COMP9313 Big Data Management

Project 2

T2, 2020

Name : Yuchen Yang

Student number : z5189310

1. Evaluation of your stacking model on the test data.

This is how I think about the design of the entire model. This project mainly investigates the use of pipeline and how to better use Estimator.

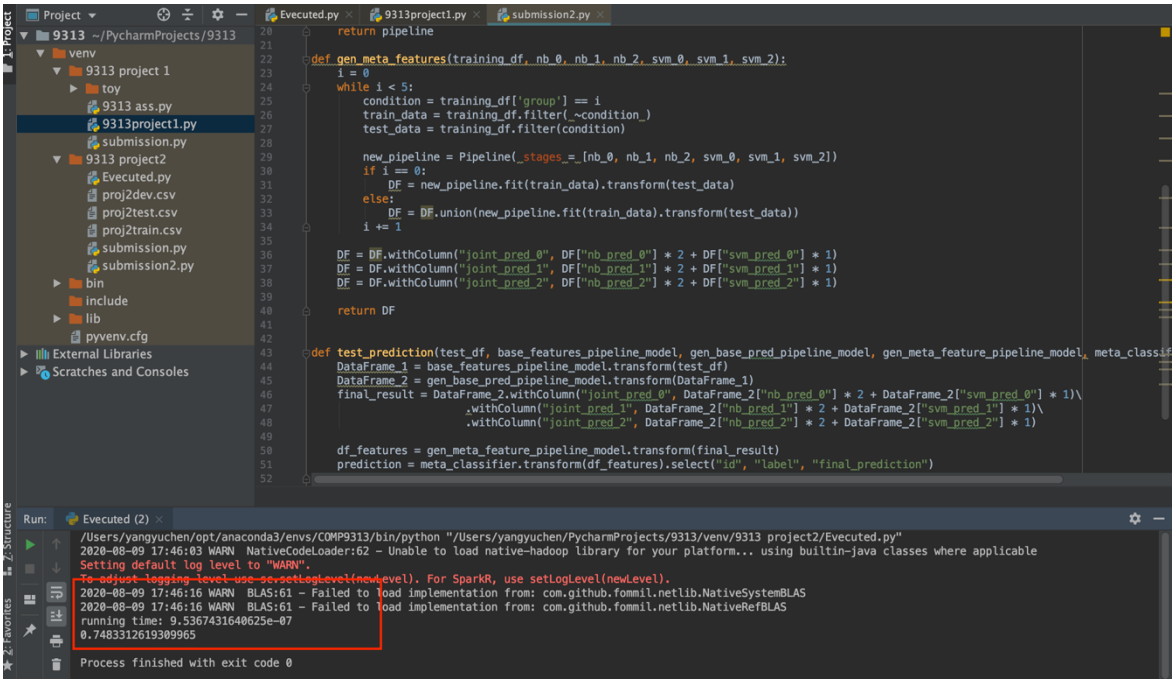
For the data preprocessing of Test1.1, I mainly used Estimator function(NaiveBayes, StringIndexer and CountVectorizer) to train the whole original dataFrame with FIT operation, changing the word in descript to a single and listing a new column according to feastures, The transfrom operation with Tokenizer is then used to generate New DataFrame.

After the operation of gen_binary_labels, I got the training_set of 5 groups and 10 columns of data among Task1.2 input. According to the generate meta feature concept, I divided the original data into 5 groups again by using the 5 groups that have been classified by the project2. Taking group1 as an example, I divide group2, group3, group4 and group5 into one group as base_model, and group1 as generate meta feature. The same is true of other groups. I use base_model to train the features and label in group2-5 again to generate the new pipeline. Then, generate a meta feature into a new pipeline to generate a new prediction column. Finally, the lable nb_pred and svm_pred were joined to get the final result.

As for Task1.3, I passed the meta_features and label I had created into the Meta Model Logistic Regression for training and prediction model generation.

Reflecting on the whole process of the project, I tried my best to avoid reducing or avoiding join operations in the code. There are two reasons. The first one is to double-calculate each sequence and then join the pipeline, which requires repeated traversal and takes a long time. The second reason is that join operation is too easy to shuffle for big data.

