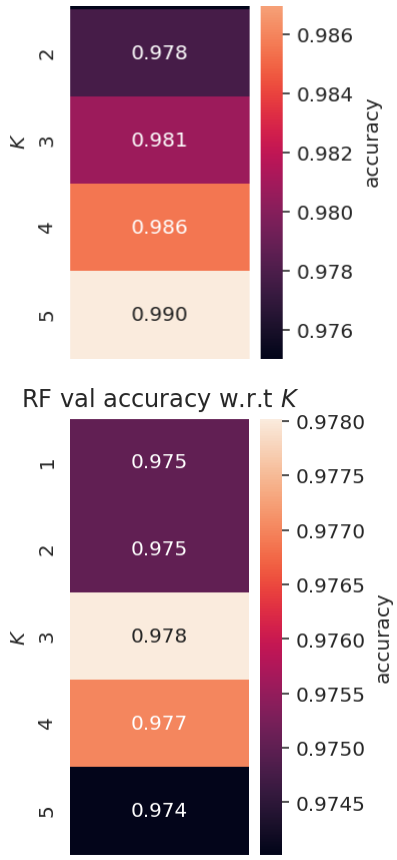
DT train accuracy w.r.t *D*



DT val accuracy w.r.t *D*



RF train accuracy w.r.t *K*



RF val accuracy w.r.t *K*


```
Test Accuracy Average for knn =  0.9656499999999999
Test Accuracy Average for Random Forest =  0.9663499999999999
```

Running Respective P and T Tests to compare the Trials Accuracy For Each Algorithm

```
[0.      0.96664 0.96624 0.96712 0.      0.      0.     ]
```

```
#Compute the difference between the results
print("P and T Test for KNN trials vs. Decision Tree")
diff = [y - x for y, x in zip(knn_test_acc, decision_tree_test_acc)]

#Compute the mean of differences
d_bar = np.mean(diff)

#Compute the variance of differences
sigma2 = np.var(diff)

#compute the number of data points used for training
n1 = len(Y_train_val)

#compute the number of data points used for testing
n2 = len(Y_test_val)

#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)
```

```
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for KNN trials vs. Decision Tree
This is the T Value
0.17892017865612797
This is the P Value
0.42900381442511615
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. Decision Tree")
diff = [y - x for y, x in zip(rand_forest_test_acc, decision_tree_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
```

```
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. Decision Tree
This is the T Value
0.27371020318139705
This is the P Value
0.39215932201463644
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. KNN")
diff = [y - x for y, x in zip(rand_forest_test_acc, knn_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. KNN
This is the T Value
0.9530316657559331
This is the P Value
0.1703100524922947
Accept the null hypothesis
```

Students Performance Dataset

This given dataset provides 8 various attributes that give us the necessary information to predict how a studnet might score on in math, reading, and writing. The reasoning behind using this dataset is for a basis for a real world scenario, where a schools want to see how students are doing based on their background

```
students_data_preserved = pd.read_csv('StudentsPerformance.csv')
students_data = students_data_preserved #Temp for manipulation
students_data #preview of dataset

#Basic Cleanup
students_data.dropna(inplace=True)
students_data.shape
students_data.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93

```
class MultiColumnLabelEncoder:
    def __init__(self,columns = None):
        self.columns = columns # array of column names to encode

    def fit(self,X,y=None):
        return self # not relevant here

    def transform(self,X):
        '''
        Transforms columns of X specified in self.columns using
        LabelEncoder(). If no columns specified, transforms all
        columns in X.
        '''
        output = X.copy()
        if self.columns is not None:
            for col in self.columns:
                output[col] = LabelEncoder().fit_transform(output[col])
        else:
            for colname,col in output.iteritems():
                output[colname] = LabelEncoder().fit_transform(col)
        return output

    def fit_transform(self,X,y=None):
```

```
        return self.fit(X,y).transform(X)


# Encoded each of the categorical features so it will be good for our testing
students_data = MultiColumnLabelEncoder(columns = ['gender','race/ethnicity',
                                    'parental level of education',
                                    'lunch',
                                    'test preparation course']).fit_transform(students_data)


#Checking what data types we have in the dataset
print(students_data.dtypes)
```

```
gender                         int64
race/ethnicity                 int64
parental level of education    int64
lunch                          int64
test preparation course        int64
math score                     int64
reading score                  int64
writing score                  int64
dtype: object
```

```
#Convert Data to a numpy Array
students_data = students_data.sample(n=1000).reset_index(drop=True)
students_data.head()#Data preview
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	1	1	5	1	1	68	54	53
1	0	3	0	1	1	71	71	74
2	0	2	2	0	1	36	53	43
3	1	2	2	1	1	90	75	69

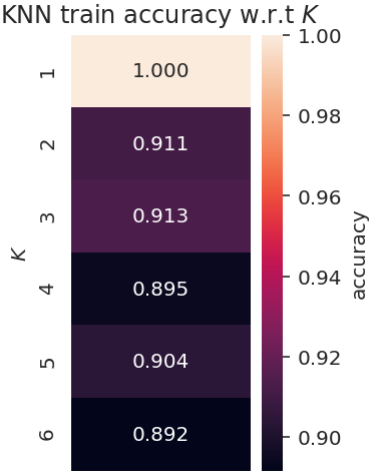
```
#Split Data By 80/20, 50/50, 20/80
partitionVal = [0.8, 0.5, 0.2]
result_table = np.zeros((3,7))
result_table1 = np.zeros((3,7))
result_table2 = np.zeros((3,7))
for i, partition in enumerate(partitionVal):
    print("Partition: ", partition)

    knn_test_acc = []
    rand_forest_test_acc = []
    decision_tree_test_acc = []
    svm_test_acc = []
```

```
NUM_TRIALS = 5
for trial in range(NUM_TRIALS):
    #Mix up the data
    students_data = students_data.sample(frac=1).reset_index(drop=True)
    #Find the point where to split the data
    breakNum = int(partition*len(students_data))

    X_train_full = students_data.loc[0:breakNum]
    X_train_val = X_train_full.drop("gender", axis = 1)
    Y_train_val = X_train_full["gender"]
    X_test_full = students_data.loc[breakNum:]
    X_test= X_test_full.drop("gender", axis = 1)
    Y_test_val = X_test_full["gender"]

    #Call the svm classifier (Took Too Long For ALL Datasets)
    #test_acc,best_train0,C0 = svm_func()
    #svm_test_acc.append(test_acc)

    #Call the knn classifier
    test_acc,best_train1,C1 = knn_classifier()
    knn_test_acc.append(test_acc)

    #Call the Decision Tree classifier
    test_acc,best_train2,C2 = decision_Tree()
    decision_tree_test_acc.append(test_acc)

    #Call the Random Forest classifier
    test_acc,best_train3,C3 = rand_Forest()
    rand_forest_test_acc.append(test_acc)


#result_table[i, 0] = sum(svm_test_acc)/NUM_TRIALS
result_table[i, 1] = sum(knn_test_acc)/NUM_TRIALS
result_table[i, 2] = sum(decision_tree_test_acc)/NUM_TRIALS
result_table[i, 3] = sum(rand_forest_test_acc)/NUM_TRIALS


#result_table1[i, 0] = best_train0
result_table1[i, 1] = best_train1
result_table1[i, 2] = best_train2
result_table1[i, 3] = best_train3


#result_table2[i, 0] = C0
result_table2[i, 1] = C1
result_table2[i, 2] = C2
result_table2[i, 3] = C3


#Average all test accuracies for all 5 trials
print("Test Accuracy Average for knn = ", sum(knn_test_acc)/NUM_TRIALS)
```

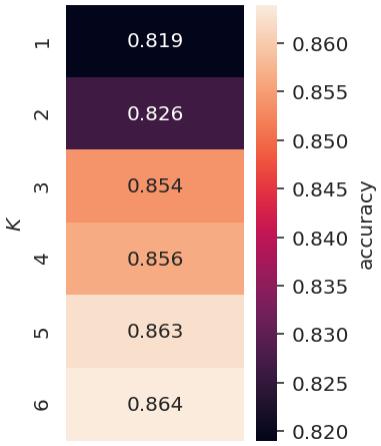
```
print("Test Accuracy Average for Random Forest = ",
      sum(rand_forest_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Decision Tree = ",
      sum(decision_tree_test_acc)/NUM_TRIALS)

#print("Test Accuracy Average for SVM = ", sum(svm_test_acc)/NUM_TRIALS)

#y-axis: partition
#x-axis: classifier
print(result_table)
print("############################")
print(result_table1)
print("############################")
print(result_table2)
```

```
Partition:  0.8
```

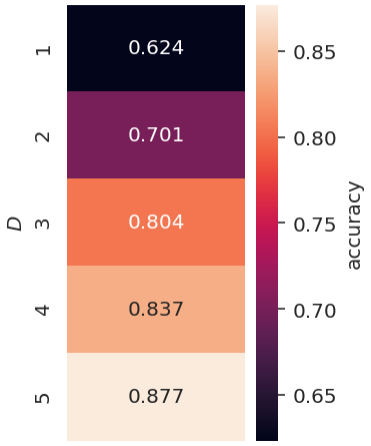
KNN train accuracy w.r.t *K*



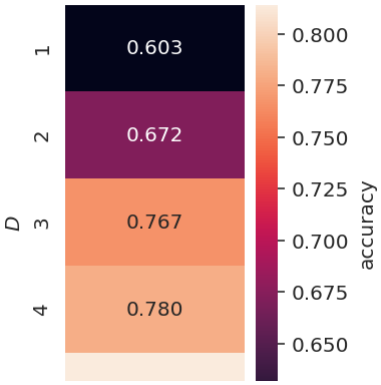
KNN val accuracy w.r.t *K*

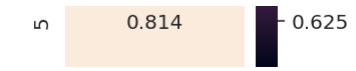


DT train accuracy w.r.t *D*

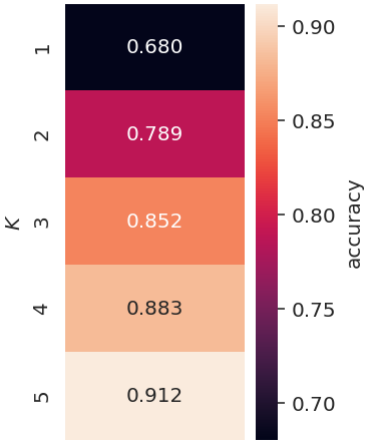


DT val accuracy w.r.t *D*

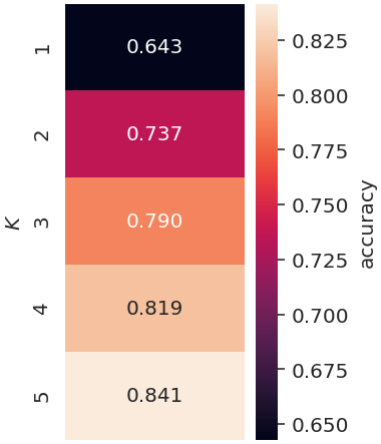




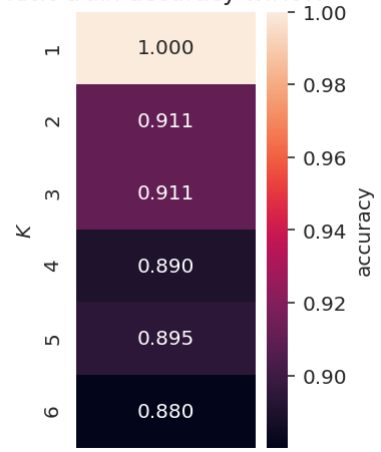
RF train accuracy w.r.t *K*



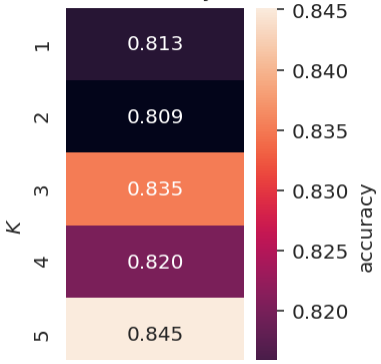
RF val accuracy w.r.t *K*



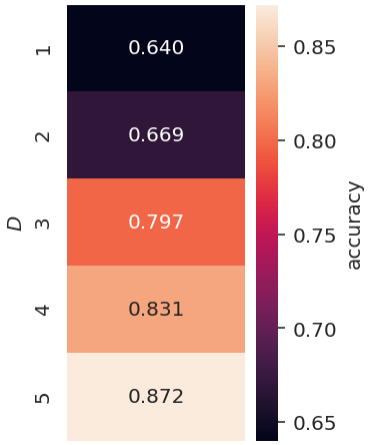
KNN train accuracy w.r.t *K*



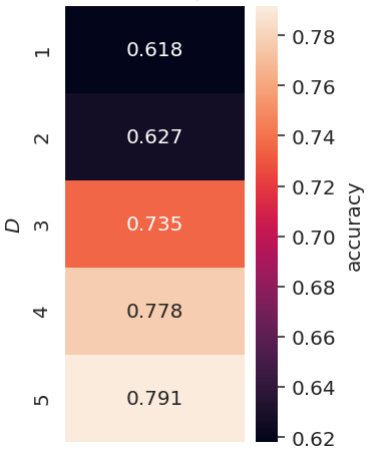
KNN val accuracy w.r.t *K*



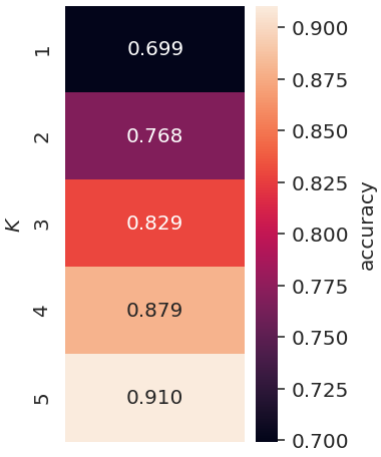
DT train accuracy w.r.t *D*



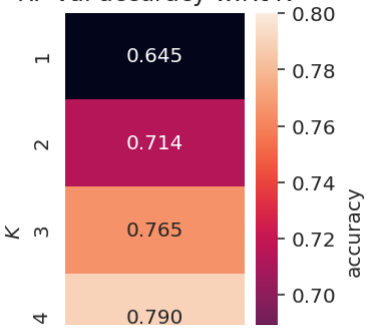
DT val accuracy w.r.t *D*

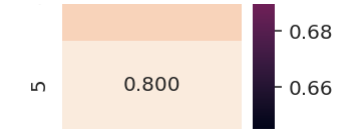


RF train accuracy w.r.t *K*

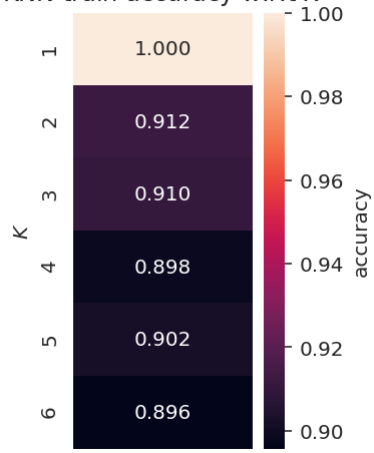


RF val accuracy w.r.t *K*

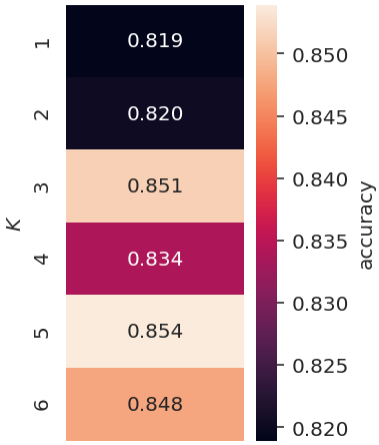




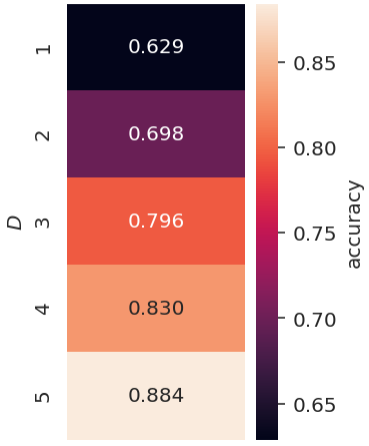
KNN train accuracy w.r.t *K*



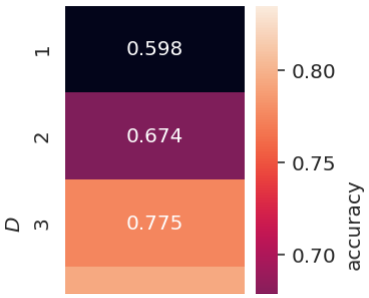
KNN val accuracy w.r.t *K*

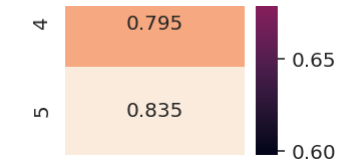


DT train accuracy w.r.t *D*

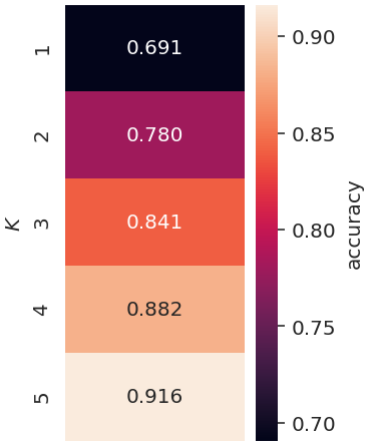


DT val accuracy w.r.t *D*

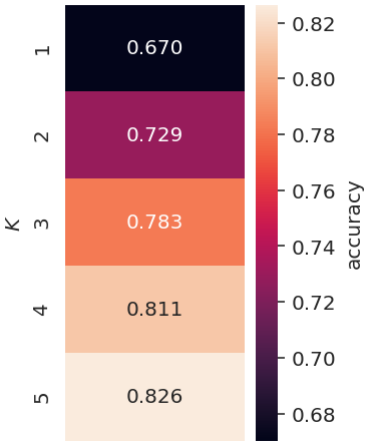




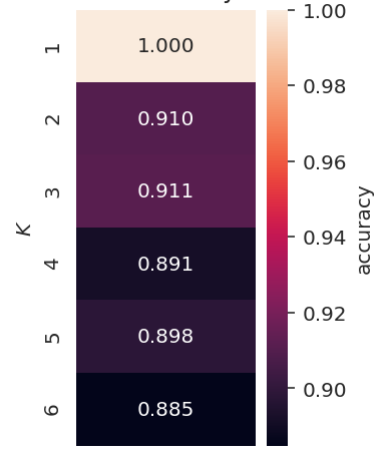
RF train accuracy w.r.t *K*



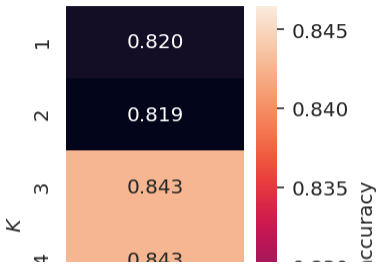
RF val accuracy w.r.t *K*

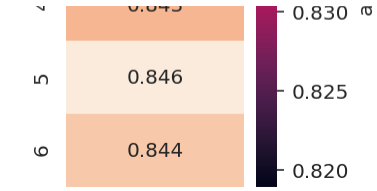


KNN train accuracy w.r.t *K*

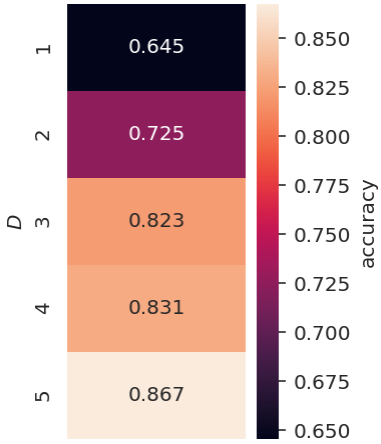


KNN val accuracy w.r.t *K*

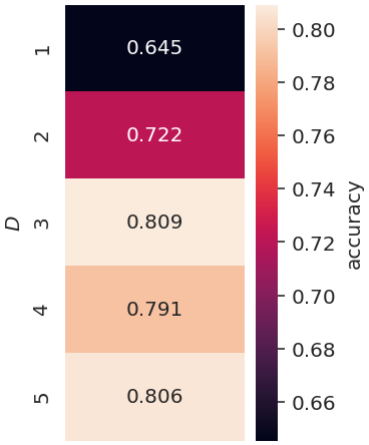




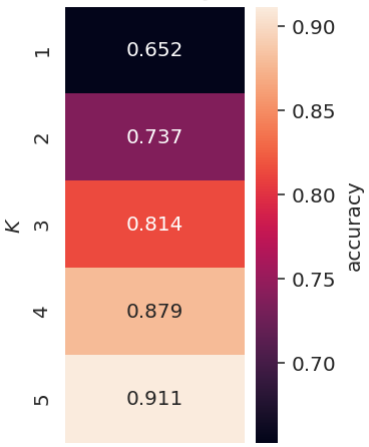
DT train accuracy w.r.t *D*



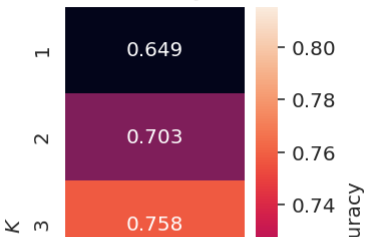
DT val accuracy w.r.t *D*

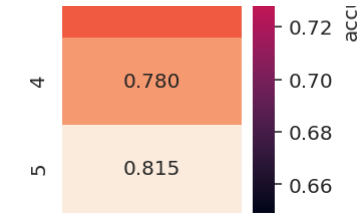


RF train accuracy w.r.t *K*

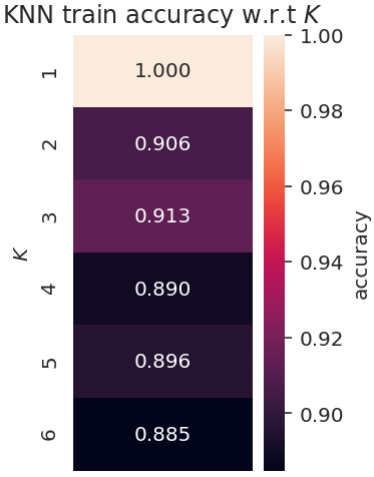


RF val accuracy w.r.t *K*

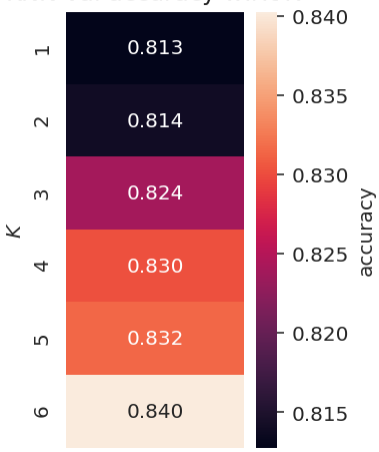




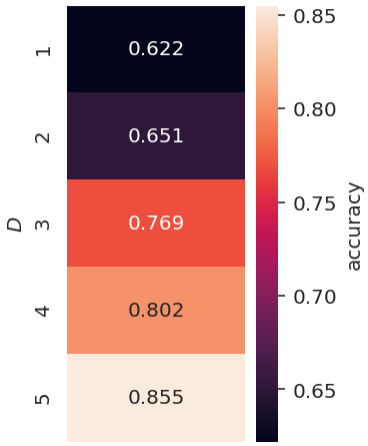
KNN train accuracy w.r.t *K*



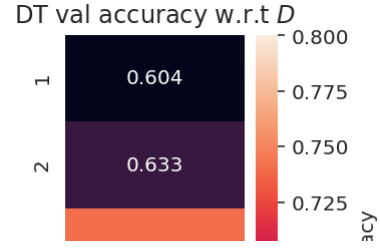
KNN val accuracy w.r.t *K*

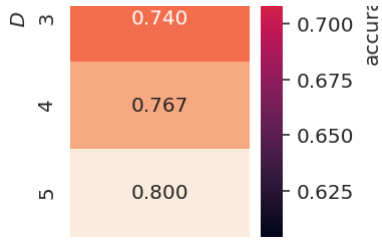


DT train accuracy w.r.t *D*

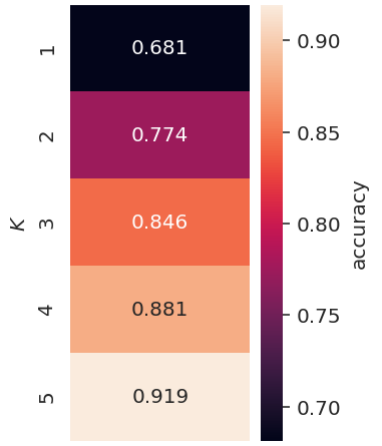


DT val accuracy w.r.t *D*

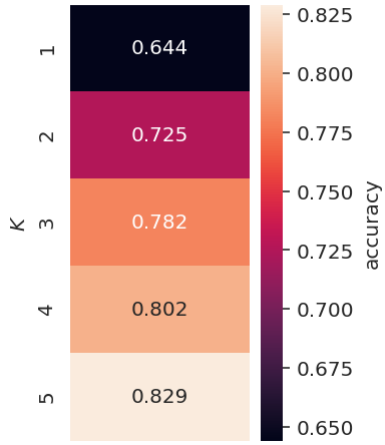




RF train accuracy w.r.t *K*

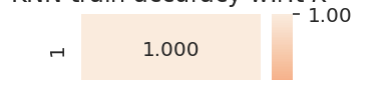


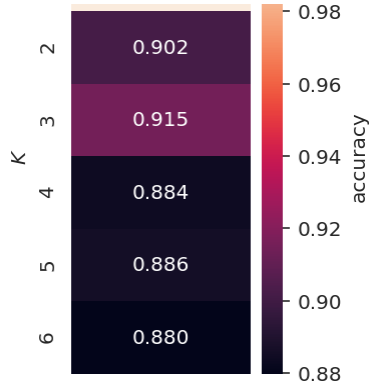
RF val accuracy w.r.t *K*



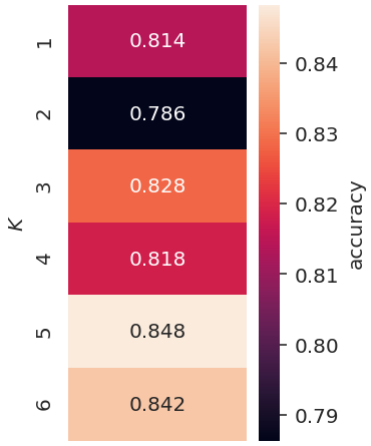

```
Test Accuracy Average for knn =  0.8240000000000001
Test Accuracy Average for Random Forest =  0.8150000000000001
Test Accuracy Average for Decision Tree =  0.804
[[0.    0.824 0.804 0.815 0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.   ]
 [0.    0.    0.    0.    0.    0.    0.   ]]
##############################
[[0.         0.88483522 0.85487178 0.91947738 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.5
```

