Practical -7

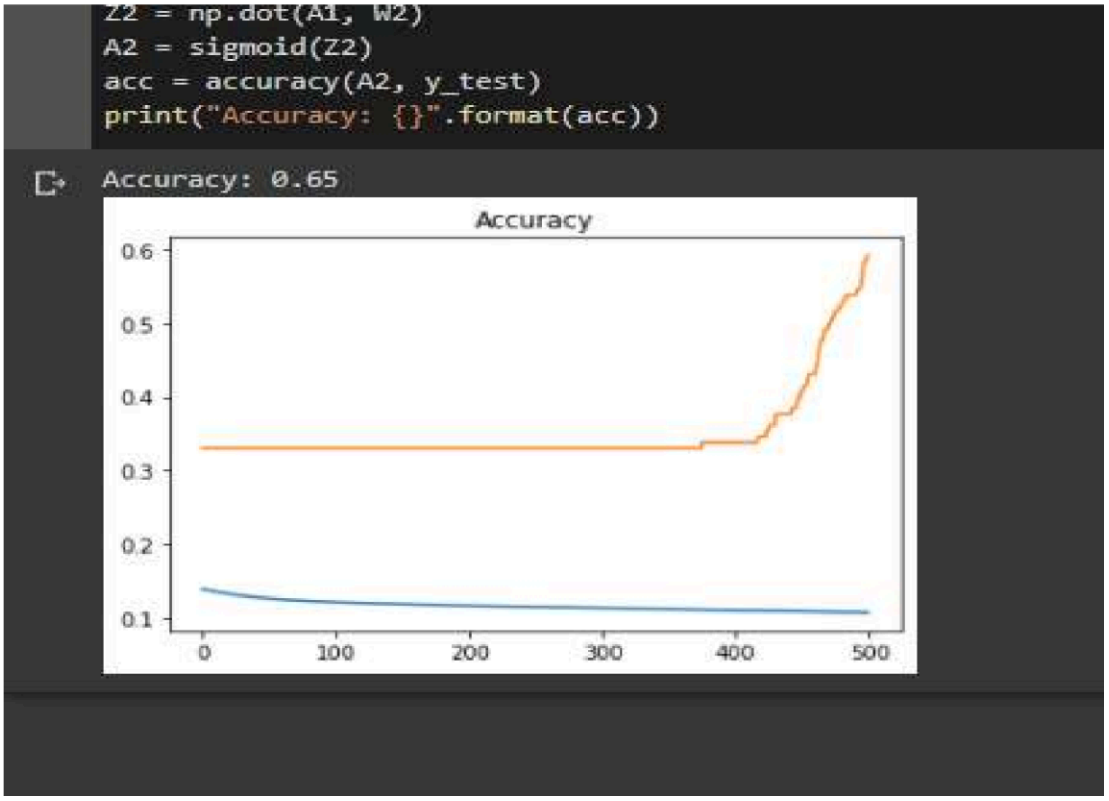
Aim: Implement BPNN Classifier in python.

Code: -

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
data = load_iris()
X=data.data
y=data.target
y = pd.get_dummies(y).values
y[:3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
learning_rate = 0.1
iterations = 500
N = y_train.size
input_size = 4
hidden_size = 2
output_size = 3
results = pd.DataFrame(columns= ["mse", "accuracy"])
np.random.seed(10)
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
```

```
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))
def sigmoid(x):
return 1 / (1 + np.exp(-x))
def mean_squared_error(y_pred, y_true):
return ((y_pred - y_true)**2). sum () / (2*y_pred.size)
def accuracy(y_pred, y_true):
acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
return acc.mean()
for itr in range(iterations):
Z1 = np.dot(X_train, W1)
A1 = sigmoid(Z1)
Z2 = np.dot (A1, W2)
A2 = sigmoid(Z2)
mse = mean_squared_error(A2, y_train)
acc = accuracy (A2, y_train)
results=results.append({"mse":mse, "accuracy":acc},ignore_index=True)
E1 = A2 - y_train
dW1 = E1 * A2 * (1 - A2)
E2 = np.dot(dW1, W2.T)
dW2 = E2 * A1 * (1 - A1)
W2_update = np.dot(A1.T, dW1) / N
W1_update = np.dot(X_train.T, dW2) / N
W2 = W2 - learning_rate * W2_update
```

```
W1 = W1 - learning_rate * W1_update
results.mse.plot(title="Mean Squared Error")
results.accuracy.plot(title="Accuracy")
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)
Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```



Practical -8

Aim: Study and Implement various Ensemble method of classifier: Bagging,Boosting and Stacking.

