EXPERIMENT NO. 5

AIM: Study and Implement Bagging using Random Forests.

Theory:

1. Bagging (Bootstrap Aggregating):

- Bagging is an ensemble technique that improves the stability and accuracy of machine learning algorithms. It reduces variance by creating multiple subsets of data from the original dataset (with replacement), training models on each subset, and then averaging their predictions.
- The idea behind bagging is to reduce overfitting by averaging out the noise in the individual models, especially in high-variance models like Decision Trees.
- In this code, Bagging is implemented using a **BaggingClassifier** with a **DecisionTreeClassifier** as the base estimator. By training multiple decision trees on random subsets of the data, Bagging provides more stable predictions than a standalone decision tree.

2. Random Forest:

- Random Forest is an extension of bagging in which a collection of decision trees (the "forest") is created, but with an added layer of randomness: each tree is trained on a random subset of features along with a random subset of data.
- Random Forests help to reduce both variance and bias, providing a robust model that is less prone to overfitting.
- In the code, **RandomForestClassifier** is used to demonstrate the effectiveness of Random Forests compared to a standalone decision tree or bagging approach.

Code:

```
# In[1]:
import pandas as pd

# In[2]:
df = pd.read_csv(r"diabetes.csv")
df.head()

# In[3]:
df.columns=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI",
"DiabetesPedigreeFunction","Age","Outcome"]
df.head()

#In[4]:
```

```
df.isnull().sum()

#In[5]:
df.describe()

#In[6]:
df.Outcome.value_counts()
```

Train test split

```
#In[7]:
X = df.drop("Outcome",axis="columns")
y = df.Outcome

#In[8]:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled[:3]

#In[9]:
from sklearn.model_selection import train_test_split
 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,  stratify=y,
random_state=10)

#In[10]:
X_train.shape

#In[11]:
X_test.shape


#In[12]:
y_train.value_counts()

#In[13]:
200/375

#In[14]:
y_test.value_counts()

#In[15]:
67/125
```

Train using stand alone model

```
#In[16]:
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

scores = cross_val_score(DecisionTreeClassifier(), X, y, cv=5)
scores

#In[17]:
scores.mean()
```

Train using Bagging

```
#In[18]:
from sklearn.ensemble import BaggingClassifier

bag_model = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=100,
    max_samples=0.8,
    oob_score=True,
    random_state=0
)
bag_model.fit(X_train, y_train)
bag_model.oob_score_

#In[19]:
bag_model.score(X_test, y_test)

#In[20]:
bag_model = BaggingClassifier(
    estimator=DecisionTreeClassifier(),
    n_estimators=100,
    max_samples=0.8,
    oob_score=True,
    random_state=0
)
scores = cross_val_score(bag_model, X, y, cv=5)
scores

#In[21]:
scores.mean()
```

Train using Random Forest

#In[22]:
```
from sklearn.ensemble import RandomForestClassifier

scores = cross_val_score(RandomForestClassifier(n_estimators=50), X, y, cv=5)
scores.mean()
```

Output Snapshots:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
Outcome
0    500
1    268
Name: count, dtype: int64
```

```
array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
         0.20401277,  0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
        -0.68442195, -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
        -1.10325546,  0.60439732, -0.10558415]])
```

```
(576, 8)  (192, 8)
```

```
Outcome
0    375
1    201
Name: count, dtype: int64
```

```
0.5333333333333333
```

```
Outcome
0    125
1     67
Name: count, dtype: int64
```

```
0.536
```

```
array([0.70779221, 0.67532468, 0.72077922, 0.79084967, 0.74509804])
```

```
0.7279687632628808      0.7534722222222222    0.7760416666666666
```

```
array([0.75324675, 0.72727273, 0.74675325, 0.82352941, 0.73856209])
```

```
0.7578728461081402    0.756599609540786
```

Learning Outcome: