Homework 6: Home Price Prediction with Regression

0 作业说明

In this homework, you are given a house dataset.

It is your job to predict the sales price for each house. For each Id in the test set, you must predict the value of the SalePrice variable.

Please load data into Spark and use Spark MLib to process them. Since there are too many attributes, you are asked to first perform the PCA dimensional deduction and only keep the most 10 important attributes. For the 10 attributes, build a linear regression model and a decision tree model (using the Mlib) for the house prices. And the precision is evaluated by the Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price.

In the report, please explain the details of your experiments and compare the performance of different models

1 PCA dimensional deduction

1.1 Data processing

1.1.1 读取文件

首先从文件中读取所需的csv文件，并去除表示Id的列：

```
train.drop(['Id'], axis=1, inplace=True)
test.drop(['Id'], axis=1, inplace=True)
```

通过直方图可以看出`SalePrice`属性并不是分布均匀的，而是左倾的，因此需要对`SalePrice`取对数处理：

```
train = train[train.GrLivArea < 4500]
train.reset_index(drop=True, inplace=True)
train["SalePrice"] = np.log1p(train["SalePrice"])
y = train['SalePrice'].reset_index(drop=True)
```

经过处理后，`SalePrice`的分布变得更加均匀，我们将处理的数值用标签`y`来表示。

1.1.2 处理噪点和缺失值

将训练集和测试集合并，处理其中的缺失值：

```
train_features = train.drop(['SalePrice'], axis=1)
test_features = test
features = pd.concat([train_features,
test_features]).reset_index(drop=True)
```

对于`MSSubClass`，`YrSold`，`MoSold`属性，这些属性本来应该是类别值，因此将它们的类型转化为字符串：

```
features['MSSubClass'] = features['MSSubClass'].apply(str)
features['YrSold'] = features['YrSold'].astype(str)
features['MoSold'] = features['MoSold'].astype(str)
```

对于某些缺失的属性，用合适的值来填充：

```
features['Functional'] = features['Functional'].fillna('Typ')
features['Electrical'] = features['Electrical'].fillna("SBrkr")
features['KitchenQual'] = features['KitchenQual'].fillna("TA")
features["PoolQC"] = features["PoolQC"].fillna("None")
features['Exterior1st'] =
features['Exterior1st'].fillna(features['Exterior1st'].mode()[0])
features['Exterior2nd'] =
features['Exterior2nd'].fillna(features['Exterior2nd'].mode()[0])
features['SaleType'] =
features['SaleType'].fillna(features['SaleType'].mode()[0])
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    features[col] = features[col].fillna(0)

for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    features[col] = features[col].fillna('None')
```

```
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
'BsmtFinType2'):
    features[col] = features[col].fillna('None')

features['MSZoning'] = features.groupby('MSSubClass')
['MSZoning'].transform(lambda x: x.fillna(x.mode()[0]))

objects = []
for i in features.columns:
    if features[i].dtype == object:
        objects.append(i)
features.update(features[objects].fillna('None'))
print(objects)

features['LotFrontage'] = features.groupby('Neighborhood')
['LotFrontage'].transform(lambda x: x.fillna(x.median()))

numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32',
'float64']
numerics = []
for i in features.columns:
    if features[i].dtype in numeric_dtypes:
        numerics.append(i)
features.update(features[numerics].fillna(0))
numerics[1:10]
```

接下来对分布不均匀的属性值进行处理：

```
numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32',
'float64']
numerics2 = []
for i in features.columns:
    if features[i].dtype in numeric_dtypes:
        numerics2.append(i)
skew_features = features[numerics2].apply(lambda x:
skew(x)).sort_values(ascending=False)

high_skew = skew_features[skew_features > 0.5]
skew_index = high_skew.index

for i in skew_index:
    features[i] = boxcox1p(features[i], boxcox_normmax(features[i] + 1))
```

1.2 Feature engineering

1.2.1 添加或移除属性

移除用处不大的属性:

```
features = features.drop(['Utilities', 'Street', 'PoolQC',], axis=1)
```

