Introduction

Below we'll be training various variation of decision tree and support vector machine on breast cancer dataset. We'll divide dataset into train set(70%) and test set(30%). We'll then repeat process of training and evaluating 20 times dividing dataset. We'll keep record of all acuracy and report average accuracy after 20 steps.

Common Library Imports

In [1]:

```
import numpy as np ## Linear algebra library
import pandas as pd ## tabular data maintaining library

import sklearn
from sklearn import datasets ## datasets has breast cancer dataset
from sklearn.model_selection import train_test_split ## train_test_split lets us split data into train/test sets.
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score

import warnings
warnings.filterwarnings('ignore')
```

Loading Breast Cancer Dataset

In [2]:

```
breast_cancer_dataset = datasets.load_breast_cancer()
X, Y = breast_cancer_dataset.data, breast_cancer_dataset.target
print('Dataset Size : ', X.shape, Y.shape)
```

```
Dataset Size :  (569, 30) (569,)
```

Generic function for training model for 20 Iterations

Below we have created generic function which takes as input machine learning model and (features, target). It then loops 20 times and each time divide dataset into train/test set, train model on train set and evaluate on test set. It also keeps track of train and test accuracy during each run and return average of both once completed.

In [3]:

```
results = []

def fit_and_evaluate_model(model, X, Y):
    train_accuracies, test_accuracies,precision, recall = [], [], [], []
    for i in range(20):
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7
, test_size=0.3, stratify=Y)
        model.fit(X_train,Y_train)
        train_accuracies.append(model.score(X_train, Y_train))
        test_accuracies.append(model.score(X_test, Y_test))
        preds = model.predict(X_test)
        precision.append(precision_score(Y_test, preds))
        recall.append(recall_score(Y_test, preds))

    return np.array(train_accuracies) .mean(), np.array(test_accuracies) .mean
(), np.array(precision) .mean(), np.array(recall) .mean()
```

Decision Tree with Information Gain

In [4]:

```
dt = DecisionTreeClassifier(criterion='entropy') ## 'entropy' as criterion lets
 calculation based on Info Gain.

avg_train_accuracy, avg_test_accuracy,precision, recall = fit_and_evaluate_model
(dt, X, Y)
print('Average Train Accuracy : %.3f'%avg_train_accuracy)
print('Average Test Accuracy : %.3f'%avg_test_accuracy)
print('Precision : %.3f'%precision)
print('Recall : %.3f'%recall)

results.append(['DT1', avg_train_accuracy, avg_test_accuracy,precision, recall])
```

```
Average Train Accuracy : 1.000
Average Test Accuracy : 0.924
Precision : 0.943
Recall : 0.935
```

We can notice above that we are getting `100%` accuracy with train data whereas accuracy with test set is `92.9%`. This shows that our model is overfitting train data. We should try to find out model which has train and test accuracies almost same or has quite low difference.

Decision Tree with Various Tree Depths

We'll below try various `max_depth` for decision tree which will create decision trees with various lengths. We'll also keep track of average train and test accuracies for each depth and report it as well.

In [5]:

```
for depth in [2,3,4,5, 6, 7, 8, 9]:
    dt = DecisionTreeClassifier(criterion='entropy', max_depth=depth)

    avg_train_accuracy, avg_test_accuracy, precision, recall = fit_and_evaluate_
model(dt, X, Y)
    print('\nTree Depth : %d'%depth)
    print('Average Train Accuracy : %.3f'%avg_train_accuracy)
    print('Average Test Accuracy : %.3f'%avg_test_accuracy)
    print('Precision : %.3f'%precision)
    print('Recall : %.3f'%recall)

    results.append(['DT2-Depth : %d'%depth, avg_train_accuracy, avg_test_accurac
y,precision, recall])
```

```
Tree Depth : 2
Average Train Accuracy : 0.937
Average Test Accuracy : 0.908
Precision : 0.936
Recall : 0.918

Tree Depth : 3
Average Train Accuracy : 0.970
Average Test Accuracy : 0.925
Precision : 0.928
Recall : 0.956

Tree Depth : 4
Average Train Accuracy : 0.985
Average Test Accuracy : 0.923
Precision : 0.929
Recall : 0.950

Tree Depth : 5
Average Train Accuracy : 0.995
Average Test Accuracy : 0.932
Precision : 0.941
Recall : 0.951

Tree Depth : 6
Average Train Accuracy : 0.999
Average Test Accuracy : 0.927
Precision : 0.943
Recall : 0.940

Tree Depth : 7
Average Train Accuracy : 1.000
Average Test Accuracy : 0.931
Precision : 0.946
Recall : 0.944

Tree Depth : 8
Average Train Accuracy : 1.000
Average Test Accuracy : 0.920
Precision : 0.939
Recall : 0.934

Tree Depth : 9
Average Train Accuracy : 1.000
Average Test Accuracy : 0.931
Precision : 0.952
Recall : 0.937
```

We can notice above that `max_depth` of 2 and 3 has less difference between train and test accuracy hence both are good choice for tree depth. All other are overfitting train data and giving `100%` accuracy whereas giving `~93%` accuracy for test data.

