DecisionTree

2020 年 12 月 16 日

```
[ ]: ## 读取数据
```

读取 HDFS 上的 train.tsv 文件并查看数据项数

```
[1]: row_df = sqlContext.read.format("csv")\
     .option("header", "true")\
     .option("delimiter", "\t")\
     .load("/input/mllib/train.tsv")
     print (row_df.count())
```

```
7395
```

查看 Schema

```
[2]: row_df.printSchema()
```

```
root
 |-- url: string (nullable = true)
 |-- urlid: string (nullable = true)
 |-- boilerplate: string (nullable = true)
 |-- alchemy_category: string (nullable = true)
 |-- alchemy_category_score: string (nullable = true)
 |-- avglinksize: string (nullable = true)
 |-- commonlinkratio_1: string (nullable = true)
 |-- commonlinkratio_2: string (nullable = true)
 |-- commonlinkratio_3: string (nullable = true)
 |-- commonlinkratio_4: string (nullable = true)
 |-- compression_ratio: string (nullable = true)
 |-- embed_ratio: string (nullable = true)
 |-- framebased: string (nullable = true)
 |-- frameTagRatio: string (nullable = true)
 |-- hasDomainLink: string (nullable = true)
 |-- html_ratio: string (nullable = true)
 |-- image_ratio: string (nullable = true)
 |-- is_news: string (nullable = true)
 |-- lengthyLinkDomain: string (nullable = true)
 |-- linkwordscore: string (nullable = true)
 |-- news_front_page: string (nullable = true)
 |-- non_markup_alphanum_characters: string (nullable = true)
 |-- numberOfLinks: string (nullable = true)
 |-- numwords_in_url: string (nullable = true)
 |-- parametrizedLinkRatio: string (nullable = true)
 |-- spelling_errors_ratio: string (nullable = true)
 |-- label: string (nullable = true)
```

使用 select 选取要查看的字段，然后查看前 10 项数据

[3]:
```
row_df.
↪select('url','alchemy_category','alchemy_category_score','is_news','label').
↪show(10)
```

```
+--------------------+------------------+----------------------+-------+-----+
|                 url|  alchemy_category|alchemy_category_score|is_news|label|
+--------------------+------------------+----------------------+-------+-----+
|http://www.bloomb...|          business|              0.789131|      1|    0|
|http://www.popsci...|        recreation|              0.574147|      1|    1|
|http://www.menshe...|            health|              0.996526|      1|    1|
|http://www.dumbli...|            health|              0.801248|      1|    1|
|http://bleacherre...|            sports|              0.719157|      1|    0|
|http://www.conven...|                 ?|                     ?|      ?|    0|
|http://gofashionl...|arts_entertainment|               0.22111|      1|    1|
|http://www.inside...|                 ?|                     ?|      ?|    0|
|http://www.valetm...|                 ?|                     ?|      1|    1|
|http://www.howswe...|                 ?|                     ?|      ?|    1|
+--------------------+------------------+----------------------+-------+-----+
only showing top 10 rows
```

编写 DataFrames UDF 用户自定义函数，将数据中的？转换为 0

[4]:
```
from pyspark.sql.functions import udf
def replace_question(x):
    return ("0" if x=="?" else x)
replace_question= udf(replace_question)
```

导入 col 模块及 pyspark.sql.types 模块，后续可以使用 col 模块读取字段数据，使用 pyspark.sql.types 模块转换数据类型

[5]:
```
from pyspark.sql.functions import col
import pyspark.sql.types
```

使用 replace_question UDF 用户自定义函数，将 row_df DataFrame 第 4 个字段至最后一个字段转换为 double。其中，最后一个字段为 label，其余是 feature。

[6]:
```
df= row_df.select(
   ['url','alchemy_category' ]+
   [replace_question(col(column)).cast("double").alias(column) for column in␣
 ↪row_df.columns[4:] ] )
```

说明: - 用 row_df.select 选取字段

- 选取字段 [‘url’ , ‘alchemy_category’], 不需要转换
- for column in row_df.columns[4:] 读取第 4 个字段至最后一个字段
- col(column) 读取字段数据并调用 replace_question 自定义函数删除问号 “？”
- .cast(“double”) 转换为 double
- .alias(column) 把别名设置为原来的字段名

查看使用 replace_question UDF 转换后的字段

[7]:
```
print (df.printSchema())
```

```
root
 |-- url: string (nullable = true)
 |-- alchemy_category: string (nullable = true)
 |-- alchemy_category_score: double (nullable = true)
 |-- avglinksize: double (nullable = true)
 |-- commonlinkratio_1: double (nullable = true)
 |-- commonlinkratio_2: double (nullable = true)
 |-- commonlinkratio_3: double (nullable = true)
 |-- commonlinkratio_4: double (nullable = true)
 |-- compression_ratio: double (nullable = true)
 |-- embed_ratio: double (nullable = true)
 |-- framebased: double (nullable = true)
 |-- frameTagRatio: double (nullable = true)
 |-- hasDomainLink: double (nullable = true)
 |-- html_ratio: double (nullable = true)
 |-- image_ratio: double (nullable = true)
 |-- is_news: double (nullable = true)
 |-- lengthyLinkDomain: double (nullable = true)
 |-- linkwordscore: double (nullable = true)
 |-- news_front_page: double (nullable = true)
 |-- non_markup_alphanum_characters: double (nullable = true)
 |-- numberOfLinks: double (nullable = true)
 |-- numwords_in_url: double (nullable = true)
 |-- parametrizedLinkRatio: double (nullable = true)
 |-- spelling_errors_ratio: double (nullable = true)
 |-- label: double (nullable = true)

None
```

使用 df.select 查看结果，我们会发现之前字段的问号都转换为了 0

```
[8]: df.select('url','alchemy_category','alchemy_category_score','is_news','label').
      ↪show(10)
```

```
+--------------------+------------------+----------------------+-------+-----+
|                 url|  alchemy_category|alchemy_category_score|is_news|label|
+--------------------+------------------+----------------------+-------+-----+
|http://www.bloomb...|          business|              0.789131|    1.0|  0.0|
|http://www.popsci...|        recreation|              0.574147|    1.0|  1.0|
|http://www.menshe...|            health|              0.996526|    1.0|  1.0|
|http://www.dumbli...|            health|              0.801248|    1.0|  1.0|
|http://bleacherre...|            sports|              0.719157|    1.0|  0.0|
|http://www.conven...|                 ?|                   0.0|    0.0|  0.0|
|http://gofashionl…|arts_entertainment|             0.22111|     1.0|   1.0|
|http://www.inside…|                 ?|                 0.0|     0.0|   0.0|
|http://www.valetm…|                 ?|                 0.0|     1.0|   1.0|
|http://www.howswe…|                 ?|                 0.0|     0.0|   1.0|
+--------------------+------------------+----------------------+-------+-----+
only showing top 10 rows
```

使用 randomSplit 将数据按照 7:3 的比例分成 train_df（训练数据）与 test_df（测试数据），并且.cache() 暂存在内存中，加快后续程序运行的速度。

[9]:
```python
train_df, test_df = df.randomSplit([0.7, 0.3])
train_df.cache()
test_df.cache()
```

[9]:
```
DataFrame[url: string, alchemy_category: string, alchemy_category_score: double,
avglinksize: double, commonlinkratio_1: double, commonlinkratio_2: double,
commonlinkratio_3: double, commonlinkratio_4: double, compression_ratio: double,
embed_ratio: double, framebased: double, frameTagRatio: double, hasDomainLink:
double, html_ratio: double, image_ratio: double, is_news: double,
lengthyLinkDomain: double, linkwordscore: double, news_front_page: double,
non_markup_alphanum_characters: double, numberOfLinks: double, numwords_in_url:
double, parametrizedLinkRatio: double, spelling_errors_ratio: double, label:
double]
```

0.1 StringIndexer

[10]:
```python
from pyspark.ml.feature import  StringIndexer
```

创建 StringIndexer

[11]:
```python
categoryIndexer = StringIndexer(
                           inputCol='alchemy_category',
                           outputCol="alchemy_category_Index")
```

stringIndexer 使用 fit 方法生成 “Transformer”

[12]:
```python
categoryTransformer=categoryIndexer.fit(df)
```

查看 categoryTransformer 的内容，categoryTransformer 的 label 属性其实就是网页分类的字典

[13]:
```
for i in range(0,len(categoryTransformer.labels)):
    print (str(i)+':'+categoryTransformer.labels[i])
```

```
0:?
1:recreation
2:arts_entertainment
3:business
4:health
5:sports
6:culture_politics
7:computer_internet
8:science_technology
9:gaming
10:religion
11:law_crime
12:unknown
13:weather
```

使用 categpryTransformer 将所有 df 转换为 df1

[14]:
```
df1=categoryTransformer.transform(df)
```

查看转换后的 df1 字段

[15]:
```
print (df1.columns)
```

```
['url', 'alchemy_category', 'alchemy_category_score', 'avglinksize',
'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkratio_3',
'commonlinkratio_4', 'compression_ratio', 'embed_ratio', 'framebased',
'frameTagRatio', 'hasDomainLink', 'html_ratio', 'image_ratio', 'is_news',
'lengthyLinkDomain', 'linkwordscore', 'news_front_page',
'non_markup_alphanum_characters', 'numberOfLinks', 'numwords_in_url',
'parametrizedLinkRatio', 'spelling_errors_ratio', 'label',
'alchemy_category_Index']
```

