REPORT

Packages used: Scikit-learn

1. Choose Dataset:

The dataset chosen is Haberman Dataset which is taken form UCI machine learning repository.

https://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.data

The dataset features includes:

Data Set Characteristics:	Multivariate	**Number of Instances:**	306	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	3	**Date Donated**	1999-03-04
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	128276

2. Pre-process the dataset:

- The first step involves dropping the null values, removing the duplicate values.(redundant tuples)
- Since there are no categorical values there is no need to for conversion of categorical into numerical values.
- Next step involves fitting and transform of data frame using standard Scaler.

3. Finding best parameters:

This table represents the best set of parameters of the classifiers and the accuracy of training set and test dataset.

SR.No	Classifier	Cross Validation fold	Parameter1	Parameter2	Parameter3	Parameter4	Parameter 5	Parameter 6	Average Training Accuracy	Average test Accuracy
1.	SVM	10	KERNEL =’Sigmoid’	Gamma=0.1	decision_function_shape='ovo'	probability=True	coef0=1		72.41	75.0
2.	Naïve Bayes	2	Priors=None						75.86	72.22
3.	Decision trees	2	Criterion=’gini’	Splitter=’best’	Max-depth=7				71.42	86.89
4.	Perceptron	10	penalty='elasticnet’	alpha=0.001	shuffle=False	n_iter =10			72.79	75.0
5.	Neural Netwrok	10	solver='lbfgs'	alpha=1e-5	hidden_layer_sizes=(5, 5)	random_state=1	activation='relu'	max_iter =200	72.41	78.22
6.	Logistic Regression	2	penalty='l2'	C=1.0	fit_intercept=True	class_weight=None	solver= ‘liblinear’	max_iter =100	77.93	79.57
7.	KNN	10	n_neighbors=2	algorithm= kd_tree	p=10				81.99	71.43
8.	Bagging	2	base_estimator=None	n_estimators=5	bootstrap=True	max_samples=1.0	oob_score=False		93.10	69.44
9.	Random Forest	5	n_estimators=10	max_depth =2	max_features="auto"	min_samples_split=2	max_leaf_nodes=None		73.27	75.43
10.	Adaboosting	10	base_estimator =None	n_estimators=40	learning_rate=0.9	algorithm='SAMME.R’			78.54	78.57
11.	Gradient Boosting	10	loss= exponential	learning_rate=0.1	n_estimators =100	criterion='friedman_mse'	min_samples_split =2	Max_depth=2	81.22	82.14
12.	Deep Learning	10	Number of layers=3	Kernel_initializer= ’uniform’	Activation=’relu’ and ‘sigmoid’	epochs=150	Loss=’binary_cross entropy’	Batch_size=10	72..41	74.12

4. Final Results

Number of instances: 306

Number of Attributes: 3

This table represents the classifiers alongwith other evaluation metrics including precision, recall and f-1 score

Classifier	Precision	Recall	F-1 Score
SVM	0.70	0.73	0.69
Naïve Bayes	0.69	0.72	0.68
Decision Trees	0.67	0.71	0.76
Perceptron	0.72	0.75	0.72
Logistic Regression	0.70	0.74	0.71
Neural Network	0.69	0.68	0.69
KNN	0.69	0.67	0.68
Bagging	0.73	0.69	0.72
Random Forest	0.71	0.67	0.68
Adaboosting	0.79	0.79	0.79
Gradient Boosting	0.83	0.82	0.83
Deep Learning	0.68	0.68	0.67

5. Analysis of Results:

Strong Classifier:

- The strongest classifier is Gradient boosting with a testing accuracy of 82.14% and training accuracy of 81.22%. The set of parameters used are :
    1. Loss=exponential
    2. n_estimators=100
    3. crtiterion=friedman_mse
    4. min_samples_split=2
    5. max_depth=2
- Reason for best performance: Gradient boosting combines weak "learners" into a single strong learner in an iterative fashion which works very strongly on classification and regression dataset and we can see the results for gradient boosting classifier and here the gradient boosting is fast and accurate than others.
- It involved various hyperparameter tuning like loss factor, total estimators, depth of the tree etc which yielded better results. Also the K fold cross validation performed on it helped in improving the accuracy.

Weak Classifier:

- The weak classifier is bagging with a testing accuracy of : 69.44% and a training accuracy of 93.10%
- Reason: On the given dataset since the number of instances is only 306 so while training through the bagging classifier the training accuracy turned out to be 93.10%, which kind of overfits the model.

  So the testing accuracy i.e., the accuracy on unseen dataset resulted into a low value. In addition, the number of hyperparameters which can be tuned through bagging are less and not giving better results for this dataset.

Attribute importance and its influence:

Attributes plays an important role in any classifier. Some attributes have more influence as compared to other attributes:

1. **SVM** : In our case the attributes like kernel , degree and gamma have more influence as compared to other attributes like coef, shrinking, probability, tol, etc.

2. **Decision trees:** The attributes criterion, splitter, max_depth played more important role as compared to other attributes.

3. **Perceptron:** The attributes like n_iter, alpha and penalty played more important role as compared to verbose, class_weight and other parameters.

4. **Neural Network:** The attributes like number of hidden layers. Alpha and activation function plays more important role as compared to other attributes.