KNN train accuracy w.r.t *K*

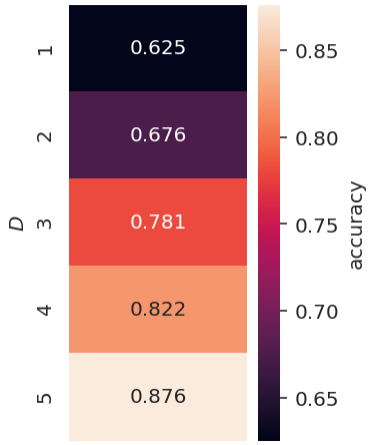




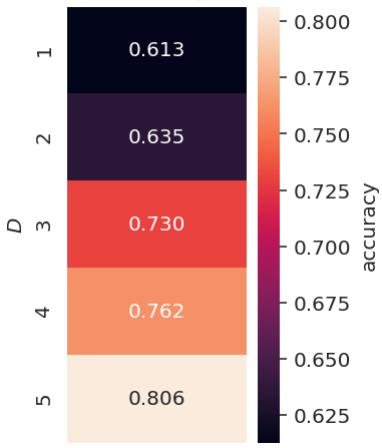
KNN val accuracy w.r.t *K*



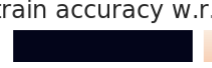
DT train accuracy w.r.t *D*

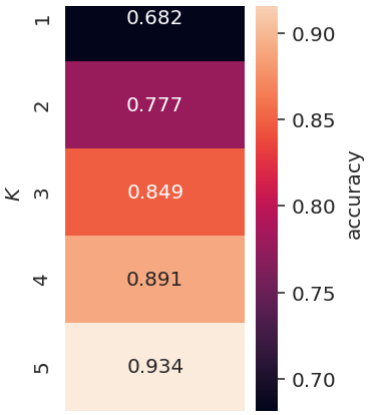


DT val accuracy w.r.t *D*

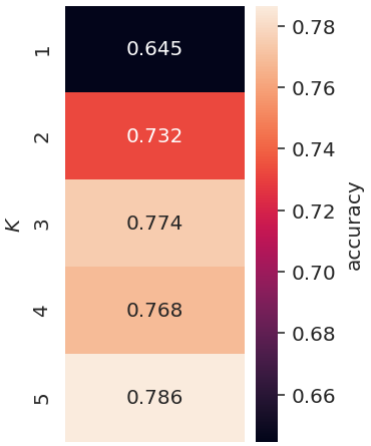


RF train accuracy w.r.t *K*

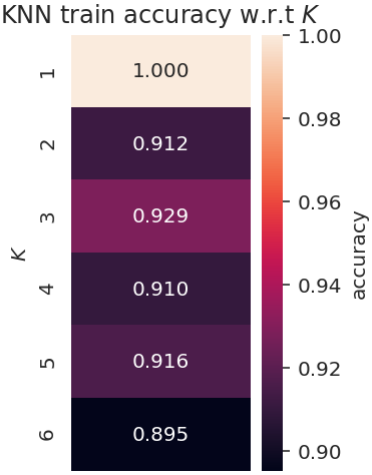




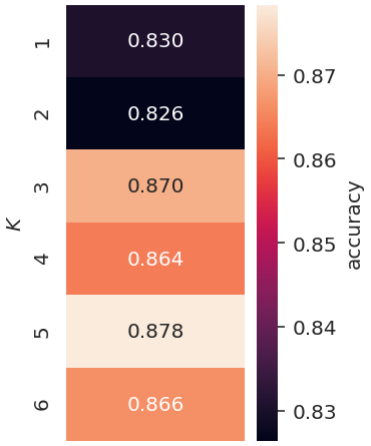
RF val accuracy w.r.t *K*



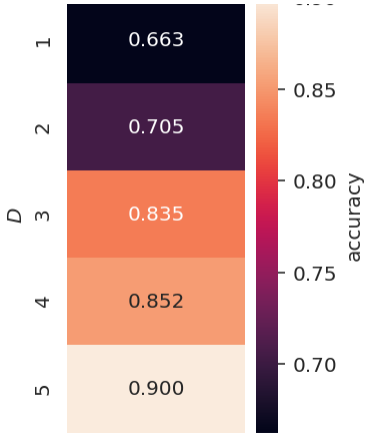
KNN train accuracy w.r.t *K*



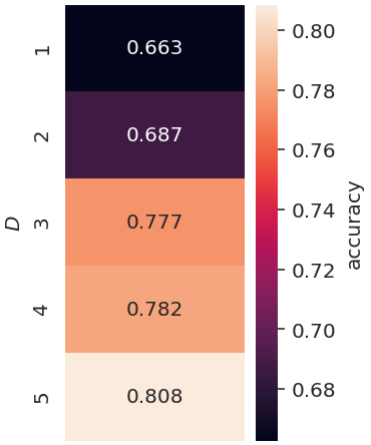
KNN val accuracy w.r.t *K*



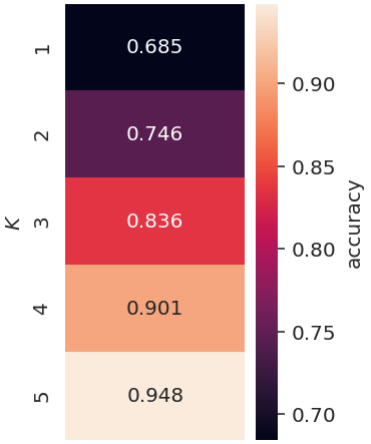
DT train accuracy w.r.t *D*



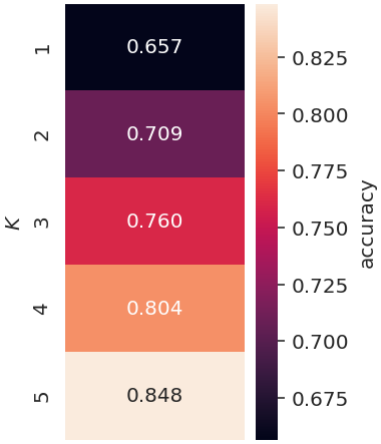
DT val accuracy w.r.t *D*



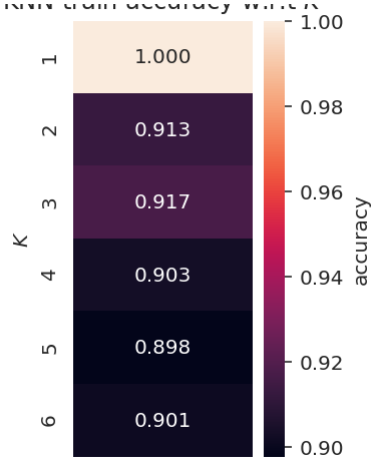
RF train accuracy w.r.t *K*



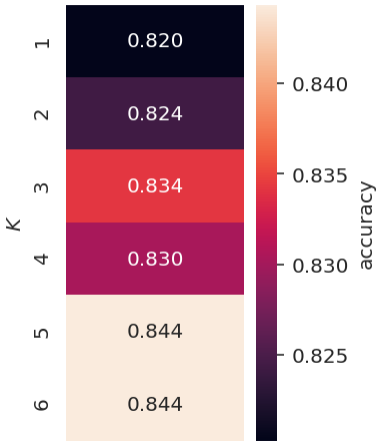
RF val accuracy w.r.t *K*



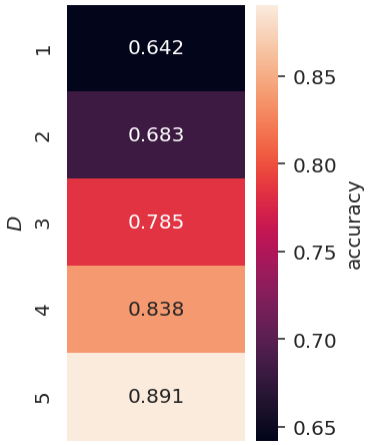
KNN train accuracy w.r.t *K*



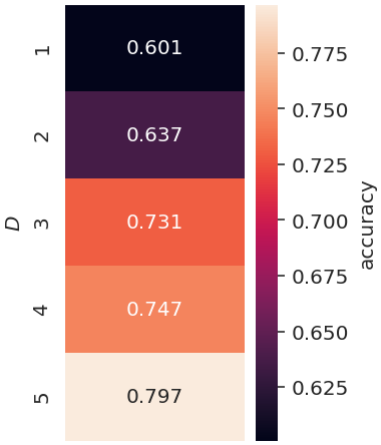
KNN val accuracy w.r.t *K*

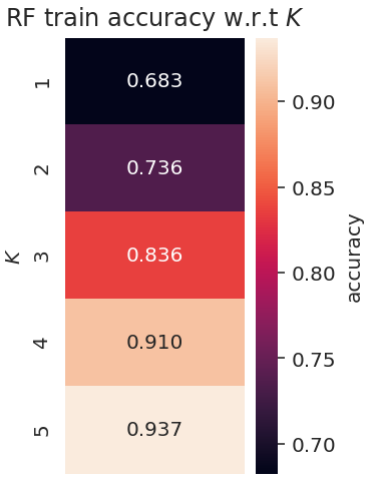


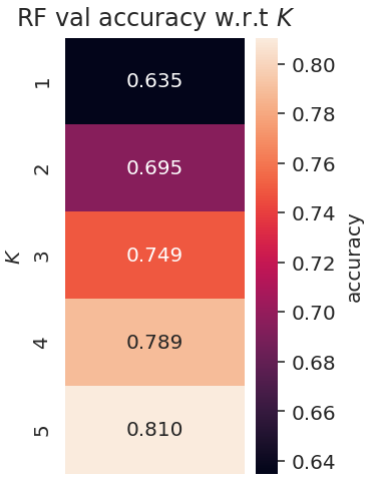
DT train accuracy w.r.t *D*

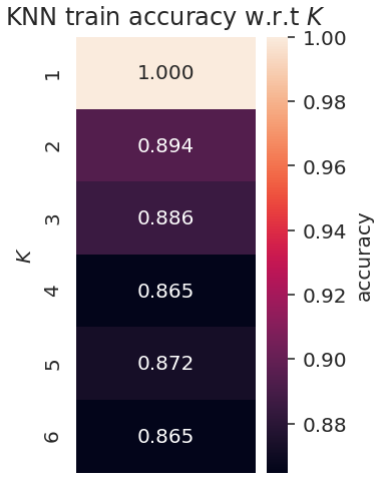


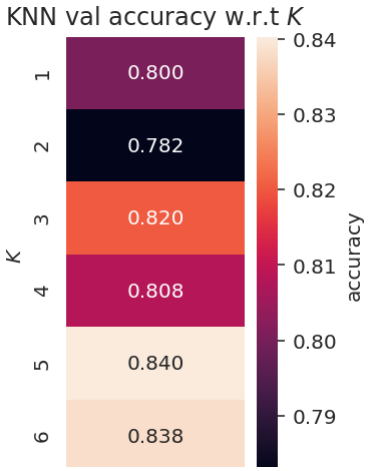
DT val accuracy w.r.t *D*



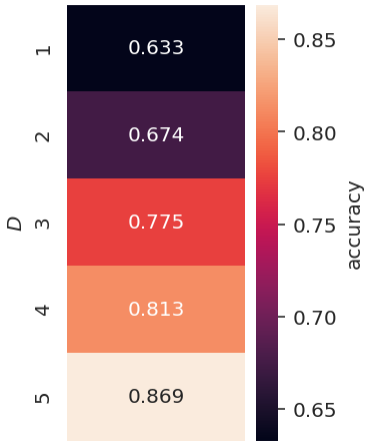

RF train accuracy w.r.t K
0.683
0.736
0.836
0.910
0.937
K
accuracy
0.90
0.85
0.80
0.75
0.70


RF val accuracy w.r.t K
0.635
0.695
0.749
0.789
0.810
K
accuracy
0.80
0.78
0.76
0.74
0.72
0.70
0.68
0.66
0.64

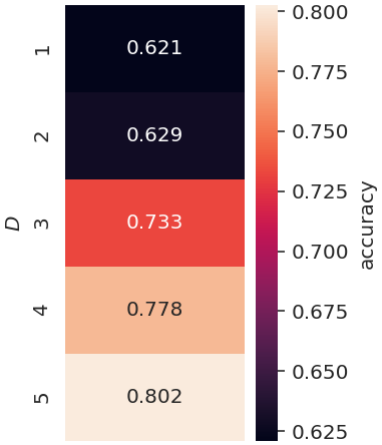

KNN train accuracy w.r.t K
1.000
0.894
0.886
0.865
0.872
0.865
K
accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88


KNN val accuracy w.r.t K
0.800
0.782
0.820
0.808
0.840
0.838
K
accuracy
0.84
0.83
0.82
0.81
0.80
0.79

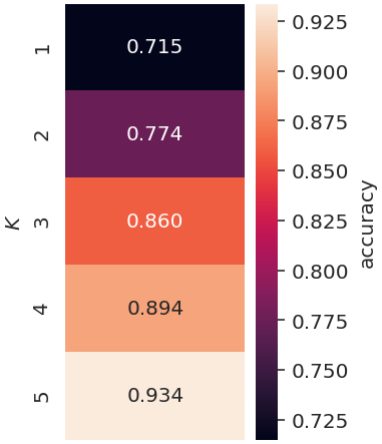
DT train accuracy w.r.t *D*



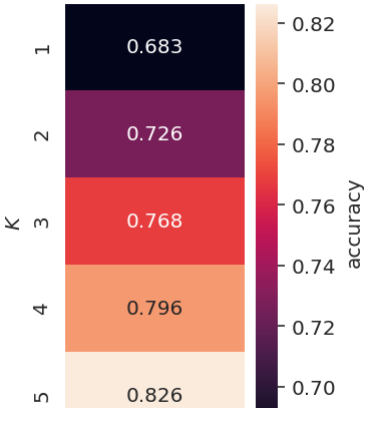
DT val accuracy w.r.t *D*



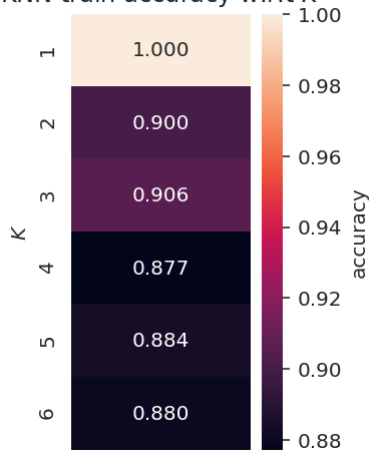
RF train accuracy w.r.t *K*



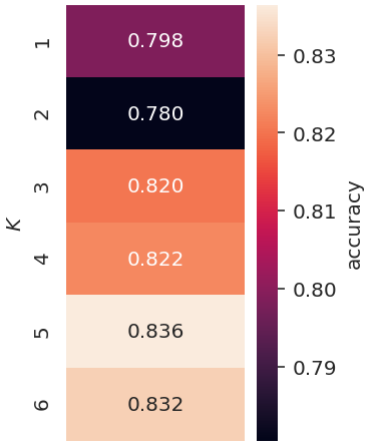
RF val accuracy w.r.t *K*



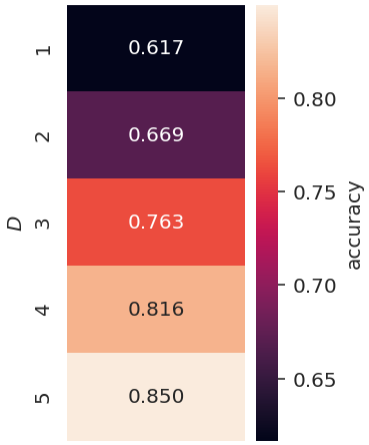
KNN train accuracy w.r.t *K*



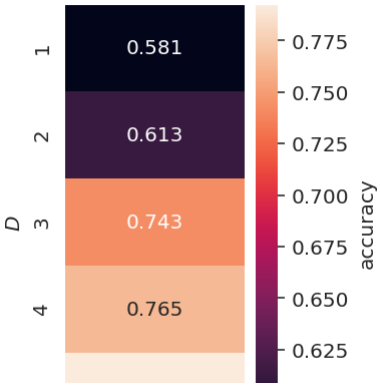
KNN val accuracy w.r.t *K*

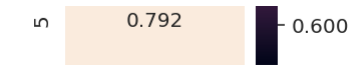


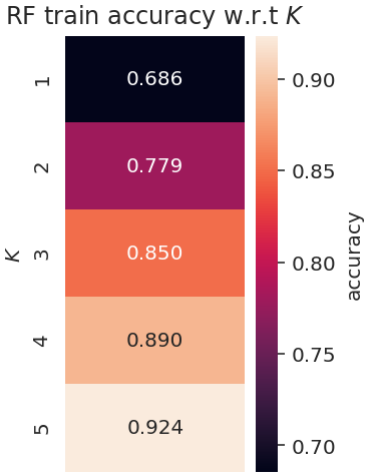
DT train accuracy w.r.t *D*

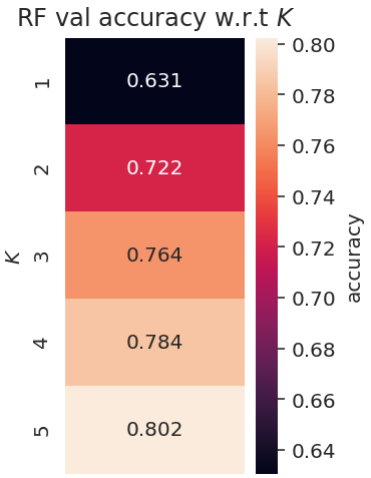


DT val accuracy w.r.t *D*

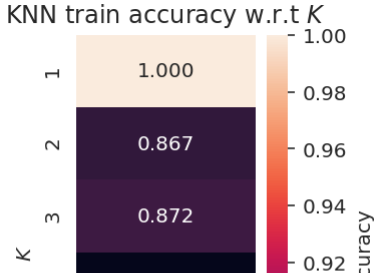


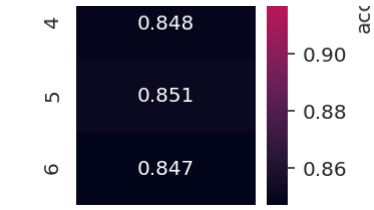




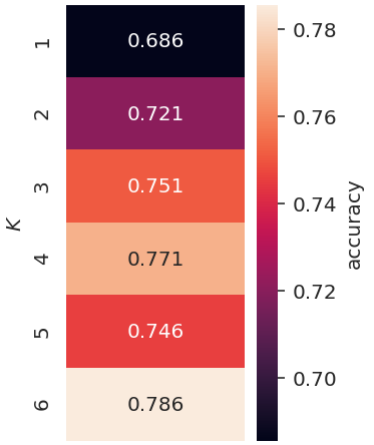



```
Test Accuracy Average for knn =  0.8512000000000001
Test Accuracy Average for Random Forest =  0.8211999999999999
Test Accuracy Average for Decision Tree =  0.7908000000000001
[[0.     0.824  0.804  0.815  0.     0.     0.    ]
 [0.     0.8512 0.7908 0.8212 0.     0.     0.    ]
 [0.     0.     0.     0.     0.     0.     0.    ]]
##############################
[[0.         0.88483522 0.85487178 0.91947738 0.         0.
  0.        ]
 [0.         0.8842394  0.84979676 0.92366584 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 5. 5. 0. 0. 0.]
 [0. 5. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.2
```

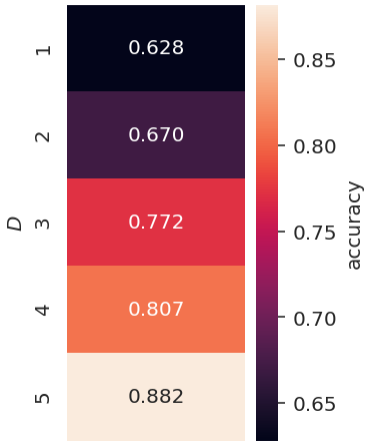




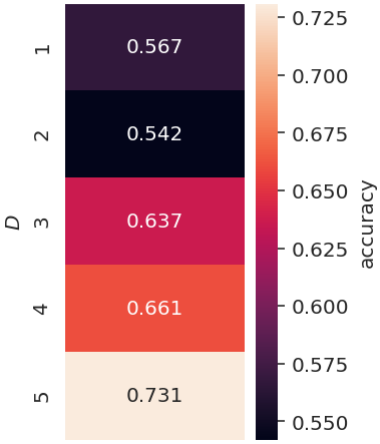
KNN val accuracy w.r.t *K*



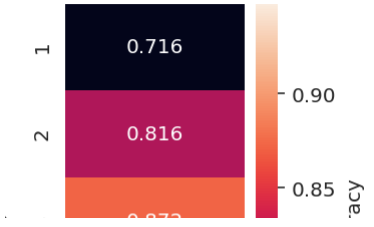
DT train accuracy w.r.t *D*

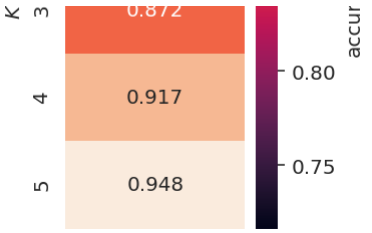


DT val accuracy w.r.t *D*

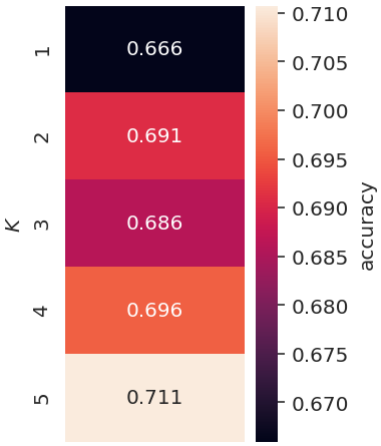


RF train accuracy w.r.t *K*

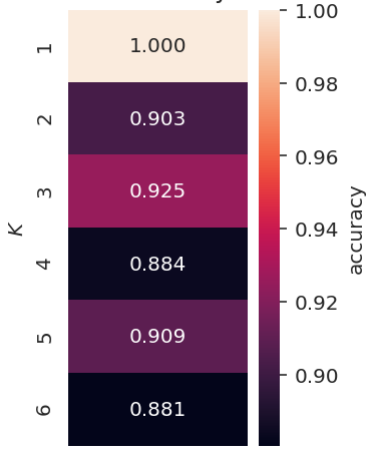




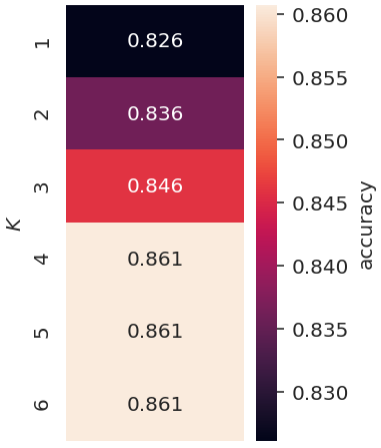
RF val accuracy w.r.t *K*



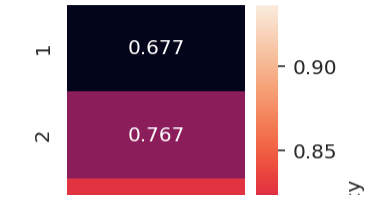
KNN train accuracy w.r.t *K*

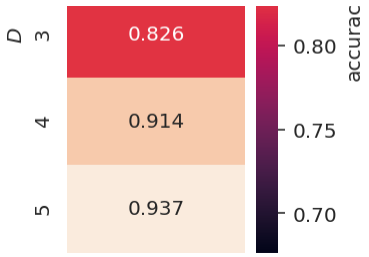


KNN val accuracy w.r.t *K*

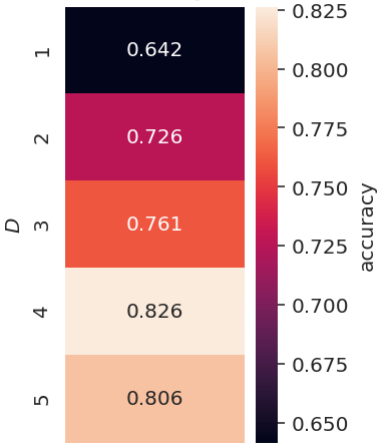


DT train accuracy w.r.t *D*

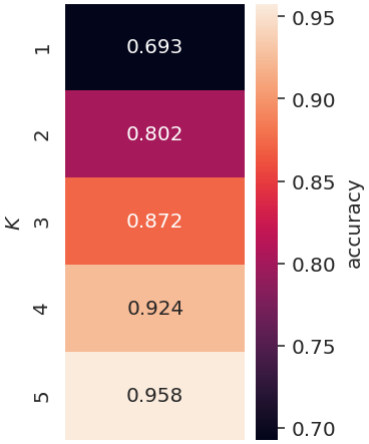




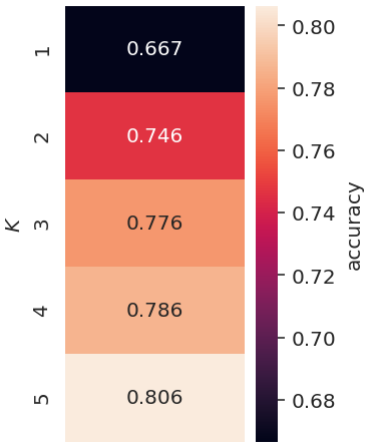
DT val accuracy w.r.t *D*



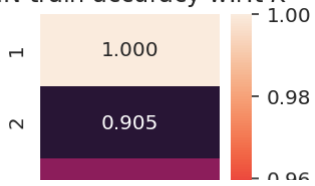
RF train accuracy w.r.t *K*

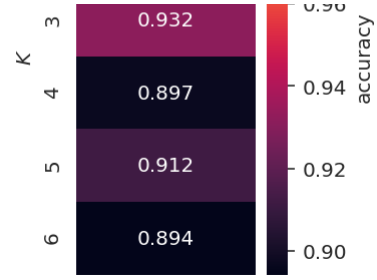


RF val accuracy w.r.t *K*

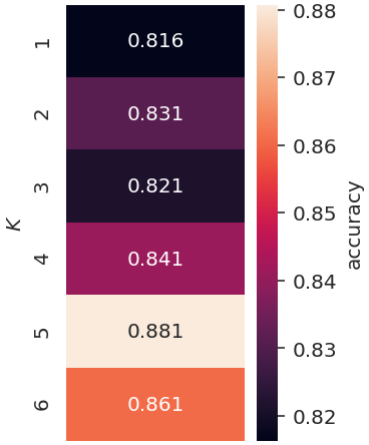


KNN train accuracy w.r.t *K*

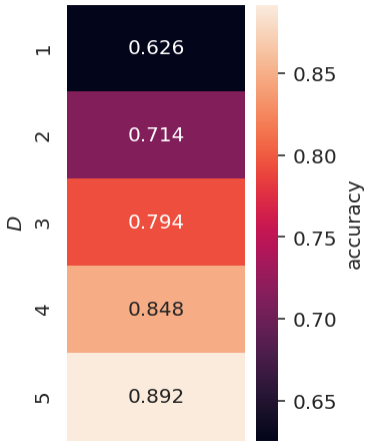




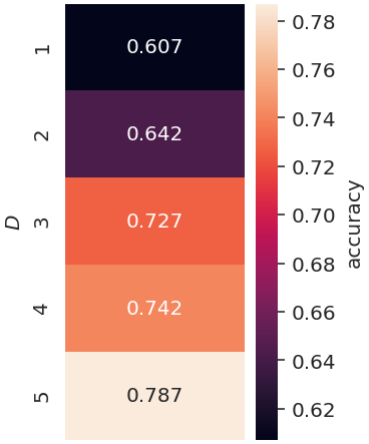
KNN val accuracy w.r.t *K*



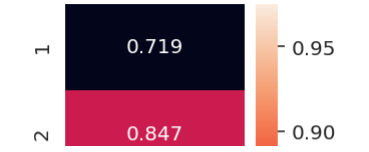
DT train accuracy w.r.t *D*

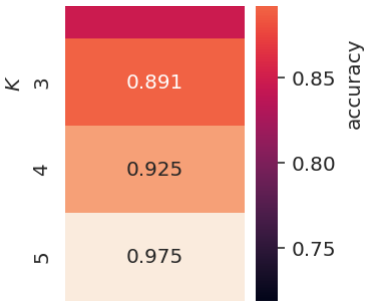


DT val accuracy w.r.t *D*

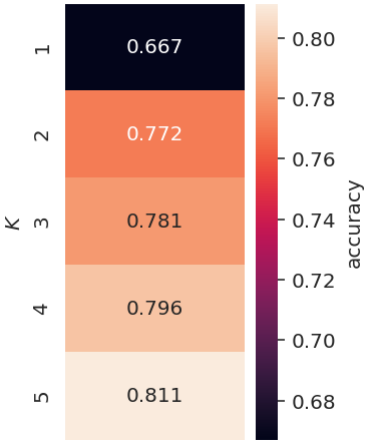


RF train accuracy w.r.t *K*

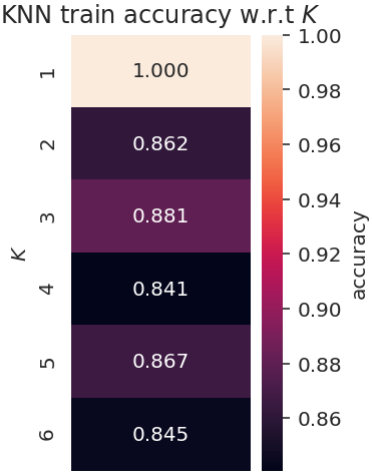




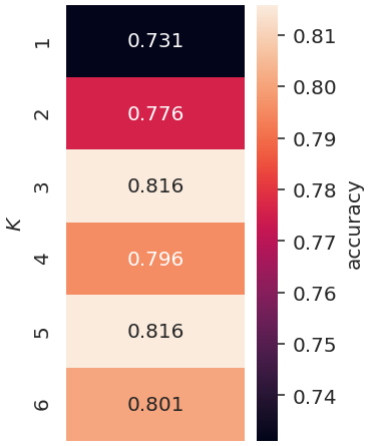
RF val accuracy w.r.t *K*



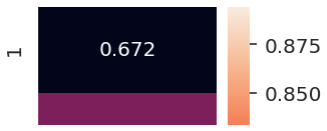
KNN train accuracy w.r.t *K*

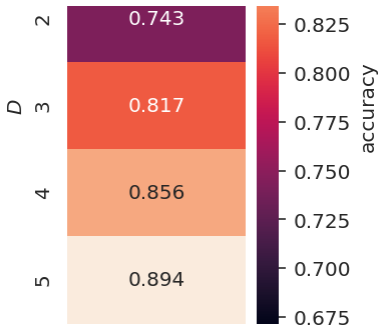


KNN val accuracy w.r.t *K*

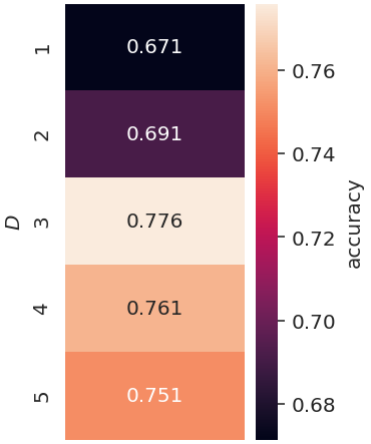


DT train accuracy w.r.t *D*

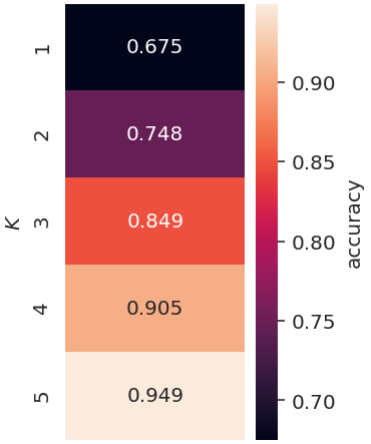




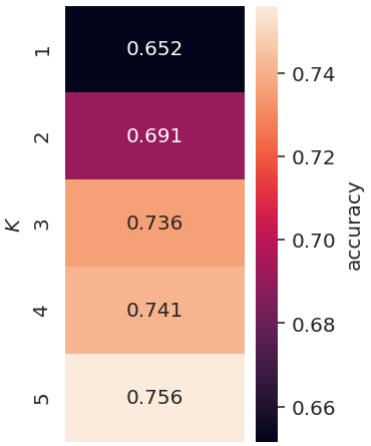
DT val accuracy w.r.t *D*



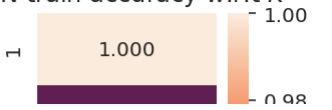
RF train accuracy w.r.t *K*

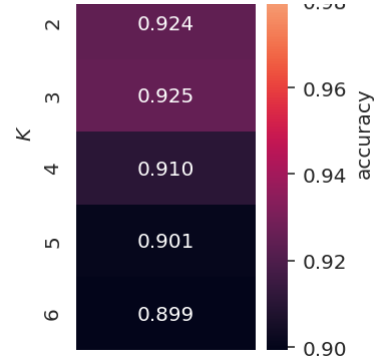


RF val accuracy w.r.t *K*

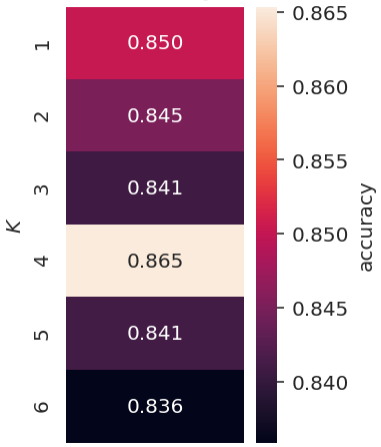


KNN train accuracy w.r.t *K*

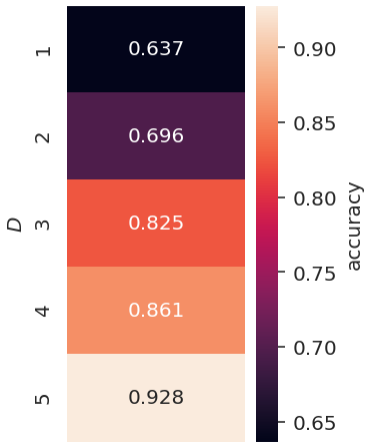




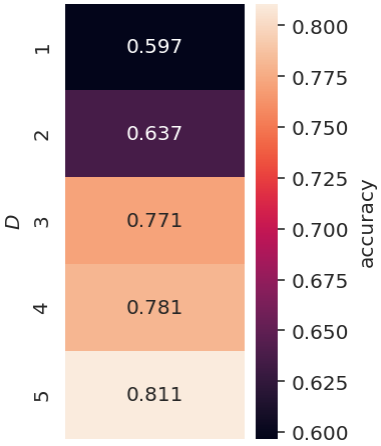
KNN val accuracy w.r.t *K*



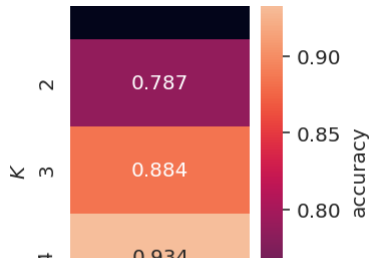
DT train accuracy w.r.t *D*



DT val accuracy w.r.t *D*



RF train accuracy w.r.t *K*



Running Respective P and T Tests to compare the Trials Accuracy For Each Algorithm


```
#Compute the difference between the results
print("P and T Test for KNN trials vs. Decision Tree")
diff = [y - x for y, x in zip(knn_test_acc, decision_tree_test_acc)]

#Compute the mean of differences
d_bar = np.mean(diff)

#Compute the variance of differences
sigma2 = np.var(diff)

#compute the number of data points used for training
n1 = len(Y_train_val)

#compute the number of data points used for testing
n2 = len(Y_test_val)

#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for KNN trials vs. Decision Tree
This is the T Value
0.7782991911502226
```

```
      This is the P Value
      0.21821474386858342
      Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. Decision Tree")
diff = [y - x for y, x in zip(rand_forest_test_acc, decision_tree_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
      P and T Test for Random Forests vs. Decision Tree
      This is the T Value
      0.2553216664871904
      This is the P Value
      0.39924258533972856
      Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. KNN")
diff = [y - x for y, x in zip(rand_forest_test_acc, knn_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
```

```
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. KNN
This is the T Value
-1.363439710455345
This is the P Value
0.9135973146909427
Accept the null hypothesis
```

