In [ ]:
```python
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import random

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

from sklearn.model_selection import cross_validate
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

Importing training and testing data sets, exploratory analysis of features

In [ ]:
```python
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

train.head() # display a few samples
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V33	V34	V35	V36	V37	V38	V39
3	3.932	3.2512	0	0.0	0	0	0	26.7	2	4	...	0	0	1	3.076	2.417	0	7.601
5	4.236	3.3944	0	0.0	0	0	0	29.4	2	4	...	0	0	0	3.351	2.405	0	8.003
6	4.236	3.4286	0	0.0	0	0	0	28.6	2	4	...	0	0	0	3.351	2.556	0	7.904
7	5.000	5.0476	1	0.0	0	0	0	11.1	0	3	...	0	0	1	4.712	4.583	0	9.303
8	4.525	3.8301	0	0.0	0	0	0	31.6	3	2	...	0	0	0	3.379	2.143	0	7.950

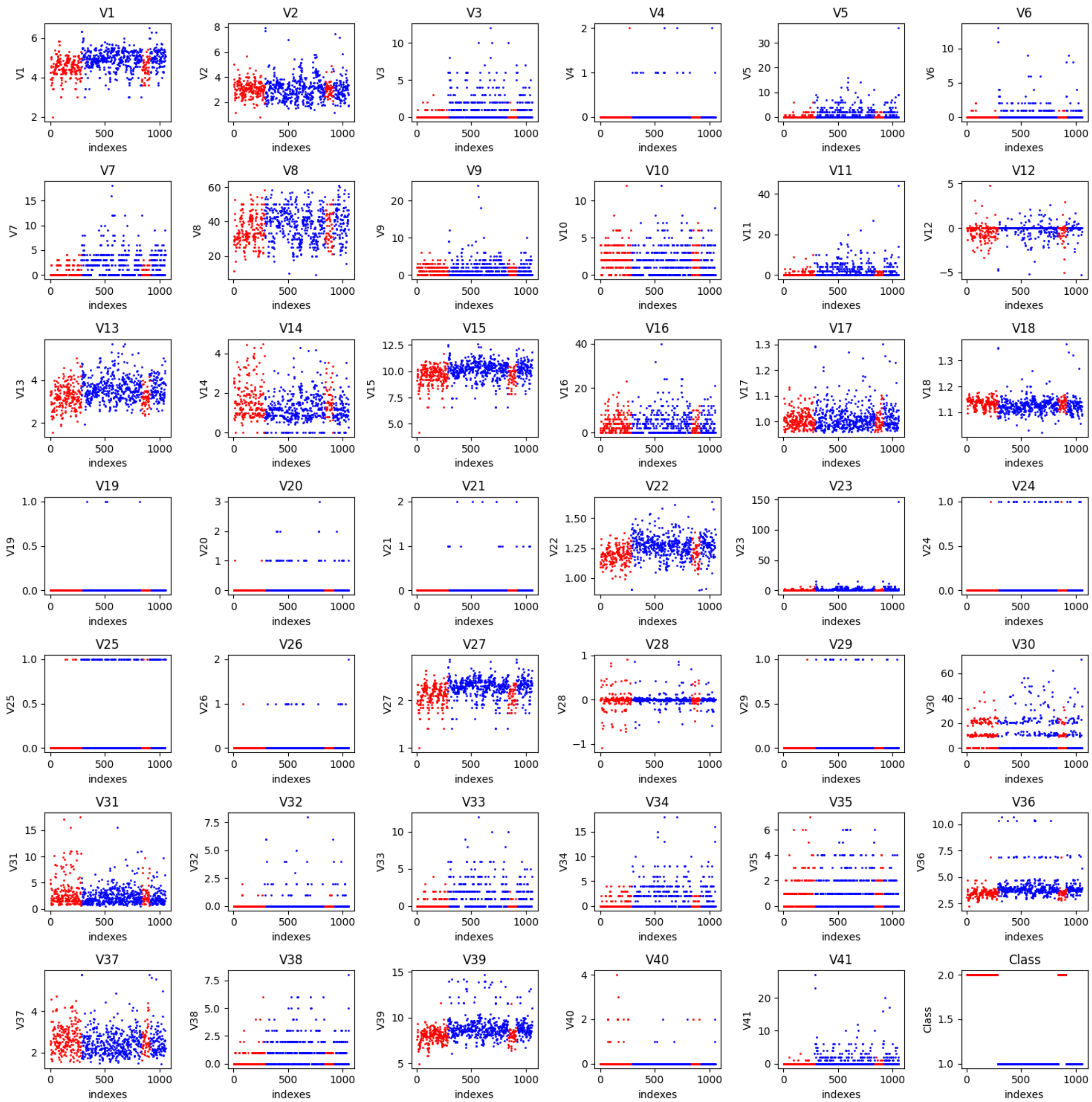
5 rows × 42 columns

Visualization of the feature space

In [ ]:
```python
f = plt.figure(figsize=(15, 15))

for i, col in enumerate(train.columns):
    print(col, len(train[col].unique()), end="; ")
    f.add_subplot(7, 6, i+1)
    plt.title(col)
    plt.ylabel(col)
```

```python
    plt.xlabel("indexes")
    plt.tight_layout()
    plt.plot(train[col][train['Class'] == 1], "bo", markersize="1")
    plt.plot(train[col][train['Class'] == 2], "ro", markersize="1")
```

