STA323

Big Data Analysis Software and Application (Hadoop or Spark)

Report on Assignment 4

12112627 李乐平

Question 1. Predict the house prices

The house pricing data is attached. Process the data and split the train.csv to train data (`80%`) and validation data (`20%`). Tune model hyperparameters with validation data. Write the predictions of the house price in the test set to a file predictions.csv (with columns of `id` and `SalePrice`). You can use all or part of features in the data.

(1). [2 point] Predict with decision tree using Pyspark MLlib package.

Answer:

首先读入训练数据和测试数据，并临时拼接以统一转换格式。对于类别列，转换为独热向量。转换完成后得到格式化的训练集和测试集，然后使用 TrainValidationSplit 包装决策树回归器对训练集进行训练。最后对测试集进行回归。

```
data = spark.read.csv("./data/Q1_house_price_data/train.csv", header=True, inferSchema=True)
test_data_origin = spark.read.csv("./data/Q1_house_price_data/test.csv", header=True, schema =
data.schema)

data = data.fillna(0)
test_data = test_data_origin.fillna(0)

def transform(df, df_test):
    df_test = df_test.withColumn("SalePrice", lit(-1))
    df_copy = df.union(df_test)
    columns = []
    stringIndexers = []
    ohes = []
    for col in df.schema:
        if isinstance(col.dataType, StringType):
            columns.append(col.name + "OHE")
            stringIndexer = StringIndexer().setInputCol(col.name).setOutputCol(col.name +
"Ind").fit(df_copy)
            df_copy = stringIndexer.transform(df_copy)
            ohe = OneHotEncoder().setInputCol(col.name + "Ind").setOutputCol(col.name +
"OHE").fit(df_copy)
            df_copy = ohe.transform(df_copy)

            stringIndexers.append(stringIndexer)
            ohes.append(ohe)
        else:
            columns.append(col.name)

    df_test = df_copy.filter(df_copy.SalePrice == -1)
    df = df_copy.filter(df_copy.SalePrice != -1)

    return df, df_test, columns, stringIndexers, ohes

train_data, test_data, cols, stringIndexers, ohes = transform(data, test_data)

va = VectorAssembler().setInputCols(cols[1 : -1]).setOutputCol("feature")
dt = DecisionTreeRegressor(featuresCol = "feature", labelCol = cols[-1])
evaluator = RegressionEvaluator().setLabelCol("SalePrice")
grid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [5, 10, 15]) \
    .addGrid(dt.maxBins, [10, 20, 30]) \
    .addGrid(dt.minInstancesPerNode, [1, 5, 10]) \
    .build()
tvs = TrainValidationSplit(
    estimator = dt,
    estimatorParamMaps = grid,
    evaluator = evaluator,
    parallelism = 1,
```

```
    trainRatio = 0.8,
    seed=42
)
va_train = va.transform(train_data)
va_test = va.transform(test_data)
tvs_model = tvs.fit(va_train)
predictions = tvs_model.transform(va_test)
```

```
+----+------------------+
|  Id|        prediction|
+----+------------------+
|1970| 489036.8461538461|
|1977|          337900.0|
|1995|189361.68333333332|
|2008|          218236.0|
|2013|          184325.0|
+----+------------------+
```

(2). [4 points] Predict with XGBoost using pyspark platform.

Answer:

对于刚才处理过的数据导入 XGBoost 回归器，用类似的方法做 TrainValidationSplit，训练并预测即可。

```
import xgboost
from xgboost.spark import SparkXGBRegressor
xgb_regressor = SparkXGBRegressor(
    features_col = "feature",
    label_col = "SalePrice",
)
grid_xgb = ParamGridBuilder() \
    .addGrid(xgb_regressor.learning_rate, [0.1, 0.3, 0.5]) \
    .addGrid(xgb_regressor.max_depth, [6, 8, 10]) \
    .build()

tvs_xgb = TrainValidationSplit(
    estimator = xgb_regressor,
    estimatorParamMaps = grid_xgb,
    evaluator = evaluator,
    parallelism = 1,
    trainRatio = 0.8,
    seed=42
)
xgb_regressor_model = tvs_xgb.fit(va_train)
predict_xgb = xgb_regressor_model.transform(va_test)
predict_xgb.select("Id", "prediction").sample(False, 0.1).show(5)
```

```
+----+-------------+
|  Id|   prediction|
+----+-------------+
|1963| 98694.640625|
|1989|187828.671875|
|2054|81152.1640625|
|2055| 157643.28125|
|2059|  119607.3125|
+----+-------------+
```

(3). [4 points] Predict with LightGBM using pyspark platform.