Parkinson's Disease Dataset

This given dataset provides 756 various attributes that give us the necessary information to analyze the state of being of Parkinson's diseased patients. The reasoning behind using this dataset is for a basis for a real world scenario of analyzing both female and male patients and their symptoms of well being. Their also is a personal motivation because my grandmother died of Parkinson's disease, so this research has definitely taught me more about how people are with the disease when I was too young to understand.

```
pd_data_preserved = pd.read_csv('pd_speech_features.csv')
pd_data = pd_data_preserved #Temp for manipulation
pd_data #preview of dataset

#Basic Cleanup
pd_data.dropna(inplace=True)
pd_data.shape
```

```
pd_data.shape
#pd_data.head()
```

```
(756, 755)
```

```
#Convert Data to a numpy Array
pd_data = pd_data.sample(n=700).reset_index(drop=True)
pd_data.head()#Data preview
```

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses
0	174	1	0.82927	0.69312	0.46323	265	264	0.007288
1	28	0	0.87005	0.69617	0.28222	439	438	0.004396
2	72	1	0.81006	0.85000	0.50814	243	242	0.007944
3	207	1	0.84190	0.76320	0.56908	273	272	0.007068
4	199	1	0.84217	0.65611	0.51994	272	271	0.007105

5 rows × 755 columns

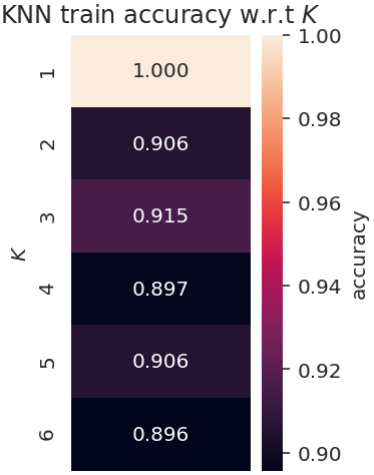
```
#Split Data By 80/20, 50/50, 20/80
partitionVal = [0.8, 0.5, 0.2]
result_table = np.zeros((3,7))
result_table1 = np.zeros((3,7))
result_table2 = np.zeros((3,7))
for i, partition in enumerate(partitionVal):
    print("Partition: ", partition)

    knn_test_acc = []
    rand_forest_test_acc = []
    decision_tree_test_acc = []
    svm_test_acc = []



    NUM_TRIALS = 5
    for trial in range(NUM_TRIALS):
        #Mix up the data
        students_data = students_data.sample(frac=1).reset_index(drop=True)
        #Find the point where to split the data
        breakNum = int(partition*len(students_data))

        X_train_full = students_data.loc[0:breakNum]
        X_train_val = X_train_full.drop("gender", axis = 1)
        Y_train_val = X_train_full["gender"]
        X_test_full = students_data.loc[breakNum:]
        X_test= X_test_full.drop("gender", axis = 1)
        Y_test_val = X_test_full["gender"]
```

```
    #Call the svm classifier (Took Too Long For ALL Datasets)
    #test_acc,best_train0,C0 = svm_func()
    #svm_test_acc.append(test_acc)

    #Call the knn classifier
    test_acc,best_train1,C1 = knn_classifier()
    knn_test_acc.append(test_acc)

    #Call the Decision Tree classifier
    test_acc,best_train2,C2 = decision_Tree()
    decision_tree_test_acc.append(test_acc)

    #Call the Random Forest classifier
    test_acc,best_train3,C3 = rand_Forest()
    rand_forest_test_acc.append(test_acc)


#result_table[i, 0] = sum(svm_test_acc)/NUM_TRIALS
result_table[i, 1] = sum(knn_test_acc)/NUM_TRIALS
result_table[i, 2] = sum(decision_tree_test_acc)/NUM_TRIALS
result_table[i, 3] = sum(rand_forest_test_acc)/NUM_TRIALS


#result_table1[i, 0] = best_train0
result_table1[i, 1] = best_train1
result_table1[i, 2] = best_train2
result_table1[i, 3] = best_train3


#result_table2[i, 0] = C0
result_table2[i, 1] = C1
result_table2[i, 2] = C2
result_table2[i, 3] = C3


#Average all test accuracies for all 5 trials
print("Test Accuracy Average for knn = ", sum(knn_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Random Forest = ",
      sum(rand_forest_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Decision Tree = ",
      sum(decision_tree_test_acc)/NUM_TRIALS)

#print("Test Accuracy Average for SVM = ", sum(svm_test_acc)/NUM_TRIALS)

#y-axis: partition
#x-axis: classifier
print(result_table)
print("############################")
print(result_table1)
```

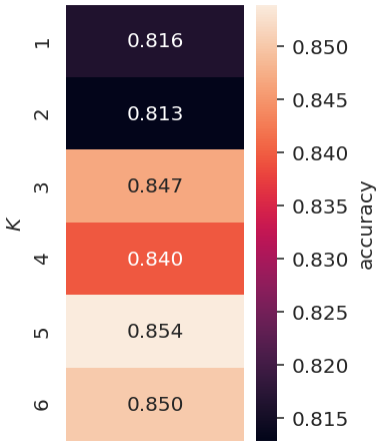
```
        print("############################")
        print(result_table2)
```

```
Partition:  0.8
```

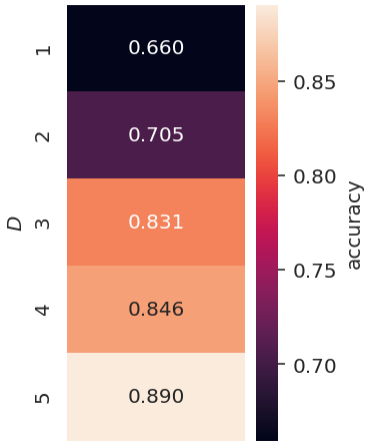
KNN train accuracy w.r.t *K*



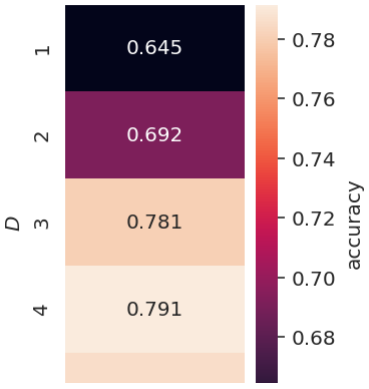
KNN val accuracy w.r.t *K*

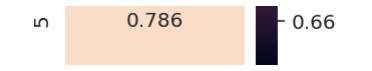


DT train accuracy w.r.t *D*

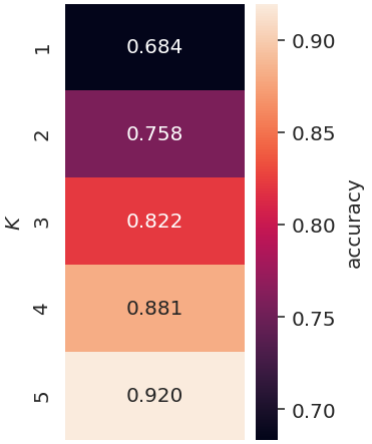


DT val accuracy w.r.t *D*

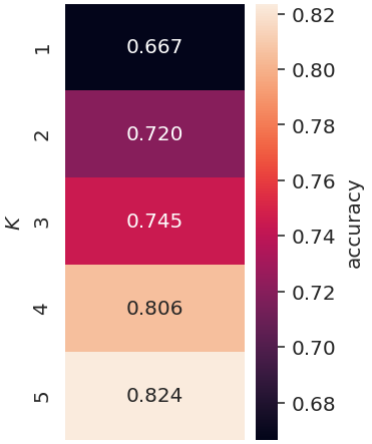




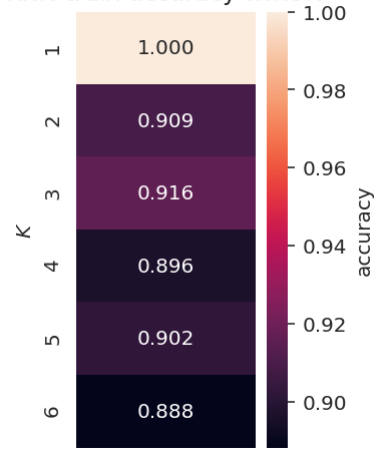
RF train accuracy w.r.t *K*



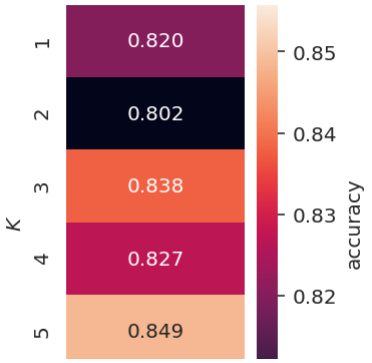
RF val accuracy w.r.t *K*



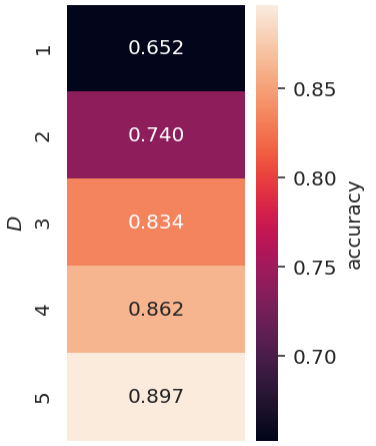
KNN train accuracy w.r.t *K*



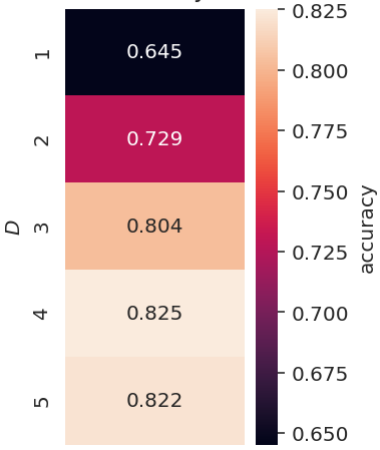
KNN val accuracy w.r.t *K*



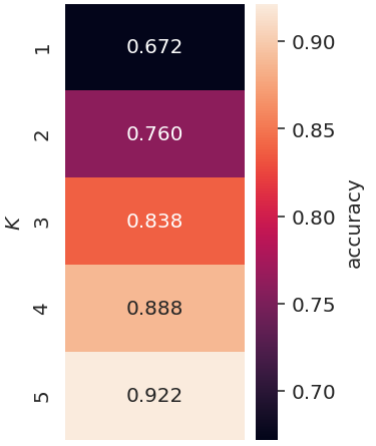
DT train accuracy w.r.t *D*



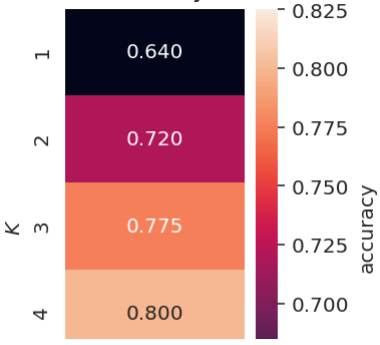
DT val accuracy w.r.t *D*

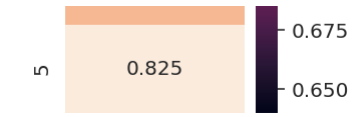


RF train accuracy w.r.t *K*

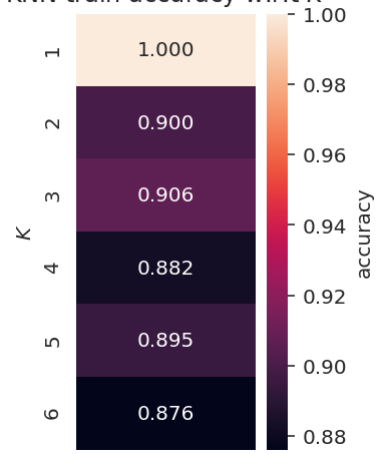


RF val accuracy w.r.t *K*

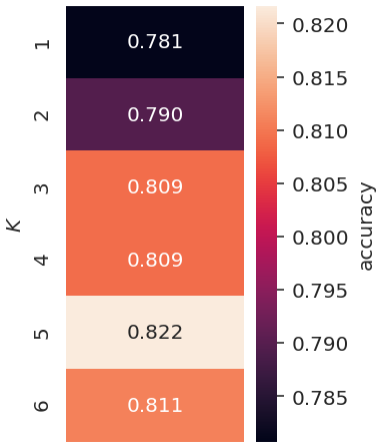




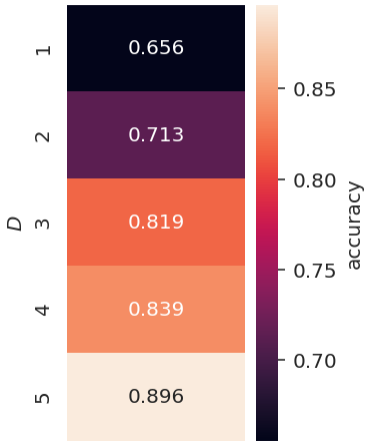
KNN train accuracy w.r.t *K*



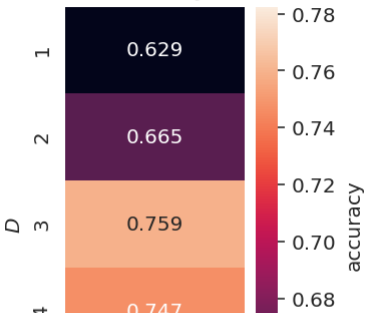
KNN val accuracy w.r.t *K*

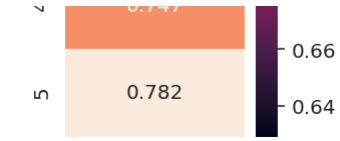


DT train accuracy w.r.t *D*

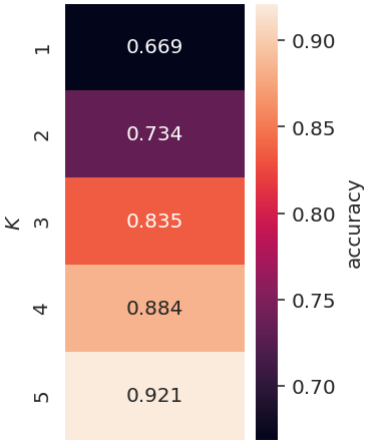


DT val accuracy w.r.t *D*

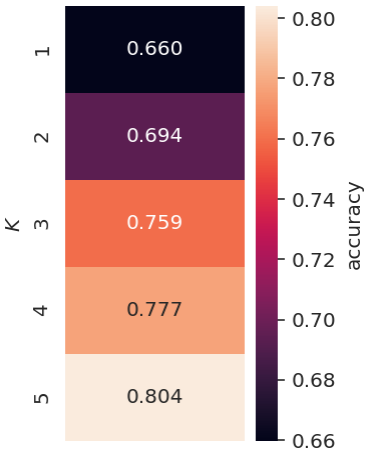




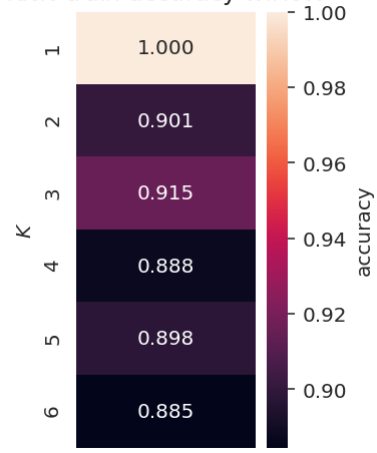
RF train accuracy w.r.t *K*



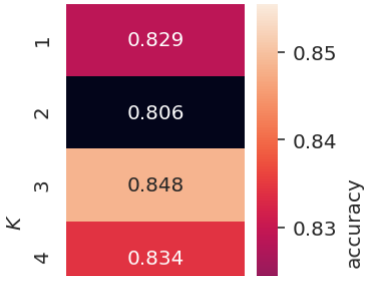
RF val accuracy w.r.t *K*

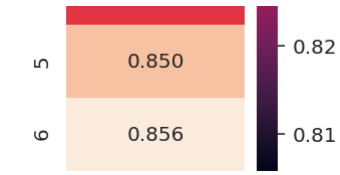


KNN train accuracy w.r.t *K*

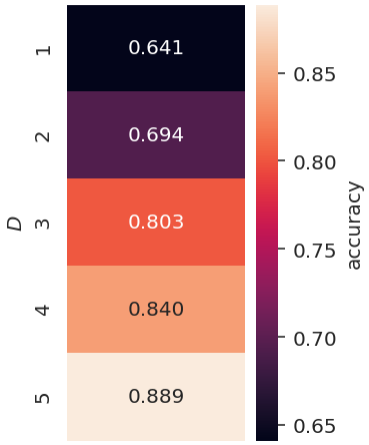


KNN val accuracy w.r.t *K*

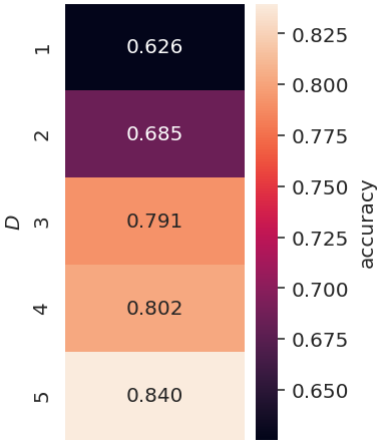




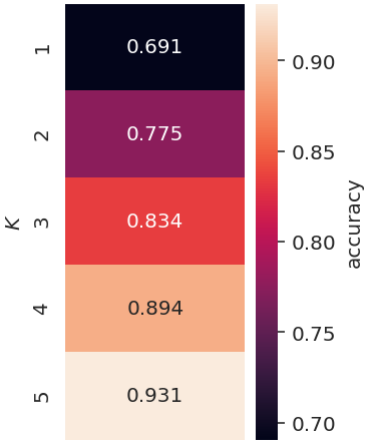
DT train accuracy w.r.t *D*



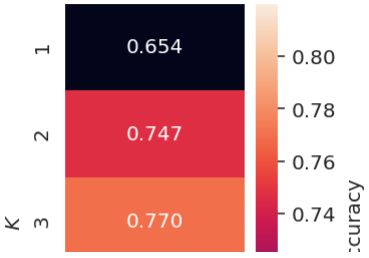
DT val accuracy w.r.t *D*

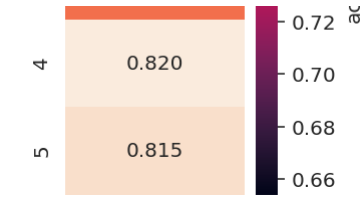


RF train accuracy w.r.t *K*

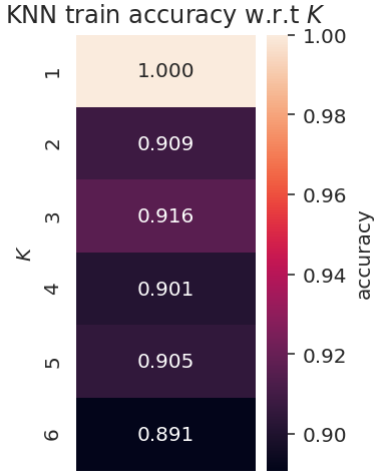


RF val accuracy w.r.t *K*

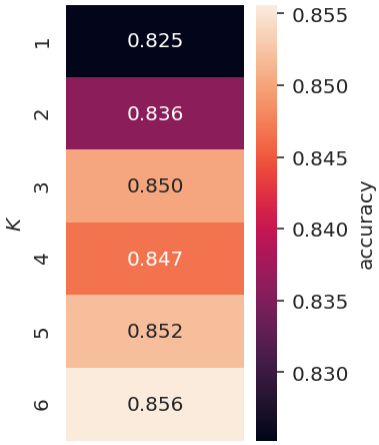




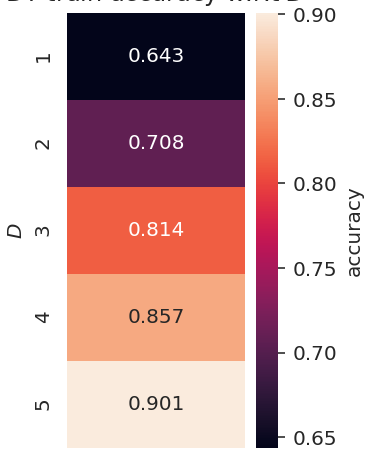
KNN train accuracy w.r.t *K*



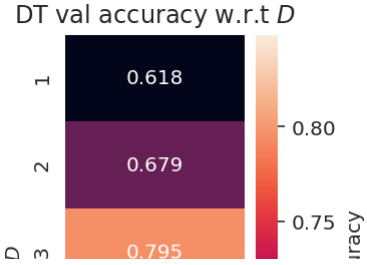
KNN val accuracy w.r.t *K*

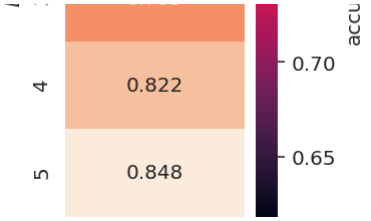


DT train accuracy w.r.t *D*

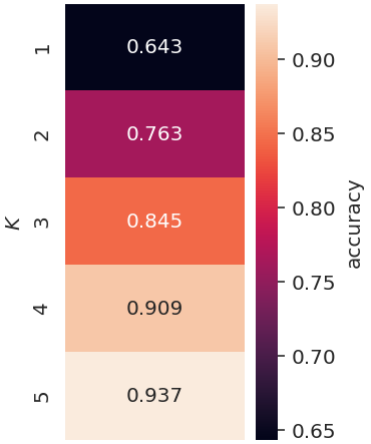


DT val accuracy w.r.t *D*

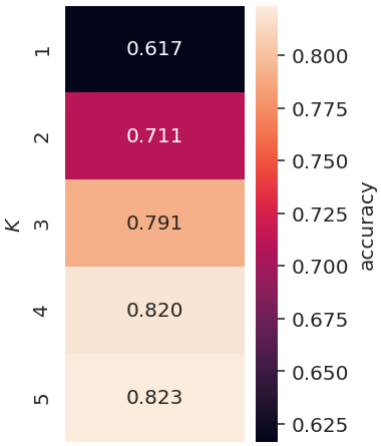




RF train accuracy w.r.t *K*

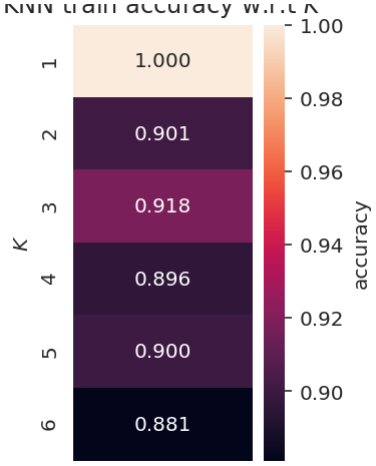


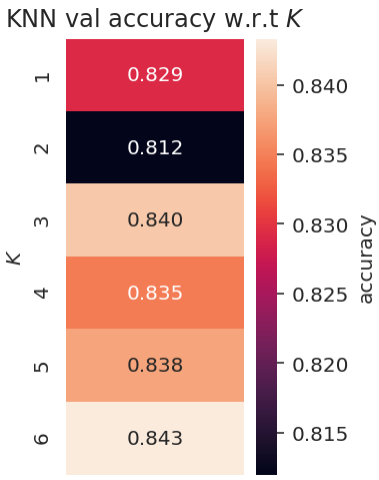
RF val accuracy w.r.t *K*

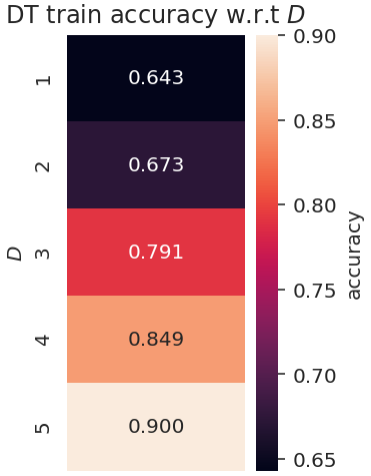


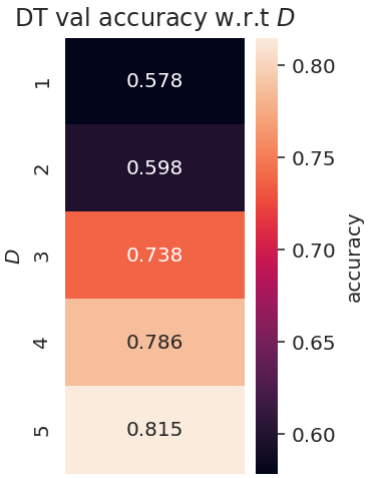

```
Test Accuracy Average for knn =  0.8514285714285714
Test Accuracy Average for Random Forest =  0.8099999999999999
Test Accuracy Average for Decision Tree =  0.7985714285714286
[[0.         0.85142857 0.79857143 0.81       0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0.         0.89082187 0.90062937 0.9371709  0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.5
```

KNN train accuracy w.r.t *K*

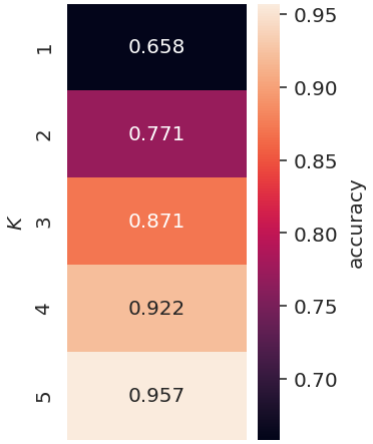

KNN train accuracy w.r.t K
1.000
0.901
0.918
0.896
0.900
0.881
K
accuracy
1.00
0.98
0.96
0.94
0.92
0.90


KNN val accuracy w.r.t K
0.829
0.812
0.840
0.835
0.838
0.843
K
accuracy
0.840
0.835
0.830
0.825
0.820
0.815

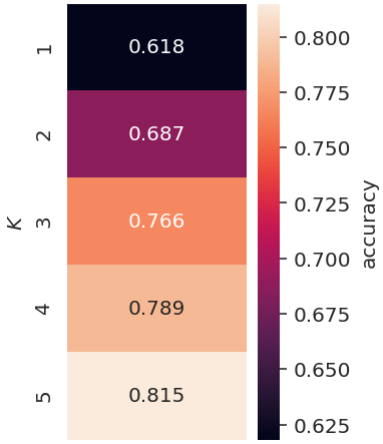

DT train accuracy w.r.t D
0.643
0.673
0.791
0.849
0.900
D
accuracy
0.90
0.85
0.80
0.75
0.70
0.65


DT val accuracy w.r.t D
0.578
0.598
0.738
0.786
0.815
D
accuracy
0.80
0.75
0.70
0.65
0.60

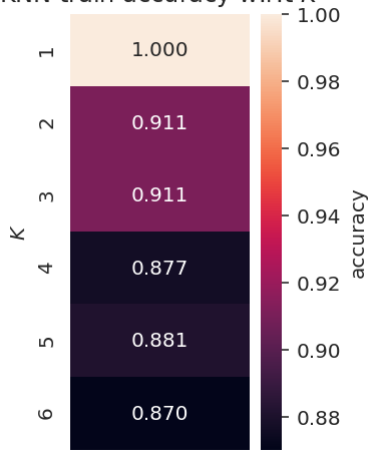
RF train accuracy w.r.t *K*



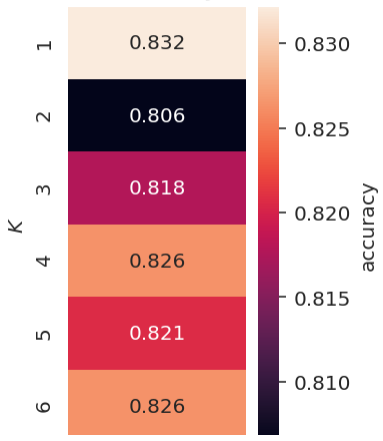
RF val accuracy w.r.t *K*

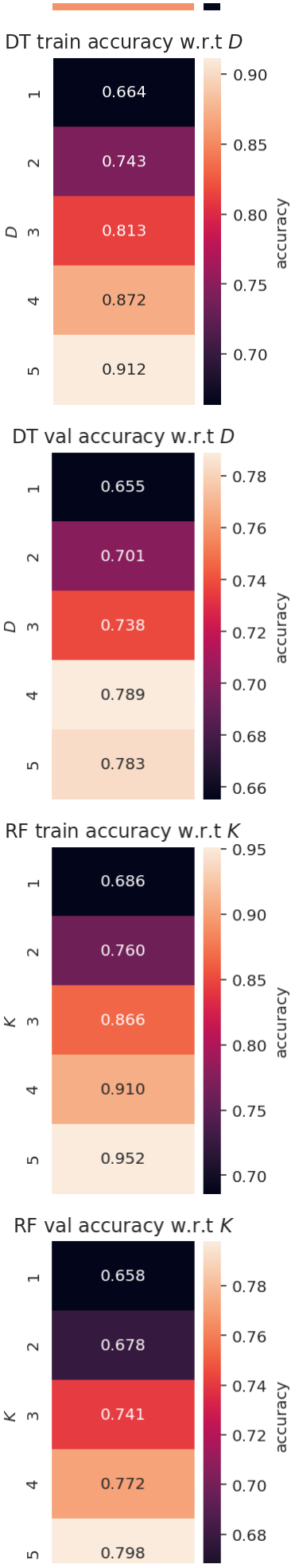


KNN train accuracy w.r.t *K*

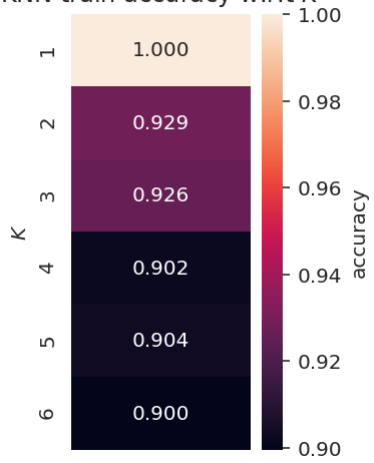


KNN val accuracy w.r.t *K*

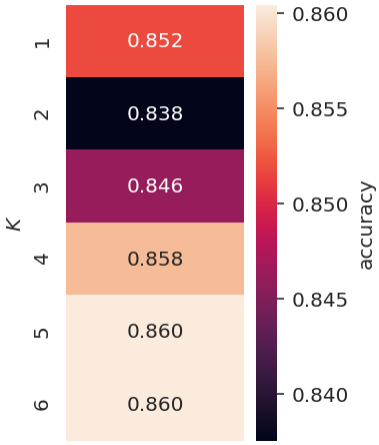



DT train accuracy w.r.t D
D
1
2
3
4
5
0.664
0.743
0.813
0.872
0.912
accuracy
0.90
0.85
0.80
0.75
0.70
DT val accuracy w.r.t D
D
1
2
3
4
5
0.655
0.701
0.738
0.789
0.783
accuracy
0.78
0.76
0.74
0.72
0.70
0.68
0.66
RF train accuracy w.r.t K
K
1
2
3
4
5
0.686
0.760
0.866
0.910
0.952
accuracy
0.95
0.90
0.85
0.80
0.75
0.70
RF val accuracy w.r.t K
K
1
2
3
4
5
0.658
0.678
0.741
0.772
0.798
accuracy
0.78
0.76
0.74
0.72
0.70
0.68

