

EECS E6893 Big Data Analytic HW1

Chong Hu ch3467

October 3, 2019

Problem 1. Iterative K-means clustering on Spark

1. L1 distance

Here is the terminal command line screenshot.

```
huchong@huchong:~/huchong/colombia/EECS_6893/Big_Data_Analytics/Homework/hw1/q15 gcloud dataproc jobs submit pyspark kmeans.py --cluster=big-data
Job [ea224ca13c6d47c884384c1002abfbc] submitted.
Waiting for job output...
19/10/03 03:23:53 INFO org.spark_project.jetty.util.log: Logging initialized @3000ms
19/10/03 03:23:53 INFO org.spark_project.jetty.server.Server: Jetty-9.3.2-SNAPSHOT, build timestamp: unknown, git hash: unknown
19/10/03 03:23:53 INFO org.spark_project.jetty.server.Server: Started @3101ms
19/10/03 03:23:53 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@725364a7(HTTP/1.1,[http/1.1])[0.0.0.0:4040]
19/10/03 03:23:53 WARN org.apache.spark.scheduler.FairSchedulerBuilder: Fair scheduler configuration file not found so jobs will be scheduled in FIFO order. To use fair scheduling, configure pools in fa
irscheduler.xml or set spark.scheduler.allocation.file to a file that contains the configuration.
19/10/03 03:23:56 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
19/10/03 03:23:57 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
Run kmeans clustering.
Plot loss.
Done!
19/10/03 03:24:47 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@725364a7(HTTP/1.1,[http/1.1])[0.0.0.0:4040]
Job [ea224ca13c6d47c884384c1002abfbc] finished successfully.
DriverControlFileURL: gs://big_data_storage/google-cloud-dataproc-meta/info/05e3acbb-384f-4e76-9979-68774f5b94f1/jobs/ea224ca13c6d47c884384c1002abfbc/
DriverOutputFileURL: gs://big_data_storage/google-cloud-dataproc-meta/info/05e3acbb-384f-4e76-9979-68774f5b94f1/jobs/ea224ca13c6d47c884384c1002abfbc/driveroutput
JobID: 324a8fdf-42eb-33db-af00-b3d456ab9d36
Placement:
  clusterName: big-data
  clusterId: 05e3acbb-384f-4e76-9979-68774f5b94f1
  pySparkJob:
    mainPythonFileURL: gs://big_data_storage/google-cloud-dataproc-meta/info/05e3acbb-384f-4e76-9979-68774f5b94f1/jobs/ea224ca13c6d47c884384c1002abfbc/staging/kmeans.py
  reference:
    JobID: ea224ca13c6d47c884384c1002abfbc
    ProjectID: hardy-symbol-252200
  status:
    state: DONE
    stateStartTime: '2019-10-03T03:24:40.322Z'
  statusHistory:
    state: PENDING
    stateStartTime: '2019-10-03T03:23:48.745Z'
    state: SETUP_DONE
    stateStartTime: '2019-10-03T03:23:48.785Z'
    details: Agent reported job success
    state: RUNNING
    stateStartTime: '2019-10-03T03:23:49.692Z'
```

