Final Project

Armin Soltan

1 Preface

In this project i use QSAR oral toxicity Data Set. In data mining we have three steps:

- preparing and pre processing data
- learning model
- evaluation

Preparing data is ready for us because use ready data set and for first step just we should implement pre processing data, for learning model we use many classifier to get best result and compare with other model and for evaluation use some tools that all of tools are exist in sklearn package.

2 Processing Data

In first step on processing data we should do pre processing, normalization and principle analysis so i do below steps:

- Since our data is big if we use python matrix our speed of processing data become very low so i use DataFrame in pandas library. Using this DataFrame cause we don't use extra for and while loop in program and this library optimize some method for instance access to the data is faster than using python builtin list. But a problem exist in the data set and i tried this library but some errors happened. So i read csv file by builtin python function and change label to zero or one to convert label from string to number.
- In this data set we have 1024 feature and 8992 sample so we have lots of feature so we should reduce dimensionality of this data set space i use PCA that exist in sklearn package and reduce dimension to 512.

Since we have one data set so seventy percents of data set belongs to training data set and thirty percent belongs to test data. Then we should split label from other data, last column belongs to labels.

3 Learning Model

Now for learning we should use classifying and clustering algorithms

Gradient boosting

Gradient boosting builds an additive model in a forward stage-wise fashion it allows for the optimization of arbitrary differentiable loss functions

```
**********gradient_boosting**********
##########accuracy##########
0.9295774647887324
##########confusion##########
[[2453   17]
 [ 173   55]]
##########precision##########
0.7638888888888888
##########recall##########
0.2412280701754386
##########f_measure##########
0.3666666666666667
```

```
def gradient_boosting(x_train, y_train, x_test, y_test):
    clf = GradientBoostingClassifier(random_state=0)
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

radius neighbor

Classifier implementing a vote among neighbors within a given radius. You can see from below information this classifier is not appropriate for this data set because set same label for all data set.

```
**********radius_neighbor**********
##########accuracy##########
0.9154929577464789
##########confusion##########
[[2470    0]
 [ 228    0]]
##########precision##########
0.0
##########recall##########
0.0
##########f_measure##########
0.0
```

```
def radius_neighbor(x_train, y_train, x_test, y_test):
    clf = RadiusNeighborsClassifier(radius=200)
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

random forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

```
**********random_forest**********
##########accuracy##########
0.927353595255745
##########confusion##########
[[2466    4]
 [ 192   36]]
##########precision##########
0.9
##########recall##########
0.15789473684210525
##########f_measure##########
0.26865671641791045
```

```
def random_forest(x_train, y_train, x_test, y_test):
    clf = RandomForestClassifier(max_depth=300)
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

ada boosting

AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

```
************************ada_boost*******************
############################accuracy#########################
0.91882876204596
############################confusion#########################
[[2426   44]
 [ 175   53]]
############################precision#########################
0.5463917525773195
############################recall#########################
0.2324561403508772
############################f_measure#########################
0.32615384615384607
```

```
def ada_boost(x_train, y_train, x_test, y_test):
    clf = AdaBoostClassifier(n_estimators=100, random_state=0)
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

KNN

KNeighbors classifiers implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user.

```
************************knn*******************
############################accuracy#########################
0.9347664936990363
############################confusion#########################
[[2412   58]
 [ 118  110]]
############################precision#########################
0.6547619047619048
############################recall#########################
0.4824561403508772
############################f_measure#########################
0.5555555555555556
```

```
def knn(x_train, y_train, x_test, y_test):
    clf = KNeighborsClassifier(n_neighbors=3)
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

MLP

Multi layer perceptron classifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. In this classifier is put two layer perceptron, first layer size is 40 and second layer is 10 neuron.