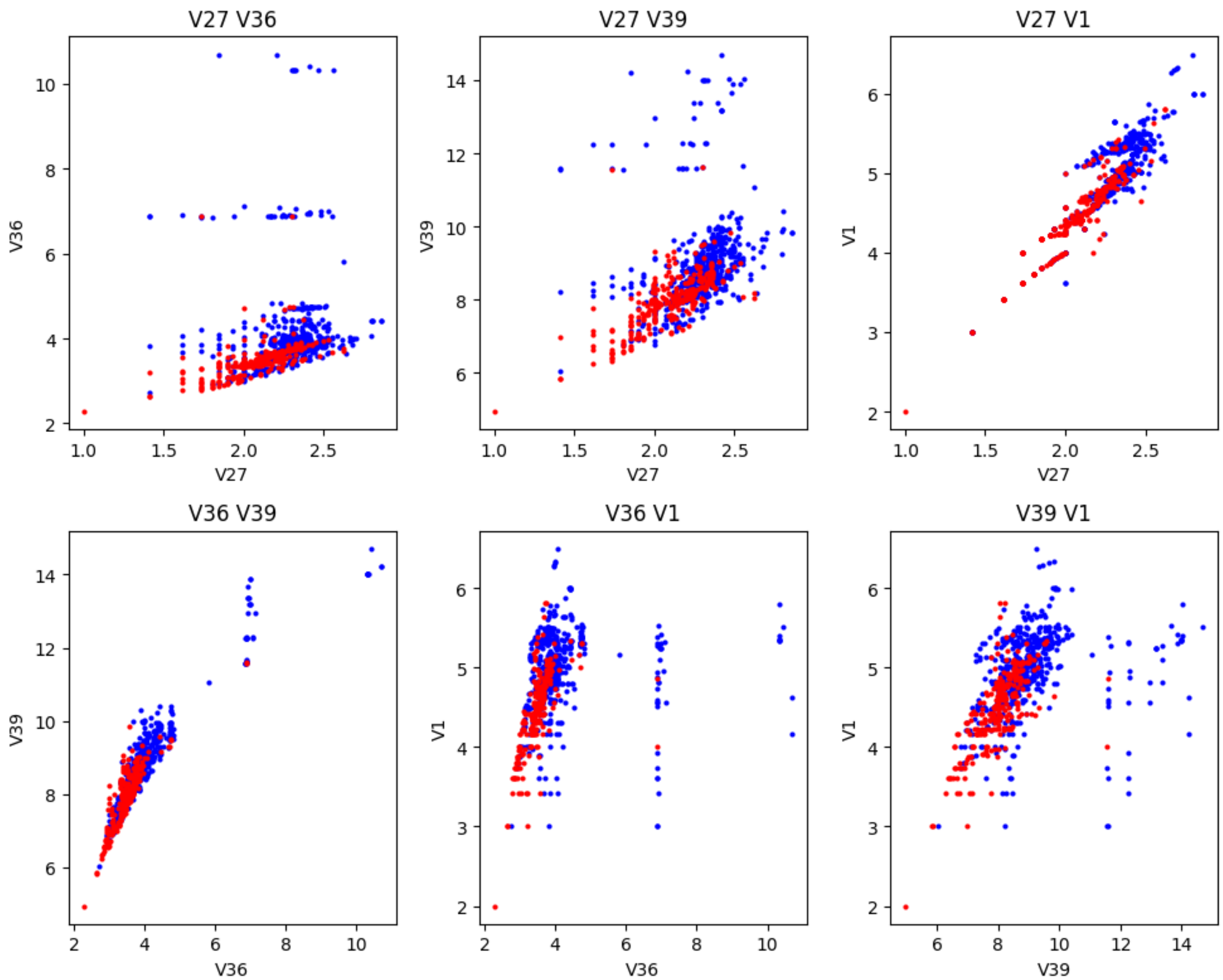
```
V1 379; V2 823; V3 11; V4 4; V5 16; V6 10; V7 14; V8 172; V9 15; V10 11; V11 21; V12 307; V13 638; V14 321; V15 436; V16 24; V17 158; V18 115; V19 2; V20 4; V21 3; V22 322; V23 13; V24 2; V25 2; V26 3; V27 291; V28 174; V29 3; V30 372; V31 468; V32 8; V33 11; V34 16; V35 8; V36 603; V37 536; V38 8; V39 711; V40 5; V41 16; Class 2;
```



In [ ]:

