Sourcecode_Inc_Qual

June 3, 2023

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[ ]:
```
traindata=pd.read_csv('train.csv')
testdata=pd.read_csv('test.csv')
```

[ ]:
```
traindata.head()
```

[ ]:
```
traindata.info()
```

[ ]:
```
# understanding the dataset
```

[ ]:
```
traindata.dtypes
```

[ ]:
```
traindata.shape
```

[ ]:
```
testdata.shape
```

[ ]:
```
## Identifying Output variable as Target
```

[ ]:
```
traindata.dtypes.value_counts()
```

[ ]:
```
traindata.Target.value_counts()
```

[ ]:
```
# Check if there are any biases in your dataset.
```

[ ]:
```
numerical_features = traindata.select_dtypes(include=['int64', 'float64']).
↪columns.tolist()
```

[ ]:
```
categorical_features = traindata.select_dtypes(include=['object']).columns.
↪tolist()
```

[ ]:
```
traindata.drop(['Id'],axis=1,inplace=True)
```

[ ]:
```
traindata.drop(['idhogar'],axis=1,inplace=True)
```

[ ]:
```
def map(i):

    if i=='yes':
        return(float(1))
    elif i=='no':
        return(float(0))
    else:
        return(float(i))
```

[ ]:
```
traindata['dependency']=traindata['dependency'].apply(map)
```

[ ]:
```
traindata['edjefe']=traindata['edjefe'].apply(map)
```

[ ]:
```
traindata['edjefa']=traindata['edjefa'].apply(map)
```

[ ]:
```
traindata.drop(['elimbasu5'],axis=1,inplace=True)
```

[ ]:
```
# Check if there is a house without a family head.
```

[ ]:
```
traindata.parentesco1.value_counts()
```

[ ]:
```
houses_without_head = traindata[traindata['parentesco1'] != 1].nunique()
print("Households without a head:", houses_without_head)
```

[ ]:
```
traindata.isna().sum()
```

[ ]:
```
traindata['v2a1'].fillna(0,inplace=True)
traindata['v18q1'].fillna(0,inplace=True)
```

[ ]:
```
traindata.drop(['tipovivi3', 'v18q','rez_esc'],axis=1,inplace=True)
```

[ ]:
```
duplicate = traindata[traindata.duplicated()]
print("Duplicate Rows :")
duplicate
```

[ ]:
```
traindata= traindata.drop_duplicates()
```

[ ]:
```
traindata['meaneduc'].fillna(np.mean(traindata['meaneduc']),inplace=True)
traindata['SQBmeaned'].fillna(np.mean(traindata['SQBmeaned']),inplace=True)
```

[ ]:
```
# Count how many null values are existing in columns.
```

[ ]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', -1)
```

```
traindata.isna().sum()
```

[ ]:
```
# Remove null value rows of the target variable.
```

[ ]:
```
traindata.Target.isna().sum()
```

[ ]:
```
sns.distplot(traindata.Target)
```

[ ]:
```
# Set poverty level of the members and the head of the house within a family.
```

[ ]:
```
Poverty_level=traindata[traindata['v2a1'] !=0]
```

[ ]:
```
poverty_level=Poverty_level.groupby('area1')['v2a1'].apply(np.median)
```

[ ]:
```
poverty_level
```

[ ]:
```
#Seperate data as features and label
features = traindata.iloc[:,:-1].values
label = traindata.iloc[:,-1].values
```

[ ]:
```
#Create train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.
↪2,random_state=4)
```

[ ]:
```
from sklearn.tree import DecisionTreeClassifier
modelTree = DecisionTreeClassifier(max_depth=5)
modelTree.fit(X_train,y_train)
```

[ ]:
```
print(modelTree.score(X_train,y_train))
print(modelTree.score(X_test,y_test))
```

[ ]:
```
from sklearn.ensemble import BaggingClassifier   #Bagging + KNN
from sklearn.neighbors import KNeighborsClassifier

algorithm = KNeighborsClassifier()

modelknn = BaggingClassifier(n_estimators=11, #No of weak learners
                         base_estimator=algorithm) #The algo to be used for␣
↪learning

modelknn.fit(X_train,y_train)
```

[ ]:
```
print(modelknn.score(X_train,y_train))
print(modelknn.score(X_test,y_test))
```

[ ]:
```
#Predict the accuracy using random forest classifier.
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier  #RandomForestClassifier
modelrfc = RandomForestClassifier(n_estimators=11)
modelrfc.fit(X_train,y_train)
```

[ ]:
```
modelrfc.score(X_train,y_train)
```

[ ]:
```
modelrfc.score(X_test,y_test)
```

[ ]:
```
from sklearn.metrics import accuracy_score

y_pred = modelrfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

[ ]:
```
# Accuracy, Precision, f1 score for RFC
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(accuracy_score(y_test,y_pred ))
print(classification_report(y_test,y_pred))
```

[ ]:
```
# Create Confusion Matrix

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

[ ]:
```
sampleSize = int(round(np.sqrt(len(X_train))))
sampleSize
```

[ ]:
```
modelrandom_forestwithoutReplacement = RandomForestClassifier(n_estimators=11,␣
↪max_samples=None, bootstrap=False, random_state=7)
```

[ ]:
```
modelrandom_forestwithoutReplacement.fit(X_train,y_train)
```

[ ]:
```
print(modelrandom_forestwithoutReplacement.score(X_train,y_train))
print(modelrandom_forestwithoutReplacement.score(X_test,y_test))
```

[ ]:
```
from sklearn.metrics import accuracy_score

y_pred = modelrandom_forestwithoutReplacement.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

