HW 4

Name :Amruta Dhondage

SJSU ID: 011416210

Rank:15

F1 Score : 0.6875

Goal :

To develop predictive models that can determine, given a particular compound, **whether it is active (1) or not (0).**

Steps followed:

1> Read the train data, test data and convert data into list. The training dataset consists of 800 records and the test dataset consists of 350 records.

2> Process the data for cleaning punctuations or extra spaces

3> As it's a classification problem, extract class values from train data and keep into cls list which are 1 and 0.

4> Perform feature selection.
Feature selection here is performed based on set. Unique identifiers are collected from train dataset and applied as features.

5> Vectorize read train and test data using having l2 norm and vocabulary list as features selected at step 4. This creates the sparse matrixes.

6> Apply classification methods on vectorized train data from step 5 and cls list from step3.
3 different classification methods are applied as explained below and prediction is calculatved.
Best of 3 is chosen as final selection.

7> Using the predictive model generated , test the test data for results.

Predict class for Y,where y is the test dataset.

8> Write the result into format file which is in the form of 0 and 1.

Classification Approached Used:

1> Using MLPClassifier:
It’s a Multi-layer Perceptron classifier. This model optimizes the log-loss function using LBFGS or stochastic gradient descent.
Parameters :
hidden_layer_sizes : tuple, length = n_layers - 2, default (100,)
The ith element represents the number of neurons in the ith hidden layer.
alpha : float, optional, default 0.0001
L2 penalty (regularization term) parameter
solver : {‘lbfgs’, ‘sgd’, ‘adam’}, default ‘adam’
The solver for weight optimization.
‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
‘sgd’ refers to stochastic gradient descent.
‘adam’ refers to a stochastic gradient-based optimizer
X: train dataset converted to sparse matrix
Y: class labels for train dataset

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(50, 20), random_state=1)
nn = clf.fit(X,y)
```

2> Using ExtraTreesClassifier(final execution):
This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.
X: train dataset converted to sparse matrix
Y: class labels for train dataset

```
clf = ExtraTreesClassifier()
clf = clf.fit(X, y)
```

3> Using SVM ,LinearSVC:
LinearSVC take as input two arrays: an array X of size [n_samples, n_features] holding the training samples, and an array y of class labels (strings or integers), size [n_samples]
X: train dataset converted to sparse matrix
Y: class labels for train dataset

```
clf = svm.LinearSVC()
clf.fit(X, y)
```

Advantages of SVMs: High accuracy, nice theoretical guarantees regarding overfitting, and with an appropriate kernel they can work well even if you're data isn't linearly separable in the base feature space.