# Machine Learning LAB - Assessment 3

20BCE0083- Jeevan Yohan Varghese

## 1. K nearest neighbour

**Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.Print both correct and wrong predictions.**

In [21]:
```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [22]:
```
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [26]:
```
def predict(X_train, y_train, x, k):
    #predicting class of a sample
    dist = np.sqrt(np.sum((X_train - x)**2, axis=1))
    nearest_indices = np.argsort(dist)[:k]
    nearest_labels = y_train[nearest_indices]
    return np.argmax(np.bincount(nearest_labels))

def predict_multiple(X_train, y_train, X_test, k):
    #utitlity function to predict multiple samples
    y_pred = np.array([predict(X_train, y_train, x, k) for x in X_test])
    return y_pred
```

In [27]:
```
# k - no of neighbours
k = 10

y_pred = predict_multiple(X_train, y_train, X_test, k)

# accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9
```

In [28]:
```
#predicting all
for i in range(len(y_pred)):
    if y_pred[i] == y_test[i]:
        print("Sample", i, ":  correct prediction as class", y_pred[i])
    else:
        print("Sample", i, "incorrect prediction as class", y_pred[i], "actu
```

```
Sample 0 :  correct prediction as class 1
Sample 1 :  correct prediction as class 1
Sample 2 :  correct prediction as class 2
Sample 3 :  correct prediction as class 1
Sample 4 :  correct prediction as class 1
Sample 5 :  correct prediction as class 0
Sample 6 :  correct prediction as class 0
Sample 7 :  correct prediction as class 0
Sample 8 :  correct prediction as class 1
Sample 9 incorrect prediction as class 1 actual class :  2
Sample 10 :  correct prediction as class 0
Sample 11 :  correct prediction as class 2
Sample 12 :  correct prediction as class 1
Sample 13 :  correct prediction as class 2
Sample 14 :  correct prediction as class 2
Sample 15 :  correct prediction as class 1
Sample 16 :  correct prediction as class 2
Sample 17 incorrect prediction as class 1 actual class :  2
Sample 18 :  correct prediction as class 1
Sample 19 :  correct prediction as class 0
Sample 20 :  correct prediction as class 0
Sample 21 :  correct prediction as class 2
Sample 22 :  correct prediction as class 0
Sample 23 :  correct prediction as class 1
Sample 24 :  correct prediction as class 1
Sample 25 :  correct prediction as class 0
Sample 26 :  correct prediction as class 1
Sample 27 :  correct prediction as class 1
Sample 28 :  correct prediction as class 0
Sample 29 incorrect prediction as class 2 actual class :  1
```

## 2. SVM

**Train SVM classifier using sklearn digits dataset( i.e from sklearn datasets import load_digits)**

In [11]:
```
from sklearn import svm
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [12]:
```
digits_data = load_digits()

X_train, X_test, y_train, y_test =train_test_split(digits_data.data, digits_
```

In [13]:
```
rbf = svm.SVC(kernel='rbf')
rbf.fit(X_train, y_train)
y_pred_rbf = rbf.predict(X_test)
rbf_accuracy = accuracy_score(y_test, y_pred_rbf)
print("RBF kernel accuracy:", rbf_accuracy)
```

```
RBF kernel accuracy: 0.9916666666666667
```

In [14]:
```
lr = svm.SVC(kernel='linear')
```

```
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
linear_accuracy = accuracy_score(y_test, y_pred_lr)
print("Linear kernel accuracy:", linear_accuracy)
```

```
Linear kernel accuracy: 0.9777777777777777
```

In [15]:
```
rbf = svm.SVC(kernel='rbf',C=0.5,gamma=0.001)
rbf.fit(X_train, y_train)
y_pred_rbf = rbf.predict(X_test)
rbf_accuracy = accuracy_score(y_test, y_pred_rbf)
print("Tuned RBF kernel accuracy:", rbf_accuracy)
```