添加新的属性:

```
features['YrBltAndRemod']=features['YearBuilt']+features['YearRemodAdd']
features['TotalSF']=features['TotalBsmtSF'] + features['1stFlrSF'] +
features['2ndFlrSF']

features['Total_sqr_footage'] = (features['BsmtFinSF1'] +
features['BsmtFinSF2'] +
                                 features['1stFlrSF'] +
features['2ndFlrSF'])

features['Total_Bathrooms'] = (features['FullBath'] + (0.5 *
features['HalfBath']) +
                               features['BsmtFullBath'] + (0.5 *
features['BsmtHalfBath']))
```

```
features['Total_porch_sf'] = (features['OpenPorchSF'] +
features['3SsnPorch'] +
features['EnclosedPorch'] + features['ScreenPorch'] +
                              features['WoodDeckSF'])

features['haspool'] = features['PoolArea'].apply(lambda x: 1 if x > 0 else
0)
features['has2ndfloor'] = features['2ndFlrSF'].apply(lambda x: 1 if x > 0
else 0)
features['hasgarage'] = features['GarageArea'].apply(lambda x: 1 if x > 0
else 0)
features['hasbsmt'] = features['TotalBsmtSF'].apply(lambda x: 1 if x > 0
else 0)
features['hasfireplace'] = features['Fireplaces'].apply(lambda x: 1 if x >
0 else 0)
```

使用pandas的get_dummies方法将类型属性转化为数值属性：

```
final_features = pd.get_dummies(features).reset_index(drop=True)
```

将数据集重新划分为训练集和测试集：

```
X = final_features.iloc[:len(y), :]
X_sub = final_features.iloc[len(y):, :]
```

通过绘制图像，去除训练集中偏离过大的点，并drop掉某种取值占全部记录99.94%以上的属性：

```
outliers = [30, 88, 462, 631, 1322]
X = X.drop(X.index[outliers])
y = y.drop(y.index[outliers])

overfit = []
for i in X.columns:
    counts = X[i].value_counts()
    zeros = counts.iloc[0]
    if zeros / len(X) * 100 > 99.94:
        overfit.append(i)

overfit = list(overfit)
X = X.drop(overfit, axis=1)
```

```
X_sub = X_sub.drop(overfit, axis=1)
overfit
```

1.2.2 PCA

对训练集进行PCA降维，接着对测试集用刚刚得到的模型进行同样的处理：

```
from sklearn.decomposition import PCA
pca = PCA(n_components=10)
X = pca.fit_transform(X)
X_sub = pca.transform(X_sub)
```

训练集降维到10个维度后的结果如下：

A	B	C	D	E	F	G	H	I	J	K
	0	1	2	3	4	5	6	7	8	9
0	1051.628	-54.8378	-83.2351	57.96432	-94.09	-48.4191	-43.5882	-1.84697	1.983916	6.674349
1	-606.944	-162.419	61.28598	-55.3017	-101.535	1.904094	39.91846	-6.2484	-8.25094	-14.7995
2	1076.848	-73.4942	-24.8214	85.19175	-37.1572	-36.7718	-44.8186	-1.43615	-7.36897	2.699076
3	775.1191	-84.9423	-99.6091	157.9212	20.75207	103.2477	-3.6218	18.9445	-9.63164	-1.22677
4	1559.444	-120.99	180.8283	199.0728	-45.6364	-9.19732	10.88516	-4.95586	-12.5466	4.374016
5	422.7106	-68.5523	-142.697	44.40537	-105.072	-41.8083	-12.2107	-0.32895	6.935612	-6.68931
6	-470.237	-265.192	322.1316	-18.9622	-125.316	-33.0718	22.80961	3.629741	-3.97861	1.676921
7	1397.631	0.878466	9.541513	-75.5407	-113.303	12.73762	47.85632	-13.2328	-4.13041	14.38091
8	765.1687	13.32562	-61.2278	-37.9906	105.451	86.41644	25.02321	-11.4593	-3.06451	-7.80803
9	-698.203	-34.1956	-173.084	-192.662	-113.98	53.47606	-18.9083	-8.87518	0.812326	-5.05219
10	-670.29	-115.215	-75.3455	-57.0036	-109.892	25.23354	-30.271	-8.65664	2.112138	-9.70139
11	1780.091	-81.9744	148.0472	103.6993	-118.135	-26.9106	-8.5496	-2.87805	1.992903	-0.1069
12	-715.884	-96.7181	-144.696	-49.533	-91.8838	14.07223	37.84689	-10.2613	-6.72379	2.590012
13	-633.322	-323.891	336.2018	183.2213	181.6728	-25.459	13.10449	-1.28818	-2.03258	6.296031
14	-635.763	-114.886	11.49519	-146.59	-59.5111	34.95631	-4.2626	-10.7245	-14.0269	20.38696
15	-805.488	-189.17	-87.333	142.9849	116.7505	34.09914	17.17293	38.02696	-4.1545	3.769222
16	-699.821	-148.008	-47.3095	28.82378	-40.4617	20.45598	-30.9564	-9.59646	-12.527	4.536239
17	-1041.49	-90.3755	-557.649	376.5921	27.02809	-9.00288	-8.9105	-12.5905	13.05476	-0.04876
18	-659.473	-217.179	43.52494	77.93683	-40.3409	-47.7321	-34.3787	6.850898	-9.22056	-6.02488
19	-711.54	-83.7116	-115.007	-135.226	-33.3726	22.33148	-26.7158	-5.71157	-10.9017	-8.62288
20	1842.194	-109.467	201.4064	183.0842	132.9532	-24.552	26.0377	-7.62864	-2.34707	8.453626