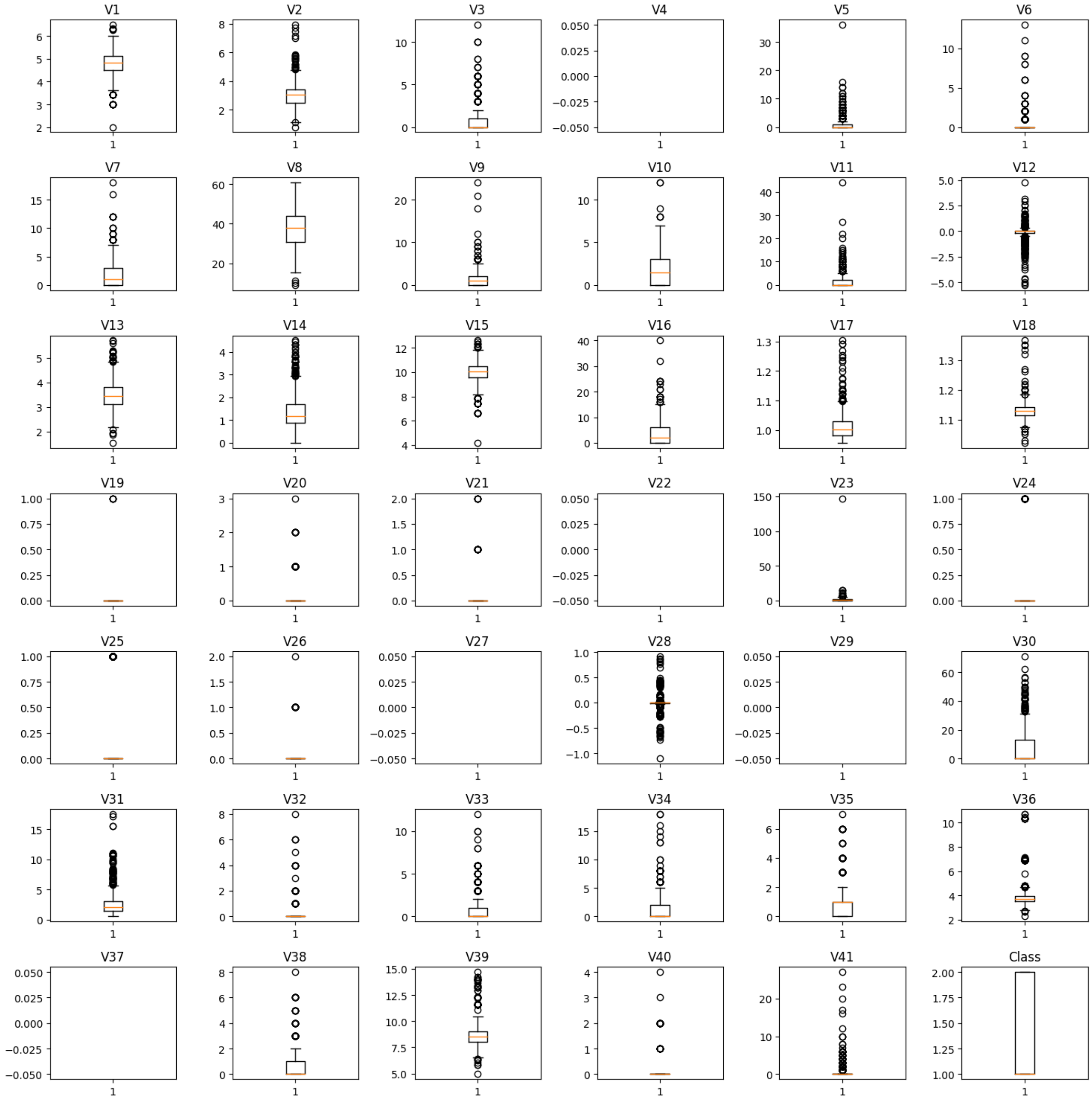
```python
f = plt.figure(figsize=(10, 8))

b = [26, 35, 38, 0]
a = train.columns[b]

counter = 1
for i, col in enumerate(a):
    for j, col2 in enumerate(a[i+1:]):
        f.add_subplot(2, 3, counter)
        plt.title(col + " " + col2)
        plt.ylabel(col2)
```

```
        plt.xlabel(col)
        plt.tight_layout()
        plt.plot(train[col][train['Class'] == 1], train[col2][train['Class'] == 1], "b
        plt.plot(train[col][train['Class'] == 2], train[col2][train['Class'] == 2], "r
        counter += 1

plt.show()
```



In [ ]:

```
f = plt.figure(figsize=(15, 15))

for i, col in enumerate(train.columns):
    f.add_subplot(7, 6, i+1)
    plt.title(col)
    plt.tight_layout()
    plt.boxplot(train[col])
```



2.1 Exploration

Inspect the dataset. How balanced is the target variable? Are there any missing values present? If there are, choose a strategy that takes this into account. Most of your data is of the numeric type. Can you identify, by adopting exploratory analysis, whether some features are directly related to the target? What about feature pairs? Produce at least three types of visualizations of the feature space and be prepared to argue why these visualizations were useful for your subsequent analysis.

Target variable distributions in test and training sets are close to [2/3 1/3].

Yes, there are some missing values. One possible strategy is to drop those that don't have all attribute values. Some classifiers however don't really need all information, so you can just ignore missing rows for that specific attributes.

The visualizations are above. We haven't found anything very concrete. Some of the features are more and some less releted to target. Random forest feature importances helped us to know which attributes are less/more important

Majority classifier

In [ ]:
```
majority = train['Class'].value_counts()
majorityArr = np.array(majority)
print("Class distribution (train)")
print(majority)
print("Percentage:")
print(np.array(majorityArr[0] / np.sum(majorityArr)))

majorityTest = test['Class'].value_counts()
majorityTestArr = np.array(majorityTest)
print("Class distribution (test)")
print(majorityTest)
print("Percentage:")
print(np.array(majorityTestArr[0] / np.sum(majorityTestArr)))
```

```
Class distribution (train)
1    564
2    282
Name: Class, dtype: int64
Percentage:
0.6666666666666666
Class distribution (test)
1    135
2     74
Name: Class, dtype: int64
Percentage:
0.645933014354067
```

Random classifier

There are two classes if we choose between them randomly accuracy is 1/2.

Preprocessing

Removing data that doesn't have all values, seperating features and target variable, choosing subsets of features. Some other possible preprocessing would be normalization of attributes, setting NA values to average/max/min/random/... values, making new features using linear/non-linear combinations of two or more attributes. Some preprocessing techniques did not improve our results so we deleted some of the code for them.

In [ ]:
```
# removing NA values from train dataframe
train = train.dropna()
# Separate input features (X) and target variable (y)
y = train.Class
X = train.drop('Class', axis=1)
testY = test.Class
testX = test.drop('Class', axis=1)
```

```
# drop bad/uninformative columns
# dropColumns = np.array(['V26', 'V29', 'V19', 'V21', 'V24', 'V20', 'V4'])
# X = X.drop(dropColumns, axis=1)
# testX = testX.drop(dropColumns, axis=1)
```

Decision tree

In [ ]:
```
clf_decitionTree = DecisionTreeClassifier(random_state=0)
clf_decitionTree.fit(X, y)

pred_y_decitionTree = clf_decitionTree.predict(testX)
print(accuracy_score(testY, pred_y_decitionTree))
prob_y_decisionTree = clf_decitionTree.predict_proba(testX)
prob_y_decisionTree = [p[1] for p in prob_y_decisionTree]
print(roc_auc_score(testY, prob_y_decisionTree))
print(f1_score(testY, pred_y_decitionTree))
print(precision_score(testY, pred_y_decitionTree))
print(recall_score(testY, pred_y_decitionTree))
```

```
0.8181818181818182
0.8043043043043043
0.8582089552238805
0.8646616541353384
0.8518518518518519
```