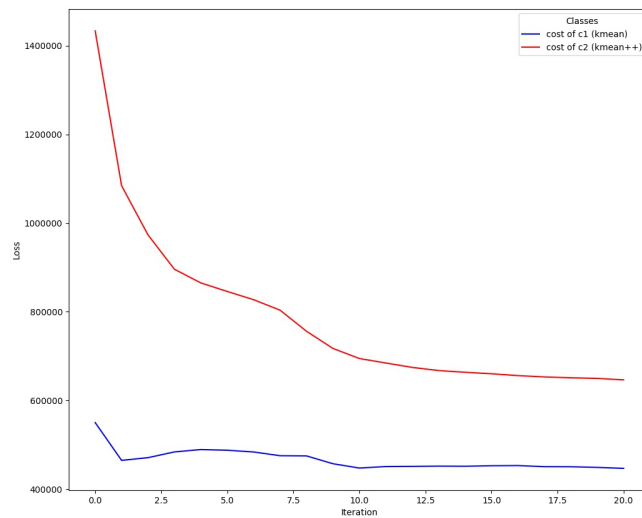


Figure 1: Loss for L1 distance

2. L2 distance

Here is the terminal command line screenshot.

```
huchong@huchong:~/Huchong/ColinBtl/ESCS_E6893_Big_Data_Analytics/Homework/hw1/q15 gcloud dataproc jobs submit pyspark kmeans.py --cluster=big-data
Job [db8bc55f497746ae894a9a1ffd57c67] submitted.
Waiting for job output...
19/10/03 03:16:57 INFO org.spark_project.jetty.util.log: Logging initialized @3997ms
19/10/03 03:16:57 INFO org.spark_project.jetty.server.Server: jetty-9.3.2-SNAPSHOT, build timestamp: unknown, git hash: unknown
19/10/03 03:16:57 INFO org.spark_project.jetty.server.Server: Started @3896ms
19/10/03 03:16:57 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@3fa1987d[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
19/10/03 03:16:57 WARN org.apache.spark.scheduler.FairSchedulerBuilder: Fair scheduler configuration file not found so jobs will be scheduled in FIFO order. To use fair scheduling, configure pools in fa
irscheduler.xml or set spark.scheduler.allocation.file to a file that contains the configuration.
19/10/03 03:17:00 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
19/10/03 03:17:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
Run kmeans clustering.
19/10/03 03:17:01 INFO org.apache.hadoop.mapred.FileInputFormat: Total input files to process : 1
Run kmeans clustering.
Plot loss.
For L2 norm, plot 2D clustering result.
Plot kmean clustering result.
Plot kmeans clustering result.
Done!
19/10/03 03:21:29 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@3fa1987d[HTTP/1.1,[http/1.1]]{0.0.0.0:4040}
Job [db8bc55f497746ae894a9a1ffd57c67] finished successfully.
DriverControlFilesUrl: gs://big_data_storage/google-cloud-dataproc-metaInfo/05e3acbd-384f-4e76-9979-68774f5b94f1/jobs/db8bc55f497746ae894a9a1ffd57c67/
DriverOutputFilesUrl: gs://big_data_storage/google-cloud-dataproc-metaInfo/05e3acbd-384f-4e76-9979-68774f5b94f1/jobs/db8bc55f497746ae894a9a1ffd57c67/driveroutput
JobId: 097f452e-a166-368d-9cde-d9a6ea77b1e
Placement:
  clusterName: big-data
  clusterUuid: 05e3acbd-384f-4e76-9979-68774f5b94f1
pysparkJob:
  mainPythonFileUrl: gs://big_data_storage/google-cloud-dataproc-metaInfo/05e3acbd-384f-4e76-9979-68774f5b94f1/jobs/db8bc55f497746ae894a9a1ffd57c67/staging/kmeans.py
references:
  jobId: db8bc55f497746ae894a9a1ffd57c67
  projectId: hardy-symbol-252200
status:
  state: DONE
  stateStartTime: '2019-10-03T03:21:32.945Z'
statusHistory:
  - state: PENDING
    stateStartTime: '2019-10-03T03:16:53.321Z'
  - state: SETUP_DONE
    stateStartTime: '2019-10-03T03:16:53.359Z'
  - details: Agent reported job success
    state: RUNNING
    stateStartTime: '2019-10-03T03:16:53.620Z'
```

