jupyter Untitled (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3 (ipykernel)

Run Code

In [180]:
```
#RUN THIS CELL
#M
import requests
from IPython.core.display import HTML

# Import libraries
%matplotlib inline
import math
import numpy as np
import pandas as pd
#import seaborn as sns
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.ensemble import RandomForestClassifier
#sns.set()

import warnings
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier

from sklearn import datasets,tree

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#import seaborn as sns

from tqdm import tqdm
import time
from sklearn.model_selection import cross_val_score, train_test_split, GridSearc
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDis
from sklearn.metrics import recall_score, precision_score, classification_report
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.inspection import permutation_importance
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.impute import SimpleImputer
#RUN THIS CELL
import requests
from IPython.core.display import HTML

# Import libraries
%matplotlib inline
import math
import numpy as np
import pandas as pd
#import seaborn as sns
from collections import Counter
```

```python
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
#sns.set()

import warnings
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier

from sklearn import datasets,tree

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#import seaborn as sns

from tqdm import tqdm
import time
from sklearn.model_selection import cross_val_score, train_test_split, GridSearc
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDis
from sklearn.metrics import recall_score, precision_score, classification_report
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.inspection import permutation_importance
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.impute import SimpleImputer

from sklearn.decomposition import TruncatedSVD
from sklearn.datasets import load_svmlight_file
from sklearn.datasets import dump_svmlight_file
from scipy import sparse as sps
from sklearn.preprocessing import MinMaxScaler
from scipy.linalg import circulant
from sklearn.model_selection import train_test_split
from keras.optimizers import *
import random
import math
import pandas as pd
from tensorflow.python.keras.utils.data_utils import Sequence
import warnings
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
warnings.filterwarnings("ignore")

import tensorflow.keras as k
import keras.backend as K
import numpy as np
from keras.layers import *
from keras.models import Sequential, Model
from keras.regularizers import l2
import matplotlib.pyplot as plt
from keras.optimizers import Adam, Adadelta

from keras.callbacks import ModelCheckpoint, EarlyStopping


from tensorflow.python.framework.ops import disable_eager_execution
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

disable_eager_execution()

%matplotlib inline
%matplotlib inline
```

In [181]:
```
import utils
import scipy
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

dictSize = 225
(X_raw, y_raw) = utils.loadData( "train", dictSize = dictSize )
```

In [182]:
```
X1=X_raw.toarray()
y1=y_raw

from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=224)
svd.fit(X1)
X_t = svd.transform(X1)
classes = np.unique(y1)
classes
```

Out[182]:
```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25., 26.,
       27., 28., 29., 30., 31., 32., 34., 35., 37., 39., 40., 41., 42.,
       43., 44., 45., 46., 47., 48., 49., 50.])
```

In [183]:
```
# print (np.unique(y))
# unique, counts = np.unique(y, return_counts=True)
# print (unique)
# print (counts)
```

In [184]:
```
print(X_t.shape)
print(y1.shape)
```

```
(10000, 224)
(10000,)
```

In [185]:
```
X_train, X_test, y_train, y_test=train_test_split(X1,y1,test_size=0.2)

# sm = SMOTE()
# X_train_SMOTE, y_train_SMOTE = sm.fit_resample(X_train, y_train)
```

In [186]:
```
dt=DecisionTreeClassifier(class_weight="balanced")
```

In [187]:
```
dt.fit(X_train,y_train)
```

Out[187]:
```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced')
```

In [188]:
```
y_pred=dt.predict(X_test)
```

In [189]:
```
y_pred
```

Out[189]:
```
array([15., 32., 13., ...,  4.,  1.,  1.])
```

In [190]:
```
acc=accuracy_score(y_test, y_pred )
```

In [191]:
```
acc
```

Out[191]: 0.693

In [192]:
```
# npzModel = np.load( "model.npz" )
# model = npzModel[npzModel.files[0]]
# 	# Let us predict a random subset of the 2k most popular labels no matter wha
# shortList = model[0:2*5]
# (i,j)=np.unique(model,return_index=True)
# print(shortList)
# print(model)
# print(i)
# print (j)
```