KNN

In [ ]:
```
clf_knn = KNeighborsClassifier(n_neighbors=9)
clf_knn.fit(X, y)

pred_y_knn = clf_knn.predict(testX)
print(accuracy_score(testY, pred_y_knn))
prob_y_knn = clf_knn.predict_proba(testX)
prob_y_knn = [p[1] for p in prob_y_knn]
print(roc_auc_score(testY, prob_y_knn))
print(f1_score(testY, pred_y_knn))
print(precision_score(testY, pred_y_knn))
print(recall_score(testY, pred_y_knn))
```

```
0.7894736842105263
0.8444944944944945
0.8307692307692307
0.864
0.8
```

SVC / SVM

In [ ]:
```
clf_svc = SVC(kernel='linear',  class_weight='balanced', probability=True)
clf_svc.fit(X, y)

pred_y_svc = clf_svc.predict(testX)
print(accuracy_score(testY, pred_y_svc))
prob_y_svc = clf_svc.predict_proba(testX)
prob_y_svc = [p[1] for p in prob_y_svc]
```

```
print(roc_auc_score(testY, prob_y_svc))
print(f1_score(testY, pred_y_svc))
print(precision_score(testY, pred_y_svc))
print(recall_score(testY, pred_y_svc))
```

```
0.8564593301435407
0.917917917917918
0.8846153846153846
0.92
0.8518518518518519
```

Random forest

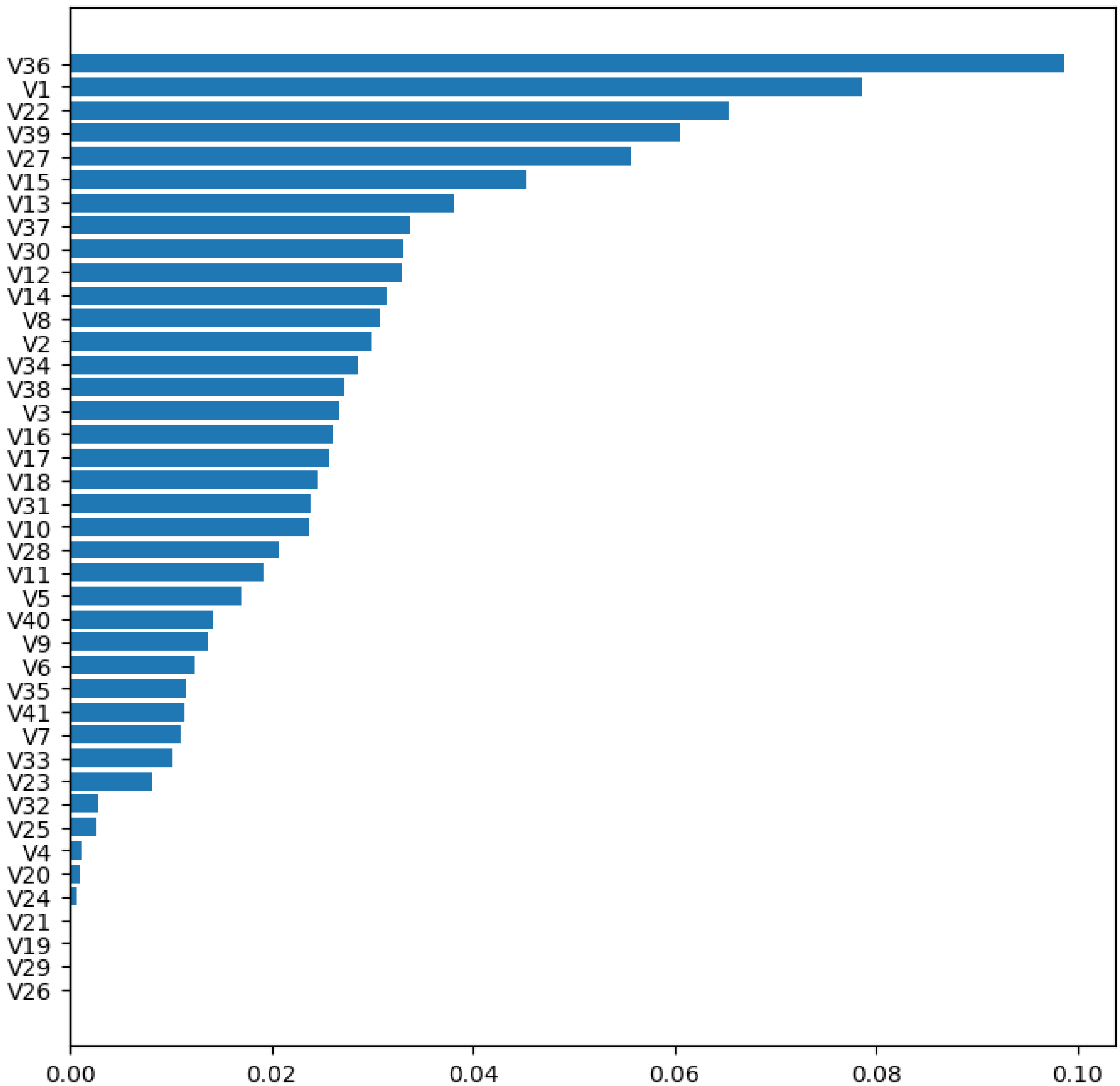
In [ ]:

```
clf_randomForest = RandomForestClassifier(random_state=1234)
clf_randomForest.fit(X, y)
pred_y_randomForest = clf_randomForest.predict(testX)
print(accuracy_score(testY, pred_y_randomForest))
prob_y_randomForest = clf_randomForest.predict_proba(testX)
prob_y_randomForest = [p[1] for p in prob_y_randomForest]
print(roc_auc_score(testY, prob_y_randomForest))
print(f1_score(testY, pred_y_randomForest))
print(precision_score(testY, pred_y_randomForest))
print(recall_score(testY, pred_y_randomForest))

f = plt.figure(figsize=(8, 8))
sorted_idx = clf_randomForest.feature_importances_.argsort()
plt.barh(train.columns[sorted_idx], clf_randomForest.feature_importances_[sorted_idx])
```

```
0.8373205741626795
0.925025025025025
0.8740740740740742
0.8740740740740741
0.8740740740740741
```