2 Regression Model

2.1 Task Analysis

使用Spark的Mllib库中的LR和DT模型，利用PCA降维后的数据进行训练，最终利用训练得到的模型预测test表中缺失的房价部分，使用RMSE作为评价指标，比较两个模型的性能差距。主要参考了Spark Mllib官方样例。

2.2 Data process

我们需要将数据格式修改为如下格式：

```
label,attr1 attr2 attr3 ...
```

读入数据：

```
import org.apache.spark.ml.linalg.Vectors
import org.apache.spark.sql.Row
import spark.implicits._
import org.apache.spark.ml.regression.LinearRegression
import org.apache.spark.sql.SparkSession
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils


val data_path = "./data"
val data = spark.read.text(data_path)
//读入数据，“，”前作为label,“,”后的属性根据“ ”分割
val training = data.map {
  case Row(line: String) =>
    var arr = line.split(',')
    (arr(0).toDouble, Vectors.dense(arr(1).split(' ').map(_.toDouble)))
}.toDF("label", "features")
```

同时也尝试了另一种libsvm格式：

```
label 1:value1 2:value2
```

```
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.tree.model.DecisionTreeModel
import org.apache.spark.mllib.util.MLUtils
// libsvm:label index1:value1 index2:value2 index3:value3 ...
//12 1:121 2:0.2112 ...升序排列
// Load and parse the data file.
// Cache the data since we will use it again to compute training error.
val data = MLUtils.loadLibSVMFile(sc, "./libsvmdata").cache()
```

2.1 Logical Regression

代码：

```
//建立模型，setRegParam:设置正则化项系数，setElasticNetParam正则化范式比
 (L1(Lasso)和L2(Ridge)。L1一般用于特征的稀疏化，L2一般用于防止过拟合)
val linearRegressionModel = new LinearRegression()
  .setMaxIter(200)
  .setRegParam(0.2)
  .setElasticNetParam(0.05)

//开始训练
val model = linearRegressionModel.fit(training)

//输出训练结果，各属性的权重
println(s"Coefficients: ${model.coefficients} Intercept:
${model.intercept}")

//输出训练过程的一些信息
val modelSummary = model.summary
println(s"numIterations: ${modelSummary.totalIterations}")
println(s"RMSE: ${modelSummary.rootMeanSquaredError}")
modelSummary.predictions.show()
```