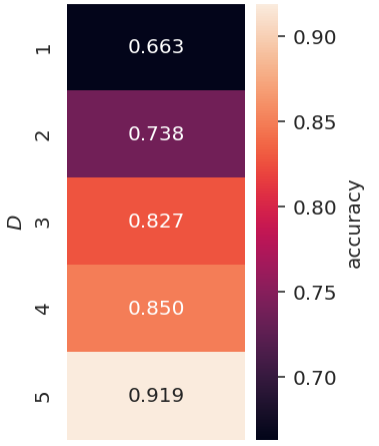
KNN train accuracy w.r.t *K*



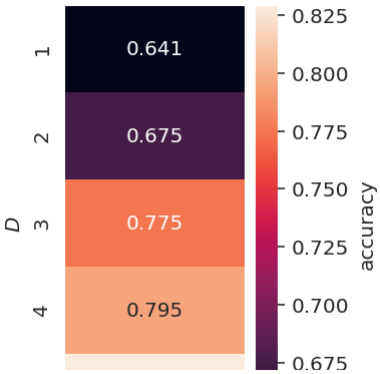
KNN val accuracy w.r.t *K*



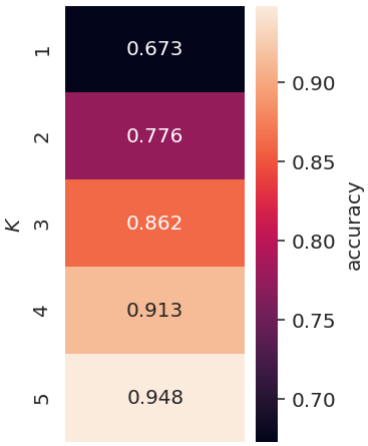
DT train accuracy w.r.t *D*



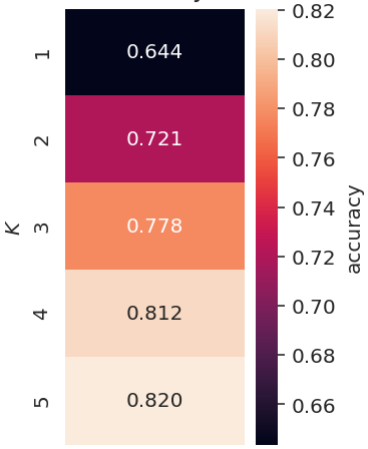
DT val accuracy w.r.t *D*



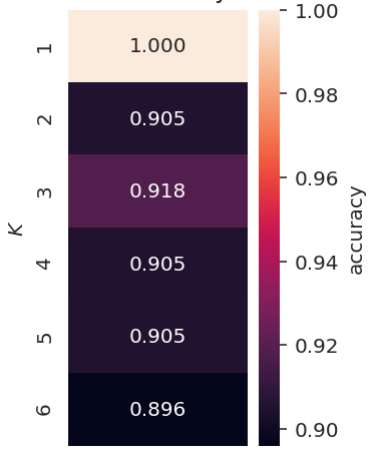
RF train accuracy w.r.t *K*



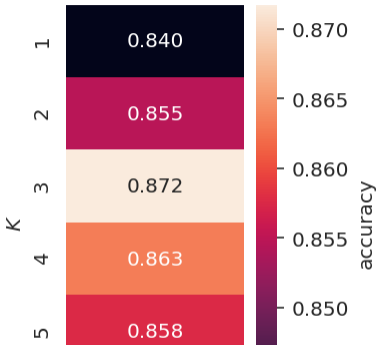
RF val accuracy w.r.t *K*

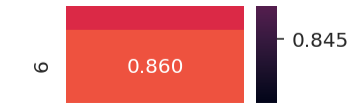


KNN train accuracy w.r.t *K*

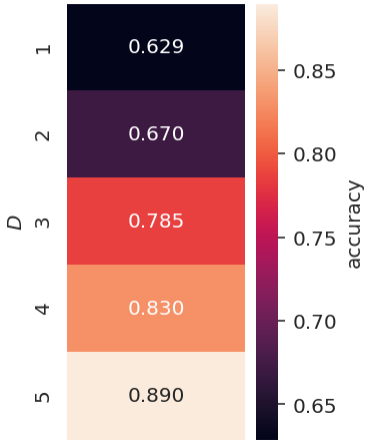


KNN val accuracy w.r.t *K*

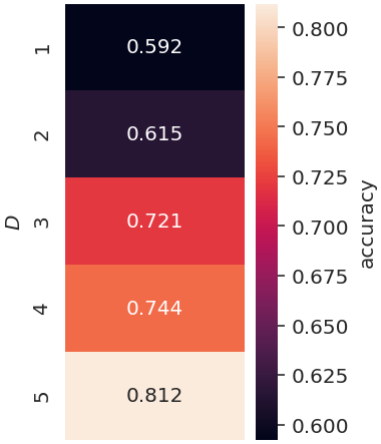




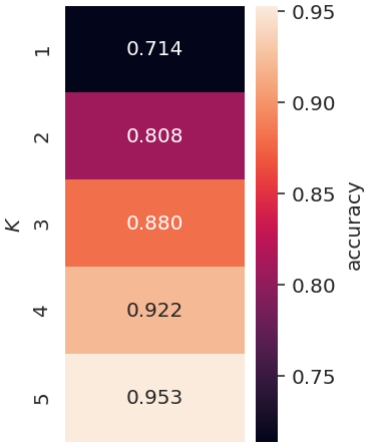
DT train accuracy w.r.t *D*



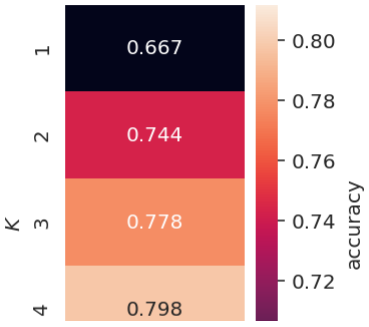
DT val accuracy w.r.t *D*

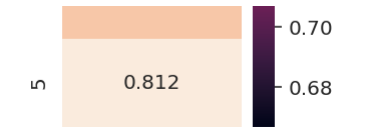


RF train accuracy w.r.t *K*

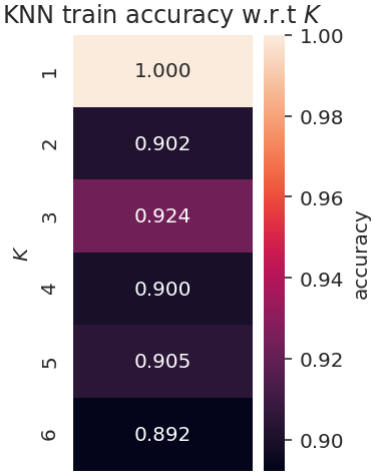


RF val accuracy w.r.t *K*

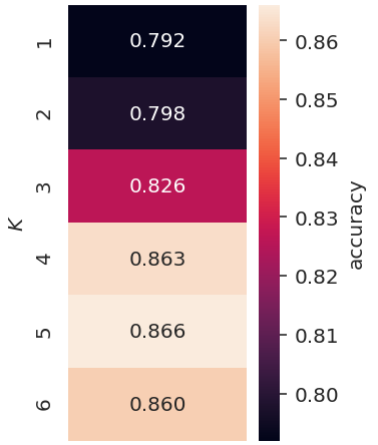




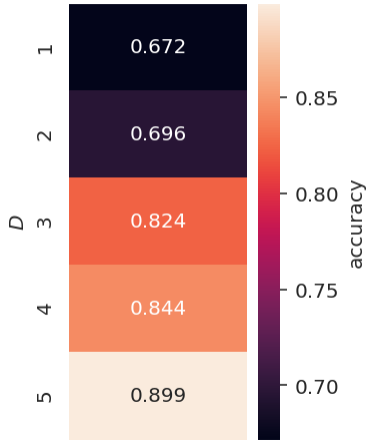
KNN train accuracy w.r.t *K*



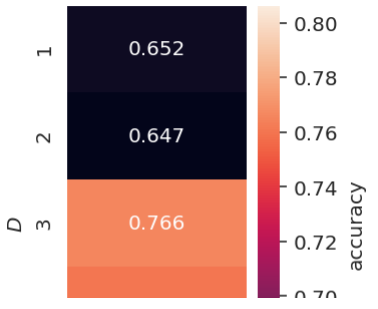
KNN val accuracy w.r.t *K*

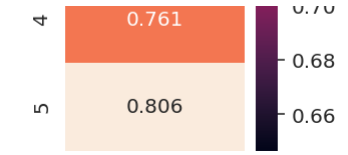


DT train accuracy w.r.t *D*

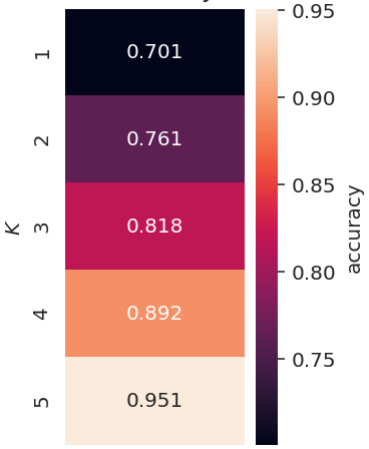


DT val accuracy w.r.t *D*

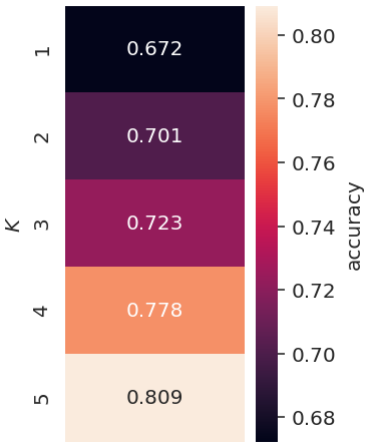




RF train accuracy w.r.t *K*

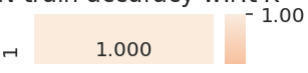


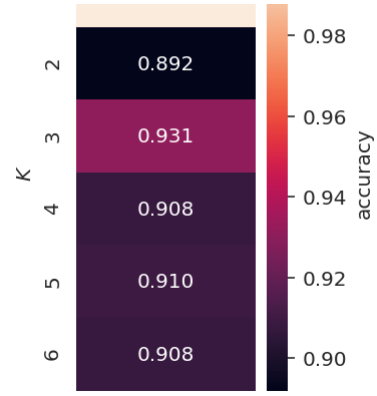
RF val accuracy w.r.t *K*



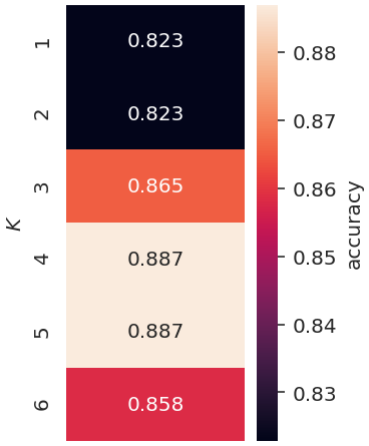

```
Test Accuracy Average for knn =  0.830857142857143
Test Accuracy Average for Random Forest =  0.8051428571428572
Test Accuracy Average for Decision Tree =  0.7902857142857143
[[0.         0.85142857 0.79857143 0.81       0.         0.
  0.        ]
 [0.         0.83085714 0.79028571 0.80514286 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0.         0.89082187 0.90062937 0.9371709  0.         0.
  0.        ]
 [0.         0.90456024 0.89885104 0.95085155 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 5. 5. 0. 0. 0.]
 [0. 5. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.2
```

KNN train accuracy w.r.t *K*

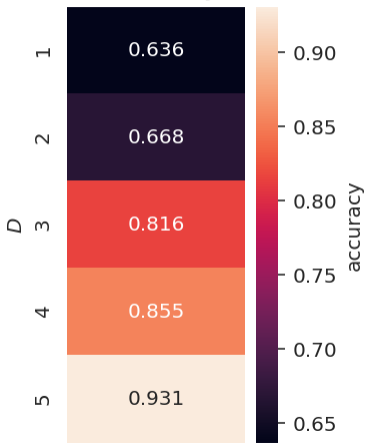




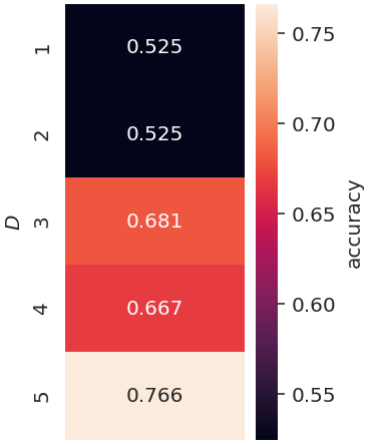
KNN val accuracy w.r.t *K*



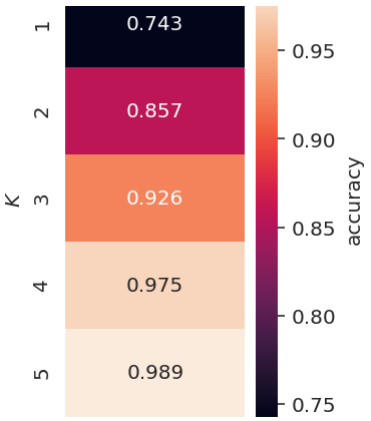
DT train accuracy w.r.t *D*



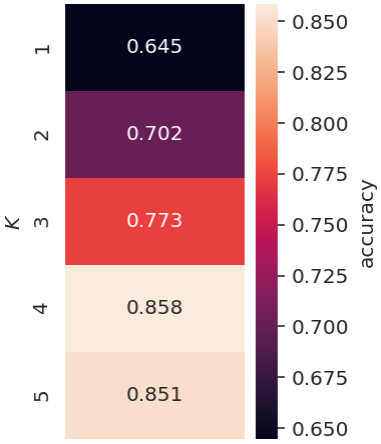
DT val accuracy w.r.t *D*



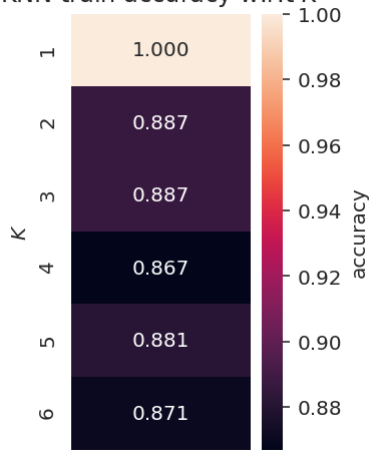
RF train accuracy w.r.t *K*



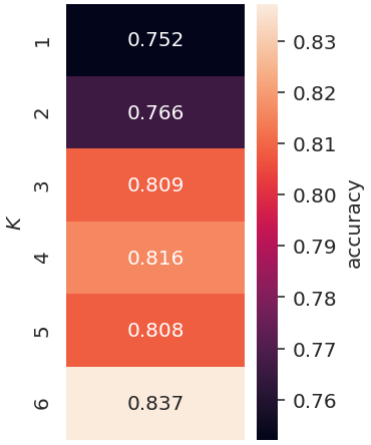
RF val accuracy w.r.t *K*



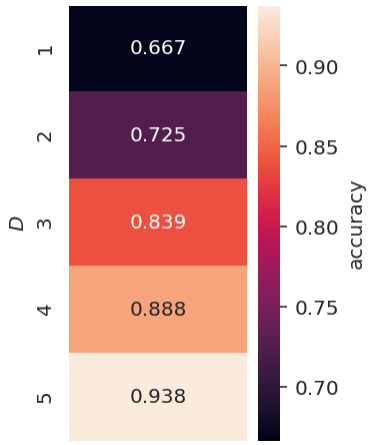
KNN train accuracy w.r.t *K*



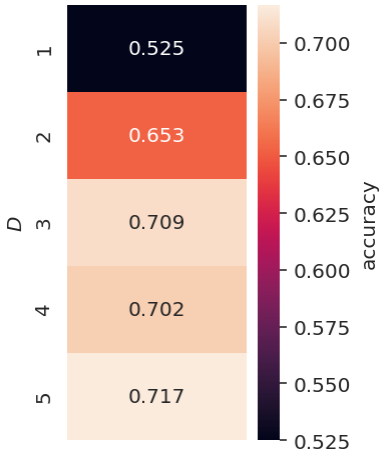
KNN val accuracy w.r.t *K*



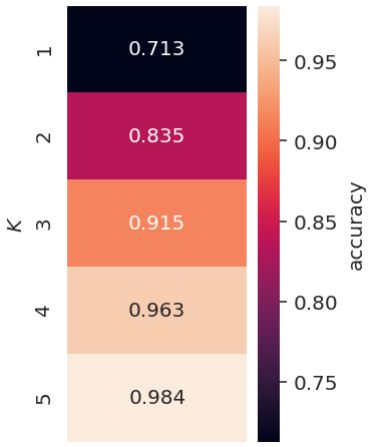
DT train accuracy w.r.t *D*



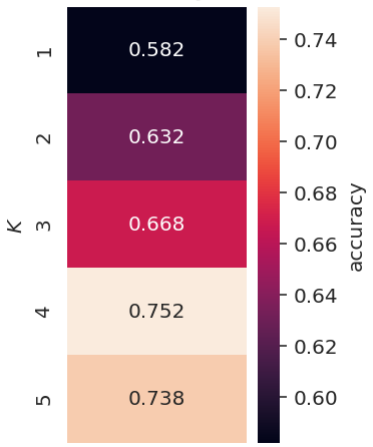
DT val accuracy w.r.t *D*



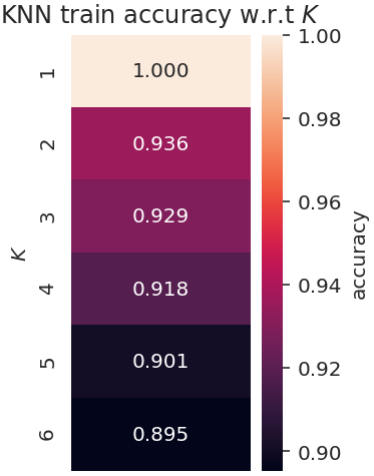
RF train accuracy w.r.t *K*



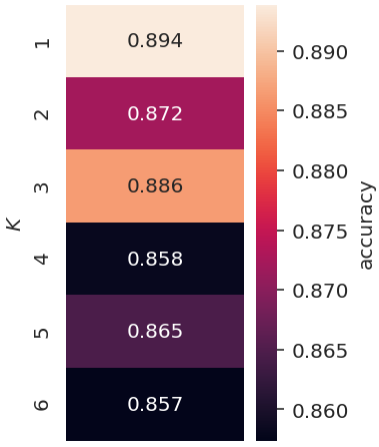
RF val accuracy w.r.t *K*



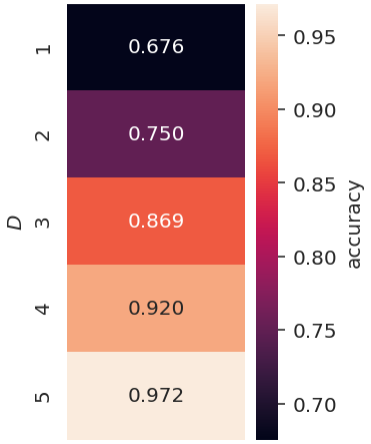
KNN train accuracy w.r.t *K*



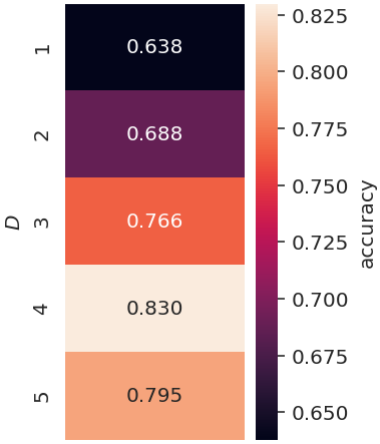
KNN val accuracy w.r.t *K*

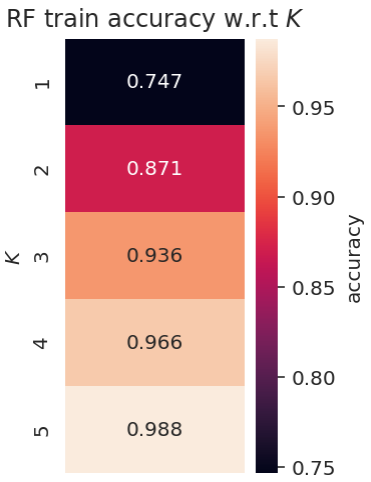


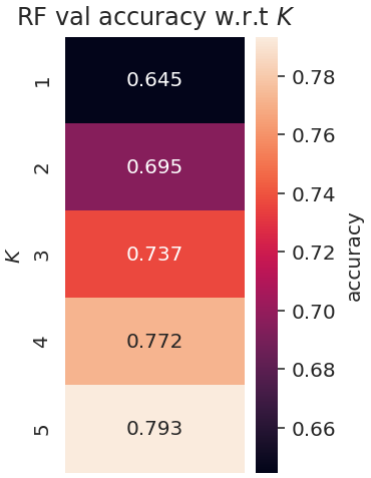
DT train accuracy w.r.t *D*

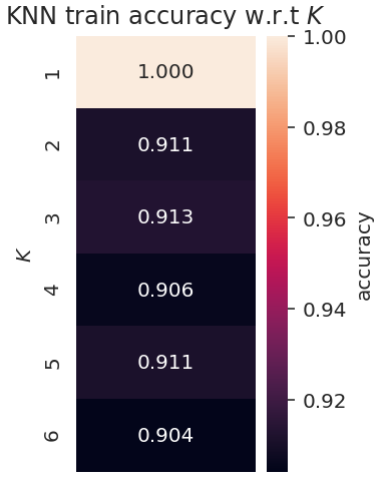


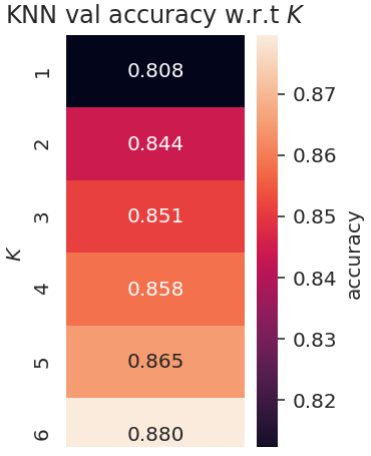
DT val accuracy w.r.t *D*



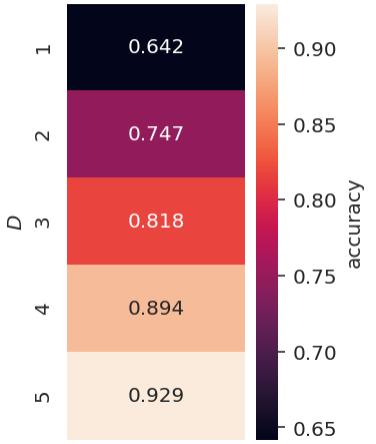

RF train accuracy w.r.t K
0.747
0.871
0.936
0.966
0.988
K
1
2
3
4
5
accuracy
0.95
0.90
0.85
0.80
0.75


RF val accuracy w.r.t K
0.645
0.695
0.737
0.772
0.793
K
1
2
3
4
5
accuracy
0.78
0.76
0.74
0.72
0.70
0.68
0.66

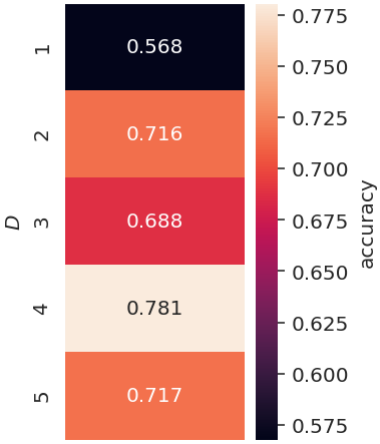

KNN train accuracy w.r.t K
1.000
0.911
0.913
0.906
0.911
0.904
K
1
2
3
4
5
6
accuracy
1.00
0.98
0.96
0.94
0.92


KNN val accuracy w.r.t K
0.808
0.844
0.851
0.858
0.865
0.880
K
1
2
3
4
5
6
accuracy
0.87
0.86
0.85
0.84
0.83
0.82

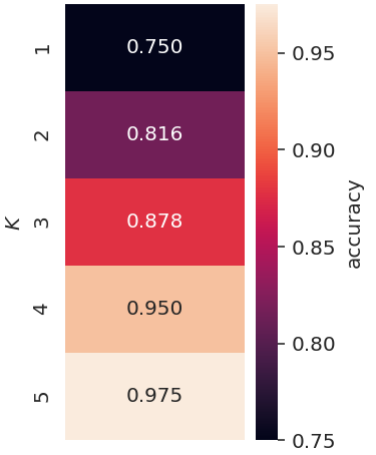
DT train accuracy w.r.t *D*



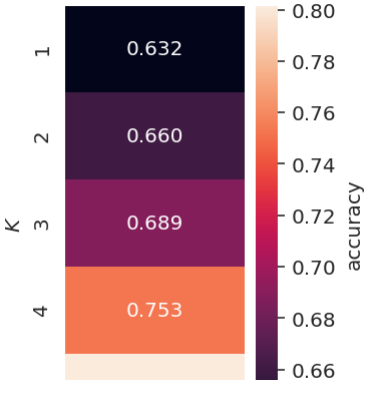
DT val accuracy w.r.t *D*

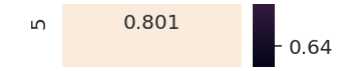


RF train accuracy w.r.t *K*

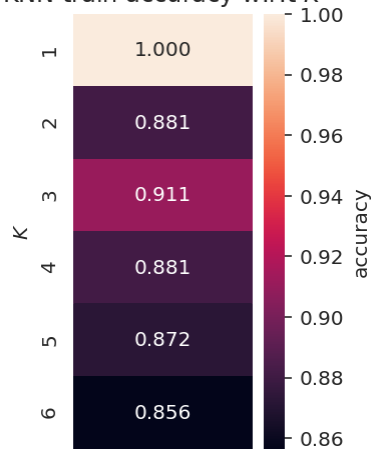


RF val accuracy w.r.t *K*

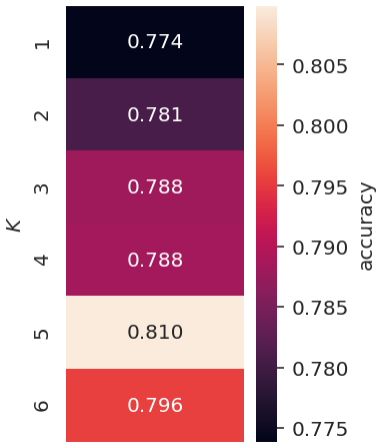




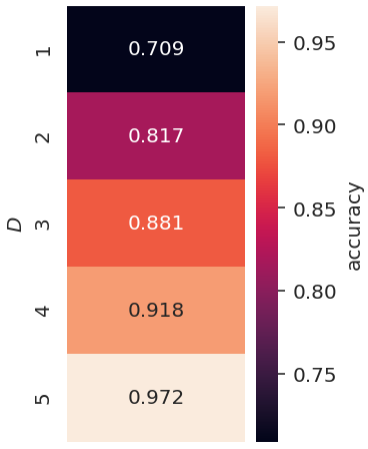
KNN train accuracy w.r.t *K*



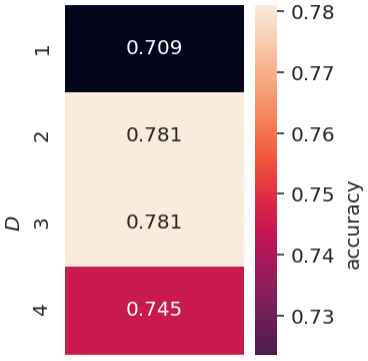
KNN val accuracy w.r.t *K*



DT train accuracy w.r.t *D*



DT val accuracy w.r.t *D*



Running Respective P and T Tests to compare the Trials Accuracy For Each Algorithm

RF train accuracy w.r.t K

```
#Compute the difference between the results
print("P and T Test for KNN trials vs. Decision Tree")
diff = [y - x for y, x in zip(knn_test_acc, decision_tree_test_acc)]

#Compute the mean of differences
d_bar = np.mean(diff)

#Compute the variance of differences
sigma2 = np.var(diff)

#compute the number of data points used for training
n1 = len(Y_train_val)

#compute the number of data points used for testing
n2 = len(Y_test_val)

#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for KNN trials vs. Decision Tree
This is the T Value
0.535669674774904
This is the P Value
0.2961053479677722
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. Decision Tree")
diff = [y - x for y, x in zip(rand_forest_test_acc, decision_tree_test_acc)]
```

```
diff = [y - x for y, x in zip(rand_forest_test_acc, decision_tree_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. Decision Tree
This is the T Value
0.1336369280258507
This is the P Value
0.44684753049812354
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. KNN")
diff = [y - x for y, x in zip(rand_forest_test_acc, knn_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
```

```
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. KNN
This is the T Value
-0.6711435814323239
This is the P Value
0.7489199627332797
Accept the null hypothesis
```

Spam Email Dataset

This given dataset provides 3 various attributes that give us the necessary information to analyze whether an email is spam or not. The reasoning behind using this dataset is for a basis for a real world scenario of analyzing spam email on the basis from knowing just the messages and the file names.

```
se_data_preserved = pd.read_csv('SpamEmail.csv')
se_data = se_data_preserved #Temp for manipulation
se_data #preview of dataset

#Basic Cleanup
se_data.dropna(inplace=True)
se_data.shape
se_data.head()
```