Out[ ]:

```
<BarContainer object of 41 artists>
```



Ada boost

In [ ]:

```
clf_adaboost = AdaBoostClassifier(n_estimators = 50, learning_rate = 0.2)
clf_adaboost.fit(X, y)

pred_y_adaboost = clf_adaboost.predict(testX)
print(accuracy_score(testY, pred_y_adaboost))
prob_y_adaboost = clf_adaboost.predict_proba(testX)
prob_y_adaboost = [p[1] for p in prob_y_adaboost]
print(roc_auc_score(testY, prob_y_adaboost))
print(f1_score(testY, pred_y_adaboost))
print(precision_score(testY, pred_y_adaboost))
print(recall_score(testY, pred_y_adaboost))
```

```
0.8516746411483254
0.8983483483483484
0.8888888888888888
0.8611111111111112
0.9185185185185185
```

In [ ]:
```
def printBestScoreOnTest(scores):
    scoresOnTestForEachEstimator = np.array([])
    for e in scores['estimator']:
        pred_y = e.predict(testX)
        ac = accuracy_score(testY, pred_y)
        prob_y = e.predict_proba(testX)
        prob_y = [p[1] for p in prob_y]
        roc = roc_auc_score(testY, prob_y)
        f1 = f1_score(testY, pred_y)
        precision = precision_score(testY, pred_y)
        recall = recall_score(testY, pred_y)
        temp = np.array([ac, roc, f1, precision, recall])
        if scoresOnTestForEachEstimator.size == 0:
            scoresOnTestForEachEstimator = temp
        else:
            scoresOnTestForEachEstimator = np.vstack((scoresOnTestForEachEstimator, te
    bestIndex = np.argmax(scoresOnTestForEachEstimator[:, 0])
    return scoresOnTestForEachEstimator[bestIndex, :]
```

In [ ]:
```
def printScores(scores):
    avgAccuracy = np.mean(scores['test_accuracy'])
    stdAccuracy = np.std(scores['test_accuracy'])
    avgF1 = np.mean(scores['test_f1_macro'])
    stdF1 = np.std(scores['test_f1_macro'])
    avgRecall = np.mean(scores['test_recall_macro'])
    stdRecall = np.std(scores['test_recall_macro'])
    avgAUC = np.mean(scores['test_roc_auc'])
    stdAUC = np.std(scores['test_roc_auc'])
    print(f"Precision average {avgAccuracy} with standard deviation {stdAccuracy}")
    print(f"F1 average {avgF1} with standard deviation {stdF1}")
    print(f"Recall average {avgRecall} with standard deviation {stdRecall}")
    print(f"AUC average {avgAUC} with standard deviation {stdAUC}")
    scoresTestArr = printBestScoreOnTest(scores)
    print(f'Test [accuracy, roc AUC, f1, precision, recall]')
    print(scoresTestArr)
    return avgAccuracy, avgF1, avgRecall, avgAUC, scoresTestArr
```

In [ ]:
```
scoring = ['accuracy', 'f1_macro', 'recall_macro', 'roc_auc']
rkf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=1234)

print("DecisionTree:")
scores = cross_validate(clf_decitionTree, X, y, cv=rkf.split(X), scoring=scoring, retu
accDT, f1DT, recallDT, aucDT, testArrDT = printScores(scores)
print("KNN:")
scores = cross_validate(clf_knn, X, y, cv=rkf.split(X), scoring=scoring, return_estima
accKNN, f1KNN, recallKNN, aucKNN, testArrKNN = printScores(scores)
print("SVC:")
scores = cross_validate(clf_svc, X, y, cv=rkf.split(X), scoring=scoring, return_estima
accSVC, f1SVC, recallSVC, aucSVC, testArrSVC = printScores(scores)
print("Random forest:")
scores = cross_validate(clf_randomForest, X, y, cv=rkf.split(X), scoring=scoring, retu
accRF, f1RF, recallRF, aucRF, testArrRF = printScores(scores)
print("Ada boost:")
scores = cross_validate(clf_adaboost, X, y, cv=rkf.split(X), scoring=scoring, return_e
accAB, f1AB, recallAB, aucAB, testArrAB = printScores(scores)
```

```
accArr = np.array([accDT, accKNN, accSVC, accRF, accAB])
f1Arr = np.array([f1DT, f1KNN, f1SVC, f1RF, f1AB])
recallArr = np.array([recallDT, recallKNN, recallSVC, recallRF, recallAB])
aucArr = np.array([aucDT, aucKNN, aucSVC, aucRF, aucAB])

testScoresArr = np.array([testArrDT, testArrKNN, testArrSVC, testArrRF, testArrAB])
```

```
DecisionTree:
Precision average 0.812156862745098 with standard deviation 0.02854376403060423
F1 average 0.784408516342152 with standard deviation 0.03202103986272087
Recall average 0.7846760605124242 with standard deviation 0.032710517604903296
AUC average 0.7846760605124244 with standard deviation 0.0327105176049033
Test [accuracy, roc AUC, f1, precision, recall]
[0.84210526 0.82892893 0.87732342 0.88059701 0.87407407]
KNN:
Precision average 0.8037908496732026 with standard deviation 0.03176470588235294
F1 average 0.7830756361992742 with standard deviation 0.03501761634187004
Recall average 0.7948762190469268 with standard deviation 0.03443701598610628
AUC average 0.8695242325608025 with standard deviation 0.029821877354123717
Test [accuracy, roc AUC, f1, precision, recall]
[0.784689   0.83363363 0.82490272 0.86885246 0.78518519]
SVC:
Precision average 0.8483660130718953 with standard deviation 0.02449018822901332
F1 average 0.8334307934166105 with standard deviation 0.025822140023746137
Recall average 0.8494180778534464 with standard deviation 0.02297552655916109
AUC average 0.9172983682699644 with standard deviation 0.021409121779670134
Test [accuracy, roc AUC, f1, precision, recall]
[0.87559809 0.92522523 0.9        0.936      0.86666667]
Random forest:
Precision average 0.8722875816993464 with standard deviation 0.026507599397837152
F1 average 0.8493273984299182 with standard deviation 0.030395319693460075
Recall average 0.8401993072677727 with standard deviation 0.03133459395358015
AUC average 0.931217265590636 with standard deviation 0.022160026982249132
Test [accuracy, roc AUC, f1, precision, recall]
[0.86124402 0.91831832 0.89454545 0.87857143 0.91111111]
Ada boost:
Precision average 0.8454901960784312 with standard deviation 0.02781458537908356
F1 average 0.8194661758682241 with standard deviation 0.03067931648303135
Recall average 0.8135451479407578 with standard deviation 0.030293738100877578
AUC average 0.9067507600767993 with standard deviation 0.02745133093093605
Test [accuracy, roc AUC, f1, precision, recall]
[0.86124402 0.9015015  0.89454545 0.87857143 0.91111111]
```