SVM with Linear Kernel

Below we'll try support vector machine with Linear kernel.

In [6]:

```
linear_svm = SVC(kernel='linear')

avg_train_accuracy, avg_test_accuracy, precision, recall = fit_and_evaluate_mode
l(linear_svm, X, Y)
print('Average Train Accuracy : %.3f'%avg_train_accuracy)
print('Average Test  Accuracy : %.3f'%avg_test_accuracy)
print('Precision : %.3f'%precision)
print('Recall : %.3f'%recall)
results.append(['SVM1', avg_train_accuracy, avg_test_accuracy,precision, recall
])
```

```
Average Train Accuracy : 0.967
Average Test  Accuracy : 0.950
Precision : 0.952
Recall : 0.970
```

We can notice above that SVM with linear kernel is performing quite well and difference between train and tet accuracy is quite less hence model has become quite generic.

SVM with RBF Kernel

We'll try below SVM with RBF kernel.

In [7]:

```
rbf_svm = SVC(kernel='rbf')

avg_train_accuracy, avg_test_accuracy, precision, recall = fit_and_evaluate_mode
l(rbf_svm, X, Y)
print('Average Train Accuracy : %.3f'%avg_train_accuracy)
print('Average Test Accuracy : %.3f'%avg_test_accuracy)
print('Precision : %.3f'%precision)
print('Recall : %.3f'%recall)
results.append(['SVM2', avg_train_accuracy, avg_test_accuracy,precision, recall
])
```

```
Average Train Accuracy : 1.000
Average Test Accuracy : 0.626
Precision : 0.626
Recall : 1.000
```

We can notice from above accuracies that SVM with rbf kernel has clearly overfit train data with 100% accuracy whereas test accuracy is only %62.6 which is not good.

SVM with RBF Kernel (Regularization parameter C tried with various values)

Below we'll try various values of regularization parameter C and check performance of SVM for each values. We'll try to check whether any combination is able to beat above bad performance.

In [8]:

```
for c in [0.01, 0.1, 1.0, 10.0]:
    rbf_svm = SVC(kernel='rbf', C=c)

    avg_train_accuracy, avg_test_accuracy, precision, recall = fit_and_evaluate_
model(rbf_svm, X, Y)

    print('\nSVC Regularization parameter C : %.2f'%c)
    print('Average Train Accuracy : %.3f'%avg_train_accuracy)
    print('Average Test Accuracy : %.3f'%avg_test_accuracy)
    print('Precision : %.3f'%precision)
    print('Recall : %.3f'%recall)
    results.append(['SVM3-C : %.2f'%c, avg_train_accuracy, avg_test_accuracy,pre
cision, recall])
```

```
SVC Regularization parameter C : 0.01
Average Train Accuracy : 0.628
Average Test Accuracy : 0.626
Precision : 0.626
Recall : 1.000

SVC Regularization parameter C : 0.10
Average Train Accuracy : 0.628
Average Test Accuracy : 0.626
Precision : 0.626
Recall : 1.000

SVC Regularization parameter C : 1.00
Average Train Accuracy : 1.000
Average Test Accuracy : 0.626
Precision : 0.626
Recall : 1.000

SVC Regularization parameter C : 10.00
Average Train Accuracy : 1.000
Average Test Accuracy : 0.626
Precision : 0.626
Recall : 1.000
```

In [9]:

```
df = pd.DataFrame(results, columns=['Model Name', 'Avg Train Accuracy', 'Avg Test Accuracy','Precision', 'Recall'])
df
```

Out[9]:

	Model Name	Avg Train Accuracy	Avg Test Accuracy	Precision	Recall
0	DT1	1.000000	0.923977	0.943396	0.935047
1	DT2-Depth : 2	0.936809	0.907602	0.936163	0.918224
2	DT2-Depth : 3	0.970477	0.925146	0.927710	0.955607
3	DT2-Depth : 4	0.985427	0.923099	0.929187	0.950467
4	DT2-Depth : 5	0.995226	0.931579	0.940510	0.951402
5	DT2-Depth : 6	0.998744	0.926608	0.943323	0.939720
6	DT2-Depth : 7	0.999874	0.930994	0.946107	0.944393
7	DT2-Depth : 8	1.000000	0.919883	0.938573	0.933645
8	DT2-Depth : 9	1.000000	0.930994	0.952359	0.936916
9	SVM1	0.966834	0.950292	0.951851	0.970093
10	SVM2	1.000000	0.625731	0.625731	1.000000
11	SVM3-C : 0.01	0.628141	0.625731	0.625731	1.000000
12	SVM3-C : 0.10	0.628141	0.625731	0.625731	1.000000
13	SVM3-C : 1.00	1.000000	0.625731	0.625731	1.000000
14	SVM3-C : 10.00	1.000000	0.625731	0.625731	1.000000

We can notice above that regularization parameter `C` has values `0.01, 0.1 and 0.5` give same accuracy for train and test sets hence genralized well. All other values are overfitting on train dataset.

Conclusion

We can notice after performing above experiments that decision tree with tree depth of `2` and `3` gives good results and SVM with `linear kernel` gives quite good results compared to rbf kernel.

In [ ]:

```

```

In [ ]:

```

```