	CATEGORY	MESSAGE	FILE_NAME
0	1	Dear Homeowner,\n\n \n\nInterest Rates are at	00249.5f45607c1bffe89f60ba1ec9f878039a
2	1	This is a multi-part message in MIME	00214.1367039e50dc6b7adb0f2aa8aba83216

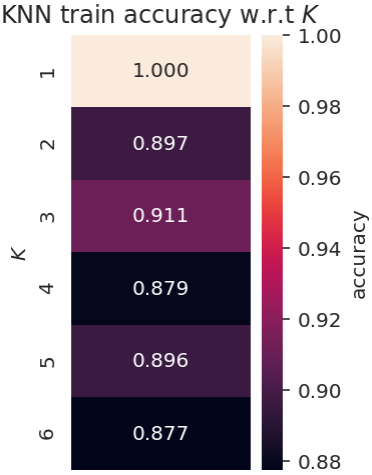
```
# Encoded each of the categorical features so it will be good for our testing
se_data = MultiColumnLabelEncoder(columns = ['MESSAGE',
                                             'FILE_NAME']).fit_transform(se_data)
```

```
#Split Data By 80/20, 50/50, 20/80
partitionVal = [0.8, 0.5, 0.2]
result_table = np.zeros((3,7))
result_table1 = np.zeros((3,7))
result_table2 = np.zeros((3,7))
for i, partition in enumerate(partitionVal):
    print("Partition: ", partition)

    knn_test_acc = []
    rand_forest_test_acc = []
    decision_tree_test_acc = []
    svm_test_acc = []



    NUM_TRIALS = 5
    for trial in range(NUM_TRIALS):
        #Mix up the data
        students_data = students_data.sample(frac=1).reset_index(drop=True)
        #Find the point where to split the data
        breakNum = int(partition*len(students_data))

        X_train_full = students_data.loc[0:breakNum]
        X_train_val = X_train_full.drop("gender", axis = 1)
        Y_train_val = X_train_full["gender"]
        X_test_full = students_data.loc[breakNum:]
        X_test= X_test_full.drop("gender", axis = 1)
        Y_test_val = X_test_full["gender"]

        #Call the svm classifier (Took Too Long For ALL Datasets)
        #test_acc,best_train0,C0 = svm_func()
        #svm_test_acc.append(test_acc)

        #Call the knn classifier
        test_acc,best_train1,C1 = knn_classifier()
        knn_test_acc.append(test_acc)

        #Call the Decision Tree classifier
        test_acc,best_train2,C2 = decision_Tree()
        decision_tree_test_acc.append(test_acc)

        #Call the Random Forest classifier
        test_acc,best_train3,C3 = rand_Forest()
```

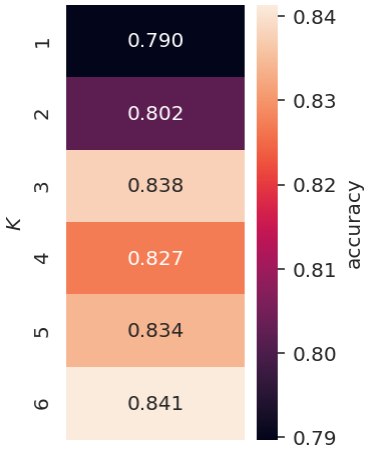
```
    test_acc,best_train3,C3 = rand_forest()
    rand_forest_test_acc.append(test_acc)


#result_table[i, 0] = sum(svm_test_acc)/NUM_TRIALS
result_table[i, 1] = sum(knn_test_acc)/NUM_TRIALS
result_table[i, 2] = sum(decision_tree_test_acc)/NUM_TRIALS
result_table[i, 3] = sum(rand_forest_test_acc)/NUM_TRIALS


#result_table1[i, 0] = best_train0
result_table1[i, 1] = best_train1
result_table1[i, 2] = best_train2
result_table1[i, 3] = best_train3


#result_table2[i, 0] = C0
result_table2[i, 1] = C1
result_table2[i, 2] = C2
result_table2[i, 3] = C3


#Average all test accuracies for all 5 trials
print("Test Accuracy Average for knn = ", sum(knn_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Random Forest = ",
      sum(rand_forest_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Decision Tree = ",
      sum(decision_tree_test_acc)/NUM_TRIALS)

#print("Test Accuracy Average for SVM = ", sum(svm_test_acc)/NUM_TRIALS)

#y-axis: partition
#x-axis: classifier
print(result_table)
print("############################")
print(result_table1)
print("############################")
print(result_table2)
```

```
Partition:  0.8
```

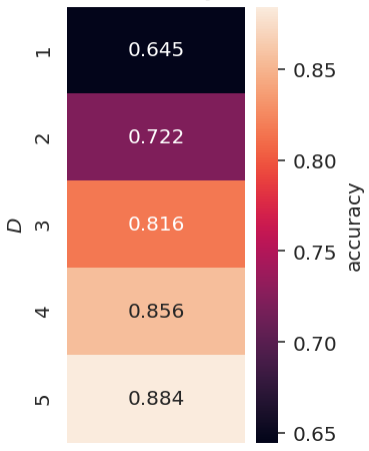
KNN train accuracy w.r.t *K*



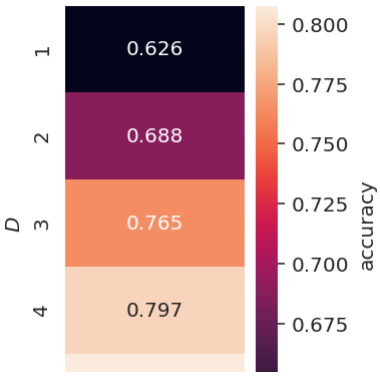
KNN val accuracy w.r.t *K*



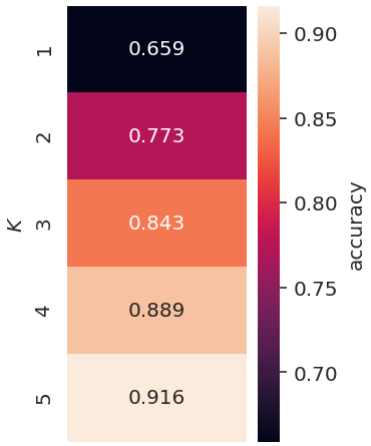
DT train accuracy w.r.t *D*



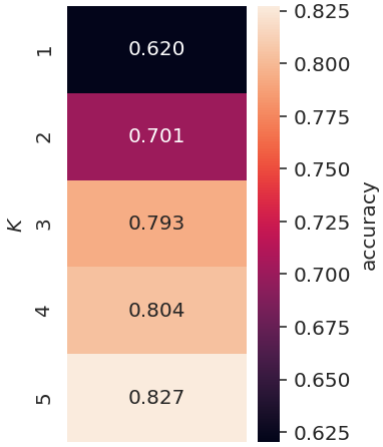
DT val accuracy w.r.t *D*



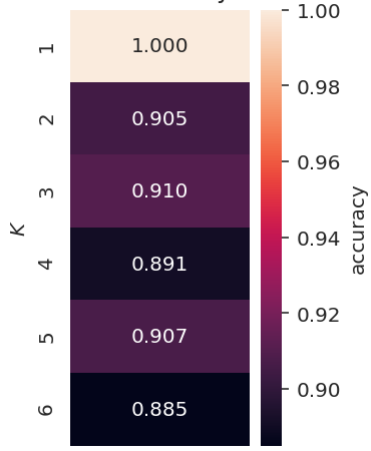
RF train accuracy w.r.t *K*



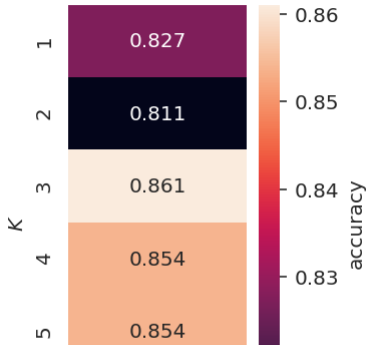
RF val accuracy w.r.t *K*

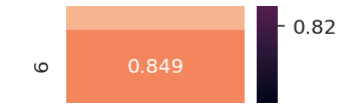


KNN train accuracy w.r.t *K*

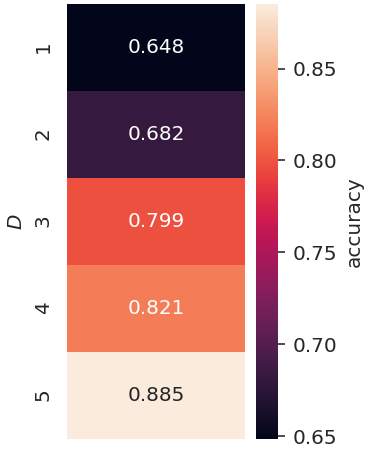


KNN val accuracy w.r.t *K*

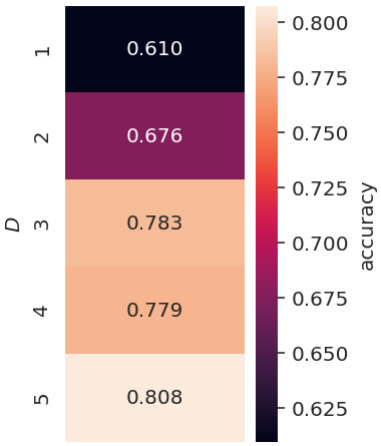




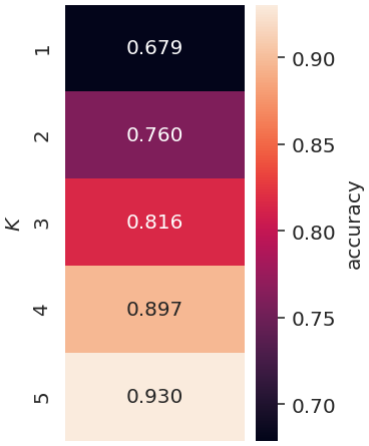
DT train accuracy w.r.t *D*



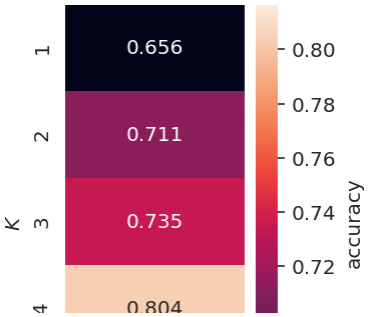
DT val accuracy w.r.t *D*

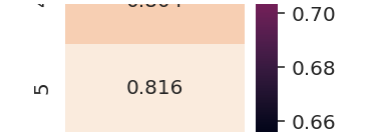


RF train accuracy w.r.t *K*

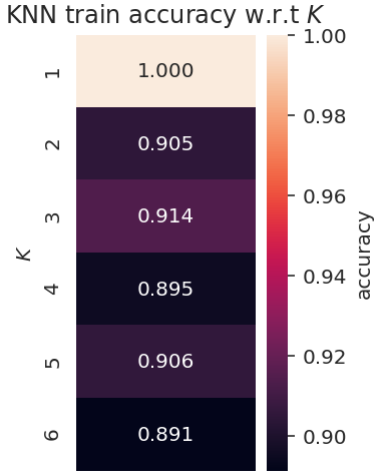


RF val accuracy w.r.t *K*

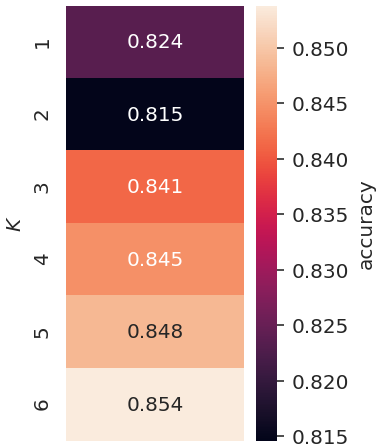




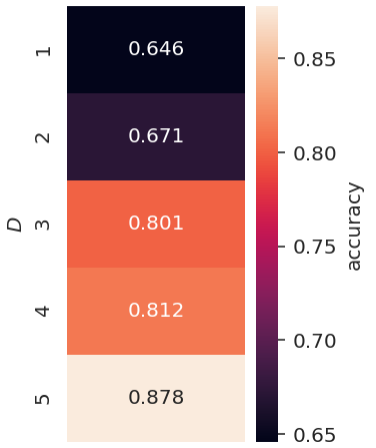
KNN train accuracy w.r.t *K*



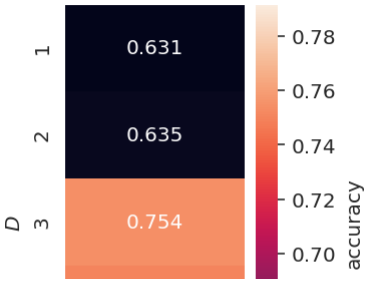
KNN val accuracy w.r.t *K*

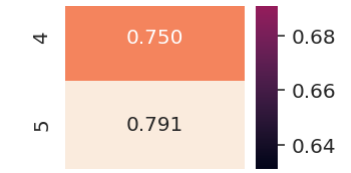


DT train accuracy w.r.t *D*

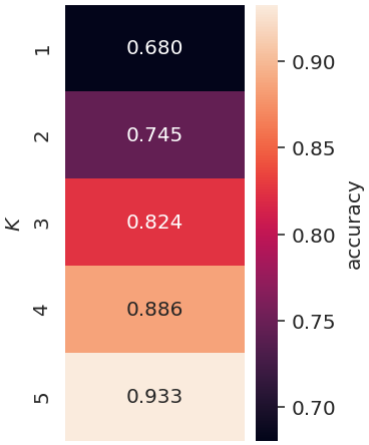


DT val accuracy w.r.t *D*

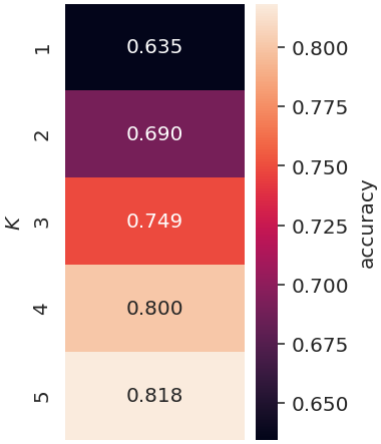




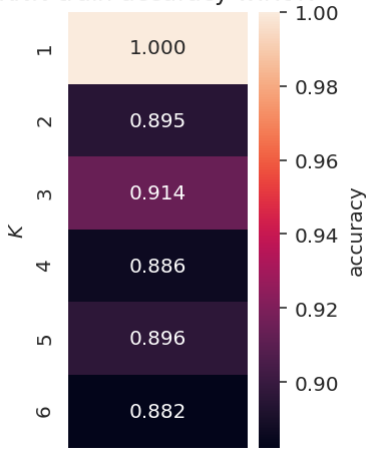
RF train accuracy w.r.t *K*



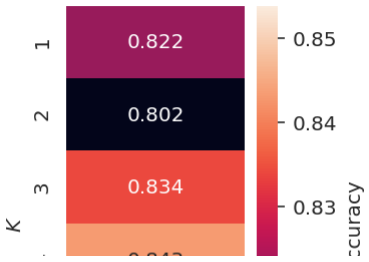
RF val accuracy w.r.t *K*

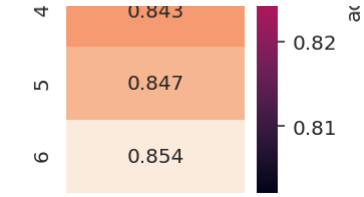


KNN train accuracy w.r.t *K*

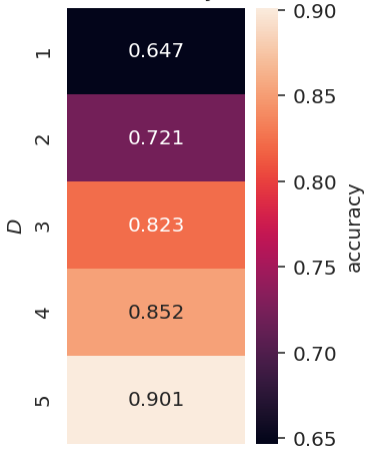


KNN val accuracy w.r.t *K*

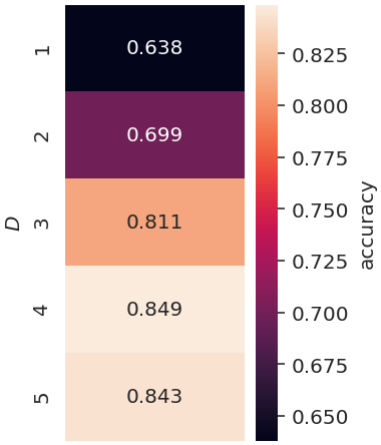




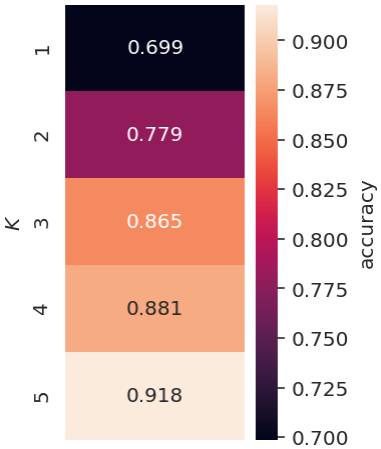
DT train accuracy w.r.t *D*



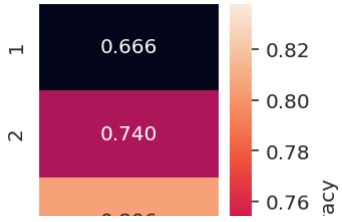
DT val accuracy w.r.t *D*

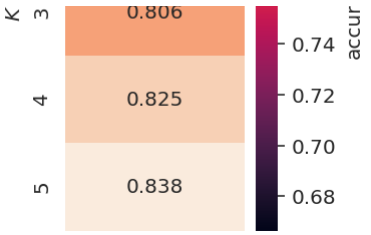


RF train accuracy w.r.t *K*

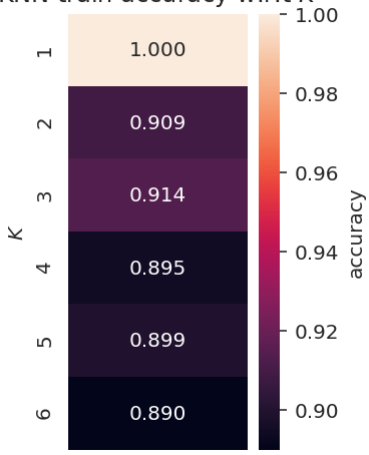


RF val accuracy w.r.t *K*

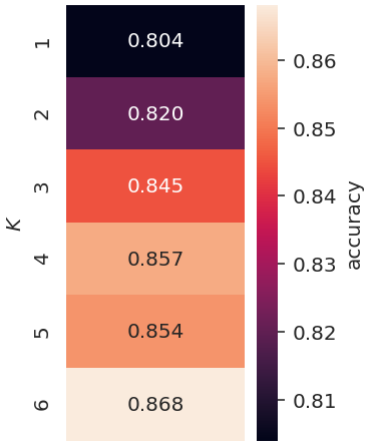




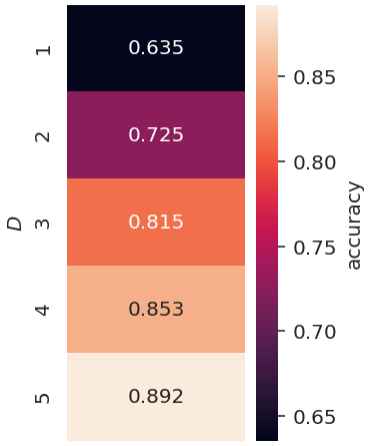
KNN train accuracy w.r.t *K*



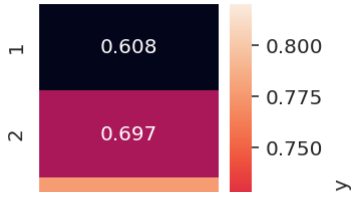
KNN val accuracy w.r.t *K*

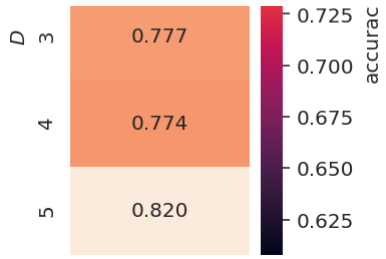


DT train accuracy w.r.t *D*

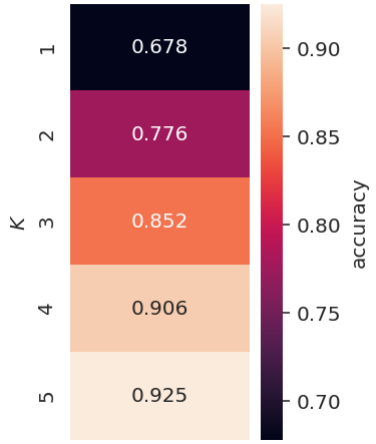


DT val accuracy w.r.t *D*

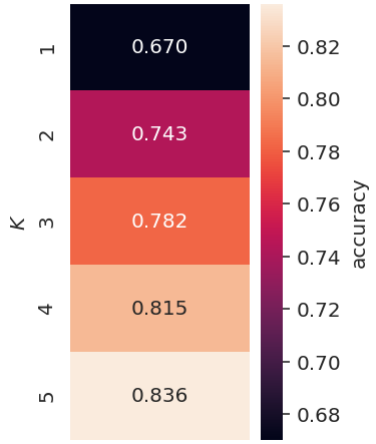




RF train accuracy w.r.t *K*

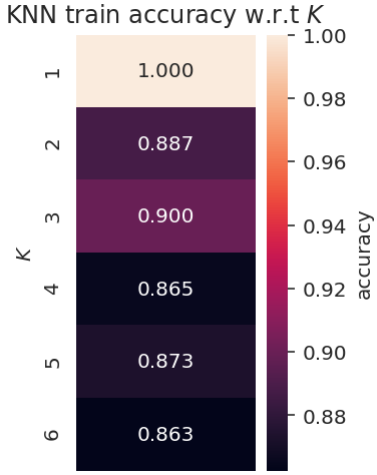


RF val accuracy w.r.t *K*

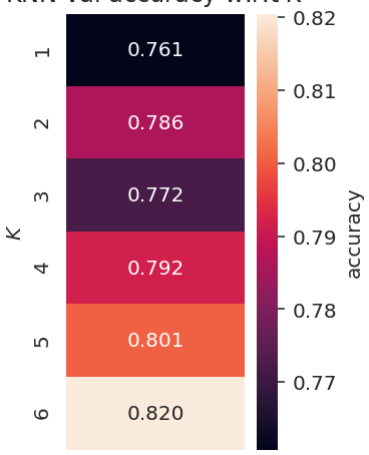



```
Test Accuracy Average for knn =  0.8442857142857143
Test Accuracy Average for Random Forest =  0.8242857142857142
Test Accuracy Average for Decision Tree =  0.8057142857142857
[[0.         0.84428571 0.80571429 0.82428571 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0.         0.88993299 0.89215519 0.92513423 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.5
```

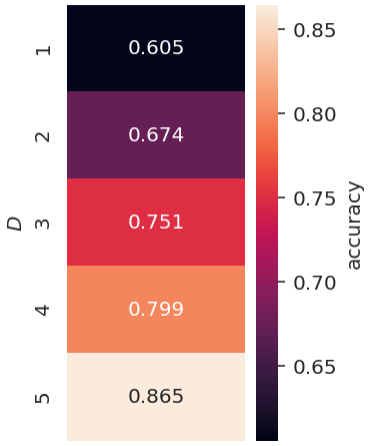
KNN train accuracy w.r.t *K*



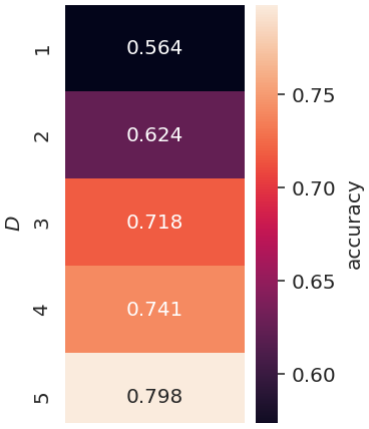
KNN val accuracy w.r.t *K*

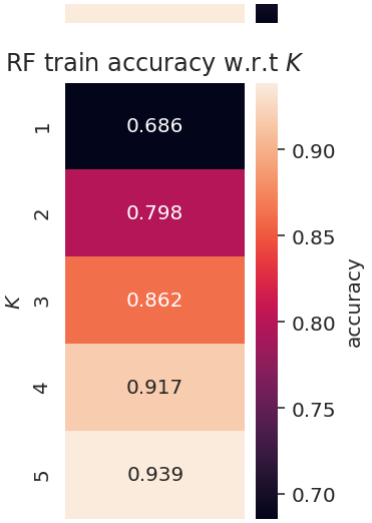


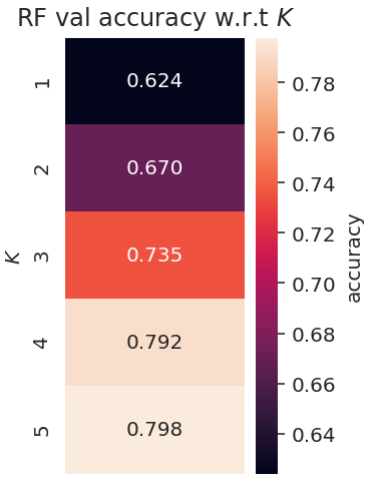
DT train accuracy w.r.t *D*

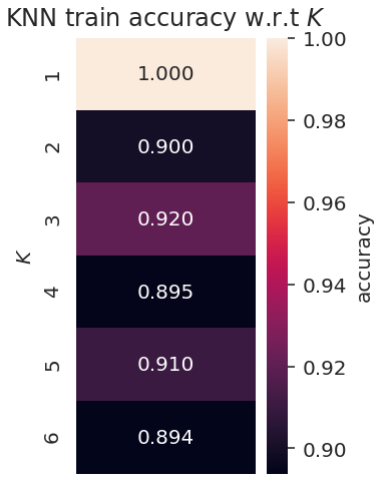


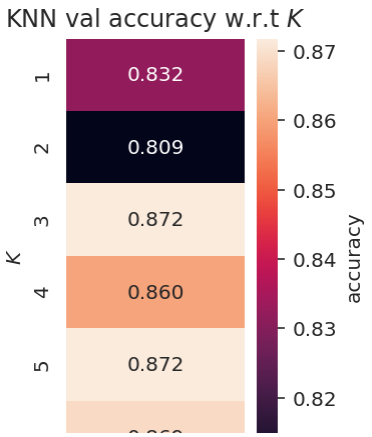
DT val accuracy w.r.t *D*

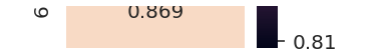



RF train accuracy w.r.t K
K
1
2
3
4
5
0.686
0.798
0.862
0.917
0.939
accuracy
0.90
0.85
0.80
0.75
0.70

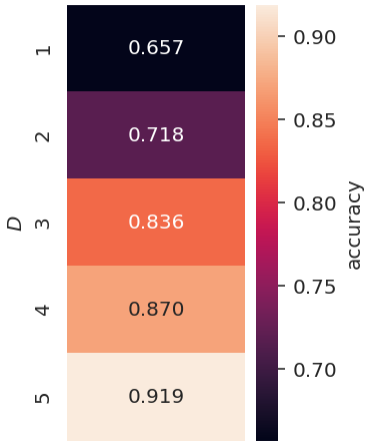

RF val accuracy w.r.t K
K
1
2
3
4
5
0.624
0.670
0.735
0.792
0.798
accuracy
0.78
0.76
0.74
0.72
0.70
0.68
0.66
0.64


KNN train accuracy w.r.t K
K
1
2
3
4
5
6
1.000
0.900
0.920
0.895
0.910
0.894
accuracy
1.00
0.98
0.96
0.94
0.92
0.90

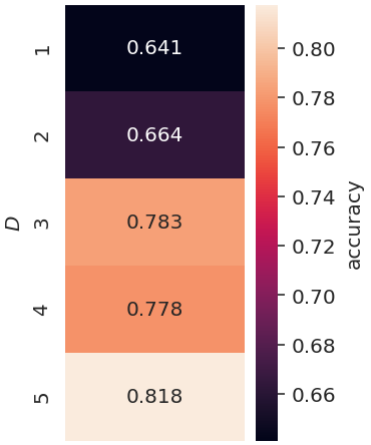

KNN val accuracy w.r.t K
K
1
2
3
4
5
0.832
0.809
0.872
0.860
0.872
accuracy
0.87
0.86
0.85
0.84
0.83
0.82