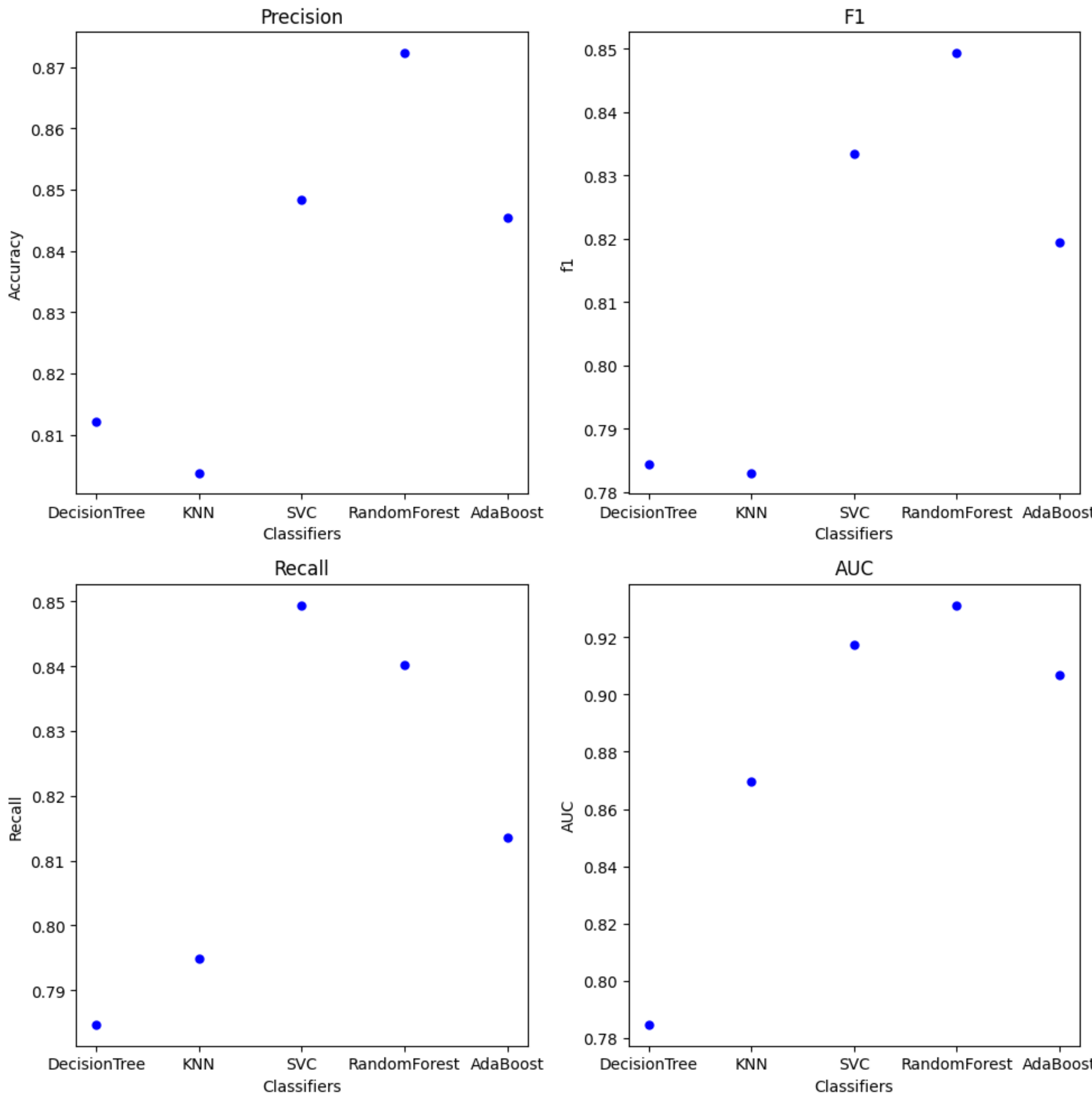
In [ ]:

```
print("Training")
xAxis = ['DecisionTree', 'KNN', 'SVC', 'RandomForest', 'AdaBoost']
f = plt.figure(figsize=(10, 10))
f.add_subplot(2, 2, 1)
plt.title("Precision")
plt.ylabel("Accuracy")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, accArr, "bo", markersize="5")

f.add_subplot(2, 2, 2)
plt.title("F1")
plt.ylabel("f1")
plt.xlabel("Classifiers")
plt.tight_layout()
```

```python
plt.plot(xAxis, f1Arr, "bo", markersize="5")

f.add_subplot(2, 2, 3)
plt.title("Recall")
plt.ylabel("Recall")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, recallArr, "bo", markersize="5")

f.add_subplot(2, 2, 4)
plt.title("AUC")
plt.ylabel("AUC")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, aucArr, "bo", markersize="5")
```

```
Training
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x244c4751930>]
```