In [193]:
```
# a=np.array([4,3])
# b=np.array([[2,3, 6,22,2],[3,9,7,49,5]])
# utils.validateAndCleanup(a,b,5)
```

In [194]:
```
dict={'C':(0.01,0.1,1,10,100), 'solver':('newton-cg', 'lbfgs', 'liblinear', 'sag
```

In [197]:
```
lr1=LogisticRegression()
clf=GridSearchCV(lr1, dict, scoring='accuracy')
clf.fit(X_train, y_train)
```

Out[197]:

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [198]:
```
print( 'Best score: %0.3f' % clf.best_score_)
print( 'Best parameters set:')
best_parameters = clf.best_estimator_.get_params()
for param_name in sorted(dict.keys()):
    print( '\t%s: %r' % (param_name, best_parameters[param_name]) )

predictions = clf.predict(X_test)
print( classification_report(y_test, predictions) )
#print(accuracy_score(y_test, predictions))
```

```
Best score: 0.792
Best parameters set:
	C: 10
	solver: 'newton-cg'
              precision    recall  f1-score   support

         1.0       0.84      0.73      0.78       311
         2.0       0.89      0.96      0.92       532
         3.0       0.86      0.91      0.88       292
         4.0       0.69      0.81      0.74       263
         5.0       0.88      0.62      0.73        24
         6.0       0.67      1.00      0.80         2
         7.0       0.98      0.94      0.96        52
         8.0       0.95      0.82      0.88        51
         9.0       0.88      0.87      0.88        70
        10.0       0.75      0.73      0.74        52
        11.0       0.78      0.85      0.82        34
        12.0       0.00      0.00      0.00        28
        13.0       0.60      0.60      0.60        30
        14.0       0.00      0.00      0.00         2
        15.0       0.55      0.42      0.48        26
        16.0       0.60      0.46      0.52        26
        17.0       0.50      0.50      0.50         6
        18.0       0.00      0.00      0.00         4
        19.0       1.00      0.64      0.78        14
        20.0       0.40      0.40      0.40        10
        21.0       0.50      0.50      0.50         8
        22.0       0.00      0.00      0.00         5
        23.0       0.93      0.82      0.87        17
        24.0       0.79      0.85      0.81        13
        25.0       0.53      0.62      0.57        13
```

```
        26.0       0.67      0.91      0.77        11
        28.0       0.89      1.00      0.94         8
        29.0       0.80      0.80      0.80         5
        30.0       1.00      0.60      0.75        10
        31.0       0.33      0.50      0.40         2
        32.0       1.00      1.00      1.00         9
        34.0       0.67      1.00      0.80         4
        35.0       1.00      1.00      1.00         9
        37.0       0.50      0.17      0.25         6
        39.0       0.17      0.33      0.22         3
        40.0       1.00      0.50      0.67         2
        41.0       0.40      0.67      0.50         3
        42.0       0.67      0.29      0.40         7
        43.0       0.57      1.00      0.73         4
        44.0       0.80      0.50      0.62         8
        45.0       0.80      0.67      0.73         6
        46.0       0.67      0.67      0.67         3
        47.0       0.00      0.00      0.00         3
        48.0       0.25      0.20      0.22         5
        49.0       0.50      0.67      0.57         3
        50.0       0.00      0.00      0.00         4

    accuracy                           0.81      2000
   macro avg       0.61      0.60      0.59      2000
weighted avg       0.80      0.81      0.80      2000
```

In [204]:

```
predic = clf.predict(X_test)
y_pred = clf.best_estimator_.predict(X_test)
predic3 = clf.best_estimator_

print(predic)
print(y_pred)
print(predic3)
y_pred=[2000,5]
```

```
[15. 32. 13. ...  7.  1.  1.]
[15. 32. 13. ...  7.  1.  1.]
LogisticRegression(C=10, solver='newton-cg')
```