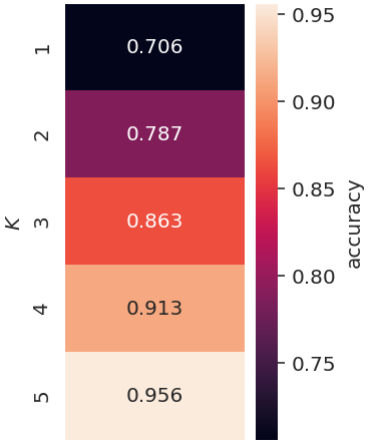
DT train accuracy w.r.t *D*



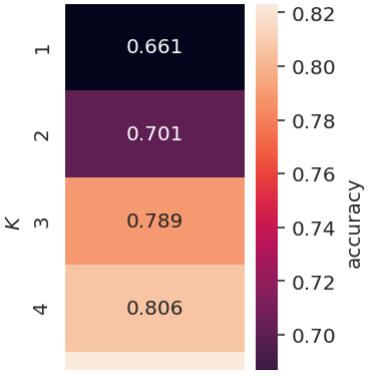
DT val accuracy w.r.t *D*



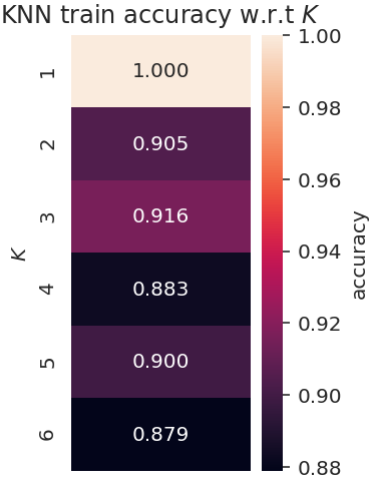
RF train accuracy w.r.t *K*



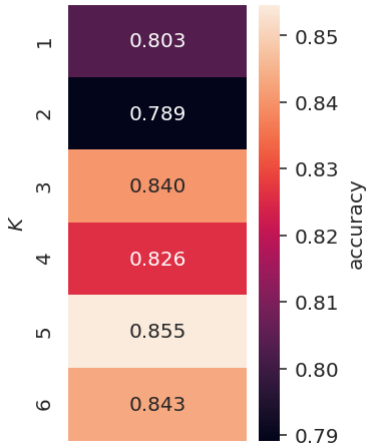
RF val accuracy w.r.t *K*



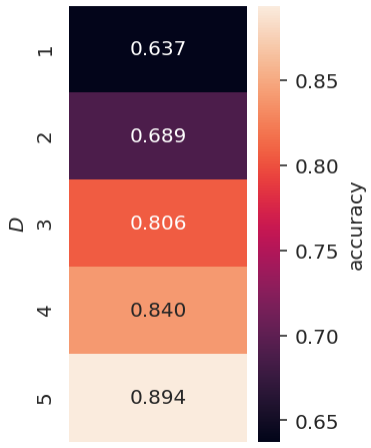
KNN train accuracy w.r.t *K*



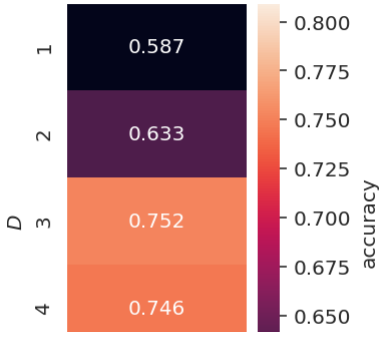
KNN val accuracy w.r.t *K*



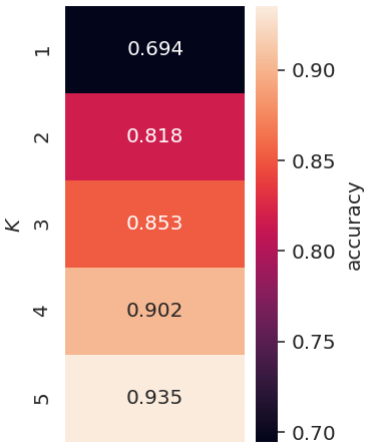
DT train accuracy w.r.t *D*



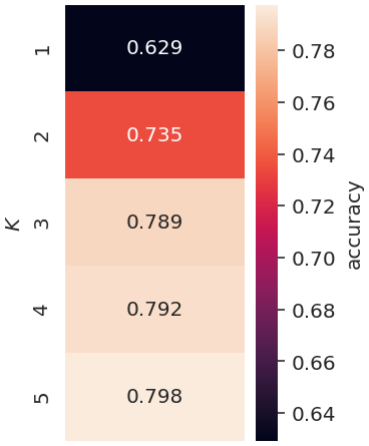
DT val accuracy w.r.t *D*



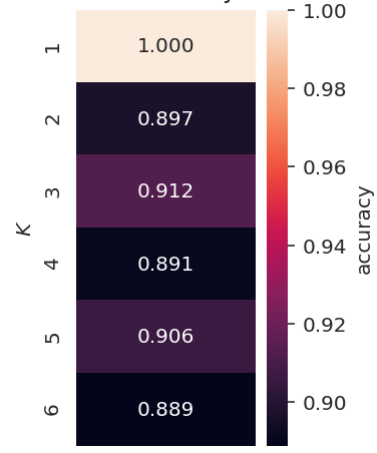
RF train accuracy w.r.t *K*



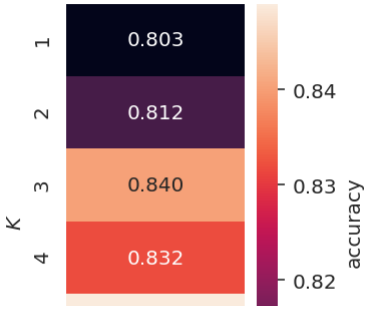
RF val accuracy w.r.t *K*

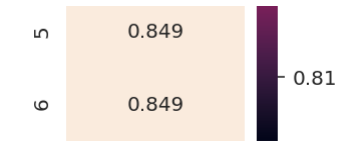


KNN train accuracy w.r.t *K*

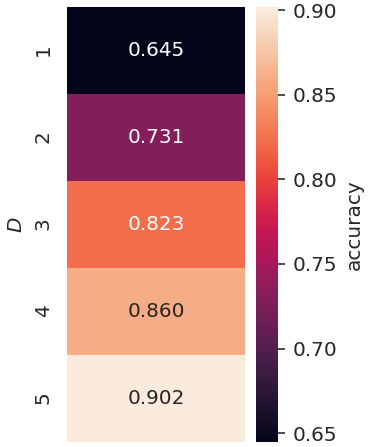


KNN val accuracy w.r.t *K*

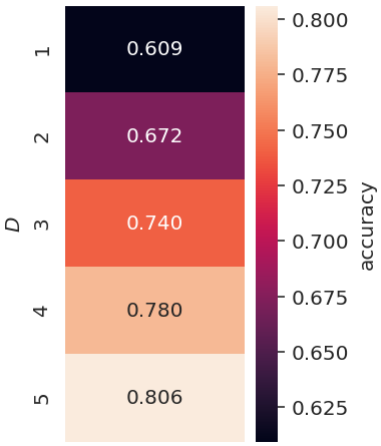




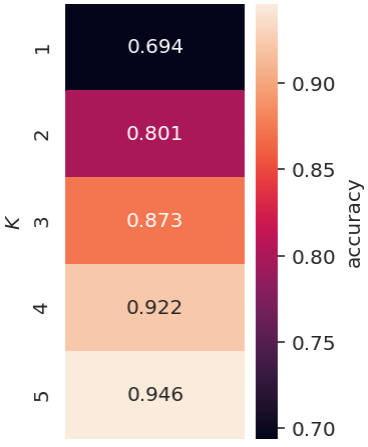
DT train accuracy w.r.t *D*



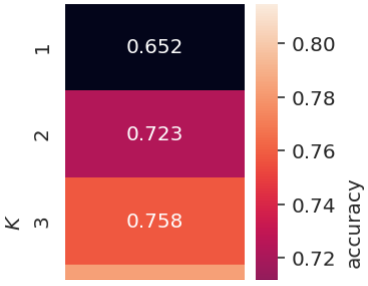
DT val accuracy w.r.t *D*

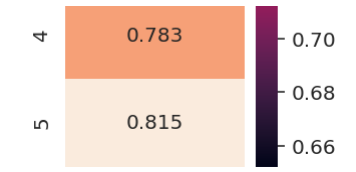


RF train accuracy w.r.t *K*

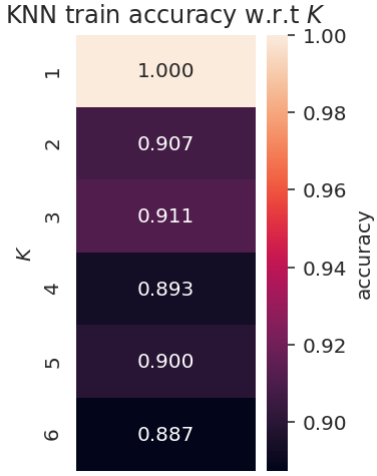


RF val accuracy w.r.t *K*

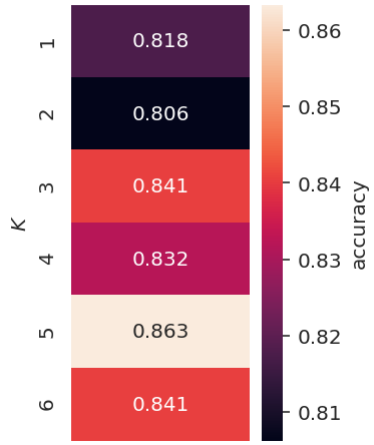




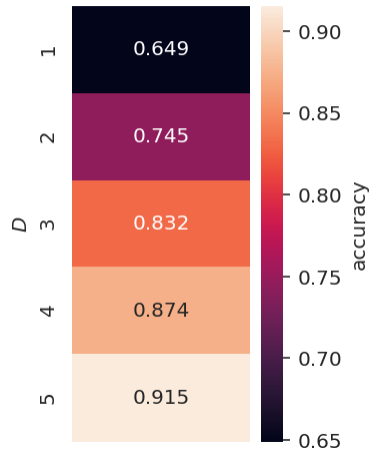
KNN train accuracy w.r.t *K*



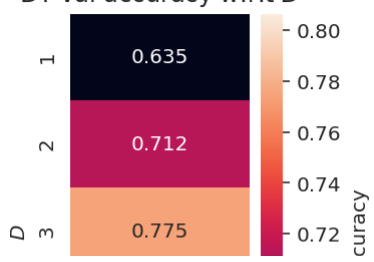
KNN val accuracy w.r.t *K*

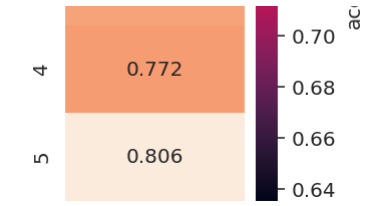


DT train accuracy w.r.t *D*

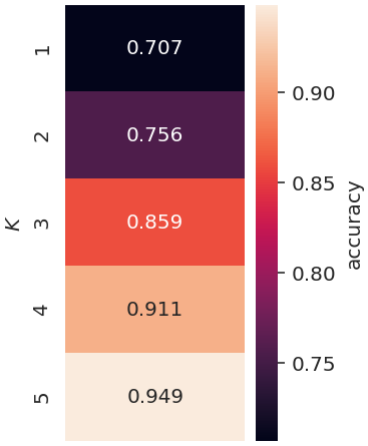


DT val accuracy w.r.t *D*

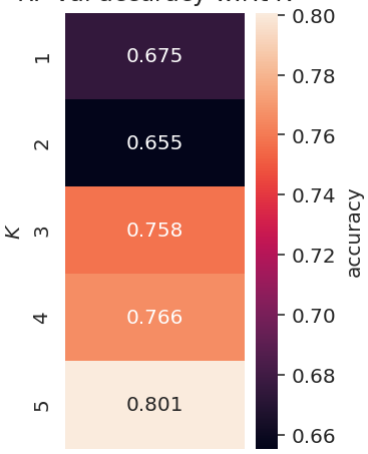




RF train accuracy w.r.t *K*

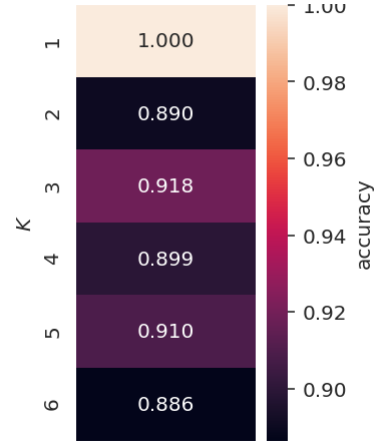


RF val accuracy w.r.t *K*

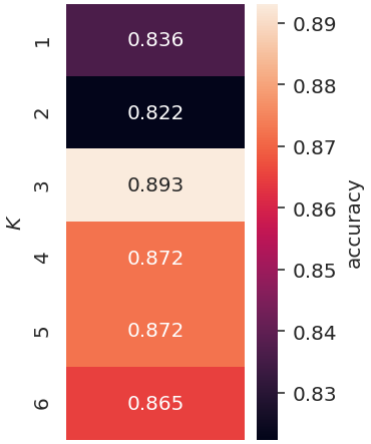



```
Test Accuracy Average for knn =  0.8377142857142857
Test Accuracy Average for Random Forest =  0.8211428571428572
Test Accuracy Average for Decision Tree =  0.8297142857142857
[[0.         0.84428571 0.80571429 0.82428571 0.         0.
  0.        ]
 [0.         0.83771429 0.82971429 0.82114286 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0.         0.88993299 0.89215519 0.92513423 0.         0.
  0.        ]
 [0.         0.89957804 0.91524657 0.94872903 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 6. 5. 5. 0. 0. 0.]
 [0. 5. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.2
```

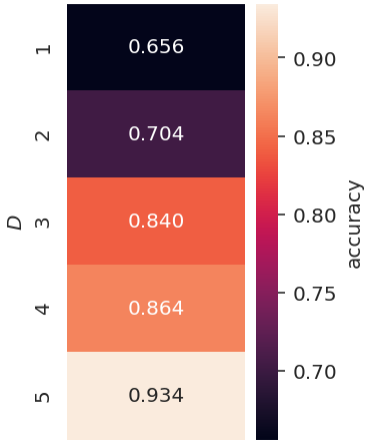
KNN train accuracy w.r.t *K*



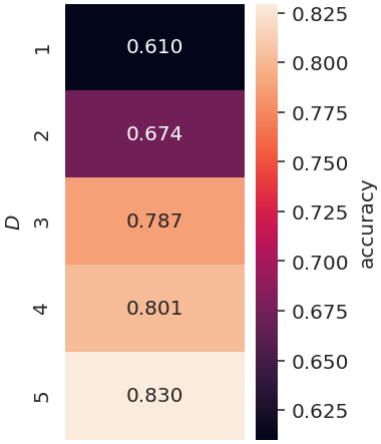
KNN val accuracy w.r.t *K*



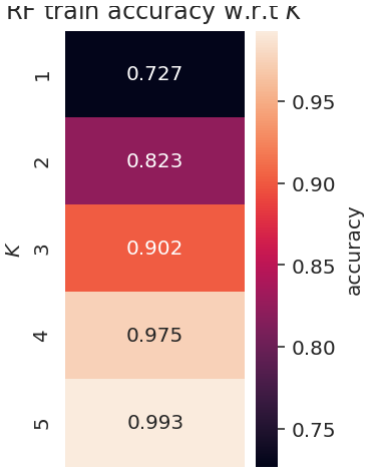
DT train accuracy w.r.t *D*

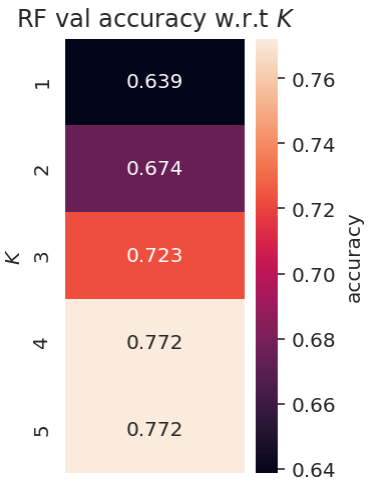


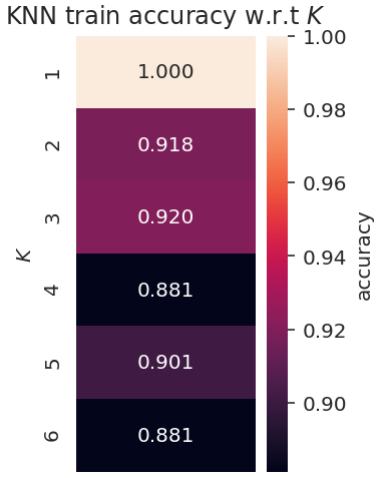
DT val accuracy w.r.t *D*

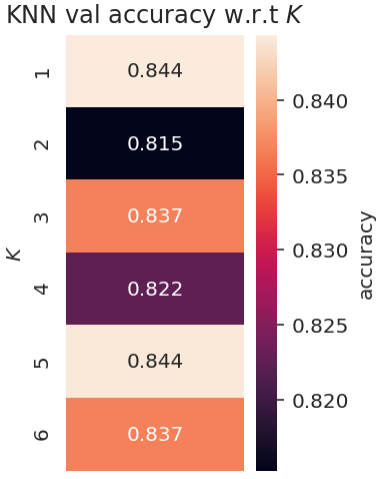


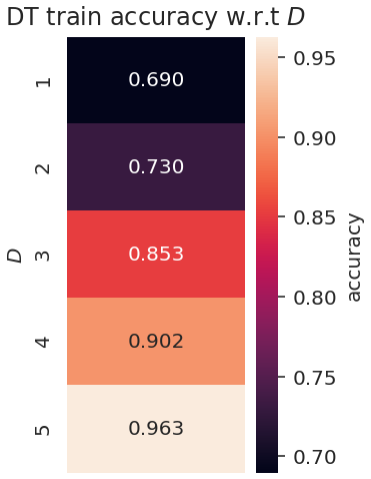
RF train accuracy w.r.t *K*

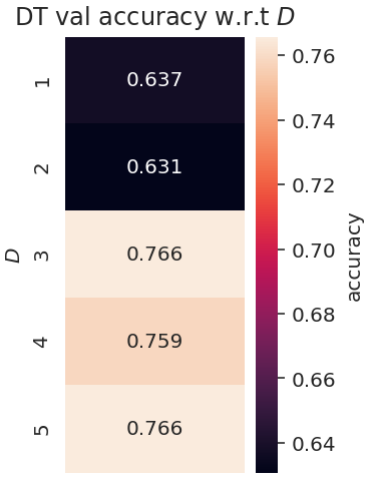

RF train accuracy w.r.t K
K
1
2
3
4
5
0.727
0.823
0.902
0.975
0.993
accuracy
0.95
0.90
0.85
0.80
0.75

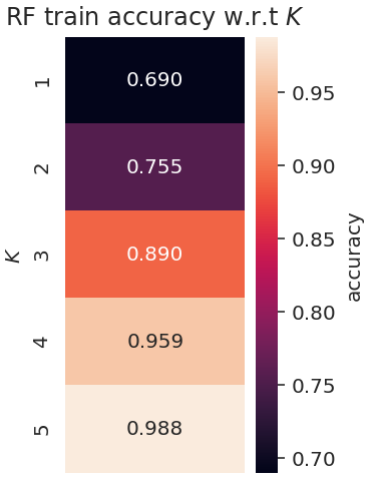

RF val accuracy w.r.t K
K
1
2
3
4
5
0.639
0.674
0.723
0.772
0.772
accuracy
0.76
0.74
0.72
0.70
0.68
0.66
0.64

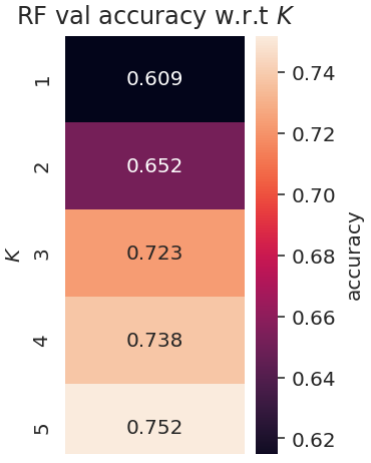

KNN train accuracy w.r.t K
K
1
2
3
4
5
6
1.000
0.918
0.920
0.881
0.901
0.881
accuracy
1.00
0.98
0.96
0.94
0.92
0.90


KNN val accuracy w.r.t K
K
1
2
3
4
5
6
0.844
0.815
0.837
0.822
0.844
0.837
accuracy
0.840
0.835
0.830
0.825
0.820

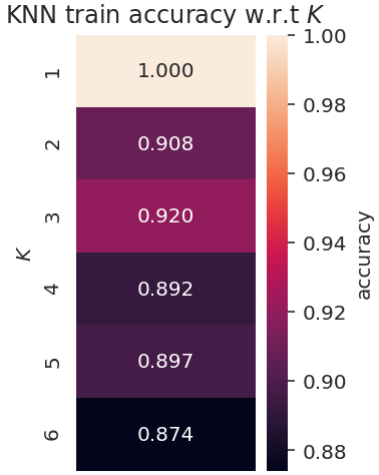

DT train accuracy w.r.t D
D
1
2
3
4
5
0.690
0.730
0.853
0.902
0.963
accuracy
0.95
0.90
0.85
0.80
0.75
0.70


DT val accuracy w.r.t D
D
1
2
3
4
5
0.637
0.631
0.766
0.759
0.766
accuracy
0.76
0.74
0.72
0.70
0.68
0.66
0.64

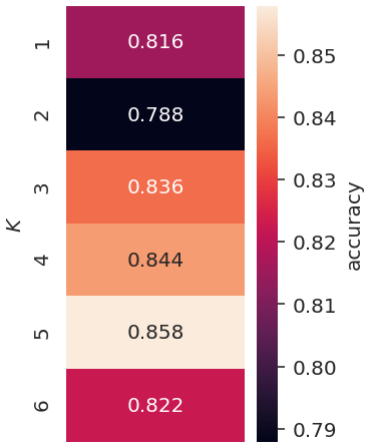

RF train accuracy w.r.t K
K
1
2
3
4
5
0.690
0.755
0.890
0.959
0.988
accuracy
0.95
0.90
0.85
0.80
0.75
0.70


RF val accuracy w.r.t K
K
1
2
3
4
5
0.609
0.652
0.723
0.738
0.752
accuracy
0.74
0.72
0.70
0.68
0.66
0.64
0.62

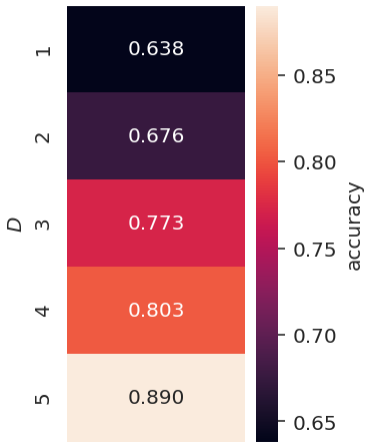
KNN train accuracy w.r.t *K*



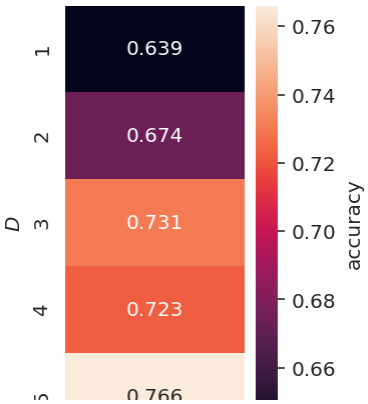
KNN val accuracy w.r.t *K*

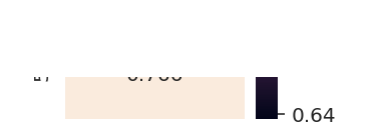


DT train accuracy w.r.t *D*

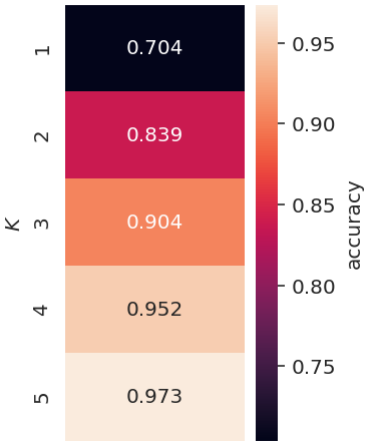


DT val accuracy w.r.t *D*

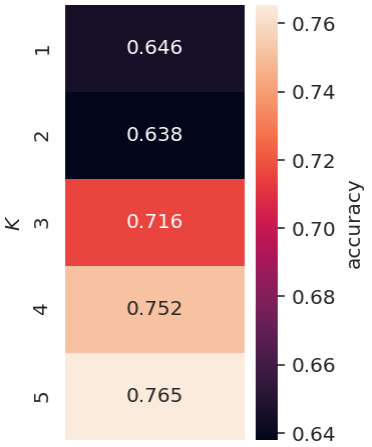




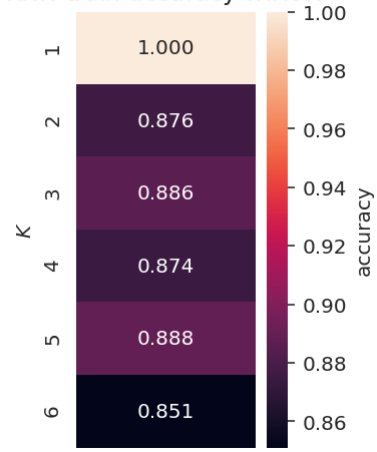
RF train accuracy w.r.t *K*



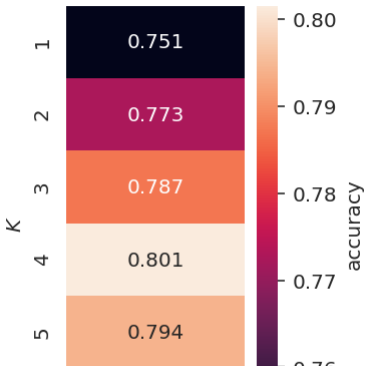
RF val accuracy w.r.t *K*



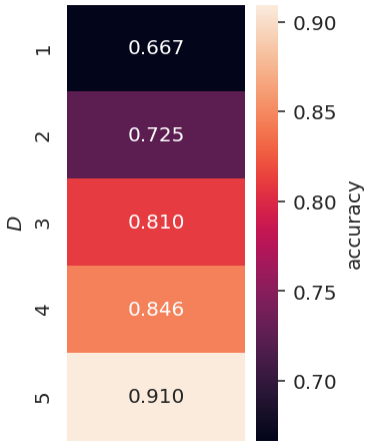
KNN train accuracy w.r.t *K*



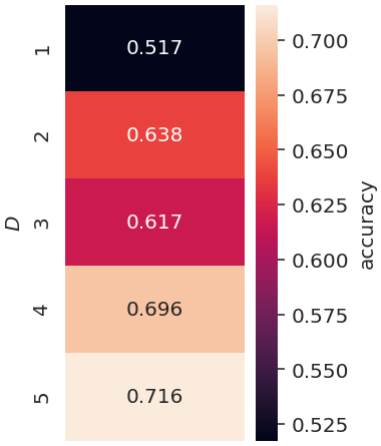
KNN val accuracy w.r.t *K*



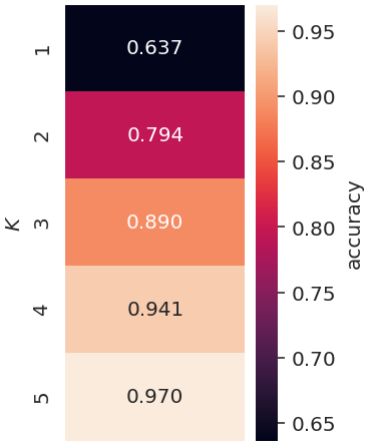
DT train accuracy w.r.t *D*



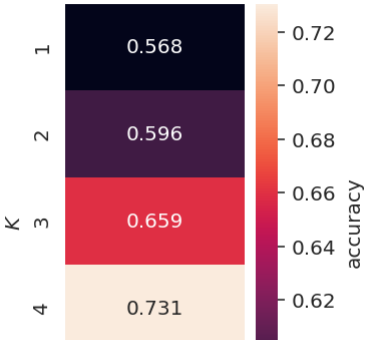
DT val accuracy w.r.t *D*



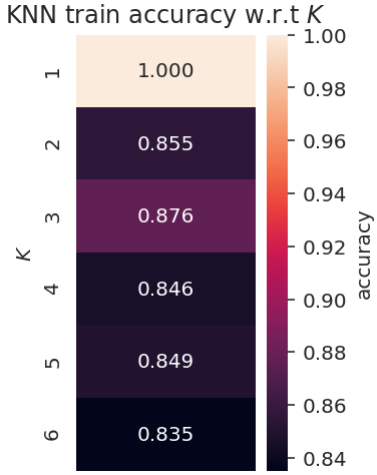
RF train accuracy w.r.t *K*



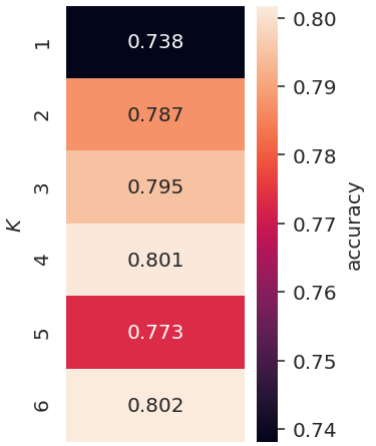
RF val accuracy w.r.t *K*



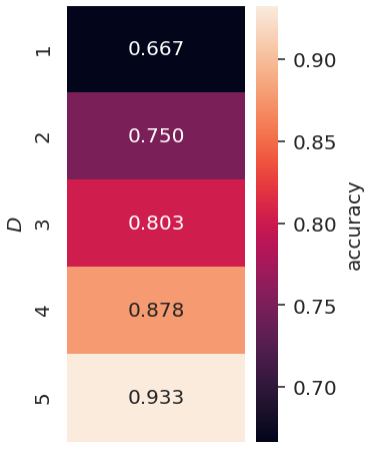
KNN train accuracy w.r.t *K*



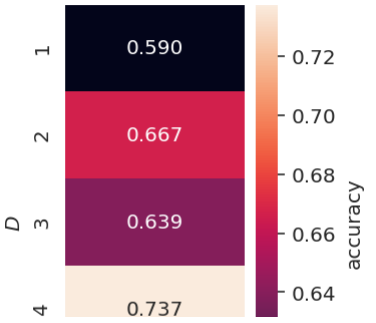
KNN val accuracy w.r.t *K*

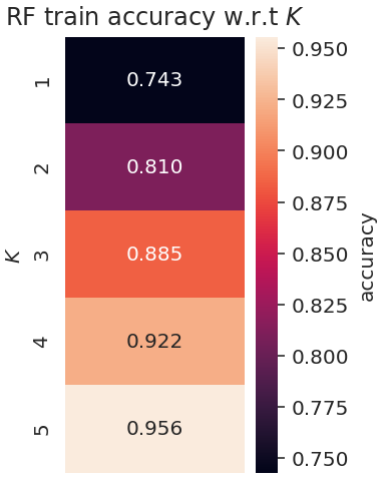


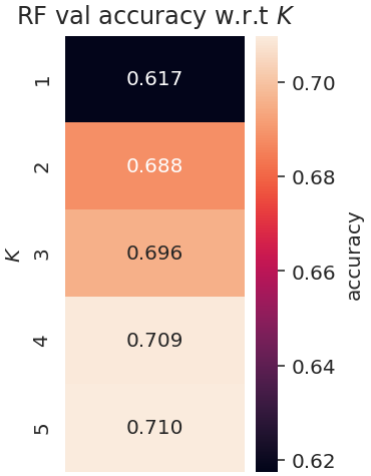
DT train accuracy w.r.t *D*



DT val accuracy w.r.t *D*








```
Test Accuracy Average for knn =  0.8146428571428572
Test Accuracy Average for Random Forest =  0.7460714285714285
Test Accuracy Average for Decision Tree =  0.7592857142857143
[[0.         0.84428571 0.80571429 0.82428571 0.         0.
  0.        ]
 [0.         0.83771429 0.82971429 0.82114286 0.         0.
  0.        ]
 [0.         0.81464286 0.75928571 0.74607143 0.         0.
  0.        ]]
##############################
[[0.         0.88993299 0.89215519 0.92513423 0.         0.
  0.        ]
```

Running Respective P and T Tests to compare the Trials Accuracy For Each Algorithm

```
#Compute the difference between the results
print("P and T Test for KNN trials vs. Decision Tree")
diff = [y - x for y, x in zip(knn_test_acc, decision_tree_test_acc)]

#Compute the mean of differences
d_bar = np.mean(diff)

#Compute the variance of differences
sigma2 = np.var(diff)
```

```
#compute the number of data points used for training
n1 = len(Y_train_val)

#compute the number of data points used for testing
n2 = len(Y_test_val)

#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for KNN trials vs. Decision Tree
This is the T Value
0.7557676986109552
This is the P Value
0.22491204793614483
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. Decision Tree")
diff = [y - x for y, x in zip(rand_forest_test_acc, decision_tree_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
```

```
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. Decision Tree
This is the T Value
-0.36298398350832245
This is the P Value
0.6416839022251927
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. KNN")
diff = [y - x for y, x in zip(rand_forest_test_acc, knn_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
```

```
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. KNN
This is the T Value
-1.1341162537024336
This is the P Value
0.8715999160193528
Accept the null hypothesis
```