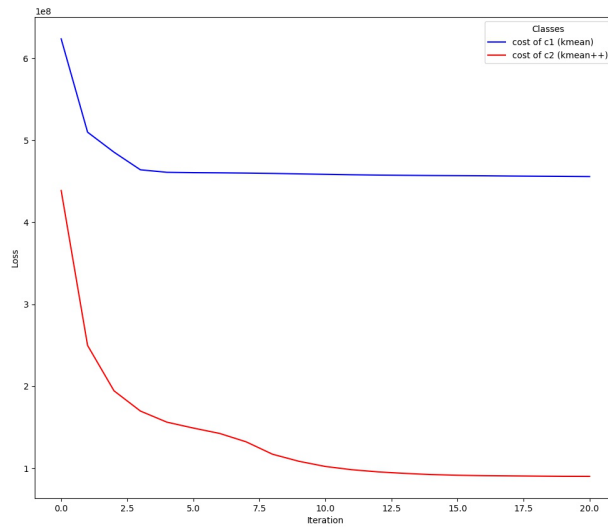


Figure 2: Loss for L2 distance

3. Visualization of high-dimensional data

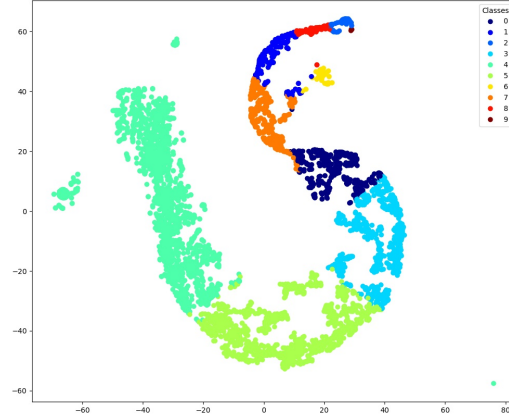


Figure 3: Visualization for kmeans clustering

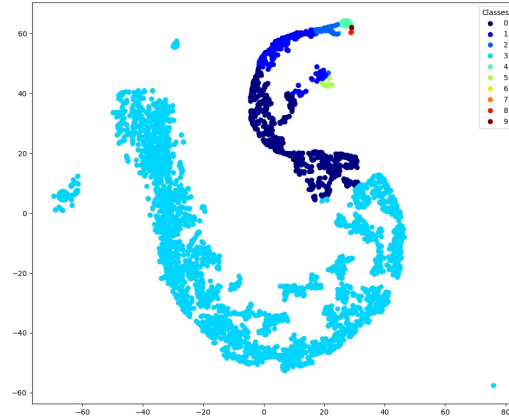


Figure 4: Visualization for kmeans++ clustering

4. For L1 loss, the kmeans++ clustering doesn't provide a better result in loss. But for L2 loss, we can clearly see that kmeans++ clustering reduces the loss sharply. In L2, initial cluster centroids which are as far away as possible could provide "good" initial points that make clustering easy to converge. Those centroids could move to the true centroids as they are re-computed as mean of points in cluster. But in L1, initial cluster centroids which are as far away as possible do not mean they are "good" initial points. Compared with L2 case, they might be too far away from most points and hard to move to the true centroids since the edge points have less contribution on the movement. In the loss term, L2 loss plays as a square term compared with L1.
5. $\mathcal{O}(t * k * n * d)$ where t is number of iterations, k is the number of clusters, n is number of data points and d is number of dimension.

```

1  import operator
2  import sys
3  from pyspark import SparkConf, SparkContext
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from scipy import linalg
7  from sklearn.manifold import TSNE
8
9  # Macros.
10 MAX_ITER = 20
11 DATA_PATH = "gs://big_data_storage/hw1/data.txt"
12 C1_PATH = "gs://big_data_storage/hw1/c1.txt"
13 C2_PATH = "gs://big_data_storage/hw1/c2.txt"
14 NORM = 1 # change to 2 for l2 loss
15
16
17 # Helper functions.
18 def closest(p, centroids, norm):
19     """
20     Compute closest centroid for a given point.
21     Args:
22         p (numpy.ndarray): input point
23         centroids (list): A list of centroids points
24         norm (int): 1 or 2
25     Returns:
26         int: The index of closest centroid.
27     """
28     closest_c = min([(i, linalg.norm(p - c, norm) ** norm)
29                     for i, c in enumerate(centroids)],
30                     key=operator.itemgetter(1))[0]
31     return closest_c
32
33
34 def loss(data, centroids, norm=2):
35     """
36
37     :param data: original data points in RDD
38     :param centroids: centroids used to calculate loss
39     :param norm: int 1 or 2
40     :return: the loss based on centroids
41     """
42     norms = data.map(lambda point: linalg.norm(np.subtract(centroids[closest(
43         point, centroids, norm=norm)], point), norm) ** norm)
44     cost = norms.reduce(lambda norm1, norm2: norm1 + norm2)