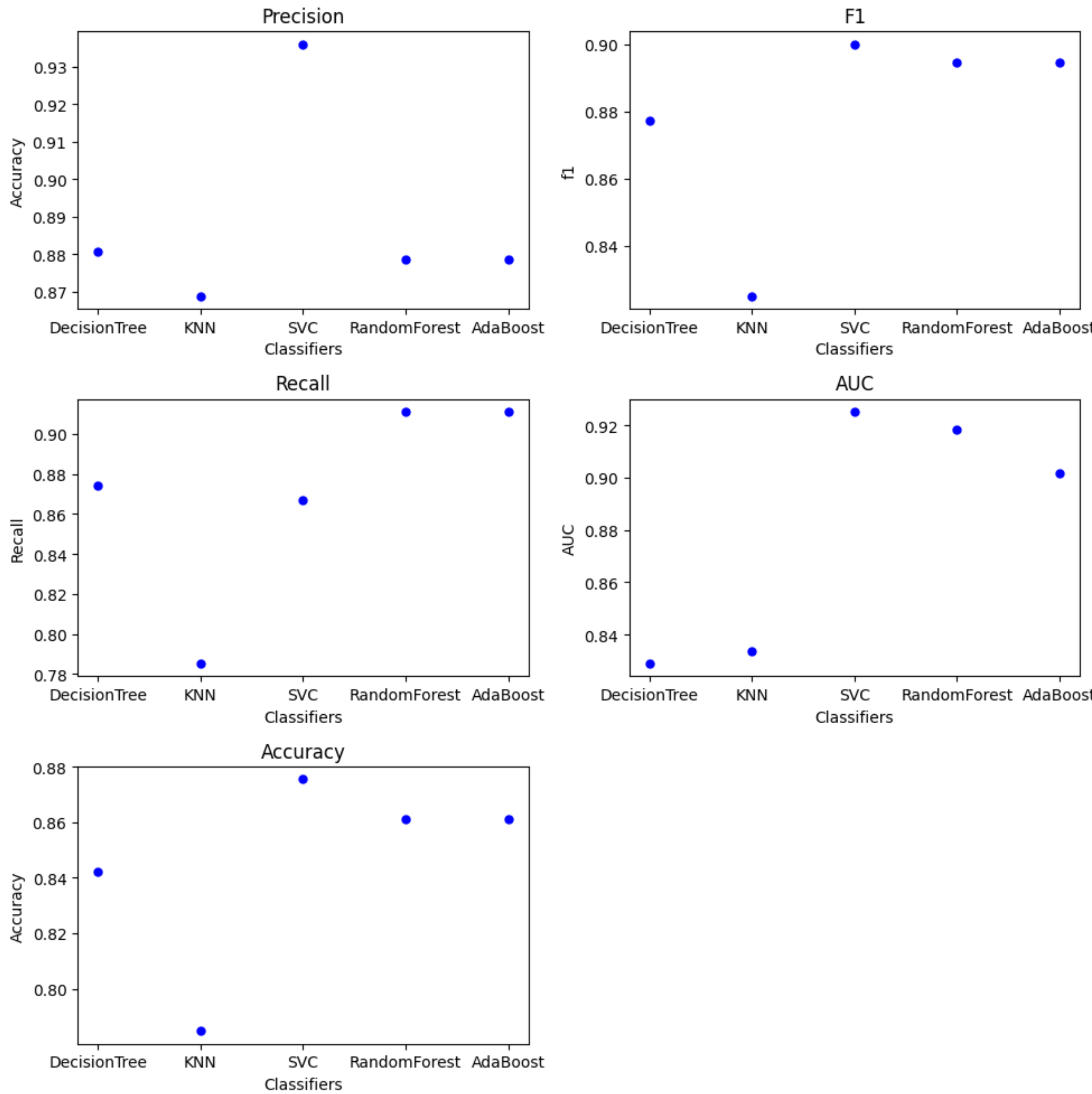
In [ ]:

```
print("Test")
xAxis = ['DecisionTree', 'KNN', 'SVC', 'RandomForest', 'AdaBoost']
f = plt.figure(figsize=(10, 10))
f.add_subplot(3, 2, 1)
plt.title("Precision")
plt.ylabel("Accuracy")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, testScoresArr[:, 3], "bo", markersize="5")

f.add_subplot(3, 2, 2)
plt.title("F1")
plt.ylabel("f1")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, testScoresArr[:, 2], "bo", markersize="5")

f.add_subplot(3, 2, 3)
plt.title("Recall")
plt.ylabel("Recall")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, testScoresArr[:, 4], "bo", markersize="5")

f.add_subplot(3, 2, 4)
plt.title("AUC")
plt.ylabel("AUC")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, testScoresArr[:, 1], "bo", markersize="5")

f.add_subplot(3, 2, 5)
plt.title("Accuracy")
plt.ylabel("Accuracy")
plt.xlabel("Classifiers")
plt.tight_layout()
plt.plot(xAxis, testScoresArr[:, 0], "bo", markersize="5")
```

```
Test
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x244bd1a0ca0>]
```



Comment on the performance of algorithms and visualize their final scores. How do they perform against the random baseline? What about the constant one? How do different learning scenarios impact the final score? Are the differences between the models statistically significant?

Random forest, SVC and AdaBoost performed better than others. DecisionTree and KNN performed significantly worse than others. They all perform better than random and constant baseline (50%/66% accuracy). Different parameters will change final scores and also different preprocessing of data might improve the final score (less noise/outliers, more accurate data,...). Differences of the RandomForest, SVC, AdaBoost are not very big, but depending on the needs of the classification different classification model might be more desirable.

