Activity prediction for chemical compounds

ID2214 Group 9. 2019.12. 10

Yuxia Wang , Hansika Attanayake, Sevket Melih Zenciroglu

This project aims to develop a model to predict the activity for chemical compounds. By using open source toolkit RDkit [1], we can get the features from the given dataset. Our work is mainly focused on four parts, as shown in Figure 1.

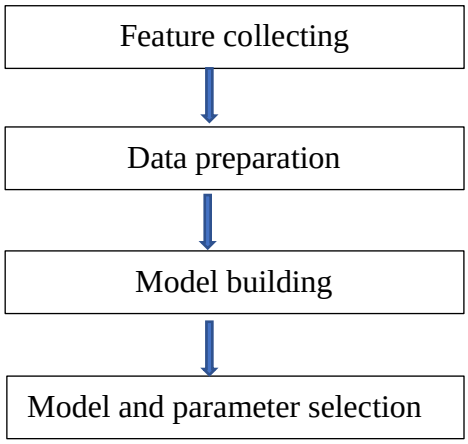

Figure 1. Workflow for this project

1. Feature collecting

The features as below are got as recommended, since it is assumed to achieve a fair score from these features, we do not explore the others now. We drop the column of HeavyAtomCount finally since it has the same values as NoAtoms.

(1) NoAtoms, CalcExactMolWt , fr_Al_COO and and HeavyAtomCount
(2) Fingerprints vector

	INDEX	SMILES	ACTIVE
0	1	Cl.O=C1C(CN(CC2:C:C:C:C:C:2)CC2:C:C:C:C:C:2)CC...	0
1	2	CCOC(=O)C1OC2:C:C:C([N+](=O)[O-]):C:C:2C1(C)O	0
2	3	COC1:C:C:C:C2:C:1OC1(N3CCOCC3)CCCCC1C2C[N+](=O...	0
3	4	CC1:C:C:C(S(=O)(=O)NC(=O)NCCSSCCNC(=O)NS(=O)(=...	0
4	5	COC1:C:C:C:C:C:1NC(=S)NC=C1C(=O)NC(=O)NC1=O	0

	NoAtoms	CalcExactMolWt	fr_Al_COO	MFbitV_0	MFbitV_1	MFbitV_2	MFbitV_3	MFbitV_4	MFbitV_5	MFbitV_6	...
0	40	552.290742	0	0	0	0	1	0	1	0	...
1	19	267.074287	0	0	0	0	0	0	1	0	...
2	26	362.184172	0	1	0	1	0	0	0	0	...
3	34	546.073519	0	0	0	0	1	0	1	1	...
4	22	320.057926	0	1	0	1	0	0	0	0	...

5 rows × 127 columns

2. Data preparation

Based on the open source of Scikit-learn [2], the imputation, normalization and discretization are used for NoAtoms, CalcExactMolWt. The preprocessing is done as follows.

- imp = SimpleImputer(missing_values=np.nan, strategy='mean')
- scaler = preprocessing.MinMaxScaler()
- kbd = KBinsDiscretizer(n_bins=10, encode="ordinal")

Note that data preparation needs to be done after the data split, so that the preprocessing can be fitted by the training data and apply to both the training data and the test data.

The original training dataset is divided into training and test data by 80% and 20%.

```
from sklearn import model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    df1, target, test_size=0.2, random_state=1)
X_train.head()
```

	NoAtoms	CalcExactMolWt	fr_Al_COO	MFbitV_0	MFbitV_1	MFbitV_2	MFbitV_3	MFbitV_4	MFbitV_5	MFbitV_6	...
17557	34	571.893145	0	1	0	0	1	0	0	0	...
15052	31	419.170771	0	1	0	1	0	0	1	0	...
1049	22	308.173607	0	0	0	0	0	0	1	0	...
13083	21	299.071306	0	0	0	1	0	0	1	0	...
18549	24	332.137222	0	0	0	0	0	0	1	0	...

5 rows × 127 columns

	NoAtoms	CalcExactMolWt	fr_Al_COO	MFbitV_0	MFbitV_1	MFbitV_2	MFbitV_3	MFbitV_4	MFbitV_5	MFbitV_6	...	I
14126	2.0	1.0	0	0	0	0	1	0	0	0	...	
11353	5.0	5.0	0	0	0	1	0	1	1	0	...	
18075	9.0	9.0	0	0	0	0	0	0	1	1	...	
8765	2.0	2.0	0	0	0	0	0	0	1	0	...	
18213	9.0	9.0	0	1	0	1	0	0	0	0	...	

5 rows × 127 columns

3. Model building

This task aims to predict the activity for chemical compounds. And from the dataset we can see the labels are [0, 1]. That means we need to do a binary classification. Considering the methods in Scikit-learn, we try four methods as below, and we do not set the parameters first, just see which models are better for this task.

BernoulliNB	DecisionTree	RandomForest	MLPClassifier(NN)

Based on the given features, with normalization of the features with number, two situations are tried:

(1) Build models based on the features as follows:

NoAtoms, CalcExactMolWt , fr_Al_COO and and HeavyAtomCount

After PCA (95%), we get two components.

(2) Build models based on the features in (1) and the fingerprints vector

After adding the feature of the fingerprints vector, PCA can be tried in this way, it does not work well, so we do not apply it later.

```
from sklearn.decomposition import PCA
pca = PCA(0.95)
pca.fit(X_train)
X_train_PCA = pca.transform(X_train)
X_test_PCA = pca.transform(X_test)
pca.singular_values_.shape
```

```
(105,)
```

The initial results are shown in table 1. We can see that the models of BernoulliNB and RandomForest give better results.