Angry Birds Review Dataset

This given dataset provides 2 various attributes that give us the necessary information to analyze whether someone likes the game angry birds or not as well as their comment on it. The reasoning behind using this dataset is not as serious as before, but more of a nostalgia of classifying a once very popular app game.

```
ab_data_preserved = pd.read_csv('AngryBirdsReview.csv')
ab_data = ab_data_preserved #Temp for manipulation
ab_data #preview of dataset

#Basic Cleanup
ab_data.dropna(inplace=True)
ab_data.shape
ab_data.head()
```

	reviewText	Positive
0	This is a one of the best apps acording to a b...	1
1	This is a pretty good version of the game for ...	1
2	this is a really cool game. there are a bunch ...	1
3	This is a silly game and can be frustrating, b...	1
4	This is a terrific game on any pad. Hrs of fun...	1

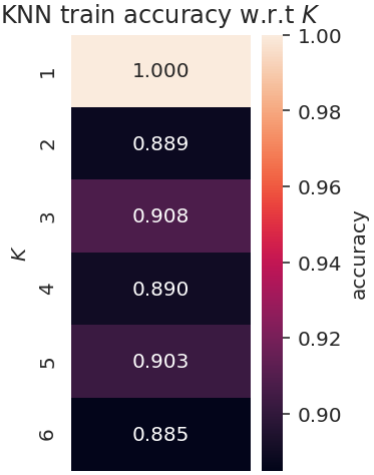
```
# Encoded each of the categorical features so it will be good for our testing
ab_data = MultiColumnLabelEncoder(columns = ['reviewText']).fit_transform(ab_data)


#Split Data By 80/20, 50/50, 20/80
partitionVal = [0.8, 0.5, 0.2]
result_table = np.zeros((3,7))
result_table1 = np.zeros((3,7))
result_table2 = np.zeros((3,7))
```

```
    for i, partition in enumerate(partitionVal):
        print("Partition: ", partition)

        knn_test_acc = []
        rand_forest_test_acc = []
        decision_tree_test_acc = []
        svm_test_acc = []



        NUM_TRIALS = 5
        for trial in range(NUM_TRIALS):
            #Mix up the data
            students_data = students_data.sample(frac=1).reset_index(drop=True)
            #Find the point where to split the data
            breakNum = int(partition*len(students_data))

            X_train_full = students_data.loc[0:breakNum]
            X_train_val = X_train_full.drop("gender", axis = 1)
            Y_train_val = X_train_full["gender"]
            X_test_full = students_data.loc[breakNum:]
            X_test= X_test_full.drop("gender", axis = 1)
            Y_test_val = X_test_full["gender"]

            #Call the svm classifier (Took Too Long For ALL Datasets)
            #test_acc,best_train0,C0 = svm_func()
            #svm_test_acc.append(test_acc)

            #Call the knn classifier
            test_acc,best_train1,C1 = knn_classifier()
            knn_test_acc.append(test_acc)

            #Call the Decision Tree classifier
            test_acc,best_train2,C2 = decision_Tree()
            decision_tree_test_acc.append(test_acc)

            #Call the Random Forest classifier
            test_acc,best_train3,C3 = rand_Forest()
            rand_forest_test_acc.append(test_acc)


        #result_table[i, 0] = sum(svm_test_acc)/NUM_TRIALS
        result_table[i, 1] = sum(knn_test_acc)/NUM_TRIALS
        result_table[i, 2] = sum(decision_tree_test_acc)/NUM_TRIALS
        result_table[i, 3] = sum(rand_forest_test_acc)/NUM_TRIALS


        #result_table1[i, 0] = best_train0
        result_table1[i, 1] = best_train1
        result_table1[i, 2] = best_train2
        result_table1[i, 3] = best_train3
```

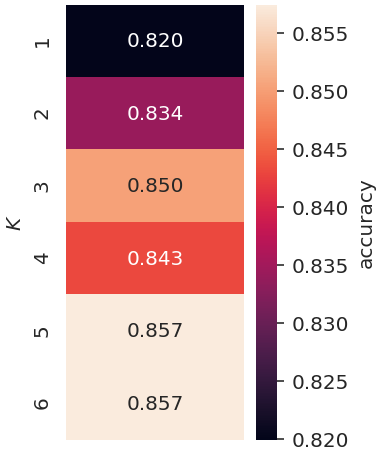
```
#result_table2[i, 0] = C0
result_table2[i, 1] = C1
result_table2[i, 2] = C2
result_table2[i, 3] = C3


#Average all test accuracies for all 5 trials
print("Test Accuracy Average for knn = ", sum(knn_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Random Forest = ",
      sum(rand_forest_test_acc)/NUM_TRIALS)

print("Test Accuracy Average for Decision Tree = ",
      sum(decision_tree_test_acc)/NUM_TRIALS)

#print("Test Accuracy Average for SVM = ", sum(svm_test_acc)/NUM_TRIALS)

#y-axis: partition
#x-axis: classifier
print(result_table)
print("############################")
print(result_table1)
print("############################")
print(result_table2)
```

```
Partition:  0.8
```

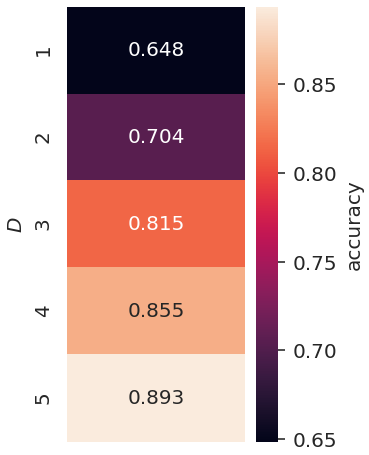
KNN train accuracy w.r.t *K*



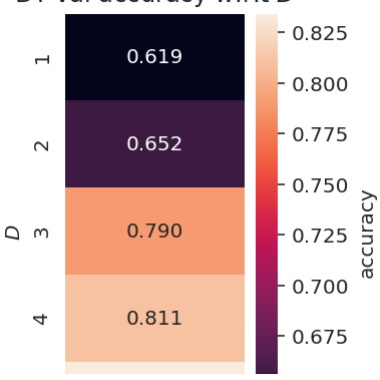
KNN val accuracy w.r.t *K*



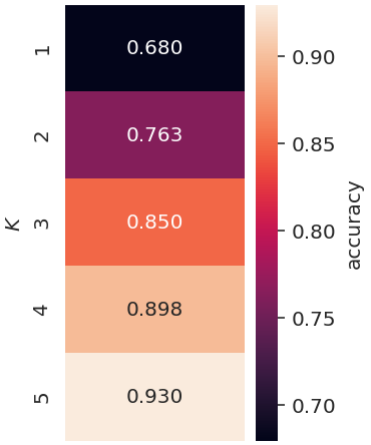
DT train accuracy w.r.t *D*



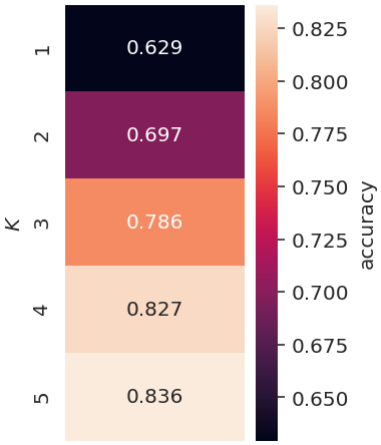
DT val accuracy w.r.t *D*



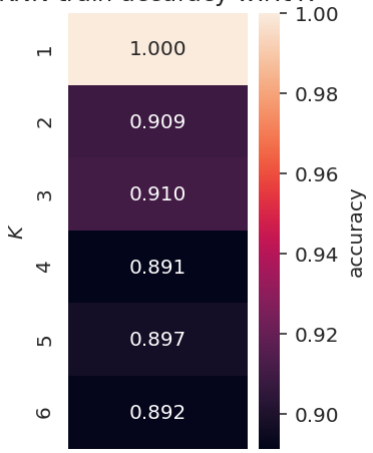
RF train accuracy w.r.t *K*



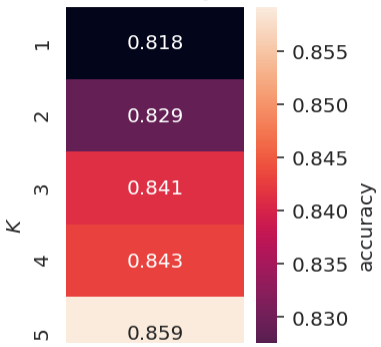
RF val accuracy w.r.t *K*

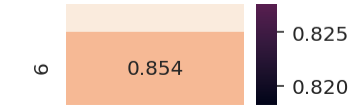


KNN train accuracy w.r.t *K*

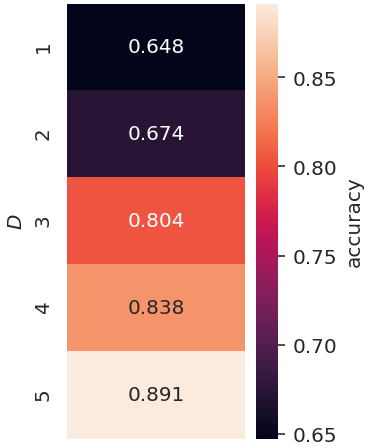


KNN val accuracy w.r.t *K*

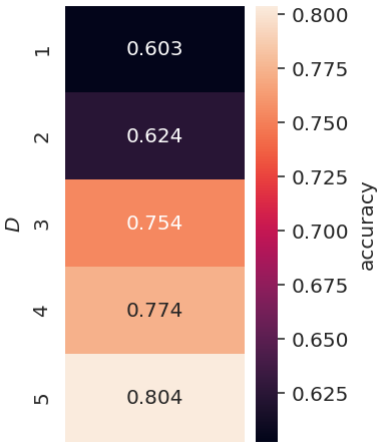




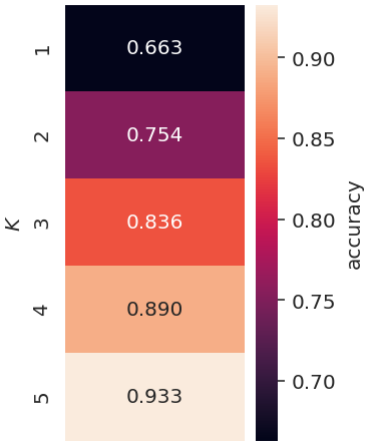
DT train accuracy w.r.t *D*



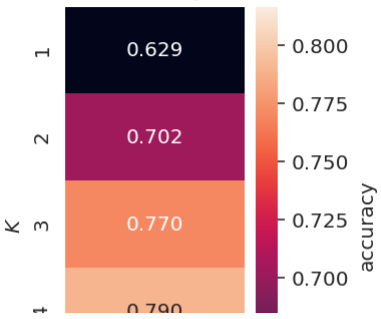
DT val accuracy w.r.t *D*

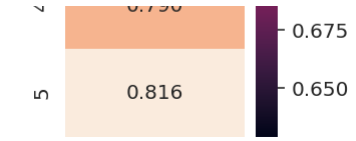


RF train accuracy w.r.t *K*

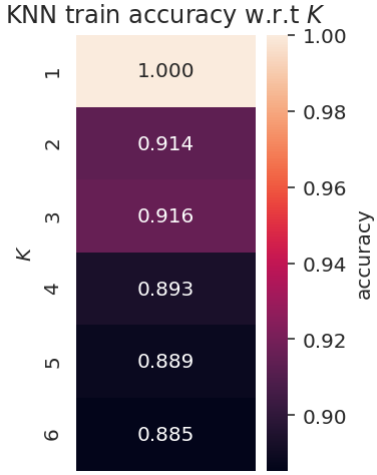


RF val accuracy w.r.t *K*

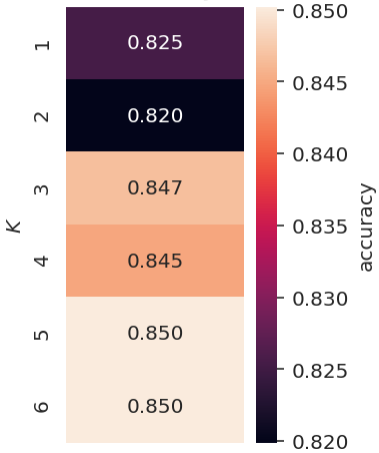




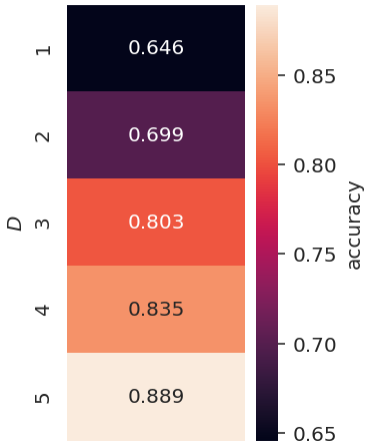
KNN train accuracy w.r.t *K*



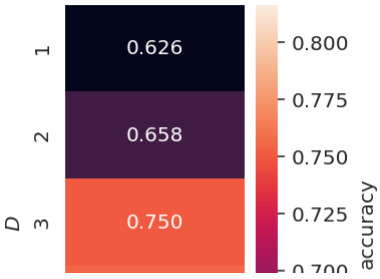
KNN val accuracy w.r.t *K*

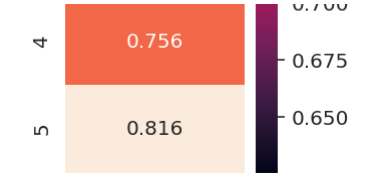


DT train accuracy w.r.t *D*

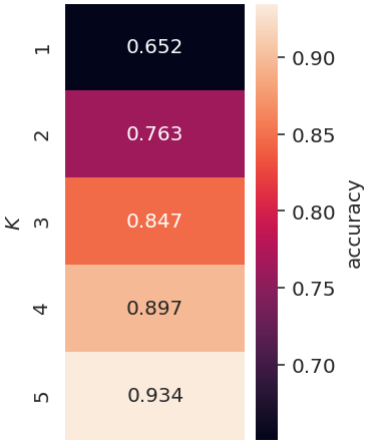


DT val accuracy w.r.t *D*

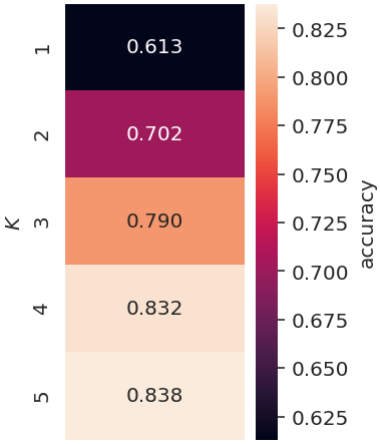




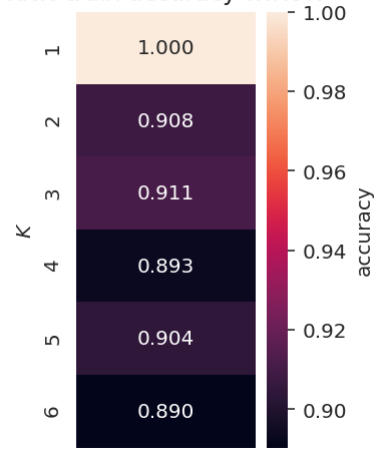
RF train accuracy w.r.t *K*



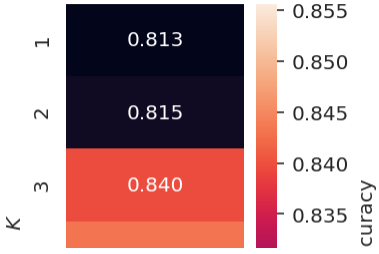
RF val accuracy w.r.t *K*

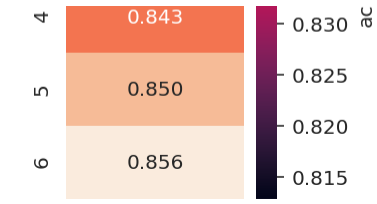


KNN train accuracy w.r.t *K*

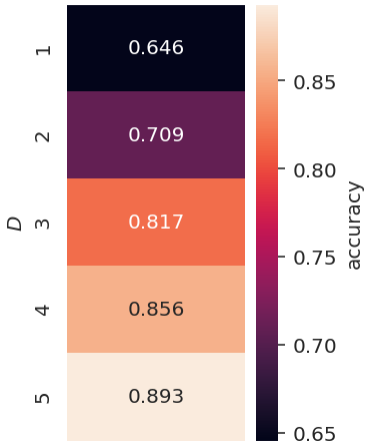


KNN val accuracy w.r.t *K*

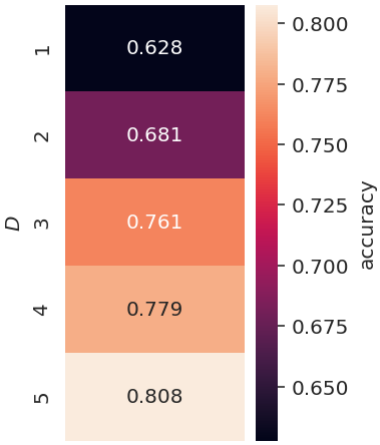




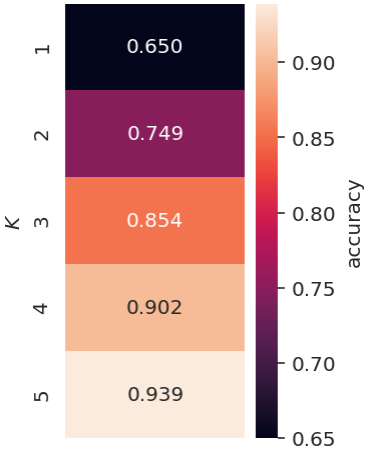
DT train accuracy w.r.t *D*



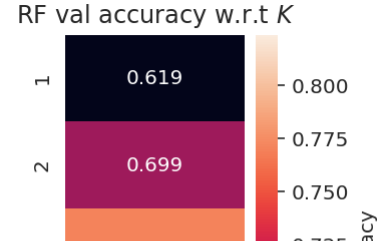
DT val accuracy w.r.t *D*

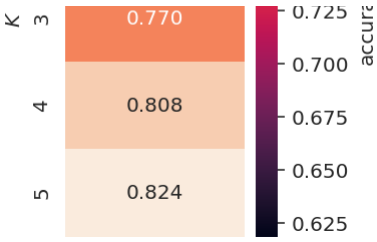


RF train accuracy w.r.t *K*

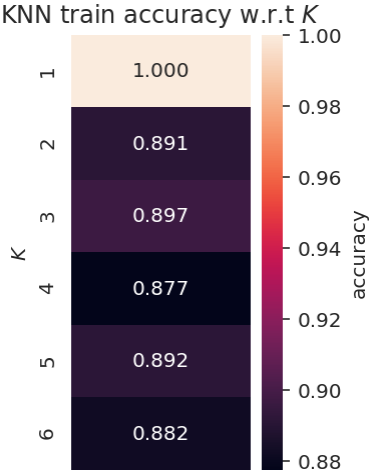


RF val accuracy w.r.t *K*

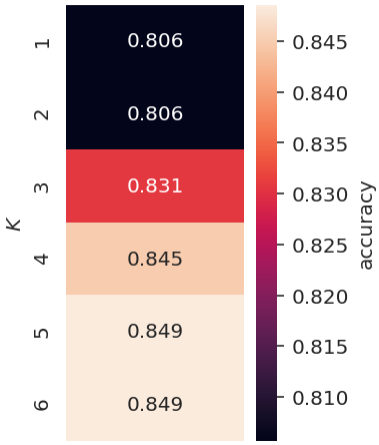




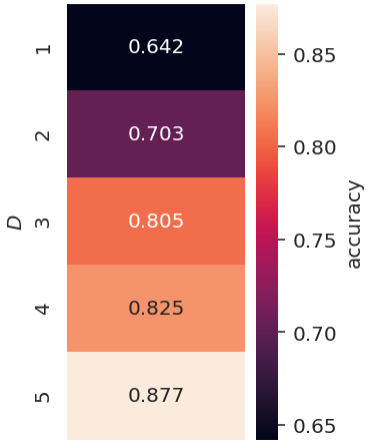
KNN train accuracy w.r.t *K*



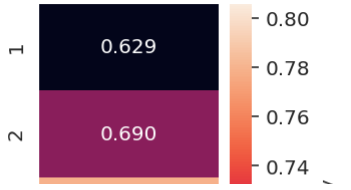
KNN val accuracy w.r.t *K*

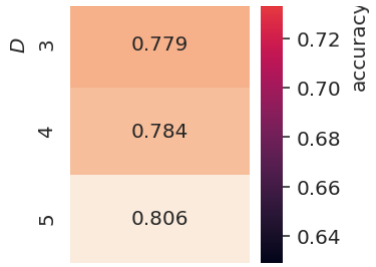


DT train accuracy w.r.t *D*

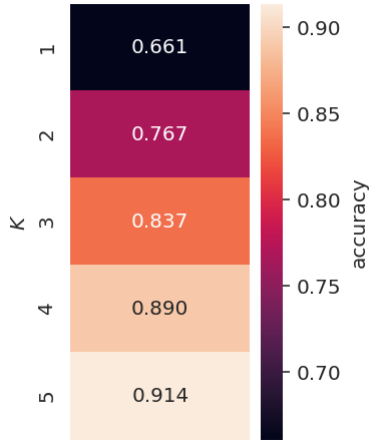


DT val accuracy w.r.t *D*

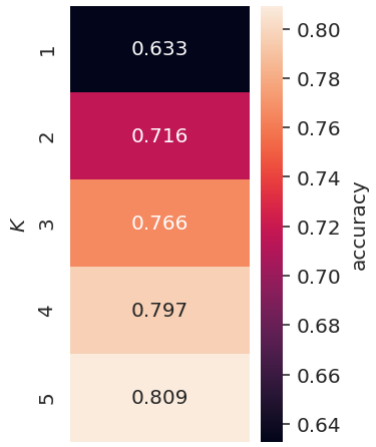




RF train accuracy w.r.t *K*



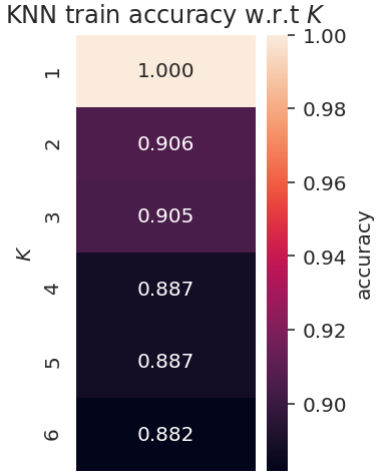
RF val accuracy w.r.t *K*



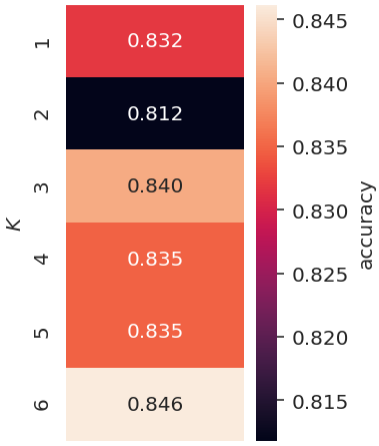

```
Test Accuracy Average for knn =  0.8385714285714286
Test Accuracy Average for Random Forest =  0.8285714285714285
Test Accuracy Average for Decision Tree =  0.8128571428571428
[[0.         0.83857143 0.81285714 0.82857143 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0.         0.89171273 0.87700644 0.91355393 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 5. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0 5
```