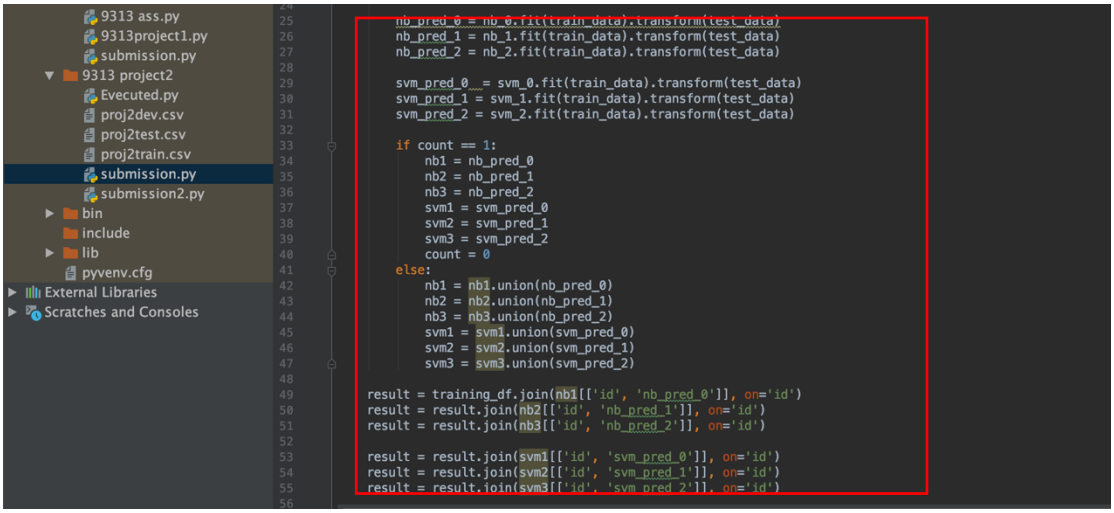
According to the test Data, my running time is 9.5367.



2. How would you improve the performance (e.g., F1) of the stacking model ?

1) Remove the join mode and use the mode of pipeline

When I wrote the project for the first time, I calculated the number of each column separately and then joined the original data one by one. But, a shuffle is very easy to happen in a join operation. In the final testing stage, I found that this was too time-consuming.



First-version in project2 (join function)

So I tried using a Pipeline, and in my second code, I created a new pipeline so that all NB and SVM computations could be done in the 4 lines of code instead of joining 6 times.

```
new_pipeline = Pipeline(stages = [nb_0, nb_1, nb_2, svm_0, svm_1, svm_2])
if i == 0:
    DF = new_pipeline.fit(train_data).transform(test_data)
else:
    DF = DF.union(new_pipeline.fit(train_data).transform(test_data))
```

2) More needs to be done in the early data preprocessing

When looking at descript data, I found that there were a lot of punctuation that greatly affected our implementation efficiency. It would be more efficient to get rid of the punctuation early on. tuning hyper para of meta model.

3) Tuning hyper para of meta model

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. I want to improve F1 by adjusting the parameters here.

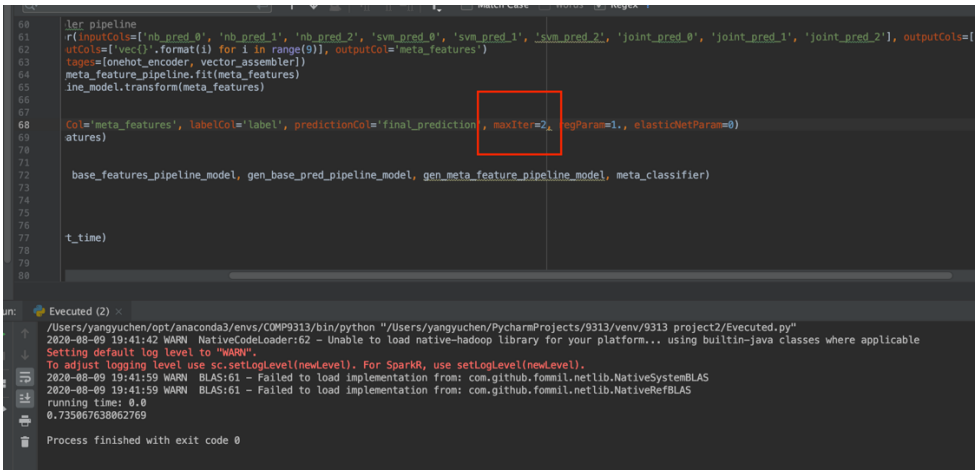
```
ne meta clasifier
= LogisticRegression(featuresCol='meta_features', labelCol='label', predictionCol='final_prediction', maxIter=20, regParam=1., elasticNetParam=0)
sifier = lr_model.fit(meta_features)
```

- maxIter

Max number of iterations (>= 0). (default: 100, current: 20). Logistic regression is optimized by iterative methods like gradient descent. I used the Development Dataset to adjust the data.

When I increase the maximum number of iterations to 100, the running time increases but the accuracy increases to 0.7502.

By reducing the maximum number of iterations to 2, the running time is reduced but the accuracy is reduced to 0.73506.



maxIter = 2 with 0.73506 prediction

- regParam
  regularization parameter (>= 0). (default: 0.0)
- elasticNetParam

The ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty. (default: 0.0)

For the purpose of hyperparameter tuning we will consider the following parameters:

- maxIter [10, 100, 1000]
- regParam [0.01, 0.5, 2.0]
- elasticNetParam [0.0, 0.5, 1.0]