2020

Project 2

COMP9313 | BIG DATA MANAGEMENT |20T2

ANANT KRISHNA MAHALE | Z5277610

In this project we need to implement several core parts of the stacking machine learning method in Pyspark.

Given Data files:

- proj2train.csv
- proj2test.csv

Looking inside the data file reveals that there are 3 columns:

id	category	descript
0	MISC	I've been there three times and have always had wonderful experiences.
1	FOOD	Stay away from the two specialty rolls on the menu, though- too much avocado and rice will fill you up right quick.
2	FOOD	Wow over 100 beers to choose from.
3	MISC	Having been a long time Ess-a-Bagel fan, I was surpised to find myself return time and time again to Murray's.
4	MISC	This is a consistently great place to dine for lunch or dinner.
5	FOOD	I ate here a week ago and found most dishes average at best and too expensive.
6	MISC	First of all Dal Bukhara Rocks.

Category being the target column, there are 3 target labels: **FOOD**, **MISC**, **PAS**.

Implementation details:

1.1

- This function completion was very similar to the Lab3. The descript column is tokenized using `Tokenizer()`, and bag of words is generated using `StringIndexer()`.
- Since there are 3 output categories, they converted into integers between 0 to 2.
- The output of the function should be pipeline, hence, above generated features are converted into pipeline using `pipeline()`
- Below is the preview of the code for the function: `base_features_gen_pipeline()`

```
class Selector(Transformer):
    def __init__(self, outputCols=['features', 'label']):
        self.outputCols = outputCols

    def _transform(self, df: DataFrame) -> DataFrame:
        return df.select(*self.outputCols)

def base_features_gen_pipeline(input_descript_col="descript", input_category_col="category", output_feature_col="features", output_label_col="label"):
    # white space expression tokenizer
    word_tokenizer = Tokenizer(inputCol=input_descript_col, outputCol="words")

    # bag of words count
    count_vectors = CountVectorizer(inputCol="words", outputCol=output_feature_col)

    # label indexer
    label_maker = StringIndexer(inputCol=input_category_col, outputCol=output_label_col)

    selector = Selector(outputCols=['id','features', 'label'])

    # build the pipeline
    pipeline = Pipeline(stages=[word_tokenizer, count_vectors, label_maker, selector])

    return pipeline
```

1.2

- This function focuses on the Meta-features. The base models defined in the main function are passed to this function along with group column. The model is trained (total_number_of_group) times, such that, the model is trained with the entries that are not in the group.

```
        condition = training_df['group'] == i
        c_train = training_df.filter(~condition).cache()
        c_test = training_df.filter(condition).cache()
```

- using the predicted values, the meta-perameters are generated like below.

```
temp_result_1 = temp_result_1.withColumn("joint_pred_0",2*temp_result_1.nb_pred_0 +temp_result_1.svm_pred_0)
temp_result_1 = temp_result_1.withColumn("joint_pred_1",2*temp_result_1.nb_pred_1 +temp_result_1.svm_pred_1)
temp_result_1 = temp_result_1.withColumn("joint_pred_2",2*temp_result_1.nb_pred_2 +temp_result_1.svm_pred_2)
```

- Preview of final code:

```
def gen_meta_features(training_df, nb_0, nb_1, nb_2, svm_0, svm_1, svm_2):
    for i in range(5):
        condition = training_df['group'] == i
        c_train = training_df.filter(~condition).cache()
        c_test = training_df.filter(condition).cache()

        nb_model_0 = nb_0.fit(c_train)
        nb_pred_0 = nb_model_0.transform(c_test)

        nb_model_1 = nb_1.fit(c_train)
        nb_pred_1 = nb_model_1.transform(c_test)

        nb_model_2 = nb_2.fit(c_train)
        nb_pred_2 = nb_model_2.transform(c_test)

        svm_model_0 = svm_0.fit(c_train)
        svm_pred_0 = svm_model_0.transform(c_test)

        svm_model_1 = svm_1.fit(c_train)
        svm_pred_1 = svm_model_1.transform(c_test)

        svm_model_2 = svm_2.fit(c_train)
        svm_pred_2 = svm_model_2.transform(c_test)

        if (i<1):
            temp_df = c_test.join(nb_pred_0, on = ['id']).select(c_test["*"],nb_pred_0["nb_pred_0"])
            temp_df = temp_df.join(nb_pred_1, on = ['id']).select(temp_df["*"],nb_pred_1["nb_pred_1"])
            temp_df = temp_df.join(nb_pred_2, on = ['id']).select(temp_df["*"],nb_pred_2["nb_pred_2"])

            temp_df = temp_df.join(svm_pred_0, on = ['id']).select(temp_df["*"],svm_pred_0["svm_pred_0"])
            temp_df = temp_df.join(svm_pred_1, on = ['id']).select(temp_df["*"],svm_pred_1["svm_pred_1"])
            temp_df = temp_df.join(svm_pred_2, on = ['id']).select(temp_df["*"],svm_pred_2["svm_pred_2"])
            result_df = temp_df
        else:
            temp_df = c_test.join(nb_pred_0, on = ['id']).select(c_test["*"],nb_pred_0["nb_pred_0"])
            temp_df = temp_df.join(nb_pred_1, on = ['id']).select(temp_df["*"],nb_pred_1["nb_pred_1"])
            temp_df = temp_df.join(nb_pred_2, on = ['id']).select(temp_df["*"],nb_pred_2["nb_pred_2"])

            temp_df = temp_df.join(svm_pred_0, on = ['id']).select(temp_df["*"],svm_pred_0["svm_pred_0"])
            temp_df = temp_df.join(svm_pred_1, on = ['id']).select(temp_df["*"],svm_pred_1["svm_pred_1"])
            temp_df = temp_df.join(svm_pred_2, on = ['id']).select(temp_df["*"],svm_pred_2["svm_pred_2"])
            result_df = result_df.union(temp_df)

    result_df = result_df.orderBy('id', ascending=True)
    result_df = result_df.withColumn("joint_pred_0",2*result_df.nb_pred_0 +result_df.svm_pred_0)
    result_df = result_df.withColumn("joint_pred_1",2*result_df.nb_pred_1 +result_df.svm_pred_1)
    result_df = result_df.withColumn("joint_pred_2",2*result_df.nb_pred_2 +result_df.svm_pred_2)
    result_df = result_df.select(result_df.id,result_df.group,result_df.features,result_df.label,result_df.label_0,result_df.label_1, result_df.label_2,result_df.nb_pred_0,result_df.nb_pre
    return result_df
```

