Implementation details:

For task1.1:

The whole process of creating a pipeline is basically the same as that of L8:slide 31, except that nb_model is not created.

For task1.2:

Firstly, we need to calculate the number of groups for subsequent loop operations, and then create a base pipeline. Secondly, traverse the group_index and pass each group_index into the prepare_data_model() I created. The training_df is divided into two parts and one part is given to training_data (when group_index not equal training_data is the value of the group column) and part is given to the predicted data (when group_index equal training_data is group the value of column). Then fit the training_data to base_pipeline and transform the prediction data. Finally, the two columns nb_pred and nvm_pred are combined and added to create a new column joint_pred, which is combined three times(index = 0,1,2). In the process of combine, we need to convert nb_pred and nvm_pred columns from float to integer, and then to string, combine the string. Then use int (x, base) to convert to decimal integer, and finally to float.

For task1.3:

Follow the lecture slides, generate meta features for prediction, using combine_column() function combine the column nb_pred and nvm_pred(same as task1.2). Finally, prediction using meta classifier.

Evaluation of stacking model on the test data:

```
lr_model = LogisticRegression(featuresCol='meta_features', labelCol='label', predictionCol='final_prediction', maxIter=20, regParam=1.,
meta_classifier = lr_model.fit(meta_features)

# task 1.3
pred_test = test_prediction(test_data, base_features_pipeline_model, gen_base_pred_pipeline_model, gen_meta_feature_pipeline_model, meta_

# Evaluation
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",metricName='f1')
end_time = time()
print('running time:', end_time - start_time)
print(evaluator.evaluate(pred_test, {evaluator.predictionCol:'final_prediction'}))
spark.stop()
```

```
running time: 342.85889649391174
0.7483312619309965
```

Get the result same as project2_spec.

Improve the performance of the stacking model:

1. When train the meta classifier we use the LogisticRegression() that may be easy to underfit and the classification accuracy is not high. There may also be data features that are missing or when the feature space is large, the effect is not good. We can reduce the number of features or regularize which may improve performance. Regularize prevent overfitting by penalizing too large parameters.

2. When we build a preprocessing pipeline, Tokenizer will not deal with the problem of punctuation, we can remove the punctuation. We can use RegexTokenizer to deal with this problem and improve the performance of stacking model.
3. We can choose different base models instead of naive Bayes and support vector machines. We can choose AdaBoost or Random forest or others to test. The advantage of AdaBoost is that it comes with feature selection and only uses features found to be effective in the training set. This reduces the number of features that need to be calculated during classification, and to a certain extent solves the problem of incomprehensible high-dimensional data.