[ ]:
```
modelrandom_forestwithReplacement = RandomForestClassifier(n_estimators=11,␣
↪max_samples=0.9, bootstrap=True, random_state=25)
```

[ ]:
```
modelrandom_forestwithReplacement.fit(X_train,y_train)
```

[ ]:
```
print(modelrandom_forestwithReplacement.score(X_train,y_train))
print(modelrandom_forestwithReplacement.score(X_test,y_test))
```

[ ]:
```
from sklearn.metrics import accuracy_score

y_pred = modelrandom_forestwithReplacement.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

[ ]:
```
# Check the accuracy using random forest with cross validation.
```

[ ]:
```
# Demonstrate the score threshold with RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
modelAlgo = RandomForestClassifier()
```

[ ]:
```
from sklearn.model_selection import cross_val_score

# Supress warnings
import warnings
warnings.filterwarnings('ignore')

scores = cross_val_score(modelAlgo,
                         features,
                         label,
                         scoring='accuracy',
                         cv = 10) #5 or 10

scores
```

[ ]:
```
print("Minimum Score Threshold is : ",scores.mean())
print("Suggested SL value to commit: ", 1-scores.mean())
```

[ ]:
```
scores.max()
```

[ ]:
```
# to extract the best training sample that gives the best score for␣
 ↪LogisticRegression

# Step1: Initialize the algo
from sklearn.ensemble import RandomForestClassifier
modelAlgo = RandomForestClassifier()

# Initialize K-Fold Cross Validation function

from sklearn.model_selection import KFold
```

```
kfold = KFold(n_splits=10, #Use the same CV values that was applied in␣
 ↪cross_val_score
             shuffle=True,
             random_state = 1) # To ensure the data is not randomized at every␣
 ↪iteration

# initialize for loop to identify which sample gives the best score and which␣
 ↪sample is the best
#.   training sample

counter = 0

for train,test in kfold.split(features):

    #Counter will help you track the sample split
    counter += 1

    #Extract the training set and testing set
    X_train,X_test = features[train],features[test]
    y_train,y_test = label[train] , label[test]

    #Fit the model
    modelAlgo.fit(X_train,y_train)

    if modelAlgo.score(X_test,y_test) >= 1.0:
        print("Test Score {} Train Score {} for Sample Split {}".
 ↪format(modelAlgo.score(X_test,y_test),modelAlgo.
 ↪score(X_train,y_train),counter))
```

[ ]:
```
# Extract the samples
# Initialize the algo
from sklearn.ensemble import RandomForestClassifier
modelAlgo = RandomForestClassifier()

# Initialize K-Fold Cross Validation function

from sklearn.model_selection import KFold

kfold = KFold(n_splits=10, #Use the same CV values that was applied in␣
 ↪cross_val_score
             shuffle=True,
             random_state = 1) # To ensure the data is not randomized at every␣
 ↪iteration

# initialize for loop to identify which sample gives the best score and which␣
 ↪sample is the best
```

```
#.   training sample

counter = 0
for train,test in kfold.split(features):

    #Counter will help you track the sample split
    counter += 1

    if counter == 1:
        X_train,X_test,y_train,y_test =␣
 ↪features[train],features[test],label[train] , label[test]
```

[ ]:
```
kfold.split(features)
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
finalModel = RandomForestClassifier()
finalModel.fit(X_train,y_train)
finalModel.score(X_test,y_test)
```

[ ]:
```
# StraifiedShuffleSplit


# Initialize the algo
from sklearn.ensemble import RandomForestClassifier
modelAlgo = RandomForestClassifier()

# Initialize StratifiedShuffleSplit Cross Validation function

from sklearn.model_selection import StratifiedShuffleSplit

ss = StratifiedShuffleSplit(n_splits=10, #Use the same CV values that was␣
 ↪applied in cross_val_score
            test_size=0.2,
            random_state = 1) # To ensure the data is not randomized at every␣
 ↪iteration

#  initialize for loop to identify which sample gives the best score and which␣
 ↪sample is the best
#.   training sample

counter = 0

for train,test in ss.split(features,label):
```

```
    #Counter will help you track the sample split
    counter += 1

    #Extract the training set and testing set
    X_train,X_test = features[train],features[test]
    y_train,y_test = label[train] , label[test]

    #Fit the model
    modelAlgo.fit(X_train,y_train)

    if modelAlgo.score(X_test,y_test) >= 1.0:
        print("Test Score {} Train Score {} for Sample Split {}".
 ↪format(modelAlgo.score(X_test,y_test),modelAlgo.
 ↪score(X_train,y_train),counter))
```

[ ]:
```
# Extract the samples
#  Initialize the algo
from sklearn.ensemble import RandomForestClassifier
modelAlgo = RandomForestClassifier()

# Initialize K-Fold Cross Validation function

from sklearn.model_selection import StratifiedShuffleSplit

ss = StratifiedShuffleSplit(n_splits=10, #Use the same CV values that was␣
 ↪applied in cross_val_score
            test_size=0.2,
            random_state = 1) # To ensure the data is not randomized at every␣
 ↪iteration

# 3. initialize for loop to identify which sample gives the best score and␣
 ↪which sample is the best
#.   training sample

counter = 0
for train,test in ss.split(features,label):

    #Counter will help you track the sample split
    counter += 1

    if counter == 7:
        X_trainSS,X_testSS,y_trainSS,y_testSS =␣
 ↪features[train],features[test],label[train] , label[test]
```

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
finalModel = RandomForestClassifier()
finalModel.fit(X_trainSS,y_trainSS)
finalModel.score(X_testSS,y_testSS)
```