Table 1. The AUC scores for different models

	BernoulliNB	DecisionTree	RandomForest	MLPClassifier(NN)
(1) Without Finger print vector	0.5934	0.5806	0.6432	0.6103
(2) With Finger print vector, without PCA	0.6821	0.5865	0.6948	0.5007

4. Models and parameters selection

Based on the results above, now all the preprocessing as shown in section 2 is done. We consider three models:

BernoulliNB, BernoulliNB+AdaBoost and RandomForest

But the result of the first one is not good, and AUC score is lower than BernoulliNB, as shown in Table 2. So we focus only on RandomForest.

Table 2. The AUC scores for different models

BernoulliNB	BernoulliNB + AdaBoost
0.6844	0.5273

(1) Cross validation

Firstly, the cross validation is used to see the performance of RandomForest. Here n_splits=5, which means we use a 5-fold cross validation. And the number of estimators is 100.

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
kf = KFold(n_splits=5, random_state=42, shuffle=False)
rf = RandomForestClassifier(n_estimators=100)
cross_val_score(rf, X_train, y_train, cv=kf, scoring='roc_auc')
```

```
array([0.73641913, 0.75147026, 0.73882872, 0.7242867 , 0.77474499])
```

From the results we get an average AUC score of 0.7451, which is acceptable without optimization. Then the parameter selection will be considered.

(2) Parameter selection by GridSearchCV

According to [6], by using the method of GridSearchCV, we can choose the best parameters for RandomForest.

```
from sklearn.model_selection import GridSearchCV
clf = RandomForestClassifier()

params = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15, 20, 30],
    'random_state': [0]
}

grid_naive_up = GridSearchCV(clf, param_grid=params, cv=kf,
                             scoring='roc_auc').fit(X_train,
                                                    y_train)
grid_naive_up.best_score_
```

```
0.7542659292324573
```

The model with best parameters will be got by the variable: grid_naive_up.best_estimator_

```
grid_naive_up.best_params_
```

```
{'max_depth': 20, 'n_estimators': 200, 'random_state': 0}
```

At the same time, we can get the AUC scores for training and test data: [0.7543, 0.7659], which are similar.

(3) Processing on imbalanced data

Here there is another factor which may influence the result, that is the imbalance of the data. It is shown the ACTIVE condition is extremely less than the non-ACTIVE situation. Thus, we would like to try processing on data.

```
: t=ch_train_df["ACTIVE"]
  t.value_counts()

: 0    18604
  1      521
  Name: ACTIVE, dtype: int64
```

Usually there are two methods to process the imbalanced data: under sampling and over sampling. Under sampling may lead to the information loss. Due the high dimension of the features, the larger dataset is preferred, so we would like to do oversampling.

But how to do oversampling? According to the references [3,4,5], two ways are chosen to do this, the first one is random oversampling and the second is the algorithm of SMOTE (Synthetic Minority Oversampling Technique), both methods are from Scikit-learn.

Based on the result of GridSearchCV, the model is fitted by three kinds of data: original data, random oversampled data and oversampled data based on SMOTE. The result is shown in Figure 2.

We also set n estimators to see the changing of the scores. It is shown that result from data of SMOTE is better than the other two, the best score is 0.8094, which is achieved when n_estimators = 200. This is same as the result of GridSearchCV algorithm.

At last we get the model:

clf = RandomForestClassifier(n_estimators=200,random_state=0, max_depth=20)

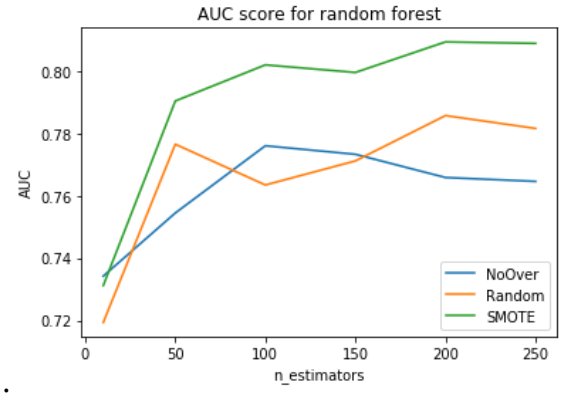

Figure 2 The AUC scores for different datasets

5. Prediction and result analysis

The last step is fitting the model with the data from SMOTE oversampling and use it to predict the AUC scores for the real test data. Note that for the test data, we need to do the same preprocessing as the training process (imputation, normalization and discretization).

As we have known, the finial test AUC score is 0.7904, which is lower than the estimated AUC score (0.8094). I think that may due to the statistical method of the estimated score. The current way is to do the test with the same 20% test data as the training process, and a better way is to do another cross validation which may reduce the deviance.

References:

[1] https://www.rdkit.org

[2] https://scikit-learn.org/stable/

[3] https://towardsdatascience.com/sampling-techniques-for-extremely-imbalanced-data-part-ii-over-sampling-d61b43bc4879

[4] https://towardsdatascience.com/sampling-techniques-for-extremely-imbalanced-data-part-i-under-sampling-a8dbc3d8d6d8

[5] https://beckernick.github.io/oversampling-modeling/

[6] https://kiwidamien.github.io/how-to-do-cross-validation-when-upsampling-data.html