Paramaters tuning (Random forest)

In [ ]:
```
params = {'n_estimators': [int(x) for x in np.linspace(start = 10, stop = 80, num = 10
               'max_features': ['sqrt'], # Number of features to consider at every spl
```

```
              'max_depth': [2,4], # Maximum number of levels in tree
              'min_samples_split':  [2, 5], # Minimum number of samples required to s
              'min_samples_leaf': [1, 2], # Minimum number of samples required at eac
              'bootstrap': [True, False]} # Method of selecting samples for training
rf_Model = RandomForestClassifier()
rf_Grid = GridSearchCV(estimator = rf_Model, param_grid = params, cv = 5, verbose=2, r
rf_Grid.fit(X, y)
rf_Grid.best_params_
rf_Random = RandomizedSearchCV(estimator = rf_Model, param_distributions = params, cv
rf_Random.fit(X, y)
rf_Random.best_params_
```

```
Fitting 5 folds for each of 160 candidates, totalling 800 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[ ]:

```
{'n_estimators': 72,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 4,
 'bootstrap': False}
```

In [ ]:

```
print (f'Train Accuracy - : {rf_Grid.score(X,y):.3f}')
print (f'Test Accuracy - : {rf_Grid.score(testX,testY):.3f}')
print (f'Train Accuracy - : {rf_Random.score(X,y):.3f}')
print (f'Test Accuracy - : {rf_Random.score(testX,testY):.3f}')

print(f'Test accuracy score, roc AUC, f1, percision, recall with the best estimator (m
pred_y_randomForest = rf_Random.best_estimator_.predict(testX)
print(accuracy_score(testY, pred_y_randomForest))
prob_y_randomForest = clf_randomForest.predict_proba(testX)
prob_y_randomForest = [p[1] for p in prob_y_randomForest]
print(roc_auc_score(testY, prob_y_randomForest))
print(f1_score(testY, pred_y_randomForest))
print(precision_score(testY, pred_y_randomForest))
print(recall_score(testY, pred_y_randomForest))
```

```
Train Accuracy - : 0.933
Test Accuracy - : 0.857
Train Accuracy - : 0.928
Test Accuracy - : 0.859
Test accuracy score, roc AUC, f1, percision, recall with the best estimator (model):
0.8133971291866029
0.925025025025025
0.8592057761732853
0.8380281690140845
0.8814814814814815
```

Parameter tuning (AdaBoost)

In [ ]:

```
adaBoostModel = AdaBoostClassifier(base_estimator=DecisionTreeClassifier())

params = {'base_estimator__max_depth':[i for i in range(2,11,2)],
          'base_estimator__min_samples_leaf':[5,10],
          'n_estimators':[10,50,250,1000],
          'learning_rate':[0.01,0.1]}

adaBoost_Grid = GridSearchCV(adaBoostModel, params, verbose=3, cv=5, scoring='f1', n_j
```

```
adaBoost_Grid.fit(X,y)
adaBoost_Grid.best_params_
adaBoost_Random = RandomizedSearchCV(estimator = adaBoostModel, param_distributions =
adaBoost_Random.fit(X, y)
adaBoost_Random.best_params_
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[ ]:

```
{'n_estimators': 250,
 'learning_rate': 0.01,
 'base_estimator__min_samples_leaf': 10,
 'base_estimator__max_depth': 10}
```

In [ ]:

```
print (f'Train Accuracy - : {adaBoost_Grid.score(X,y):.3f}')
print (f'Test Accuracy - : {adaBoost_Grid.score(testX,testY):.3f}')
print (f'Train Accuracy - : {adaBoost_Random.score(X,y):.3f}')
print (f'Test Accuracy - : {adaBoost_Random.score(testX,testY):.3f}')

print(f'Test accuracy score, roc AUC, f1, percision, recall with the best estimator (m
pred_y_adaboost = adaBoost_Grid.best_estimator_.predict(testX)
print(accuracy_score(testY, pred_y_adaboost))
prob_y_adaboost = clf_adaboost.predict_proba(testX)
prob_y_adaboost = [p[1] for p in prob_y_adaboost]
print(roc_auc_score(testY, prob_y_adaboost))
print(f1_score(testY, pred_y_adaboost))
print(precision_score(testY, pred_y_adaboost))
print(recall_score(testY, pred_y_adaboost))
```

```
Train Accuracy - : 0.998
Test Accuracy - : 0.887
Train Accuracy - : 1.000
Test Accuracy - : 0.863
Test accuracy score, roc AUC, f1, percision, recall with the best estimator (model):
0.8564593301435407
0.8983483483483484
0.8872180451127819
0.9007633587786259
0.8740740740740741
```

Parameter tuning (SVC)

In [ ]:

```
from scipy import stats

svcModel = SVC(kernel='linear',  class_weight='balanced', probability=True)
params = {"C": np.arange(2, 10, 2),
             "gamma": np.arange(0.1, 1, 0.2)}

svc_Grid = GridSearchCV(svcModel, param_grid = params, n_jobs = 4, cv = 5, scoring='f1
svc_Grid.fit(X, y)
svc_Grid.best_params_

params = {"C": stats.uniform(2, 10),
            "gamma": stats.uniform(0.1, 1)}

svc_Random = RandomizedSearchCV(svcModel, param_distributions = params, n_iter = 20, r
```

```
svc_Random.fit(X, y)
svc_Random.best_params_
```

Out[ ]: {'C': 11.307729610869862, 'gamma': 0.7495504104423961}

In [ ]:
```
print (f'Train Accuracy - : {svc_Grid.score(X,y):.3f}')
print (f'Test Accuracy - : {svc_Grid.score(testX,testY):.3f}')
print (f'Train Accuracy - : {svc_Random.score(X,y):.3f}')
print (f'Test Accuracy - : {svc_Random.score(testX,testY):.3f}')


print(f'Test accuracy score, roc AUC, f1, percision, recall with the best estimator (m
pred_y_svc = svc_Random.best_estimator_.predict(testX)
print(accuracy_score(testY, pred_y_svc))
prob_y_svc = clf_svc.predict_proba(testX)
prob_y_svc = [p[1] for p in prob_y_svc]
print(roc_auc_score(testY, prob_y_svc))
print(f1_score(testY, pred_y_svc))
print(precision_score(testY, pred_y_svc))
print(recall_score(testY, pred_y_svc))
```

```
Train Accuracy - : 0.914
Test Accuracy - : 0.893
Train Accuracy - : 0.913
Test Accuracy - : 0.889
Test accuracy score, roc AUC, f1, percision, recall with the best estimator (model):
0.861244019138756
0.917917917917918
0.8888888888888888
0.9206349206349206
0.8592592592592593
```