```
Tuned RBF kernel accuracy: 0.9944444444444445
```

In [16]:
```
lr = svm.SVC(kernel='linear',C=1,gamma=0.001)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
linear_accuracy = accuracy_score(y_test, y_pred_lr)
print("Tuned Linear kernel accuracy:", linear_accuracy)
```

```
Tuned Linear kernel accuracy: 0.9777777777777777
```

## 3. ANN

**Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.**

In [17]:
```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
y = y/100
def sigmoid (x):
 return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x):
 return x * (1 - x)
```

In [18]:
```
epoch=5
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

In [19]:
```
for i in range(epoch):

 hinp1=np.dot(X,wh)
 hinp=hinp1 + bh
 hlayer_act = sigmoid(hinp)
 outinp1=np.dot(hlayer_act,wout)
 outinp= outinp1+bout
```

```python
output = sigmoid(outinp)

EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO * outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad

wout += hlayer_act.T.dot(d_output) *lr
wh += X.T.dot(d_hiddenlayer) *lr

print ("Epoch-", i+1, "Starting")
print("Input: \n" + str(X))
print("Actual: \n" + str(y))
print("Predicted Output: \n" ,output)
print ("Epoch-", i+1, "Ending\n")
```

```
Epoch- 1 Starting
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84845988]
 [0.83935393]
 [0.84840029]]
Epoch- 1 Ending

Epoch- 2 Starting
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84891601]
 [0.83978841]
 [0.84884992]]
Epoch- 2 Ending

Epoch- 3 Starting
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84936544]
 [0.84021661]
 [0.84929297]]
Epoch- 3 Ending

Epoch- 4 Starting
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.84980831]
```

```
 [0.84063868]
 [0.84972956]]
Epoch- 4 Ending

Epoch- 5 Starting
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.85024476]
 [0.84105475]
 [0.85015984]]
Epoch- 5 Ending
```

In [20]:

```
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.85024476]
 [0.84105475]
 [0.85015984]]
```

## 4. Bagging Ensembles including Bagged Decision Trees, Random Forest and Extra Trees

In [30]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classificatior
%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

In [31]:

```
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[31]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunc |
|---|---|---|---|---|---|---|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2 |

In [32]:

```
pd.set_option('display.float_format', '{:.2f}'.format)
df.describe()
```

Out[32]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigr |
|---|---|---|---|---|---|---|---|
| count | 768.00 | 768.00 | 768.00 | 768.00 | 768.00 | 768.00 | |
| mean | 3.85 | 120.89 | 69.11 | 20.54 | 79.80 | 31.99 | |
| std | 3.37 | 31.97 | 19.36 | 15.95 | 115.24 | 7.88 | |
| min | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| 25% | 1.00 | 99.00 | 62.00 | 0.00 | 0.00 | 27.30 | |
| 50% | 3.00 | 117.00 | 72.00 | 23.00 | 30.50 | 32.00 | |
| 75% | 6.00 | 140.25 | 80.00 | 32.00 | 127.25 | 36.60 | |
| max | 17.00 | 199.00 | 122.00 | 99.00 | 846.00 | 67.10 | |

In [34]:

```
categorical_val = []
continous_val = []
for column in df.columns:
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
df.columns
```

Out[34]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [35]:

```
feature_columns = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'
]

for column in feature_columns:
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
Pregnancies ==> Missing zeros : 111
Glucose ==> Missing zeros : 5
BloodPressure ==> Missing zeros : 35
SkinThickness ==> Missing zeros : 227
Insulin ==> Missing zeros : 374
BMI ==> Missing zeros : 11
DiabetesPedigreeFunction ==> Missing zeros : 0
Age ==> Missing zeros : 0
```

In [36]:

```python
fill_values = SimpleImputer(missing_values=0, strategy="mean", copy=False)
df[feature_columns] = fill_values.fit_transform(df[feature_columns])

for column in feature_columns:
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
```