Answer:

对于刚才处理过的数据导入 TrainValidationSplit 处理过的 LightGBM 回归器训练并预测即可。由于 LightGBM 对空值异常敏感，需要注意去除空值。

```
from synapse.ml.lightgbm import LightGBMRegressor

LC = LightGBMRegressor(
    featuresCol="feature",
    labelCol = "SalePrice"
)
```

```
grid_lgbm = ParamGridBuilder() \
    .addGrid(LC.alpha, [0.3, 0.5]) \
    .addGrid(LC.learningRate, [0.3, 0.5]) \
    .build()

tvs_LC = TrainValidationSplit(
    estimator = LC,
    estimatorParamMaps = grid_lgbm,
    evaluator = evaluator,
    parallelism = 1,
    trainRatio = 0.8,
    seed=42
)

LCmodel = tvs_LC.fit(va_train.select("feature", "SalePrice").filter(col("feature").isNotNull()))

pred_lgbm = LCmodel.transform(va_test)
pred_lgbm.select("Id", "prediction").sample(False, 0.1).show(5)
```

```
+----+------------------+
|  Id|        prediction|
+----+------------------+
|1961|  116094.404021644|
|1964|113483.04518325946|
|1967|301002.78605465195|
|1998| 344904.4351506987|
|2006|242136.72580701215|
+----+------------------+
```

Question 2. User analyses

The customer data is attached. Analyze the data to better understand customer behavior. Before analyses, you may need to observe the data and do some data cleaning.

(1). [5 points] Use Pyspark K-means to cluster the users into groups. You can use all features or part of the features for clustering. Also, you will decide the optimal number of clusters.

Answer:

首先去除 InvoiceNo 以 c 开头的列，去除有异常 ID 的列。选定股票代码、数量、单价等三个特征，利用先前的方法做向量化处理得到 feature 列。利用 K-means 分类器进行不同分类数的分类，用 silhouette score 得到最佳分类数。最佳分类数为 3。最后用最佳分类数再进行一次分类。

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import PCA

user_df = spark.read.csv("./data/Q2_customer_data.csv", header=True, inferSchema=True)
user_df = user_df.filter(~col("InvoiceNo").startswith("c"))\
                 .filter(col("CustomerID").rlike("^[0-9]+$"))
user_df = user_df.select("StockCode", "Quantity", "UnitPrice", "CustomerID")
selected_columns = ["StockCodeOHE", "Quantity", "UnitPrice", "CustomerID"]

def trans(df):
    df_copy = df.select("*")
    columns = []
    stringIndexers = []
    ohes = []
    for col in df.schema:
        if isinstance(col.dataType, StringType):
            columns.append(col.name + "OHE")
            stringIndexer = StringIndexer().setInputCol(col.name).setOutputCol(col.name +
"Ind").fit(df_copy)
            df_copy = stringIndexer.transform(df_copy)
            ohe = OneHotEncoder().setInputCol(col.name + "Ind").setOutputCol(col.name +
"OHE").fit(df_copy)
            df_copy = ohe.transform(df_copy)

            stringIndexers.append(stringIndexer)
            ohes.append(ohe)
        else:
```

```
            columns.append(col.name)

    return df_copy, columns, stringIndexers, ohes

user_df, user_cols, stringIndexers, ohes = trans(user_df)

user_df = user_df \
    .groupBy("CustomerID") \
    .agg(
        F.avg(col("Quantity")).alias("TotalQuantity"),
        F.avg(col("UnitPrice")).alias("TotalAmount"),
        F.max("StockCodeOHE").alias("MaxStockCodeOHE")
    )

assembler = VectorAssembler(inputCols=["TotalQuantity", "TotalAmount", "MaxStockCodeOHE"],
outputCol="features")
user_df = assembler.transform(user_df)