5. **Logistic Regression:** The parameter like solver = liblinear have more influence.

6. **KNN:** number of neighbours and algorithm plays more important role.

7. **Bagging:** The number of estimators are more important(n_estimators).

8. **Random Forest:** Influential attributes are max_depth, min_samples_split and n_estimators.

9. **Adaboosting:** Influential Attributes are n_estimators, learning rate, algorithm.

10. **Gradient Boosting:** Influential attributes are max_depth, learning-rate, criterion and min_sample_split.

11. **Deep learning:** Influential attributes are kernel_initializer and activation.

Measuring Performance:

- For measuring the performance, only calculating the accuracy is not a good option. This is because with exactly **zero predictive power**, and yet, we got an increase in accuracy. So for this reason we also calculated Precision, recall and F Score.

- **Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

  Precision = TP/TP+FP (TP=true positive,FP=False positive)
- **Recall** (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in target class.

  Recall = TP/TP+FN FP (TP=true positive,FN=False negative)

- These quantities are also related to the (F_1) score, which is defined as the harmonic mean of precision and recall.

ROC Curve:

We have also made ROC curve for each classifier to visualize the performance. **ROC curve**, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied i.e., the plot against FPR and TPR.

Classifier	Area under ROC Curve
SVM	0.64
Naïve Bayes	0.69
Perceptron	0.59
Logistic Regression	0.59
Neural Network	0.59
Deep learning	0.50
KNN	0.70
Bagging	0.65
Random Forest	0.72
Adaboosting	0.72
Gradient Boosting	0.75
Decision Tree	0.61

6.Psuedo Code:

HabermanDataset (https://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.data):

```
{
        dataframe read_csv;
        df.dropna()    #removing null values
        df.drop_duplicates()  #removing duplicate values
        check for categorical values  # no categorical values in our dataset
        check for impure values                 # no impure values in our dataset
        Fit the data
        Transform the data
}
        Build_model()

def build_model():
        svm()
        nb()
        KNN()
        bagging()
        random_forest()
        Neurak_network()
        deep_learning()
        dt()
        gradient_boosting()
```

```
        adaboosting()
        LR()
        perceptron()



def svm():
        K_fold()
        clf = svm.SVC(kernel='sigmoid',gamma=0.1,coef0=1,decision_function_shape='ovo',probability=True)
        clf.fit(X_train, Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def KNN():
        K_fold()
        clf = KNeighborsClassifier(n_neighbors=2,algorithm='kd_tree',p=10)
        clf.fit(X_train, Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def nb():               #naive bayes
        K_fold()
        clf=GaussianNB(priors=None)
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def random_forest():
        K_fold()
        clf=RandomForestClassifier(n_estimators=10,max_depth=2, random_state=0, max_features="auto", min_samples_split=2,
        max_leaf_nodes=None)
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def bagging():
        K_fold()
        clf=BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False, max_features=1.0, max_samples=1.0,
                n_estimators=5, n_jobs=1, oob_score=False, random_state=None, verbose=0)
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def Neural_network():
        K_fold()
        clf = MLPClassifier(activation='relu',solver='lbfgs',hidden_layer_sizes=(5,5), random_state=1)  #finding best set of parameters
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def LR():
        K_fold()
        clf=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr',
        n_jobs=2, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()


def perceptron():
        K_fold()
        clf = Perceptron(penalty='elasticnet',alpha=0.001,n_iter=10)
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def adaboosting():
        K_fold()
        clf=AdaBoostClassifier(base_estimator=None, n_estimators=40, learning_rate=0.9, algorithm='SAMME.R', random_state=None)
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def gradient_boosting():
        K_fold()
        clf=GradientBoostingClassifier(loss='exponential',learning_rate=0.1, n_estimators=100, criterion='friedman_mse',
        min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=2, min_impurity_split=None, init=None,
        random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')
        clf.fit(X_train,Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def deep_learning():
        K_fold()
        # create model
        model = Sequential()
        model.add(Dense(8, input_dim=3, activation='relu'))
        model.add(Dense(4, input_dim=8, activation='relu'))
        model.add(Dense(6, input_dim=8, activation='relu'))
        model.add(Dense(3, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))
        # Compile model
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def decisionTree():
        K_fold()
        clf = tree.DecisionTreeClassifier(criterion='gini', splitter='best',max_depth=7,presort=True)
        clf.fit(X_train, Y_train)
        train_accuracy()
        test_accuracy()
        roc()
        area_roc()

def K_fold():
        k_fold = KFold(n_splits=best_split)
        for train_indices, test_indices in k_fold.split(X):
                X_train,X_test=X[train_indices],X[test_indices]
                Y_train,Y_test=Y[train_indices],Y[test_indices]
        return X_train,X_test,Y_train,Y_test
```

```
def train_accuracy():
        ta=clf.score(X_train,Y_train,sample_weight=None)*100
        return ta

def test_accuracy():
        tb=clf.score(X_test,Y_test,sample_weight=None)*100
        return tb

def roc():
        fpr, tpr, _ = roc_curve(Y_test, pred)
        plt.plot(fpr,tpr)
        plt.show()

def area_roc():
        from sklearn.metrics import roc_auc_score
        pred=clf.predict(X_test)
        area=roc_auc_score(Y_test,pred)
        return area
```