```
Pregnancies ==> Missing zeros : 0
Glucose ==> Missing zeros : 0
BloodPressure ==> Missing zeros : 0
SkinThickness ==> Missing zeros : 0
Insulin ==> Missing zeros : 0
BMI ==> Missing zeros : 0
DiabetesPedigreeFunction ==> Missing zeros : 0
Age ==> Missing zeros : 0
```

In [37]:

```python
X = df[feature_columns]
y = df.Outcome

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar
```

In [44]:

```python
def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("Training Results: \n")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, c
    print(f"Confusion Matrix:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"Accuracy:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"Classification Report:\n{clf_report}")

    print("Testing results: \n")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, out
    print(f"Confusion Matrix:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"Accuracy:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"Classification Report:\n{clf_report}")
```

In [39]:

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [45]:

```python
tree = DecisionTreeClassifier()
bagging_clf = BaggingClassifier(estimator=tree, n_estimators=1500, random_st
bagging_clf.fit(X_train, y_train)

evaluate(bagging_clf, X_train, X_test, y_train, y_test)
```

```
Training Results:

Confusion Matrix:
[[349   0]
 [  0 188]]
Accuracy:
1.0000
Classification Report:
                0      1  accuracy  macro avg  weighted avg
precision    1.00   1.00      1.00       1.00          1.00
recall       1.00   1.00      1.00       1.00          1.00
f1-score     1.00   1.00      1.00       1.00          1.00
support    349.00 188.00      1.00     537.00        537.00
Testing results:

Confusion Matrix:
[[119  32]
 [ 24  56]]
Accuracy:
0.7576
Classification Report:
                0     1  accuracy  macro avg  weighted avg
precision    0.83  0.64      0.76       0.73          0.76
recall       0.79  0.70      0.76       0.74          0.76
f1-score     0.81  0.67      0.76       0.74          0.76
support    151.00 80.00      0.76     231.00        231.00
```

In [46]:

```
#Random Forest
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(random_state=42, n_estimators=1000)
rf_clf.fit(X_train, y_train)
evaluate(rf_clf, X_train, X_test, y_train, y_test)
```

```
Training Results:

Confusion Matrix:
[[349   0]
 [  0 188]]
Accuracy:
1.0000
Classification Report:
                0      1  accuracy  macro avg  weighted avg
precision    1.00   1.00      1.00       1.00          1.00
recall       1.00   1.00      1.00       1.00          1.00
f1-score     1.00   1.00      1.00       1.00          1.00
support    349.00 188.00      1.00     537.00        537.00
Testing results:

Confusion Matrix:
[[123  28]
 [ 29  51]]
Accuracy:
0.7532
Classification Report:
                0     1  accuracy  macro avg  weighted avg
precision    0.81  0.65      0.75       0.73          0.75
recall       0.81  0.64      0.75       0.73          0.75
f1-score     0.81  0.64      0.75       0.73          0.75
support    151.00 80.00      0.75     231.00        231.00
```

In [47]:

```
#Extra Trees
from sklearn.ensemble import ExtraTreesClassifier
ex_tree_clf = ExtraTreesClassifier(n_estimators=1000, max_features=7, random
ex_tree_clf.fit(X_train, y_train)
evaluate(ex_tree_clf, X_train, X_test, y_train, y_test)
```

```
Training Results:

Confusion Matrix:
[[349   0]
 [  0 188]]
Accuracy:
1.0000
Classification Report:
                0      1  accuracy  macro avg  weighted avg
precision    1.00   1.00      1.00       1.00          1.00
recall       1.00   1.00      1.00       1.00          1.00
f1-score     1.00   1.00      1.00       1.00          1.00
support    349.00 188.00      1.00     537.00        537.00
Testing results:

