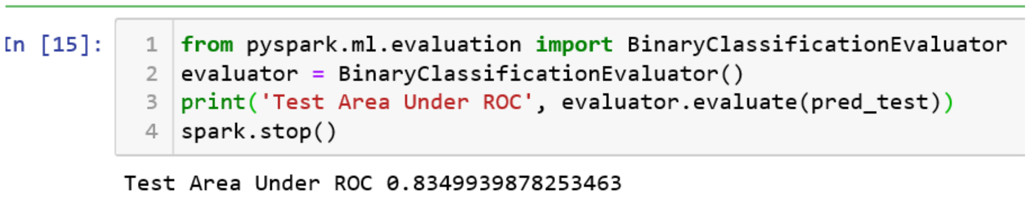
COMP9313
Clare Xinyu Xu
z5175081

1. Evaluation of your stacking model on the test data

The result shows that the performance on test data is 0.7483312619309965.
Which means that the precision of test data with the F1 score method is around 74.83%
that is a moderate performance but frankly speaking not very outstanding.
I have also tried other matrixes on the performance of the test data.
By select one more column which is rawPrediction, I use the BinaryClassificationEvaluato
r method to calculate the ROC accuracy.

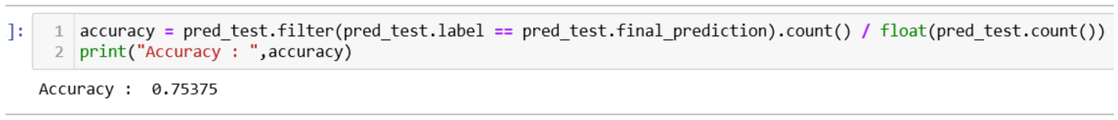


```
In [15]:
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(pred_test))
spark.stop()
```

```
Test Area Under ROC 0.8349939878253463
```

Here I use AUC to measure logistic regression model. The AUC is the area under the ROC curve. Typically, an AUC of 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding. Thus, the model here performs not bad for distinguishing the class and the result of test data is also acceptable.

Besides, I calculate the accuracy to evaluate the model on test data.



```
]:
accuracy = pred_test.filter(pred_test.label == pred_test.final_prediction).count() / float(pred_test.count())
print("Accuracy : ",accuracy)
```

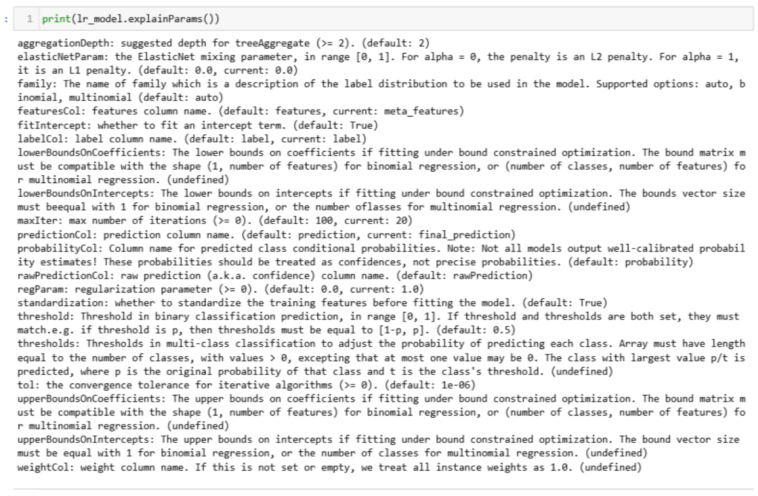
```
Accuracy :  0.75375
```

While the percentage of accuracy is the number of correct prediction (for which the label is same as the one that be predicted) divided the total number of predictions. It turns out that accuracy is 75.3% which is considerate acceptable but need to improved somehow.

2. How would you improve the performance of the stacking model?

In order to increase the performance of the stacking model, I am trying to tune the parameter of model with gridSearch and the cross validation.