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(user_df)
user_df = scaler_model.transform(user_df)

evaluator = ClusteringEvaluator()

silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(featuresCol="scaledFeatures", k=k)
    model = kmeans.fit(user_df)
    predictions = model.transform(user_df)
    silhouette_score = evaluator.evaluate(predictions)
    silhouette_scores.append(silhouette_score)

optimal_k = np.argmax(silhouette_scores) + 2
k = optimal_k
kmeans = KMeans(featuresCol="scaledFeatures", k = k)
model = kmeans.fit(user_df)
predictions = model.transform(user_df)

print("Cluster Centers: ")
centers = model.clusterCenters()
for center in centers:
    print(center)

predictions.select("CustomerID", "prediction").show()
```

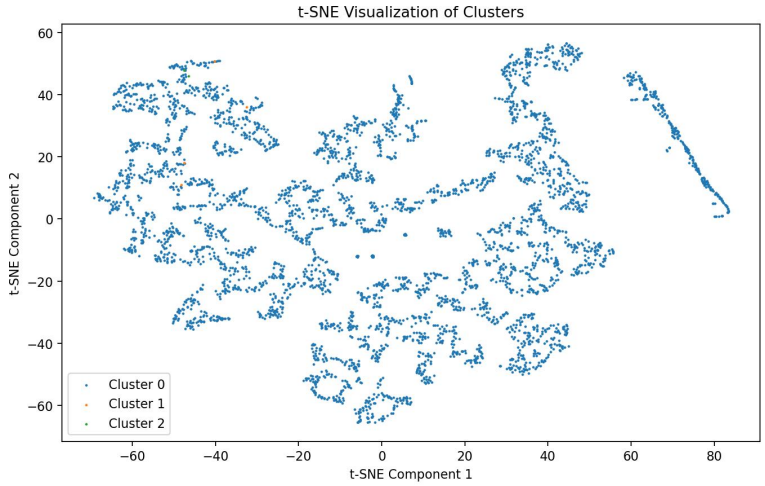
```
Cluster Centers:
[0.10512723 0.05508396 0.01514455 ... 0.         0.         0.01514455]
[0.00067605 0.02486922 0.         ... 0.         0.         0.        ]
[0.0077554  0.01681575 0.         ... 0.         0.         0.        ]
[Stage 636:>                                                        (0 + 1
+----------+----------+
|CustomerID|prediction|
+----------+----------+
|     12346|         0|
|     12347|         0|
|     12348|         0|
|     12349|         0|
|     12350|         0|
```

(2). [2 point] Visualize the clustering with t-SNE.

Answer:

```
pca = PCA(k = k, inputCol="scaledFeatures", outputCol="pcaFeatures")
pca_model = pca.fit(predictions)
predictions = pca_model.transform(predictions)

pred_data = predictions.select("pcaFeatures", "prediction").rdd.map(lambda x: (x.prediction,
x.pcaFeatures.toArray())).collect()
clusters, features = zip(*pred_data)
features = np.array(features)

tsne = TSNE(n_components = 2, random_state = 42)
tsne_results = tsne.fit_transform(features)

plt.figure(figsize=(10, 6), dpi = 160)
```

```
for cluster in range(k):
    cluster_indices = np.where(np.array(clusters) == cluster)[0]
    plt.scatter(tsne_results[cluster_indices, 0], tsne_results[cluster_indices, 1], label=f'Cluster
{cluster}', s = 1)

plt.title('t-SNE Visualization of Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()
```



(3). [3 points] Observe the customer clusters and give your explanations to the results.

Answer:

观察分类结果可以发现，绝大多数的客户都被归为了一类。其他两个类只有零星的几个点。这有可能是因为将股票代码展开为独热向量后导致空间维数太高，从而使特征无法很好地提取导致的。

观察聚类中心发现，后两类的平均交易数量远小于第一类，这也可能昭示着这些用户进行了某些异常的交易行为，需要加强关注。