- Output of gen_meata_features()

id	group	features	label	label_0	label_1	label_2	nb_pred_0	nb_pred_1	nb_pred_2	svm_pred_0	svm_pred_1	svm_pred_2	joint_pred_0	joint_pred_1	joint_pred_2
0	4	(5421,[1,18,31,39 ...	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0
1	4	(5421,[0,1,15,20, ...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4	(5421,[3,109,556, ...	0.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	2.0
3	0	(5421,[1,2,3,5,6, ...	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
4	2	(5421,[2,3,4,8,11 ...	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
5	0	(5421,[1,2,5,25,4 ...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0	0.0
6	4	(5421,[7,40,142,1 ...	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	3.0	2.0	0.0
7	4	(5421,[8,13,19,25 ...	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0	0.0
8	4	(5421,[2,3,7,8,21 ...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
9	4	(5421,[2,16,22,49 ...	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0

1.3

- In this task, stacked model is used to predict the values using test dataset.
- Also, meta features for the test_df is generated in this function.
- Preview of the function test_prediction()

```
def test_prediction(test_df, base_features_pipeline_model, gen_base_pred_pipeline_model, gen_meta_feature_pipeline_model, meta_classifier):
    temp_result_0 = base_features_pipeline_model.transform(test_df)
    temp_result_1 = gen_base_pred_pipeline_model.transform(temp_result_0)

    #find the joint probability or generate meta-parameters.
    temp_result_1 = temp_result_1.withColumn("joint_pred_0",2*temp_result_1.nb_pred_0 +temp_result_1.svm_pred_0)
    temp_result_1 = temp_result_1.withColumn("joint_pred_1",2*temp_result_1.nb_pred_1 +temp_result_1.svm_pred_1)
    temp_result_1 = temp_result_1.withColumn("joint_pred_2",2*temp_result_1.nb_pred_2 +temp_result_1.svm_pred_2)

    temp_result_2 = gen_meta_feature_pipeline_model.transform(temp_result_1)
    temp_final = meta_classifier.transform(temp_result_2)
    final_result = temp_final.select(temp_final.id, temp_final.label, temp_final.final_prediction)
    return final_result
```

- Output of the function:

```
+---+-----+----------------+
| id|label|final_prediction|
+---+-----+----------------+
|  0|  0.0|             0.0|
|  1|  2.0|             0.0|
|  2|  0.0|             0.0|
|  3|  0.0|             0.0|
|  4|  0.0|             0.0|
|  5|  1.0|             1.0|
|  6|  0.0|             0.0|
|  7|  0.0|             0.0|
|  8|  0.0|             0.0|
|  9|  0.0|             0.0|
+---+-----+----------------+
```

The evaluation of the stacking model gives the f1-score of 0.7483312619309965

What can be improved?

- Preprocessing the text:
    1. Instead of using the text as is, the symbols, non-ascii chars, numbers along with white spaces can be removed.
    2. Lemmatization and stemming techniques can be applied on the text, to make it uniform.
- Improving the F1 score.
    1. Using the above said pre-processing techniques
    2. Instead of using model with default parameters, we can use hyper parameters to suit the data

- Hyper -Parameters tuning:
- I tried many hyper parameters with NB and SVC models. I got best results when maxIter = 100, regParam = 0.08 in SVC. For NB changing the params like smoothing did not have significant improvement on F1-score.