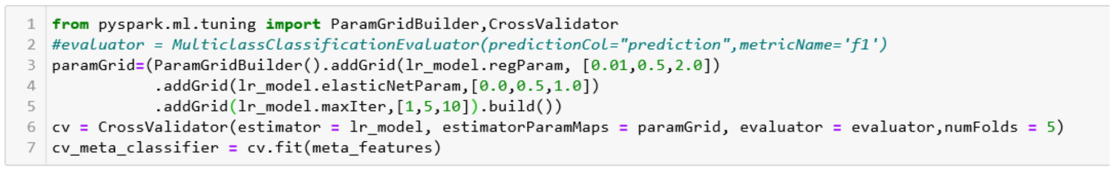
To be clear about what params are available for tuning, I print out a list of the params and the definitions via explainParams().



```
:
print(lr_model.explainParams())
```

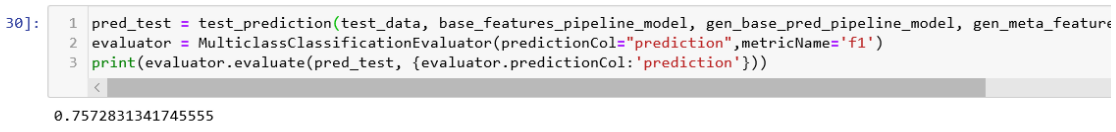
```
aggregationDepth: suggested depth for treeAggregate (>= 2). (default: 2)
elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1,
it is an L1 penalty. (default: 0.0, current: 0.0)
family: The name of family which is a description of the label distribution to be used in the model. Supported options: auto, b
inomial, multinomial (default: auto)
featuresCol: features column name. (default: features, current: meta_features)
fitIntercept: whether to fit an intercept term. (default: True)
labelCol: label column name. (default: label, current: label)
lowerBoundsOnCoefficients: The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix m
ust be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) fo
r multinomial regression. (undefined)
lowerBoundsOnIntercepts: The lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size
must beequal with 1 for binomial regression, or the number oflasses for multinomial regression. (undefined)
maxIter: max number of iterations (>= 0). (default: 100, current: 20)
predictionCol: prediction column name. (default: prediction, current: final_prediction)
probabilityCol: Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probabil
ity estimates! These probabilities should be treated as confidences, not precise probabilities. (default: probability)
rawPredictionCol: raw prediction (a.k.a. confidence) column name. (default: rawPrediction)
regParam: regularization parameter (>= 0). (default: 0.0, current: 1.0)
standardization: whether to standardize the training features before fitting the model. (default: True)
threshold: Threshold in binary classification prediction, in range [0, 1]. If threshold and thresholds are both set, they must
match.e.g. if threshold is p, then thresholds must be equal to [1-p, p]. (default: 0.5)
thresholds: Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length
equal to the number of classes, with values > 0, excepting that at most one value may be 0. The class with largest value p/t is
predicted, where p is the original probability of that class and t is the class's threshold. (undefined)
tol: the convergence tolerance for iterative algorithms (>= 0). (default: 1e-06)
upperBoundsOnCoefficients: The upper bounds on coefficients if fitting under bound constrained optimization. The bound matrix m
ust be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) fo
r multinomial regression. (undefined)
upperBoundsOnIntercepts: The upper bounds on intercepts if fitting under bound constrained optimization. The bound vector size
must be equal with 1 for binomial regression, or the number of classes for multinomial regression. (undefined)
weightCol: weight column name. If this is not set or empty, we treat all instance weights as 1.0. (undefined)
```

I indicate three values for regularization parameter (regParam), three values for number of iterations (maxIter) and two values for elastic net parameter(elasticNetParam) which in total is 3^3 = 27 parameters. I also create the five-cross validator.



```python
from pyspark.ml.tuning import ParamGridBuilder,CrossValidator
#evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",metricName='f1')
paramGrid=(ParamGridBuilder().addGrid(lr_model.regParam, [0.01,0.5,2.0])
           .addGrid(lr_model.elasticNetParam,[0.0,0.5,1.0])
           .addGrid(lr_model.maxIter,[1,5,10]).build())
cv = CrossValidator(estimator = lr_model, estimatorParamMaps = paramGrid, evaluator = evaluator,numFolds = 5)
cv_meta_classifier = cv.fit(meta_features)
```

Therefore, the performance works better, which indicate that the F1 score increases a bit to around 0.757.



```python
30]:
pred_test = test_prediction(test_data, base_features_pipeline_model, gen_base_pred_pipeline_model, gen_meta_feature
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction",metricName='f1')
print(evaluator.evaluate(pred_test, {evaluator.predictionCol:'prediction'}))
```

```
0.7572831341745555
```