运行结果：

```
scala> println(s"RMSE: ${modelSummary.rootMeanSquaredError}")
RMSE: 38662.87499328574

scala> println(s"r2: ${modelSummary.r2}")
r2: 0.7627126775297407

scala> modelSummary.predictions.show()
+--------+--------------------+------------------+
|   label|            features|        prediction|
+--------+--------------------+------------------+
|208500.0|[1051.628432,-54....| 231313.5587684692|
|181500.0|[-606.9437734,-16...|177125.63458597162|
|223500.0|[1076.847725,-73....| 232675.5330030265|
|140000.0|[775.1191252,-84....|164410.16102530225|
|250000.0|[1559.443568,-120...| 299633.7889502668|
|143000.0|[422.7106284,-68....|188824.88428051627|
|307000.0|[-470.2373823,-26...| 271262.4509548309|
|200000.0|[1397.631168,0.87...| 254392.4641387563|
|129900.0|[765.168670899999...|153858.02067576922|
|118000.0|[-698.20338459999...| 99332.41747345532|
|129500.0|[-670.2900365,-11...| 133184.4510166205|
|345000.0|[1780.09081700000...| 306592.0633001253|
|144000.0|[-715.8840175,-96...| 132757.2843930832|
|279500.0|[-633.32214579999...| 248405.8591385332|
|157000.0|[-635.7630073,-11...| 164898.8444083497|
|132000.0|[-805.4876367,-18...| 135464.5439140512|
|149000.0|[-699.82134400000...|144274.48779844938|
| 90000.0|[-1041.489553,-90...| 34754.67332634117|
|159000.0|[-659.4728134,-21...|187344.56223870267|
|139000.0|[-711.5401744,-83...|112215.35595472218|
+--------+--------------------+------------------+
only showing top 20 rows
```

获得参数：

[38.613836412356484,-51.523550792333104,216.31558977106528,26.262472009856885,-88.8468093375649,-311.1558912069379,124.04560893777025,202.656545037169,235.6719856487446,746.5914393447789]

2.2 Decision Tree

代码：

```
// Train a DecisionTree model-Since we want a regression task instead of a classification model.
//  Empty categoricalFeaturesInfo indicates all features are continuous.
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "variance"
val maxDepth = 5
val maxBins = 32
```

```
val model = DecisionTree.trainRegressor(trainingData,
categoricalFeaturesInfo, impurity,
  maxDepth, maxBins)
// Evaluate model on test instances and compute test error
val labelsAndPredictions = testData.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val testMSE = labelsAndPredictions.map{ case (v, p) => math.pow(v - p, 2)
}.mean()
//RMSE (Root Mean Squard Error)
//模型单位为万元，MSE表现出的差值是千万元，开根后误差的结果和数据是一个级别的，看起来比较正常
val RMSE= math.sqrt(testMSE)
println("Root Mean Squared Error = " + RMSE)
println("Learned regression tree model:\n" + model.toDebugString)
```

