Learning to Smell Report

Author: Huaping Ding

# **Methodology**

## **2. Auto-classifier**

As discussed earlier, this learning to smell problem is obviously a multiple-class and multiple-label classification problem, and can be properly solved with many machine learning algorithms. Therefore, in this section, we proposed to use the auto-classifier approach—applying many classic classification algorithms with different thresholds to identify the best algorithm and parameters that can produce the best predictions of smell, evaluated by the required performance metric for this challenge.

Classic classification algorithms have been implemented in many open-source libraries, such as scikit-learn[1], and there is no need to reinvent the wheels. We have been suggested that a useful Python library Polyssifier [2] had leveraged the scikit-learn package to easily run multiple algorithms and produce nice graphic report to get results. We now discuss the details in the following sub-sections.

### **2.1 Using Polyssifier**

Polyssifier is an open-source Python package which can be downloadable from GitHub[2]. It includes the following classifiers: Multiplayer Perceptron, KNN, SVM (Linear or RBF kernel), Decision Tree, Random Forest, Logistic Regression, Naïve Bayes, and Voting classifier.

It also has the following features:

* Cross validated scores and graphic reporting
* Feature ranking for compatible models
* Parallel processing
* Model saving for future use
* Feature selection available and automatic data scale

We will be mainly using its parallel processing and reporting feature. Once Polyssifier reports the best algorithm and parameters, we will use that settings to re-train the model to generate the final performance with Jaccard similarity score.

### **2.2 Feature Engineering with Molecular Descriptors[3]**

As showed earlier, the data has about 4000 chemicals in SMILE string format. A proper feature engineering process has to be involved to convert them properly and meaningfully into numbers.

Fortunately, we found the tool Mordred—a molecular descriptor calculator[3], which can compute thousands of chemical property indexes from molecular structures, such as Aromatic, Polarizability, number of Acids, etc. Those indexes are naturally describing the chemical or biological property and are very related to the smell of such chemical.

The following code shows how to convert SMILES string into molecular structure, and then calculate the chemical indexes.

*from rdkit import Chem*

*from mordred import Calculator, descriptors*

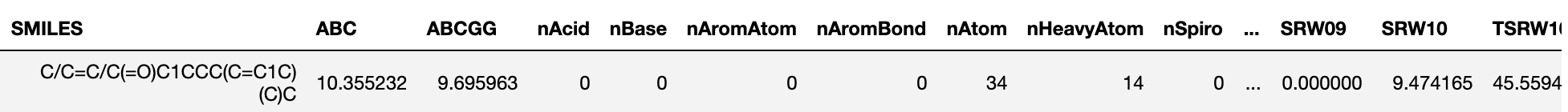
*mol = Chem.MolFromSmiles(SMILES\_STRING)*

*calc = Calculator(descriptors, ignore\_3D=True)*

*calc(mol)*

However, due to some chemical properties may not necessarily available for the chemicals in this SMILES data, we only get about **641** indexes as features in the end.

Take a SMILES string “C/C=C/C(=O)C1CCC(C=C1C)(C)C” as example, it has the following features:



### **2.3 Target Label Encoding**

There are total about 108 different kinds of smells in the data set as our prediction target. Some chemicals may have multiple types of smells, and the maximum number of smells in the training data is 11. Therefore, a proper encoding strategy is used to convert the multiple-class and multiple-label target into prediction labels.

We used a simple numeric encoding schema, based on the order of smells listed in the vocabulary file (the whole list of 108 smells). For example, the first smell “alcoholic” is coded as 1, 2nd smell “aldehydic” is coded as 2, and so on. The final smell “woody” is coded as 108.

For the multiple labels problem in many chemicals, we simply encode each smell sequentially, toe the target label is an array of integers, up to 11.

Take one smell sentence as example, “fruity, rose” will be encoded as [48, 89] based on our vocabulary mapping.

### **2.4 Experiment Settings**

We used all the classifiers that are supported by Polyssifier, and use the default parameter settings. A 3-fold cross-validations is used in the first round of experiments. Scoring function is chosen as Jaccard to match the challenge evaluation metrics.

The following code is an example of calling the Polyssifier main loop.

*report = poly(data, label, n\_folds=3, verbose=True, save=True, scale=True, feature\_selection=False, scoring='jaccard', concurrency=4, exclude=exclude\_list)*

*report.plot\_scores()*

In the data exploration stage, we notice that a few smell categories only show up a few times in the whole training data set. Therefore, during the random cross-validation stage, the training set have different label categories than the testing data set, leading to different shapes of confusion matrix to calculate the Jaccard similarity.

We updated the Polyssifier code by identifying the label difference between training data and testing data, and then insert a count of 0 for those labels before computing confusion matrix.

# **Experiment Results & Comparison**

### **Results of Polyssifier**

The full run time with default Polyssifier with 3-fold cross validation is about 2 hours with 4 concurrency threads using a i7 quad-core 2.8G Hz computer. Majority of the time is spent on training SVM models.

This following table and graph shows the average training and testing results of prediction the first smell of the chemical.

mean std min max

Multilayer Perceptron train 0.851364 0.005463 0.845159 0.855450

test 0.191342 0.007135 0.184296 0.198562

Nearest Neighbors train 0.966250 0.000802 0.965473 0.967075

test 0.202481 0.004226 0.198154 0.206598

SVM train 0.252679 0.006062 0.248859 0.259669

test 0.123779 0.012964 0.115292 0.138701

Linear SVM train 0.549309 0.099997 0.434209 0.614827

test 0.131784 0.012137 0.120776 0.144800

Decision Tree train 0.859836 0.003611 0.855695 0.862334

test 0.165131 0.003712 0.161613 0.169010

Random Forest train 0.858795 0.003485 0.854772 0.860865

test 0.202491 0.007352 0.195804 0.210364

Logistic Regression train 0.490507 0.012060 0.477522 0.501356

test 0.216362 0.006067 0.210282 0.222417

Naive Bayes train 0.116289 0.003263 0.112750 0.119178

test 0.037227 0.010612 0.030472 0.049459

Voting train 0.876612 0.003722 0.872580 0.879916

test 0.212953 0.006799 0.206813 0.220260

Chart, bar chart

Description automatically generated

The same process is executed again to predict the 2nd smell, 3rd smell, etc.

We have some observations after running all the results:

* Performance are significantly higher in training set than in testing set, meaning that overfitting is happening here.
* Random Forest are normally ranked among the top 3 of all classifiers.
* Naïve Bayes is really not performing well in all cases.
* SVM always takes a lot of time, but didn’t perform very well

### **Results of Best Random Forest**

Based on the conclusion of Polyssifier experiment, we choose the best parameters of Random Forest to rebuilt models to see performance.

Meanwhile, we did not simply pick the prediction of random forest as predicted label, but using its prediction probability to select top K classes as the prediction results, and then compute the Jaccard similarity with true labels. We tried K to be 4, 3, 2, 1.

In this experiment, we use a 5-fold cross validation, and the final result shows that K = 1 producing the highest Jaccard score. The following number shows the results of best random forest.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Fold** | **1** | **2** | **3** | **4** | **5** | **Average** |
| Jaccard Similarity | 0.2063 | 0.2177 | 0.2228 | 0.2141 | 0.1952 | 0.211 ± 0.01 |

## References:

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

[2] Polyssifier GitHub code repository: https://github.com/alvarouc/polyssifier

[3] Mordred GitHub repository: https://github.com/mordred-descriptor/mordred