Partition: 0.5

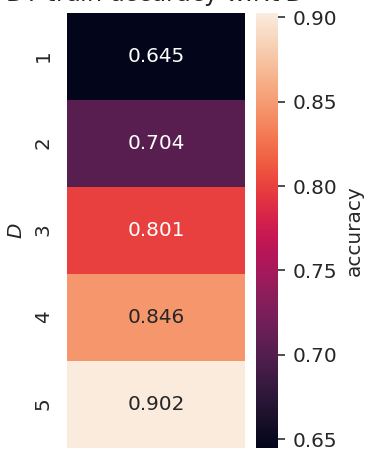
KNN train accuracy w.r.t *K*



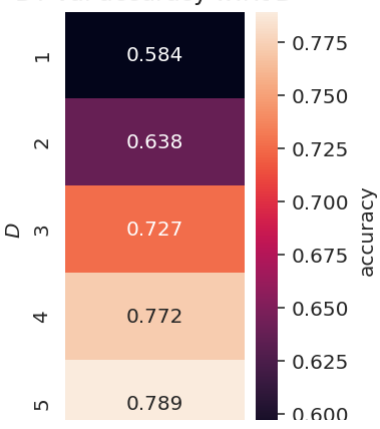
KNN val accuracy w.r.t *K*



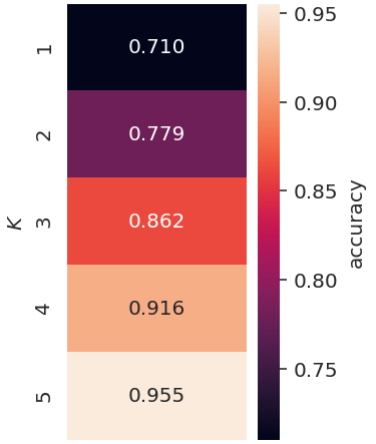
DT train accuracy w.r.t *D*



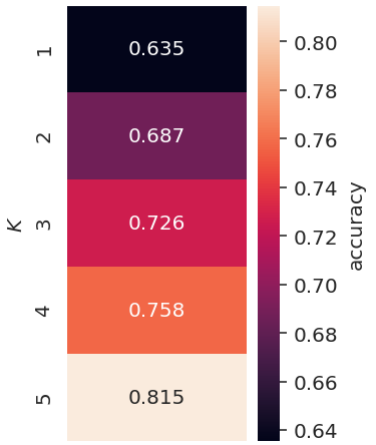
DT val accuracy w.r.t *D*



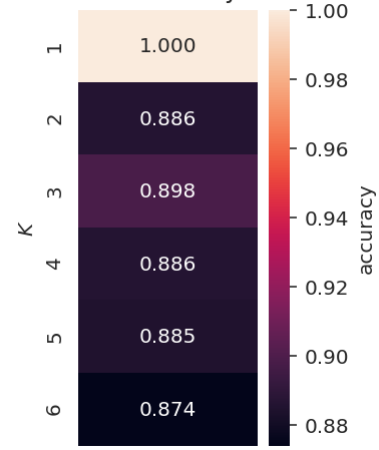
RF train accuracy w.r.t *K*



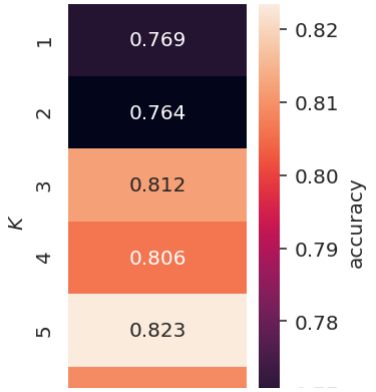
RF val accuracy w.r.t *K*

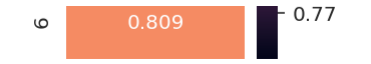


KNN train accuracy w.r.t *K*

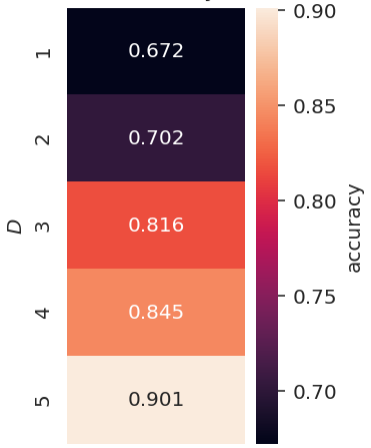


KNN val accuracy w.r.t *K*

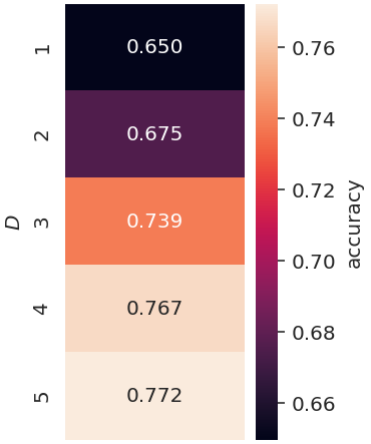




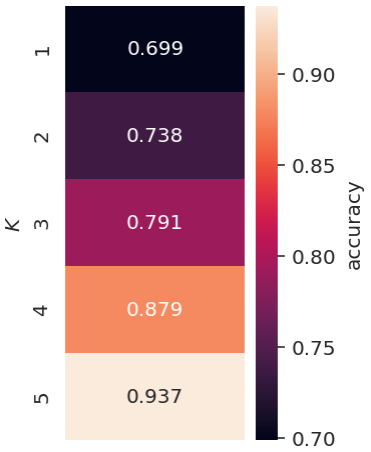
DT train accuracy w.r.t *D*



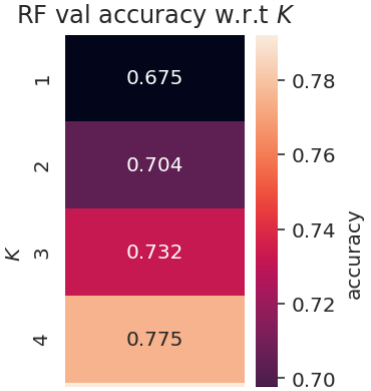
DT val accuracy w.r.t *D*



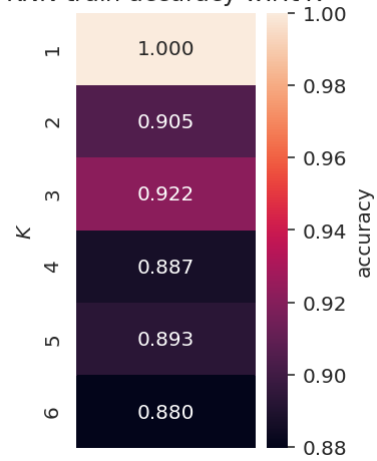
RF train accuracy w.r.t *K*



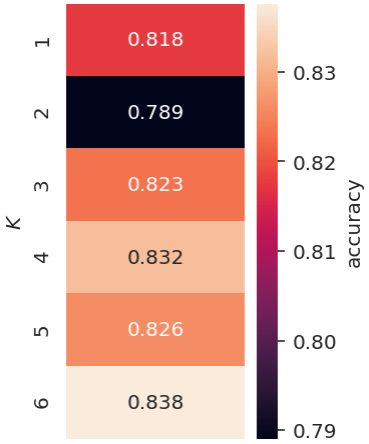
RF val accuracy w.r.t *K*



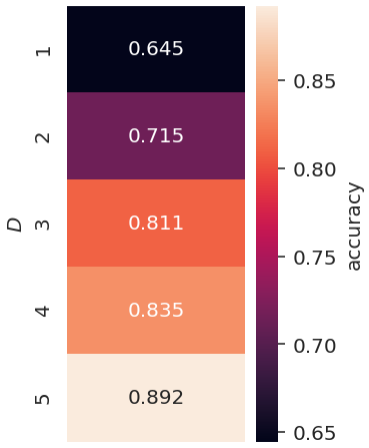
KNN train accuracy w.r.t *K*



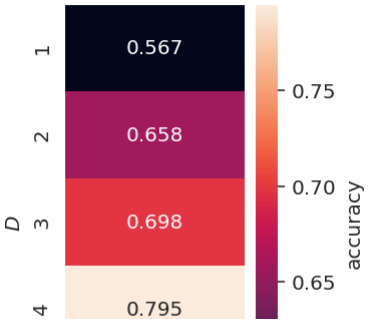
KNN val accuracy w.r.t *K*



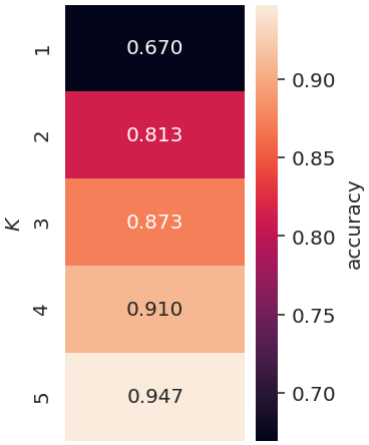
DT train accuracy w.r.t *D*



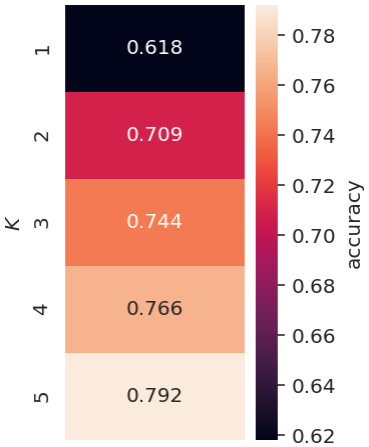
DT val accuracy w.r.t *D*



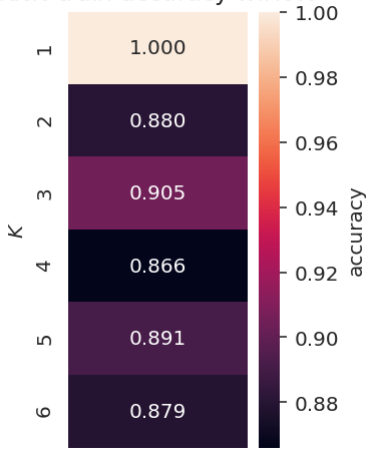
RF train accuracy w.r.t *K*



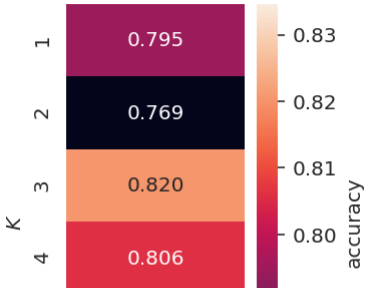
RF val accuracy w.r.t *K*

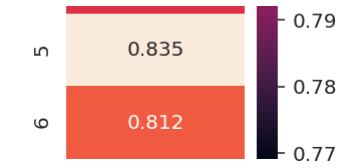


KNN train accuracy w.r.t *K*

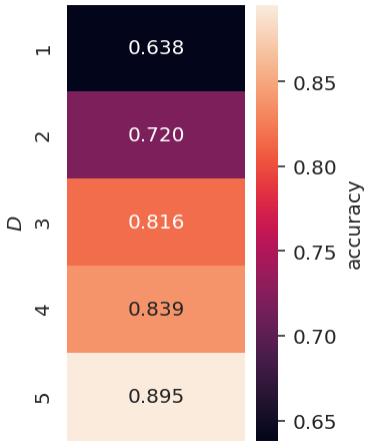


KNN val accuracy w.r.t *K*

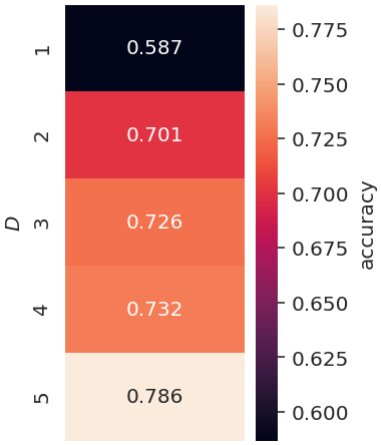




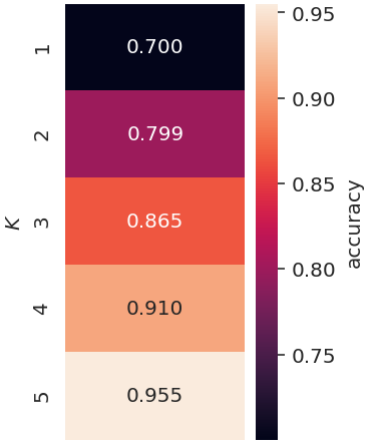
DT train accuracy w.r.t *D*



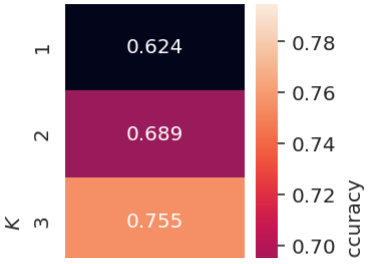
DT val accuracy w.r.t *D*

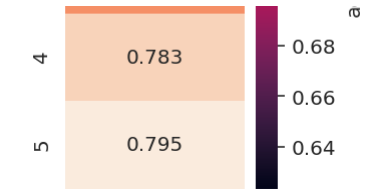


RF train accuracy w.r.t *K*

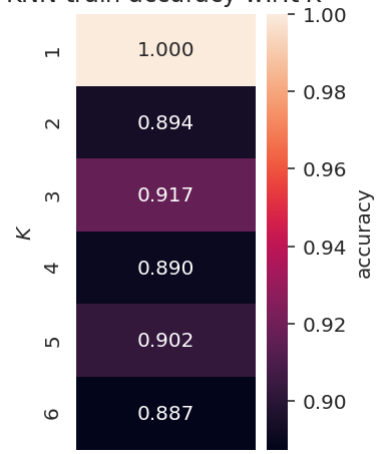


RF val accuracy w.r.t *K*

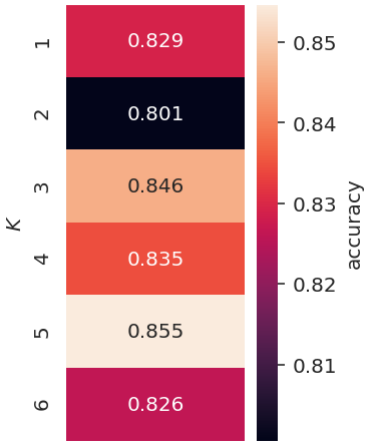




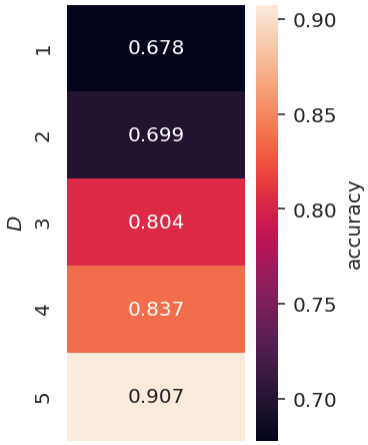
KNN train accuracy w.r.t *K*



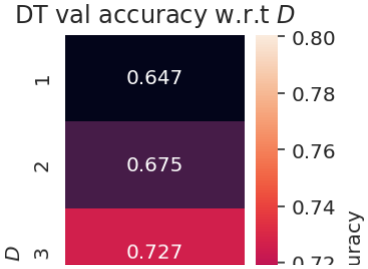
KNN val accuracy w.r.t *K*

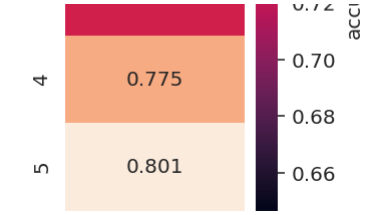


DT train accuracy w.r.t *D*

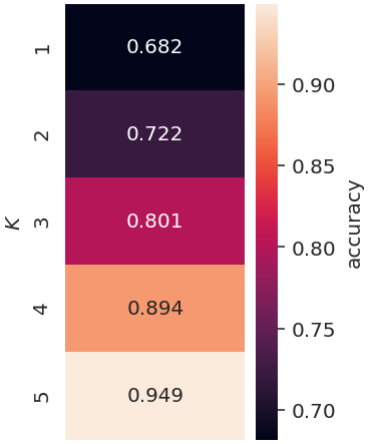


DT val accuracy w.r.t *D*

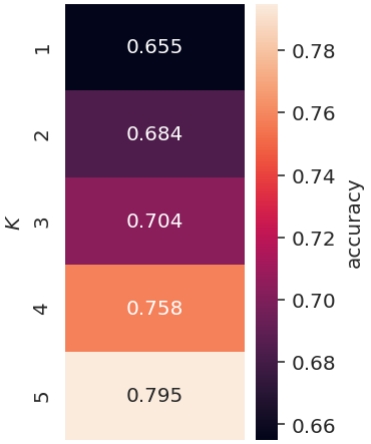




RF train accuracy w.r.t *K*

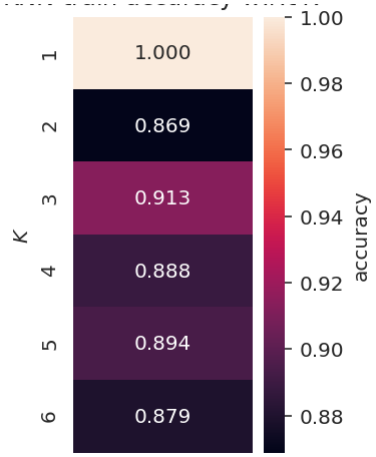


RF val accuracy w.r.t *K*

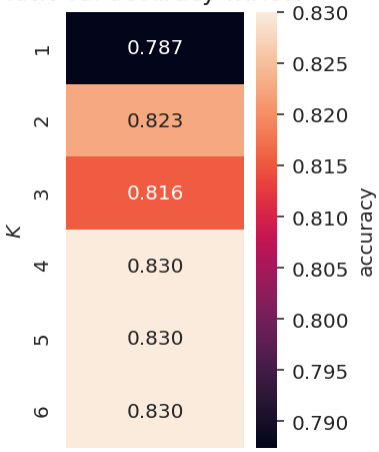



```
Test Accuracy Average for knn =  0.8371428571428572
Test Accuracy Average for Random Forest =  0.8182857142857143
Test Accuracy Average for Decision Tree =  0.7965714285714286
[[0.         0.83857143 0.81285714 0.82857143 0.         0.
  0.        ]
 [0.         0.83714286 0.79657143 0.81828571 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0.         0.89171273 0.87700644 0.91355393 0.         0.
  0.        ]
 [0.         0.9024301  0.90741484 0.94943823 0.         0.
  0.        ]
 [0.         0.         0.         0.         0.         0.
  0.        ]]
##############################
[[0. 5. 5. 5. 0. 0. 0.]
 [0. 5. 5. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
Partition:  0.2
```

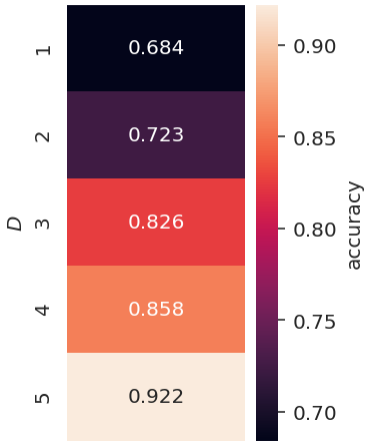
KNN train accuracy w.r.t *K*



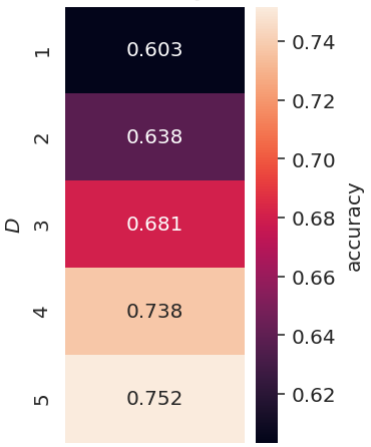
KNN val accuracy w.r.t *K*



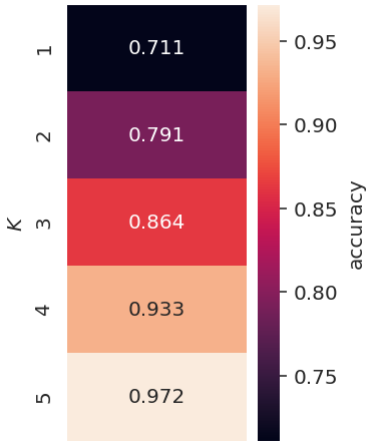
DT train accuracy w.r.t *D*



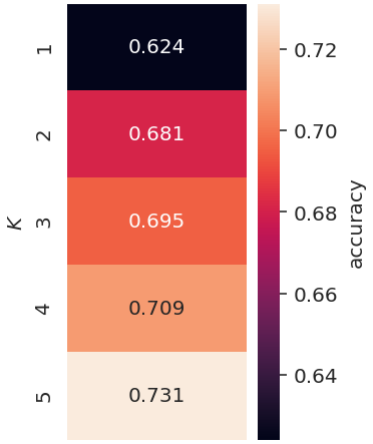
DT val accuracy w.r.t *D*



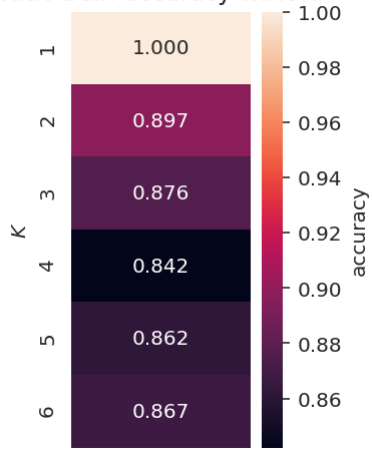
RF train accuracy w.r.t *K*



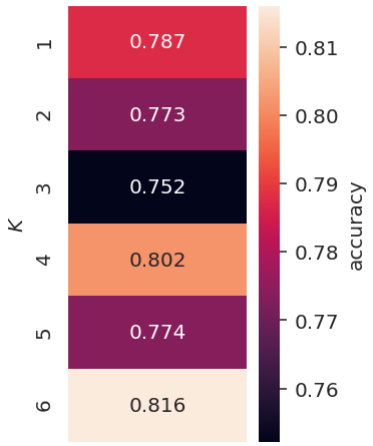
RF val accuracy w.r.t *K*

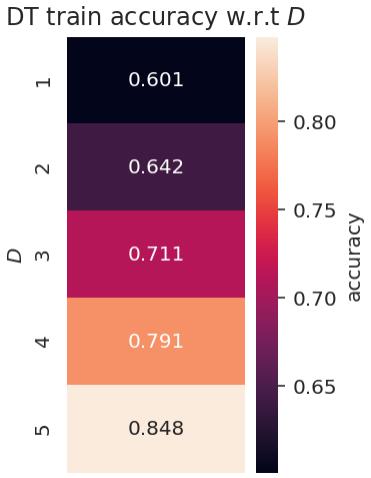


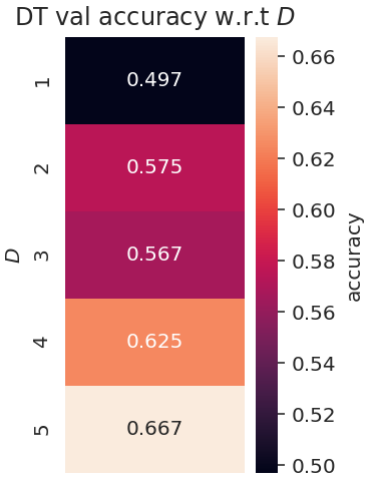
KNN train accuracy w.r.t *K*

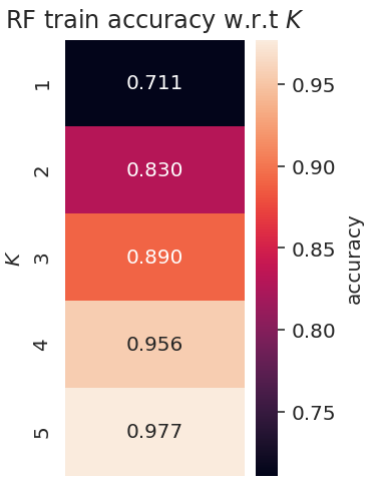


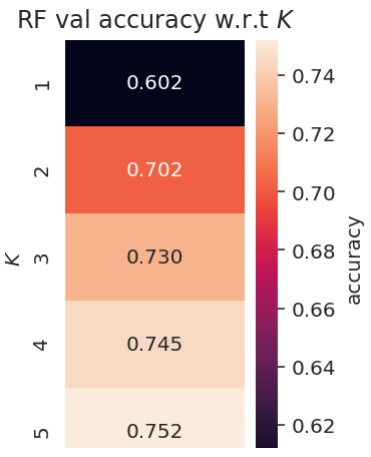
KNN val accuracy w.r.t *K*



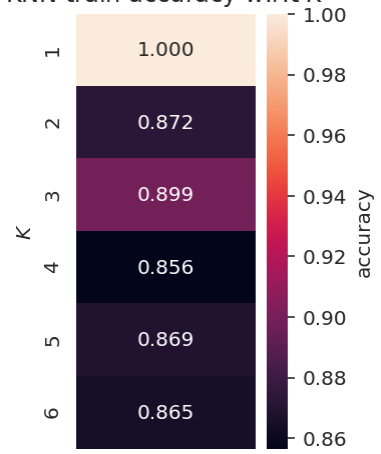

DT train accuracy w.r.t D
D
1
2
3
4
5
0.601
0.642
0.711
0.791
0.848
accuracy
0.80
0.75
0.70
0.65


DT val accuracy w.r.t D
D
1
2
3
4
5
0.497
0.575
0.567
0.625
0.667
accuracy
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
0.50

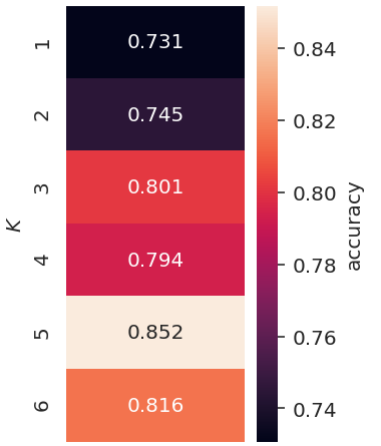

RF train accuracy w.r.t K
K
1
2
3
4
5
0.711
0.830
0.890
0.956
0.977
accuracy
0.95
0.90
0.85
0.80
0.75


RF val accuracy w.r.t K
K
1
2
3
4
5
0.602
0.702
0.730
0.745
0.752
accuracy
0.74
0.72
0.70
0.68
0.66
0.64
0.62

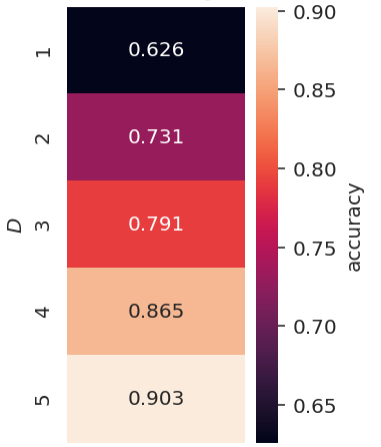
KNN train accuracy w.r.t *K*



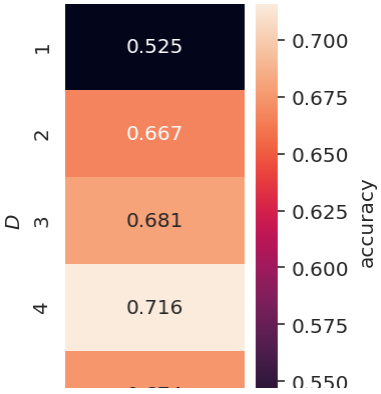
KNN val accuracy w.r.t *K*



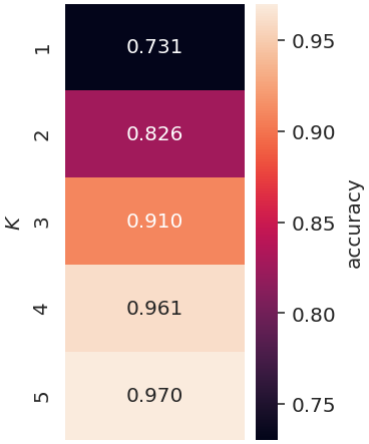
DT train accuracy w.r.t *D*



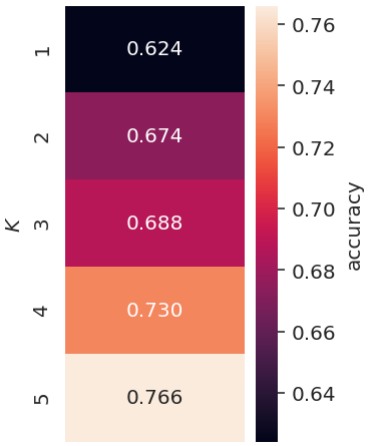
DT val accuracy w.r.t *D*



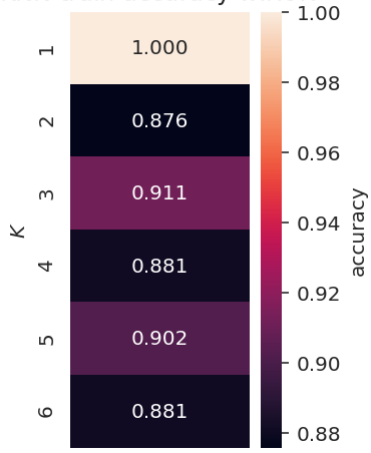
RF train accuracy w.r.t *K*



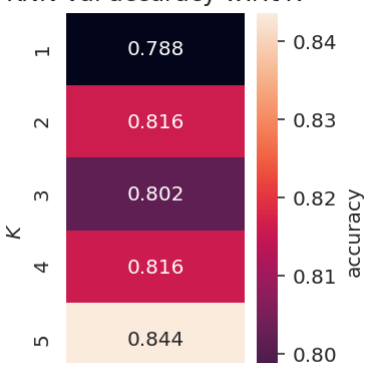
RF val accuracy w.r.t *K*



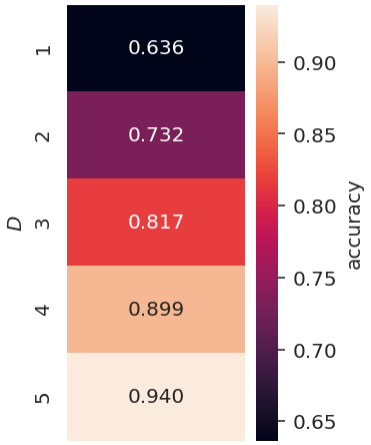
KNN train accuracy w.r.t *K*



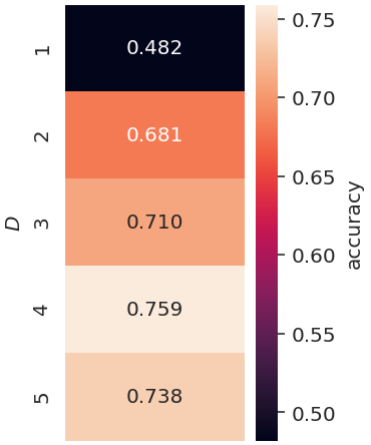
KNN val accuracy w.r.t *K*



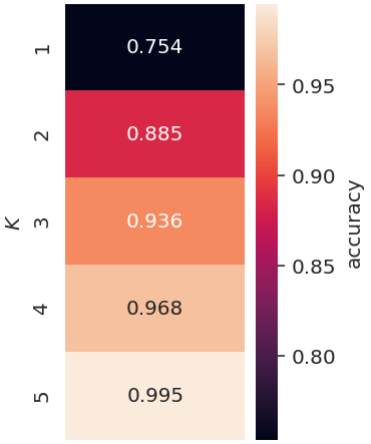
DT train accuracy w.r.t *D*



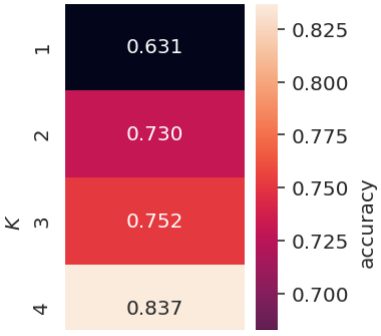
DT val accuracy w.r.t *D*

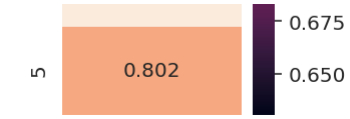


RF train accuracy w.r.t *K*

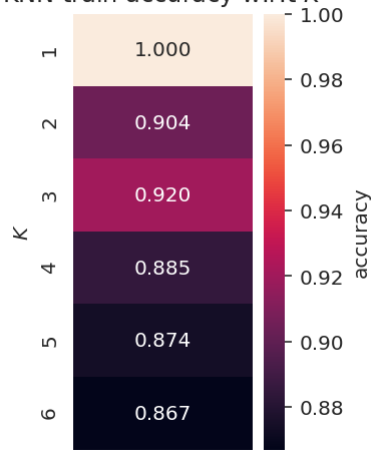


RF val accuracy w.r.t *K*

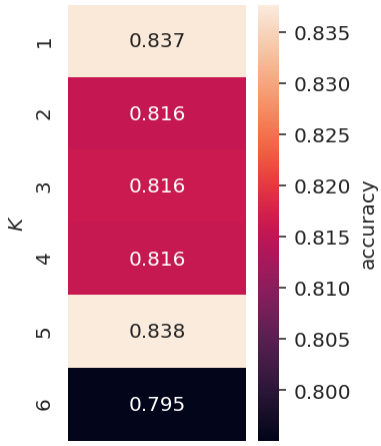




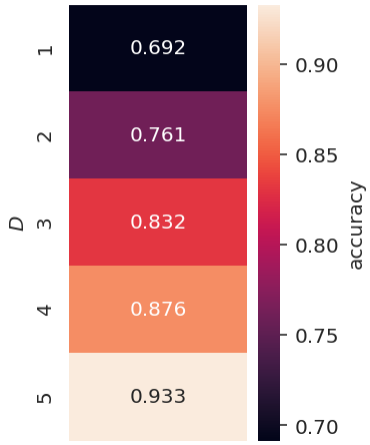
KNN train accuracy w.r.t *K*



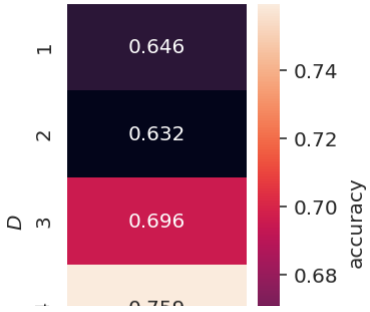
KNN val accuracy w.r.t *K*

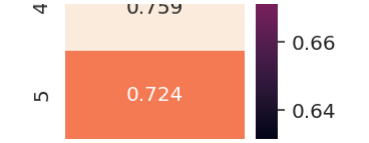


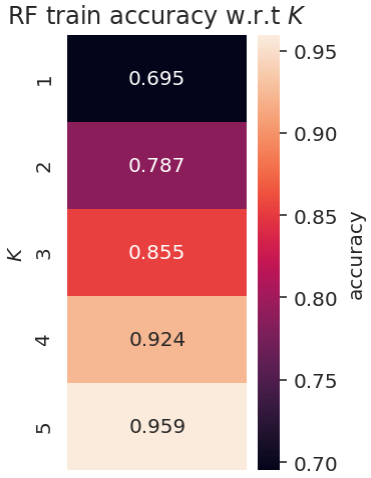
DT train accuracy w.r.t *D*

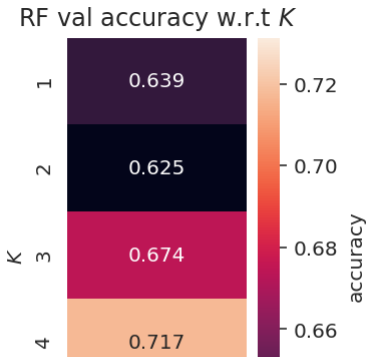


DT val accuracy w.r.t *D*









Running Respective P and T Tests to compare the Trials Accuracy For Each Algorithm

```
#Compute the difference between the results
print("P and T Test for KNN trials vs. Decision Tree")
diff = [y - x for y, x in zip(knn_test_acc, decision_tree_test_acc)]

#Compute the mean of differences
d_bar = np.mean(diff)

#Compute the variance of differences
sigma2 = np.var(diff)

#compute the number of data points used for training
n1 = len(Y_train_val)

#compute the number of data points used for testing
n2 = len(Y_test_val)

#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static = d_bar / np.sqrt(sigma2_mod)
```

```
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for KNN trials vs. Decision Tree
This is the T Value
1.0390872231678294
This is the P Value
0.1494072134636163
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. Decision Tree")
diff = [y - x for y, x in zip(rand_forest_test_acc, decision_tree_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
```

```
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. Decision Tree
This is the T Value
0.5286692426151534
This is the P Value
0.2985291936330071
Accept the null hypothesis
```

```
#Compute the difference between the results
print("P and T Test for Random Forests vs. KNN")
diff = [y - x for y, x in zip(rand_forest_test_acc, knn_test_acc)]
#Comopute the mean of differences
d_bar = np.mean(diff)
#compute the variance of differences
sigma2 = np.var(diff)
#compute the number of data points used for training
n1 = len(Y_train_val)
#compute the number of data points used for testing
n2 = len(Y_test_val)
#compute the total number of data points
n = 5000
#compute the modified variance
sigma2_mod = sigma2 * (1/n + n2/n1)
#compute the t_static
t_static =  d_bar / np.sqrt(sigma2_mod)
print("This is the T Value"),
print(t_static)

from scipy.stats import t
#Compute p-value and plot the results
Pvalue = ((1 - t.cdf(t_static, n-1)))
print("This is the P Value"),
print(Pvalue)

if Pvalue < 0.05:
  # Statistically Significant
  print("We reject null hypothesis")
else:
  # Statistically insignificant
  print("Accept the null hypothesis")
```

```
P and T Test for Random Forests vs. KNN
This is the T Value
-0.8198207030633983
This is the P Value
0.7938212937912209
Accept the null hypothesis
```