```

```

45     return cost
46
47
48 def plot_loss(loss1, loss2, img_path):
49     fig = plt.figure(figsize=(12, 10))
50     plt.plot(range(len(loss1)), loss1, "b", label="cost of c1 (kmean)")
51     plt.plot(range(len(loss2)), loss2, "r", label="cost of c2 (kmean++)")
52     plt.legend(loc="upper right", title="Classes")
53     plt.xlabel("Iteration")
54     plt.ylabel("Loss")
55     fig.savefig(img_path)
56
57
58 def plot_cluster(data, img_path):
59     index = data.map(lambda x: x[0]).collect()
60     points = data.map(lambda x: x[1][0]).collect()
61     points_embedded = TSNE(n_components=2, perplexity=50,
62                             random_state=100).fit_transform(points)
63     fig = plt.figure(figsize=(12, 10))
64     scatter = plt.scatter(points_embedded[:, 0], points_embedded[:, 1],
65                             marker='o', c=index, cmap='jet')
66     plt.legend(*scatter.legend_elements(),
67               loc="upper right", title="Classes")
68     fig.savefig(img_path)
69
70
71 # K-means clustering
72 def kmeans(data, centroids, norm=2):
73     """
74     Conduct k-means clustering given data and centroid.
75     This is the basic version of k-means, you might need more
76     code to record cluster assignment to plot TSNE, and more
77     data structure to record cost.
78     Args:
79         data (RDD): RDD of points
80         centroids (list): A list of centroids points
81         norm (int): 1 or 2
82     Returns:
83         RDD: assignment information of points, a RDD of (centroid, (point, 1))
84         centroids: a list of centroids
85         loss: a list of loss for each steps.
86     """
87     # iterative k-means
88     # k = len(centroids)
89

```

```

90     cost_ls = [loss(data=data, centroids=centroids, norm=norm)]
91
92     for _ in range(MAX_ITER):
93         # Transform each point to a combo of point, closest centroid, count=1
94         # point -> (closest_centroid, (point, 1))
95
96         # Re-compute cluster center
97         # For each cluster center (key), aggregate its values
98         # by summing up points and count
99
100        # Average the points for each centroid: divide sum of points by count
101
102        # Use collect() to turn RDD into list
103        combo = data.map(lambda point: (closest(
104            point, centroids, norm=norm), (point, 1)))
105        centroids = combo.reduceByKey(lambda combo1, combo2: (
106            np.add(combo1[0], combo2[0]),
107            combo1[1] + combo2[1])).map(lambda x: np.divide(x[1][0], x[1][1]))
108        centroids = centroids.collect()
109        cost = loss(data=data, centroids=centroids, norm=norm)
110        cost_ls.append(cost)
111    combo = data.map(lambda point: (closest(
112        point, centroids, norm=norm), (point, 1)))
113    return combo, centroids, cost_ls
114
115
116    def main():
117        # Spark settings
118        conf = SparkConf().setMaster("local").setAppName("kmeans")
119        sc = SparkContext(conf=conf)
120
121        # Load the data, cache this since we're accessing this each iteration
122        data = sc.textFile(DATA_PATH).map(
123            lambda line: np.array([float(x) for x in line.split(' ')]))
124        ).cache()
125        # Load the initial centroids c1, split into a list of np arrays
126        centroids1 = sc.textFile(C1_PATH).map(
127            lambda line: np.array([float(x) for x in line.split(' ')]))
128        ).collect()
129        # Load the initial centroids c2, split into a list of np arrays
130        centroids2 = sc.textFile(C2_PATH).map(
131            lambda line: np.array([float(x) for x in line.split(' ')]))
132        ).collect()
133        print("Run kmean clustering.")
134        combo1, centroids1, cost1 = kmeans(data=data, centroids=centroids1,