运行结果：

```
scala> println("Root Mean Squared Error = " + RMSE)
Root Mean Squared Error = 49286.8655706226

scala> println("Learned regression tree model:\n" + model.toDebugString)
Learned regression tree model:
DecisionTreeModel regressor of depth 5 with 63 nodes
  If (feature 2 <= 213.91469424999997)
   If (feature 0 <= 911.7122502499999)
    If (feature 0 <= -743.2343985499999)
     If (feature 1 <= -58.313174945)
      If (feature 5 <= 43.694424275)
       Predict: 126535.29333333333
      Else (feature 5 > 43.694424275)
       Predict: 106648.75
     Else (feature 1 > -58.313174945)
      If (feature 3 <= 24.735496425)
       Predict: 104502.0
      Else (feature 3 > 24.735496425)
       Predict: 83716.48387096774
    Else (feature 0 > -743.2343985499999)
     If (feature 2 <= 30.763199895)
      If (feature 0 <= 631.29282345)
       Predict: 138383.8432055749
      Else (feature 0 > 631.29282345)
       Predict: 174371.88235294117
     Else (feature 2 > 30.763199895)
      If (feature 5 <= 37.943170235)
       Predict: 191433.4960629921
      Else (feature 5 > 37.943170235)
       Predict: 147206.4516129032
   Else (feature 0 > 911.7122502499999)
    If (feature 2 <= 30.763199895)
     If (feature 5 <= 21.855451895)
      If (feature 7 <= -15.440891615)
       Predict: 154817.66666666666
      Else (feature 7 > -15.440891615)
       Predict: 215726.83823529413
     Else (feature 5 > 21.855451895)
      If (feature 9 <= -5.895590142)
```

```
     If (feature 9 <= -5.895590142)
      Predict: 212733.33333333334
     Else (feature 9 > -5.895590142)
      Predict: 159353.34615384616
   Else (feature 2 > 30.763199895)
    If (feature 1 <= 29.04796851)
     If (feature 0 <= 1938.6513694999999)
      Predict: 258525.16129032258
     Else (feature 0 > 1938.6513694999999)
      Predict: 323128.8
    Else (feature 1 > 29.04796851)
     If (feature 3 <= -26.337818910000003)
      Predict: 113000.0
     Else (feature 3 > -26.337818910000003)
      Predict: 160000.0
 Else (feature 2 > 213.91469424999997)
  If (feature 0 <= -540.32929015)
   If (feature 0 <= -743.2343985499999)
    If (feature 6 <= -21.83691679)
     If (feature 8 <= -9.242429489500001)
      Predict: 108833.33333333333
     Else (feature 8 > -9.242429489500001)
      Predict: 138500.0
    Else (feature 6 > -21.83691679)
     If (feature 6 <= -7.932696878)
      Predict: 87175.0
     Else (feature 6 > -7.932696878)
      Predict: 112250.0
   Else (feature 0 > -743.2343985499999)
    If (feature 5 <= 7.9684938075)
     If (feature 2 <= 279.19972279999996)
      Predict: 212230.92307692306
     Else (feature 2 > 279.19972279999996)
      Predict: 247858.46666666667
    Else (feature 5 > 7.9684938075)
     If (feature 4 <= 27.906270284999998)
      Predict: 200180.0
     Else (feature 4 > 27.906270284999998)
      Predict: 159200.5
  Else (feature 0 > -540.32929015)
```

```
      Predict: 200180.0
     Else (feature 4 > 27.906270284999998)
      Predict: 159200.5
  Else (feature 0 > -540.32929015)
   If (feature 2 <= 469.15040419999997)
    If (feature 5 <= 21.855451895)
     If (feature 2 <= 388.45985944999995)
      Predict: 294468.12727272726
     Else (feature 2 > 388.45985944999995)
      Predict: 360953.53846153844
    Else (feature 5 > 21.855451895)
     If (feature 0 <= 1292.31167)
      Predict: 195101.7142857143
     Else (feature 0 > 1292.31167)
      Predict: 274555.55555555556
   Else (feature 2 > 469.15040419999997)
    If (feature 0 <= 1292.31167)
     If (feature 6 <= -44.0181032)
      Predict: 515777.75
     Else (feature 6 > -44.0181032)
      Predict: 353113.2380952381
    Else (feature 0 > 1292.31167)
     If (feature 1 <= -92.20168225)
      Predict: 492644.3333333333
     Else (feature 1 > -92.20168225)
      Predict: 685000.0


scala>
```

获得参数：

参数即如上。

预测结果：

```
 //预测TEST数据
val input = MLUtils.loadLibSVMFile(sc, "./libsvmTestData").cache()

val predictResult = input.map { point =>
    val prediction = model.predict(point.features)
    (point.label, prediction)
}
predictResult.collect()
predictResult.saveAsTextFile("./DTPredicionData")
```

预测结果被存储到了/root/DTPredicionData

```
root@hadoop1:~/DTPredicionData# ls
part-00000  part-00001  _SUCCESS
root@hadoop1:~/DTPredicionData#
```

3 Model comparasion

根据预测结果显示，decision tree model表现正常，预测结果为训练数据中出现的数据；linear regression model预测结果中出现了负值。因此，linear regression发生了过拟合，原因可能有：1. 前期筛选输入变量不合理，存在噪声数据：样本数量少或者抽样方法错误。2.自变量和因变量之间不存在明显的线性关系。3. 训练集的数据规模太小，没有达到深度学习的要求。

决策树模型相比于使用神经网络的模型，是白盒模型。决策树的优点在于需要相对较少的数据准备。决策树的缺点在于：1. 模型容易过拟合。修剪机制，设置叶子节点所需的最小样本数目或设置数的最大深度是避免决策树过拟合的必要条件。2. 决策树可能不稳定，因为数据中的小变化可能导致生成完全不同的树，不能保证返回全局最优决策树，可以通过训练多个决策树减轻这种情况。

线性回归假设自变量和因变量之间的关系是线性相关的，否则要经过一些变换。线性回归产生过拟合产生原因：样本里噪声数据干扰过大，模型过分记住了噪声特征，反而忽略了真实的输入输出间的关系。如果数据量太少，本质上是一个浅度学习的模型。