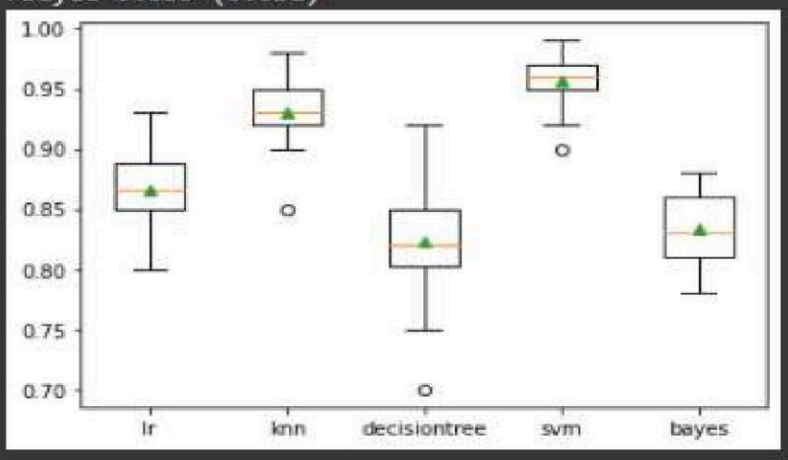
Code: -

```
from numpy import mean

from numpy import std

from sklearn.datasets import make_classification

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import RepeatedStratifiedKFold

from sklearn.ensemble import BaggingClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.naive_bayes import GaussianNB

from matplotlib import pyplot

#Bagging

X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5,

random_state=5)

model = BaggingClassifier()

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
```

```
print('Accuracy: %.3f (%.3f)' % (mean(n_scores),
std(n_scores)))

#Boosting

from sklearn.ensemble import GradientBoostingClassifier

X, y = make_classification(n_samples=1000, n_features=20,
n_informative=15, n_redundant=5,

random_state=7)

model = GradientBoostingClassifier()

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
random_state=1)

n_scores = cross_val_score(model, X, y, scoring='accuracy',
cv=cv, n_jobs=-1)

print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores),
std(n_scores)))

def evaluate_model(model, X, y):

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
random_state=1)

scores = cross_val_score(model, X, y, scoring='accuracy',
cv=cv, n_jobs=-1, error_score='raise')

return scores

#Stacking

X, y = make_classification(n_samples=1000, n_features=20,
n_informative=15, n_redundant=5,

random_state=1)

models = dict()

models['lr'] = LogisticRegression()

models['knn'] = KNeighborsClassifier()
```

```
models['decisiontree'] = DecisionTreeClassifier()
models['svm'] = SVC ()
models['bayes'] = GaussianNB()
results, names = list(), list()
for name, model in models.items():
scores = evaluate_model(model, X, y)
results.append(scores)
names.append(name)
print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

```
pyplot.show()

Accuracy: 0.855 (0.046)
Mean Accuracy: 0.899 (0.030)
>lr 0.866 (0.029)
>knn 0.931 (0.025)
>decisiontree 0.824 (0.045)
>svm 0.957 (0.020)
>bayes 0.833 (0.031)
```

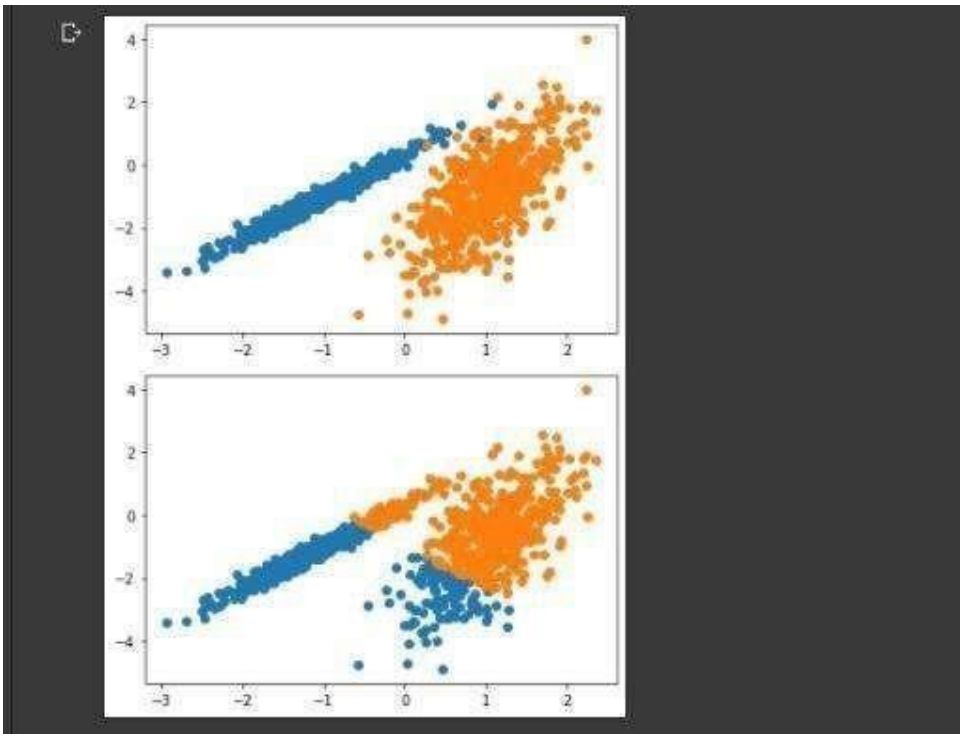


Practical -9

Aim: Implement various Clustering algorithm in python.

Code: -

```
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot
from numpy import unique
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
n_clusters_per_class=1, random_state=4)
for class_value in range(2):
row_ix = where(y == class_value)
pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
#KMeans
from sklearn.cluster import KMeans
model = KMeans(n_clusters=2)
model.fit(X)
ypredicted = model.predict(X)
clusters = unique(ypredicted)
for cluster in clusters:
  row_ix = where(ypredicted == cluster)
  pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()
```


4
2
0
−2
−4
−3
−2
−1
0
1
2
4
2
0
−2
−4
−3
−2
−1
0
1
2