```

```

135                                     norm=NORM)
136
137     print("Run kmean++ clustering.")
138     combo2, centroids2, cost2 = kmeans(data=data, centroids=centroids2,
139                                     norm=NORM)
140     print("Plot loss.")
141     plot_loss(cost1, cost2, "loss-1%d.jpg" % NORM)
142
143     if NORM == 2:
144         print("For L2 norm, plot 2D clustering result.")
145         print("Plot kmean clustering result.")
146         plot_cluster(combo1, "kmeans-2Dpoints.jpg")
147         print("Plot kmean++ clustering result.")
148         plot_cluster(combo2, "kmeans+-2Dpoints.jpg")
149
150     print("Done!")
151
152
153 if __name__ == "__main__":
154     main()

```

Problem 2. Binary classification with Spark MLlib

For part II, I run the code on my own localhost for convenience.

1. As for the data loading part, please see the python script in detail.

```

1 def load(sqlContext, csv_path):
2     column_names = ["age", "workclass", "fnlwgt", "education", "education_num",
3                     "marital_status", "occupation", "relationship", "race",
4                     "sex",
5                     "capital_gain", "capital_loss", "hours_per_week",
6                     "native_country", "income"]
7     income_df = sqlContext.read.format("com.databricks.spark.csv").options(
8         header="false", inferSchema="true").load(csv_path)
9     for old_name, new_name in zip(income_df.columns, column_names):
10         income_df = income_df.withColumnRenamed(old_name, new_name)
11         income_df = income_df.dropna()
12     # print(income_df.dtypes)
13     print("Load csv file correctly.")
14     return income_df

```

2. As for the data preprocessing part, please see the python script in detail.

```

1 def preprocess(data_frame):
2     category_columns = ["workclass", "education", "marital_status",
3                          "occupation", "relationship", "race", "sex",
4                          "native_country", "income"]
5     index_columns = [col + "_index" for col in category_columns]
6     vec_columns = [col + "_vec" for col in category_columns]
7     for col in category_columns:
8         stringIndexer = StringIndexer(inputCol=col,
9                                         outputCol=col + "_index",
10                                         handleInvalid='error')
11         model = stringIndexer.fit(data_frame)
12         data_frame = model.transform(data_frame)
13         data_frame = data_frame.drop(col)
14     index_columns.pop(-1)
15     vec_columns.pop(-1)
16
17     ohe = OneHotEncoderEstimator(inputCols=index_columns,
18                                 outputCols=vec_columns)
19     ohe_model = ohe.fit(data_frame)
20     ohe_df = ohe_model.transform(data_frame)
21     ohe_df = ohe_df.drop(*index_columns)
22     # ohe_df.show()
23     cols = ohe_df.columns
24     cols.remove("income_index")
25     vector_assembler = VectorAssembler(inputCols=cols, outputCol="features")
26     vdata_frame = vector_assembler.transform(ohe_df)
27     vdata_frame = vdata_frame.drop(*cols)
28     # vdata_frame.show()
29     print("Preprocess input data correctly.")
30     return vdata_frame