```
************************mlp*******************
############################accuracy#########################
0.9373610081541883
############################confusion########################
[[2412   58]
 [ 111  117]]
############################precision########################
0.6685714285714286
############################recall###########################
0.5131578947368421
############################f_measure########################
0.5806451612903226
```

```
def mlp(x_train, y_train, x_test, y_test):
    clf = MLPClassifier(random_state=1, max_iter=300, hidden_layer_sizes=(40, 10))
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

suport vector machine

```
************************svm*******************
############################accuracy#########################
0.9318013343217197
############################confusion########################
[[2446   24]
 [ 160   68]]
############################precision########################
0.7391304347826086
############################recall###########################
0.2982456140350877
############################f_measure########################
0.425
```

```
def svm(x_train, y_train, x_test, y_test):
    clf = svm_clf.SVC(gamma='scale')
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

logistic regression

This class implements regularized logistic regression using the liblinear library.

```
************************logistic_regression********************
############################accuracy#########################
0.927353595255745
############################confusion########################
[[2415   55]
 [ 141   87]]
############################precision########################
0.6126760563380281
############################recall###########################
0.3815789473684211
############################f_measure########################
0.47027027027027024
```

```
def logistic_regression(x_train, y_train, x_test, y_test):
    clf = LogisticRegression(random_state=0)
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

naive bayes

```
************************naive_bayes********************
############################accuracy#########################
0.9277242401779096
############################confusion########################
[[2393   77]
 [ 118  110]]
############################precision########################
0.5882352941176471
############################recall###########################
0.4824561403508772
############################f_measure########################
0.5301204819277109
```

```
def naive_bayes(x_train, y_train, x_test, y_test):
    clf = GaussianNB()
    return compute_validation(x_train, y_train, x_test, y_test, clf)
```

K mean clustering

The KMeans algorithm clusters data by trying to sseparate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum of squares .

```
************************k_mean_clustering*******************
############################accuracy########################
0.6067457375833951
############################confusion#######################
[[1542  928]
 [ 133   95]]
############################precision#######################
0.09286412512218964
############################recall##########################
0.4166666666666667
############################f_measure#######################
0.15187849720223823
```

```
def k_mean_clustering(x_train, y_train, x_test, y_test):
    k_means = KMeans(n_clusters=2, random_state=0)
    k_means.fit(x_train, y_train)
    y_predict = k_means.predict(x_test)
    return _compute_score(y_test, y_predict)
```

4 Evaluation

10-cross validation

In this section I evaluate classifiers by k-cross validation in sklearn package and get below result. First we should specify the number of splits so in the final project assignment is written 10 so i put 10 split and then create train data set and test data set and then compute accuracy for each classifier and sum accuracy for each data set and at the end i divide accuracy by number of split that's value is 10. You can see mlp and svm classifier have better result from the other classifier.

```
.............accuracy  gradient_boosting classifier .............
0.928824001977506
.............accuracy  radius_neighbor classifier .............
0.9175902855024101
.............accuracy  random_forest classifier .............
0.9274891855147696
.............accuracy  ada_boost classifier .............
0.919593622543567
.............accuracy  knn classifier .............
0.9307149919663825
.............accuracy  mlp classifier .............
0.9361646273637373
.............accuracy  svm classifier .............
0.9366098133728833
.............accuracy  logistic_regression classifier .............
0.9239307872945247
```

```python
def k_fold(x_data, y_data):
    classifiers = [
        gradient_boosting,
        radius_neighbor,
        random_forest,
        ada_boost,
        knn,
        mlp,
        svm,
        logistic_regression
    ]
    accuracy_score = dict()
    for clf in classifiers:
        accuracy_score[clf.__name__] = 0
    kf = KFold(n_splits=10)
    for train_index, test_index in kf.split(x_data):
        x_train, y_train, x_test, y_test = list(), list(), list(), list()
        for i in train_index:
            x_train.append(x_data[i])
            y_train.append(y_data[i])
        for j in test_index:
            x_test.append(x_data[j])
            y_test.append(y_data[j])
        for clf in classifiers:
            accuracy_score[clf.__name__] += clf(x_train, y_train, x_test, y_test)['accuracy']
    for clf in classifiers:
        print("..............accuracy  {} classifier .............".format(clf.__name__))
        print(accuracy_score[clf.__name__] / 10)
```

5 Code repository

You can see my code in my github repository and check process of writing project
`https://github.com/arminsoltan/data-mining`