Confusion Matrix:
[[124  27]
 [ 25  55]]
Accuracy:
0.7749
Classification Report:
                0     1  accuracy  macro avg  weighted avg
precision    0.83  0.67      0.77       0.75          0.78
recall       0.82  0.69      0.77       0.75          0.77
f1-score     0.83  0.68      0.77       0.75          0.78
support    151.00 80.00      0.77     231.00        231.00
```

## 5. Boosting Ensembles including AdaBoost and Stochastic Gradient Boosting

In [48]:

```python
from sklearn.ensemble import AdaBoostClassifier
ada_boost_clf = AdaBoostClassifier(n_estimators=30)
ada_boost_clf.fit(X_train, y_train)
evaluate(ada_boost_clf, X_train, X_test, y_train, y_test)
```

```
Training Results:

Confusion Matrix:
[[310  39]
 [ 51 137]]
Accuracy:
0.8324
Classification Report:
                0      1  accuracy  macro avg  weighted avg
precision   0.86   0.78      0.83       0.82          0.83
recall      0.89   0.73      0.83       0.81          0.83
f1-score    0.87   0.75      0.83       0.81          0.83
support   349.00 188.00      0.83     537.00        537.00
Testing results:

Confusion Matrix:
[[123  28]
 [ 27  53]]
Accuracy:
0.7619
Classification Report:
                0     1  accuracy  macro avg  weighted avg
precision   0.82  0.65      0.76       0.74          0.76
recall      0.81  0.66      0.76       0.74          0.76
f1-score    0.82  0.66      0.76       0.74          0.76
support   151.00 80.00      0.76     231.00        231.00
```

In [49]:

```
#Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
grad_boost_clf = GradientBoostingClassifier(n_estimators=100, random_state=4
grad_boost_clf.fit(X_train, y_train)
evaluate(grad_boost_clf, X_train, X_test, y_train, y_test)
```

```
Training Results:

Confusion Matrix:
[[342   7]
 [ 19 169]]
Accuracy:
0.9516
Classification Report:
               0      1  accuracy  macro avg  weighted avg
precision   0.95   0.96      0.95       0.95          0.95
recall      0.98   0.90      0.95       0.94          0.95
f1-score    0.96   0.93      0.95       0.95          0.95
support   349.00 188.00      0.95     537.00        537.00
Testing results:

Confusion Matrix:
[[116  35]
 [ 26  54]]
Accuracy:
0.7359
Classification Report:
               0     1  accuracy  macro avg  weighted avg
precision   0.82  0.61      0.74       0.71          0.74
recall      0.77  0.68      0.74       0.72          0.74
f1-score    0.79  0.64      0.74       0.72          0.74
support   151.00 80.00      0.74     231.00        231.00
```

## 6. Voting Ensembles for averaging the predictions for any arbitrary models.

In [52]:

```python
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
estimators = []
log_reg = LogisticRegression(solver='liblinear')
estimators.append(('Logistic', log_reg))

tree = DecisionTreeClassifier()
estimators.append(('Tree', tree))

svm_clf = SVC(gamma='scale')
estimators.append(('SVM', svm_clf))

voting = VotingClassifier(estimators=estimators)
voting.fit(X_train, y_train)

evaluate(voting, X_train, X_test, y_train, y_test)
```

```
Training Results:

Confusion Matrix:
[[327  22]
 [ 82 106]]
Accuracy:
0.8063
Classification Report:
               0      1  accuracy  macro avg  weighted avg
precision   0.80   0.83      0.81       0.81          0.81
recall      0.94   0.56      0.81       0.75          0.81
f1-score    0.86   0.67      0.81       0.77          0.80
support   349.00 188.00      0.81     537.00        537.00
Testing results:

Confusion Matrix:
[[131  20]
 [ 38  42]]
Accuracy:
0.7489
Classification Report:
               0     1  accuracy  macro avg  weighted avg
precision   0.78  0.68      0.75       0.73          0.74
recall      0.87  0.53      0.75       0.70          0.75
f1-score    0.82  0.59      0.75       0.71          0.74
support   151.00 80.00      0.75     231.00        231.00
```

In [ ]: