DATA WAREHOUSE AND DATA MINING

LAB DA-4

NAME: HRITHIK HEM SUNDAR.B

REGNO: 19MID0021

CODE:

```
#Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
from sklearn. preprocessing import StandardScaler
from sklearn.model_selection import train_test_spliT
#Reading the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
```

```
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])

labelencoder_X_2 = LabelEncoder()

X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])

from sklearn.compose import ColumnTransformer

t = ColumnTransformer([("Geography", OneHotEncoder(), [1])], remainder = 'passthrough')

X = ct.fit_transform(X)

X = X[:, 1:]

# Splitting the dataset into the Training set and Test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15, random_state = 0, stratify = y)

# Feature Scaling

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

#Defining the machine learning models

model1 = LogisticRegression()

model2 = DecisionTreeClassifier(max_depth = 2)

model3 = SVC()

model4 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)

model5 = GaussianNB()

#Training the machine learning models

model1.fit(X_train, y_train)

model2.fit(X_train, y_train)

model3.fit(X_train, y_train)

model4.fit(X_train, y_train)

model5.fit(X_train, y_train)

#Making the prediction

y_pred1 = model1.predict(X_test)
```

```
y_pred2 = model2.predict(X_test)

y_pred3 = model3.predict(X_test)

y_pred4 = model4.predict(X_test)

y_pred5 = model5.predict(X_test)

#Confusion matrix

cm_LogisticRegression = confusion_matrix(y_test, y_pred1)

cm_DecisionTree = confusion_matrix(y_test, y_pred2)

cm_SupportVectorClass = confusion_matrix(y_test, y_pred3)

cm_KNN = confusion_matrix(y_test, y_pred4)

cm_NaiveBayes = confusion_matrix(y_test, y_pred5)

#10-fold cross-validation

kfold = model_selection.KFold(n_splits=10, random_state = 0)

result1 = model_selection.cross_val_score(model1, X_train, y_train, cv=kfold)

result2 = model_selection.cross_val_score(model2, X_train, y_train, cv=kfold)

result3 = model_selection.cross_val_score(model3, X_train, y_train, cv=kfold)

result4 = model_selection.cross_val_score(model4, X_train, y_train, cv=kfold)

result5 = model_selection.cross_val_score(model5, X_train, y_train, cv=kfold)

#Printing the accuracies achieved in cross-validation

print('Accuracy of Logistic Regression Model = ',result1.mean())

print('Accuracy of Decision Tree Model = ',result2.mean())

print('Accuracy of Support Vector Machine = ',result3.mean())

print('Accuracy of k-NN Model = ',result4.mean())

print('Accuracy of Naive Bayes Model = ',result5.mean())

estimators = []


#Defining 5 Logistic Regression Models

model11 = LogisticRegression(penalty = 'l2', random_state = 0)
```

```
estimators.append(('logistic1', model11))

model12 = LogisticRegression(penalty = 'l2', random_state = 0)

estimators.append(('logistic2', model12))

model13 = LogisticRegression(penalty = 'l2', random_state = 0)

estimators.append(('logistic3', model13))

model14 = LogisticRegression(penalty = 'l2', random_state = 0)

estimators.append(('logistic4', model14))

model15 = LogisticRegression(penalty = 'l2', random_state = 0)

estimators.append(('logistic5', model15))


#Defining 5 Decision Tree Classifiers

model16 = DecisionTreeClassifier(max_depth = 3)

estimators.append(('cart1', model16))

model17 = DecisionTreeClassifier(max_depth = 4)

estimators.append(('cart2', model17))

model18 = DecisionTreeClassifier(max_depth = 5)

estimators.append(('cart3', model18))

model19 = DecisionTreeClassifier(max_depth = 2)

estimators.append(('cart4', model19))

model20 = DecisionTreeClassifier(max_depth = 3)

estimators.append(('cart5', model20))


#Defining 5 Support Vector Classifiers

model21 = SVC(kernel = 'linear')

estimators.append(('svm1', model21))

model22 = SVC(kernel = 'poly')

estimators.append(('svm2', model22))
```

```
model23 = SVC(kernel = 'rbf')
estimators.append(('svm3', model23))
model24 = SVC(kernel = 'rbf')
estimators.append(('svm4', model24))
model25 = SVC(kernel = 'linear')
estimators.append(('svm5', model25))

#Defining 5 K-NN classifiers
model26 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
estimators.append(('knn1', model26))
model27 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
estimators.append(('knn2', model27))
model28 = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p = 2)
estimators.append(('knn3', model28))
model29 = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p = 1)
estimators.append(('knn4', model29))
model30 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 1)
estimators.append(('knn5', model30))

#Defining 5 Naive Bayes classifiers
model31 = GaussianNB()
estimators.append(('nbs1', model31))
model32 = GaussianNB()
estimators.append(('nbs2', model32))
model33 = GaussianNB()
estimators.append(('nbs3', model33))
model34 = GaussianNB()
```

```
estimators.append(('nbs4', model34))

model35 = GaussianNB()

estimators.append(('nbs5', model35))


# Defining the ensemble model

ensemble = VotingClassifier(estimators)

ensemble.fit(X_train, y_train)

y_pred = ensemble.predict(X_test)


#Confusion matrix

cm_HybridEnsembler = confusion_matrix(y_test, y_pred)


#Cross-Validation

seed = 7

kfold = model_selection.KFold(n_splits=10, random_state=seed)

results = model_selection.cross_val_score(ensemble, X_train, y_train, cv=kfold)

print(results.mean())
```

SCREEN SHOT WITH DESCRIPTION AND OUTPUT:

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.svm import SVC
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.naive_bayes import GaussianNB
        from sklearn.ensemble import VotingClassifier
        from sklearn import model_selection
        from sklearn.metrics import confusion_matrix
        from sklearn. preprocessing import StandardScaler
        from sklearn.model_selection import train_test_split
```

In the above cell the libraries like numpy and pandas were imported which is used to deal with datasets. And classifiers like

- Decision tree classifier which classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.
- Logistic regression which is used to model the probability of a certain class or event existing.
- Support vector classifier which maximizes a soft margin
- KN Classifier is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution.
- GaussianNB Can perform online updates to model parameters via partial_fit
- Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.
- Model selection is the process of selecting one final machine learning modelfrom among a collection of candidate machine learning models for a training dataset.
- Confusion matrix is imported in order to display the confusion matrix for the model
- Standard scalar is used to scale down the values.
- Train test split is used to split the whole data set into 2 parts.

```
In [2]: #Reading the dataset
        dataset = pd.read_csv('D:\datasets\Churn_Modelling.csv')
        X = dataset.iloc[:, 3:13].values
        y = dataset.iloc[:, 13].values
```

Dataset is imported and it was splitted for train test split X Y

```
In [4]: # Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
from sklearn.compose import ColumnTransformer
t = ColumnTransformer([("Geography", OneHotEncoder(), [1])], remainder = 'passthrough')
X = t.fit_transform(X)
X = X[:, 1:]
```

Label encoder is used to label the categorical variables in the feature to numerical values in order to fit in the model.

One hot encoder is a encoding technique, It creates the dummy variables for each category in the particular column. It deals with binary numbers 0 and 1

Column Transformer is a sciket-learn class used to create and apply separate transformers for numerical and categorical data

```
In [16]: # Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0, stratify = y)
```

Train test split is implemented

```
In [17]: # Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Fit_transform scales down the values with standard scalar which means the values were scaled down under the criteria of mean=0 and standard deviation=1

```
In [18]: #Defining the machine learning models
model1 = LogisticRegression()
model2 = DecisionTreeClassifier(max_depth = 2)
model3 = SVC()
model4 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
model5 = GaussianNB()
```

Object has been created for each models in order to train the model

```
In [19]: #Training the machine learning models
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)
model3.fit(X_train, y_train)
model4.fit(X_train, y_train)
model5.fit(X_train, y_train)

Out[19]: GaussianNB()
```

The fit() module applies the operations of each models to the respective objects.x train and y train were passed as arguments.

```
In [20]:
#Making the prediction
y_pred1 = model1.predict(X_test)
y_pred2 = model2.predict(X_test)
y_pred3 = model3.predict(X_test)
y_pred4 = model4.predict(X_test)
y_pred5 = model5.predict(X_test)
```

After fit() module trained the data the prediction has to be done

Prediction is the final step and our expected outcome of the **model** generation.

```
In [21]: #Confusion matrix
cm_LogisticRegression = confusion_matrix(y_test, y_pred1)
cm_DecisionTree = confusion_matrix(y_test, y_pred2)
cm_SupportVectorClass = confusion_matrix(y_test, y_pred3)
cm_KNN = confusion_matrix(y_test, y_pred4)
cm_NaiveBayes = confusion_matrix(y_test, y_pred5)
```

```
CONFUSION MATRIX FOR LOGISTIC REGRESSION
[[1914   77]
 [ 401  108]]
CONFUSION MATRIX FOR DECISION TREE CLASSIFIER
[[1849  142]
 [ 300  209]]
CONFUSION MATRIX FOR SUPPORT VECTOR CLASSIFIER
[[1951   40]
 [ 330  179]]
CONFUSION MATRIX FOR KNN
[[1873  118]
 [ 320  189]]
CONFUSION MATRIX FOR NAIVE BAYES CLASSIFIER
[[1888  103]
 [ 336  173]]
```

Confusion matrix has been generated for each models

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

```
In [22]: #10-fold cross-validation
kfold = model_selection.KFold(n_splits=10, random_state = 0)
result1 = model_selection.cross_val_score(model1, X_train, y_train, cv=kfold)
result2 = model_selection.cross_val_score(model2, X_train, y_train, cv=kfold)
result3 = model_selection.cross_val_score(model3, X_train, y_train, cv=kfold)
result4 = model_selection.cross_val_score(model4, X_train, y_train, cv=kfold)
result5 = model_selection.cross_val_score(model5, X_train, y_train, cv=kfold)
```

k-Fold Cross-Validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

```
In [23]: #Printing the accuracies achieved in cross-validation
print('Accuracy of Logistic Regression Model = ',result1.mean())
print('Accuracy of Decision Tree Model = ',result2.mean())
print('Accuracy of Support Vector Machine = ',result3.mean())
print('Accuracy of k-NN Model = ',result4.mean())
print('Accuracy of Naive Bayes Model = ',result5.mean())

Accuracy of Logistic Regression Model =  0.8094666666666667
Accuracy of Decision Tree Model =  0.8314666666666666
Accuracy of Support Vector Machine =  0.8577333333333333
Accuracy of k-NN Model =  0.8318666666666668
Accuracy of Naive Bayes Model =  0.8240000000000001

In [ ]:
```

Accuracy is displayed for performance of all models

```
In [1]: estimators = []
```

```
In [14]: #Defining 5 Logistic Regression Models
model11 = LogisticRegression(penalty = 'l2', random_state = 0)
estimators.append(('logistic1', model11))
model12 = LogisticRegression(penalty = 'l2', random_state = 0)
estimators.append(('logistic2', model12))
model13 = LogisticRegression(penalty = 'l2', random_state = 0)
estimators.append(('logistic3', model13))
model14 = LogisticRegression(penalty = 'l2', random_state = 0)
estimators.append(('logistic4', model14))
model15 = LogisticRegression(penalty = 'l2', random_state = 0)
estimators.append(('logistic5', model15))
```

In [15]:
```
#Defining 5 Decision Tree Classifiers
model16 = DecisionTreeClassifier(max_depth = 3)
estimators.append(('cart1', model16))
model17 = DecisionTreeClassifier(max_depth = 4)
estimators.append(('cart2', model17))
model18 = DecisionTreeClassifier(max_depth = 5)
estimators.append(('cart3', model18))
model19 = DecisionTreeClassifier(max_depth = 2)
estimators.append(('cart4', model19))
model20 = DecisionTreeClassifier(max_depth = 3)
estimators.append(('cart5', model20))
```

In [16]:
```
#Defining 5 Support Vector Classifiers
model21 = SVC(kernel = 'linear')
estimators.append(('svm1', model21))
model22 = SVC(kernel = 'poly')
estimators.append(('svm2', model22))
model23 = SVC(kernel = 'rbf')
estimators.append(('svm3', model23))
model24 = SVC(kernel = 'rbf')
estimators.append(('svm4', model24))
model25 = SVC(kernel = 'linear')
estimators.append(('svm5', model25))
```

In [17]:
```
#Defining 5 K-NN classifiers
model26 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
estimators.append(('knn1', model26))
model27 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
estimators.append(('knn2', model27))
model28 = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p = 2)
estimators.append(('knn3', model28))
model29 = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p = 1)
estimators.append(('knn4', model29))
model30 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 1)
estimators.append(('knn5', model30))
```

In [18]:
```
#Defining 5 Naive Bayes classifiers
model31 = GaussianNB()
estimators.append(('nbs1', model31))
model32 = GaussianNB()
estimators.append(('nbs2', model32))
model33 = GaussianNB()
estimators.append(('nbs3', model33))
model34 = GaussianNB()
estimators.append(('nbs4', model34))
model35 = GaussianNB()
estimators.append(('nbs5', model35))
```

In [19]:
```
# Defining the ensemble model
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train, y_train)
y_pred = ensemble.predict(X_test)
```

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output. It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting.

```
In [24]: #Confusion matrix
         cm_HybridEnsembler = confusion_matrix(y_test, y_pred)
         cm_HybridEnsembler

Out[24]: array([[1954,   37],
                [ 343,  166]], dtype=int64)
```

Hybrid ensemble combines two different ensemble models to enhance the prediction/ generalization capability of the ensemble model.

```
In [23]: seed=7
         kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
         results = model_selection.cross_val_score(ensemble, X_train, y_train, cv=kfold)
         print(results.mean())

         0.8422666666666666
```