In [205]:

```
preck = utils.getPrecAtK( y_test, y_pred, 5 )
# The macro precision code takes a bit longer to execute due to the for loop ove
mpreck = utils.getMPrecAtK( y_test,y_pred, 5 )

# According to our definitions, both prec@k and mprec@k should go up as k goes u
# method, prec@i > prec@j if i > j and mprec@i > mprec@j if i > j. See the assig
# to convince yourself why this must be the case.

print( "prec@1: %0.3f" % preck[0], "prec@3: %0.3f" % preck[2], "prec@5: %0.3f" %
# Dont be surprised if mprec is small -- it is hard to do well on rare error cla
print( "mprec@1: %0.3e" % mpreck[0], "mprec@3: %0.3e" % mpreck[2], "mprec@5: %0.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In [205], line 1
----> 1 preck = utils.getPrecAtK( y_test, y_pred, 5 )
      2 # The macro precision code takes a bit longer to execute due to the for
loop over labels
      3 mpreck = utils.getMPrecAtK( y_test,y_pred, 5 )

File ~\OneDrive - IIT Kanpur\Desktop\Lecture Notes\Intro to ML\Assignments\Assn
2\Madhav\utils.py:62, in getPrecAtK(yGold, yPred, k)
     60 def getPrecAtK( yGold, yPred, k ):
     61         n = len(yGold)
---> 62         (yGoldNew, yPredNew) = validateAndCleanup( yGold, yPred, k )
     64         # Use some fancy indexing (yes, this is the formal term for the
technique)
     65         # to find out where all did we predict the correct error class
     66         # Python indexing with arrays creates copies of data so we are
safe
     67         # The -1 step is required since predicted labels are indexed 1
... 50 whereas Python expects zero_based indices
     68         wins = yGoldNew[ np.arange( n )[: np.newaxis], yPredNew.astype
```

```
(int) - 1 ]

File ~\OneDrive - IIT Kanpur\Desktop\Lecture Notes\Intro to ML\Assignments\Assn 2\Madhav\utils.py:45, in validateAndCleanup(yGold, yPred, k)
     42 n = len(yGold)
     44 # Make sure the prediction matrix is in correct shape
---> 45 assert yPred.shape[0] == n, "Mismatch in number of test data points and number of predictions"
     46 assert yPred.shape[1] == k, "Mismatch in number of predictions received and number expected"
     48 # Penalize duplicates in yPred by replacing them with predictions of the dummy error class 0
     49 # Since error classes are numbered from 1 to 50, the 0 error class is a safe dummy choice

AttributeError: 'list' object has no attribute 'shape'
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```python
from sklearn.decomposition import TruncatedSVD
from sklearn.svm import SVC
from sklearn.cluster import KMeans

#svd = TruncatedSVD(n_components=224)
#svd.fit(X1)
#X_t = svd.transform(X1)
X_train, X_test, y_train, y_test=train_test_split(X_t,y1,test_size=0.2)

#kms  = KMeans()
#kms.fit(X_train)

#y_train = kms.labels_
#print(y_train)
#y_test = kms.predict(X_test)

lr=LogisticRegression(class_weight="balanced", solver='liblinear')
lr.fit(X_train, y_train)
y_pred2=lr.predict(X_test)
acc2=accuracy_score(y_test,y_pred2)
proba=lr.predict_proba(X_test)
ind = np.argsort(proba)[:,-1:-6:-1]

print (acc2)
print (ind.shape)
yPred = lr.classes_[ind]
```

In [ ]:

```python
X_train.shape
```

In [ ]:

```python
y_train[:].shape
```

In [ ]:

```python
preck = utils.getPrecAtK( y_test, yPred, 5 )
# The macro precision code takes a bit longer to execute due to the for loop ove
mpreck = utils.getMPrecAtK( y_test,yPred, 5 )

# According to our definitions, both prec@k and mprec@k should go up as k goes u
# method, prec@i > prec@j if i > j and mprec@i > mprec@j if i > j. See the assig
# to convince yourself why this must be the case.

print( "prec@1: %0.3f" % preck[0], "prec@3: %0.3f" % preck[2], "prec@5: %0.3f" %
# Dont be surprised if mprec is small -- it is hard to do well on rare error cla
print( "mprec@1: %0.3e" % mpreck[0], "mprec@3: %0.3e" % mpreck[2], "mprec@5: %0.
```

In [ ]:
```
#plt.plot('proba', label = 'accuracy')
#plt.grid(True)
#plt.legend()
```

In [ ]:
```
print(accuracy_score(y_test, y_pred))
```

In [ ]:
```
```