```

3. As for the data modelling part, please see the python script in detail.

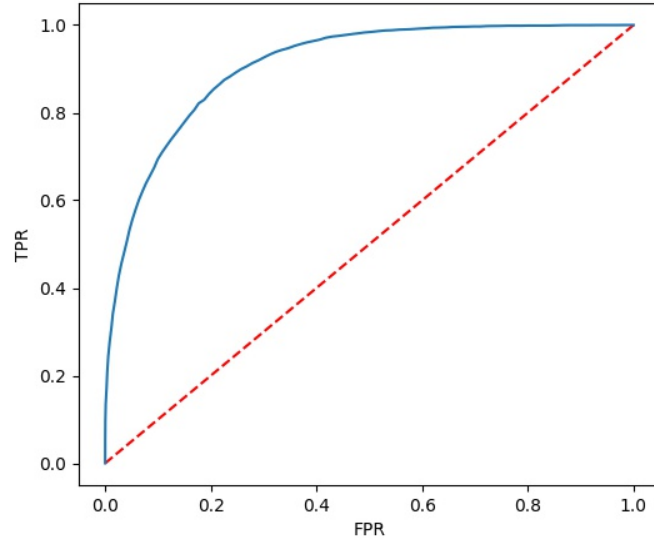


Figure 5: ROC curve of training process

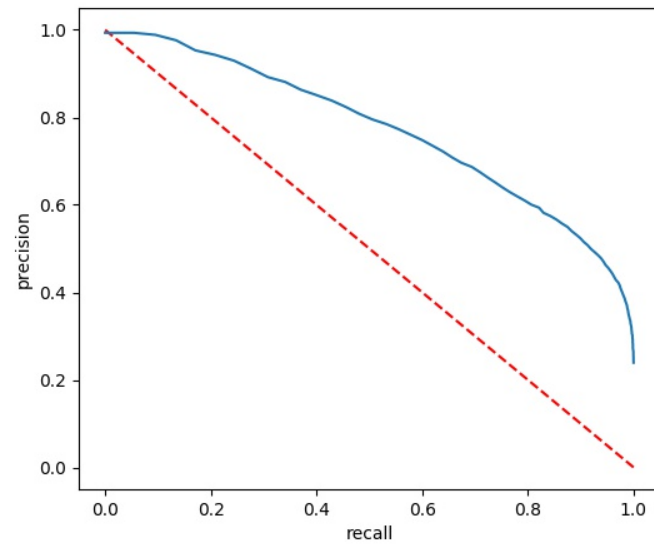


Figure 6: Precision-Recall curve of training process

4. Evaluation part:

-
- 1 Training:
 - 2 Training Area under ROC = 0.9084271679599486
 - 3 Training Accuracy = 0.8558104912864524

```

4
5 Evaluation:
6 Testing Area under ROC = 0.9076680514568874
7 Testing Accuracy = 0.8445952895196955
8 Testing Confusion Matrix:
9 [[6820.  528.]
10  [ 983. 1392.]]
11 Total cost 29.132921s
12 Done!

```

Here we can see value of area under ROC, accuracy, and confusion matrix on test data set.

Here is whole code for part II.

```

1 import numpy as np
2 import time
3 from pyspark import SparkConf, SparkContext
4 from pyspark.sql import SQLContext
5 from pyspark.ml.feature import OneHotEncoderEstimator, StringIndexer, \
6     VectorAssembler
7 from pyspark.ml.classification import LogisticRegression
8 from pyspark.mllib.classification import LogisticRegressionWithLBFGS
9 from pyspark.mllib.evaluation import BinaryClassificationMetrics, \
10     MulticlassMetrics
11 from matplotlib import pyplot as plt
12
13 CSV_PATH = "data/adult.data.csv"
14
15
16 def load(sqlContext, csv_path):
17     column_names = ["age", "workclass", "fnlwgt", "education", "education_num",
18                     "marital_status", "occupation", "relationship", "race",
19                     "sex",
20                     "capital_gain", "capital_loss", "hours_per_week",
21                     "native_country", "income"]
22     income_df = sqlContext.read.format("com.databricks.spark.csv").options(
23         header="false", inferSchema="true").load(csv_path)
24     for old_name, new_name in zip(income_df.columns, column_names):
25         income_df = income_df.withColumnRenamed(old_name, new_name)
26     income_df = income_df.dropna()
27     # print(income_df.dtypes)
28     print("Load csv file correctly.")
29     return income_df
30

```

```

31
32 def preprocess(data_frame):
33     category_columns = ["workclass", "education", "marital_status",
34                         "occupation", "relationship", "race", "sex",
35                         "native_country", "income"]
36     index_columns = [col + "_index" for col in category_columns]
37     vec_columns = [col + "_vec" for col in category_columns]
38     for col in category_columns:
39         stringIndexer = StringIndexer(inputCol=col,
40                                       outputCol=col + "_index",
41                                       handleInvalid='error')
42         model = stringIndexer.fit(data_frame)
43         data_frame = model.transform(data_frame)
44         data_frame = data_frame.drop(col)
45     index_columns.pop(-1)
46     vec_columns.pop(-1)
47
48     ohe = OneHotEncoderEstimator(inputCols=index_columns,
49                                 outputCols=vec_columns)
50     ohe_model = ohe.fit(data_frame)
51     ohe_df = ohe_model.transform(data_frame)
52     ohe_df = ohe_df.drop(*index_columns)
53     # ohe_df.show()
54     cols = ohe_df.columns
55     cols.remove("income_index")
56     vector_assembler = VectorAssembler(inputCols=cols, outputCol="features")
57     vdata_frame = vector_assembler.transform(ohe_df)
58     vdata_frame = vdata_frame.drop(*cols)
59     # vdata_frame.show()
60     print("Preprocess input data correctly.")
61     return vdata_frame
62
63
64 def plot_roc(FPR, TPR, img_path):
65     fig = plt.figure(figsize=(6, 5))
66     plt.plot([0, 1], [0, 1], 'r--')
67     plt.plot(FPR, TPR)
68     plt.xlabel('FPR')
69     plt.ylabel('TPR')
70     fig.savefig(img_path)
71
72
73 def plot_pr(recall, precision, img_path):
74     fig = plt.figure(figsize=(6, 5))
75     plt.plot([0, 1], [1, 0], 'r--')

```

```

76     plt.plot(recall, precision)
77     plt.xlabel('recall')
78     plt.ylabel('precision')
79     fig.savefig(img_path)
80
81
82 def main():
83     start = time.time()
84     conf = SparkConf().setMaster("local").setAppName("income")
85     sc = SparkContext(conf=conf)
86     sqlContext = SQLContext(sc)
87     income_df = load(sqlContext, csv_path=CSV_PATH)
88     # income_df.show()
89     # print(income_df.dtypes)
90     # print(income_df.count())
91
92     features_df = preprocess(data_frame=income_df)
93
94     # train, test split
95     train_df, test_df = features_df.randomSplit([7.0, 3.0], 100)
96
97     # logistic regression
98
99     income_lr = LogisticRegression(featuresCol="features",
100                                   labelCol="income_index",
101                                   regParam=0.0, elasticNetParam=0.0,
102                                   maxIter=200)
103     income_model = income_lr.fit(train_df)
104
105     # modeling
106     print("Training:")
107     training_summary = income_model.summary
108     training_FPR = training_summary.roc.select('FPR').collect()
109     training_TPR = training_summary.roc.select('TPR').collect()
110     plot_roc(training_FPR, training_TPR, "pic/training_roc.jpg")
111
112     training_recall = training_summary.pr.select('recall').collect()
113     training_precision = training_summary.pr.select('precision').collect()
114     # Area under ROC curve
115     print("Training Area under ROC = %s" % training_summary.areaUnderROC)
116     # accuracy
117     print("Training Accuracy = %s" % training_summary.accuracy)
118     plot_pr(training_recall, training_precision, "pic/training_pr.jpg")
119
120     # evaluation

```

```

121     print()
122     print("Evaluation:")
123     pred_df = income_model.transform(test_df).select("prediction",
124                                                       "income_index")
125     raw_pred_df = income_model.transform(test_df).select("probability",
126                                                         "income_index"
127                                                         ).rdd.map(
128         lambda l: (float(l[0][1]), l[1]))
129     metrics = BinaryClassificationMetrics(raw_pred_df)
130     # Area under ROC curve
131     print("Testing Area under ROC = %s" % metrics.areaUnderROC)
132     # accuracy
133     metrics = MulticlassMetrics(pred_df.rdd)
134     print("Testing Accuracy = %s" % metrics.accuracy)
135
136     # confusion matrix
137     print("Testing Confusion Matrix:")
138     print(metrics.confusionMatrix().toArray())
139     print("Total cost %fs" % (time.time() - start))
140     print("Done!")
141
142
143 if __name__ == '__main__':
144     main()

```