查看转换后的结果

[16]:
```
df1.select("alchemy_category","alchemy_category_Index").show(10)
```

```
+------------------+----------------------+
|  alchemy_category|alchemy_category_Index|
+------------------+----------------------+
|          business|                   3.0|
|        recreation|                   1.0|
|            health|                   4.0|
|            health|                   4.0|
|            sports|                   5.0|
|                 ?|                   0.0|
|arts_entertainment|                   2.0|
|                 ?|                   0.0|
|                 ?|                   0.0|
|                 ?|                   0.0|
+------------------+----------------------+
only showing top 10 rows
```

0.2 OneHotEncoder

导入 OneHotEncoder 模块

[17]:
```
from pyspark.ml.feature import  OneHotEncoder
```

创建 OneHotEncoder

[18]:
```
encoder = OneHotEncoder(dropLast=False,
                            inputCol='alchemy_category_Index',
                            outputCol="alchemy_category_IndexVec")
```

OneHotEncoder 使用 transform 转换，结果是 df2，我们可以使用下列指令查看字段

[19]:
```
df2=encoder.transform(df1)
print (df2.columns)
```

```
['url', 'alchemy_category', 'alchemy_category_score', 'avglinksize',
'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkratio_3',
'commonlinkratio_4', 'compression_ratio', 'embed_ratio', 'framebased',
'frameTagRatio', 'hasDomainLink', 'html_ratio', 'image_ratio', 'is_news',
'lengthyLinkDomain', 'linkwordscore', 'news_front_page',
'non_markup_alphanum_characters', 'numberOfLinks', 'numwords_in_url',
'parametrizedLinkRatio', 'spelling_errors_ratio', 'label',
'alchemy_category_Index', 'alchemy_category_IndexVec']
```

结果显示新增了 alchemy_category_IndexVec 字段

查看转换后新增的字段

```
[20]: df2.select("alchemy_category","alchemy_category_Index",
                   "alchemy_category_IndexVec").show(10)
```

```
+------------------+----------------------+-------------------------+
|  alchemy_category|alchemy_category_Index|alchemy_category_IndexVec|
+------------------+----------------------+-------------------------+
|          business|                   3.0|           (14,[3],[1.0])|
|        recreation|                   1.0|           (14,[1],[1.0])|
|            health|                   4.0|           (14,[4],[1.0])|
|            health|                   4.0|           (14,[4],[1.0])|
|            sports|                   5.0|           (14,[5],[1.0])|
|                 ?|                   0.0|           (14,[0],[1.0])|
|arts_entertainment|                   2.0|           (14,[2],[1.0])|
|                 ?|                   0.0|           (14,[0],[1.0])|
|                 ?|                   0.0|           (14,[0],[1.0])|
|                 ?|                   0.0|           (14,[0],[1.0])|
+------------------+----------------------+-------------------------+
only showing top 10 rows
```

0.3 VectorAssembler

VectorAssembler 可以将多个特征字段整合成一个特征的 Vector

```
[21]: from pyspark.ml.feature import  VectorAssembler
```

创建全部特征字段 List

```
[22]: assemblerInputs =['alchemy_category_IndexVec']  +  row_df.columns[4:-1]
      print (assemblerInputs)
```

```
['alchemy_category_IndexVec', 'alchemy_category_score', 'avglinksize',
'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkratio_3',
'commonlinkratio_4', 'compression_ratio', 'embed_ratio', 'framebased',
'frameTagRatio', 'hasDomainLink', 'html_ratio', 'image_ratio', 'is_news',
'lengthyLinkDomain', 'linkwordscore', 'news_front_page',
'non_markup_alphanum_characters', 'numberOfLinks', 'numwords_in_url',
'parametrizedLinkRatio', 'spelling_errors_ratio']
```

创建 VectorAssembler

```
[23]: assembler = VectorAssembler(inputCols=assemblerInputs,
                                  outputCol="features")
```

运行 VectorAssembler 转换

```
[24]: df3=assembler.transform(df2)
```

查看整合后的新增字段

```
[25]: print (df3.columns)
```

```
['url', 'alchemy_category', 'alchemy_category_score', 'avglinksize',
'commonlinkratio_1', 'commonlinkratio_2', 'commonlinkratio_3',
'commonlinkratio_4', 'compression_ratio', 'embed_ratio', 'framebased',
'frameTagRatio', 'hasDomainLink', 'html_ratio', 'image_ratio', 'is_news',
'lengthyLinkDomain', 'linkwordscore', 'news_front_page',
'non_markup_alphanum_characters', 'numberOfLinks', 'numwords_in_url',
'parametrizedLinkRatio', 'spelling_errors_ratio', 'label',
'alchemy_category_Index', 'alchemy_category_IndexVec', 'features']
```

查看 features 特征字段

```
[26]: df3.select('features').take(1)
```

```
[26]: [Row(features=SparseVector(36, {3: 1.0, 14: 0.7891, 15: 2.0556, 16: 0.6765, 17:
0.2059, 18: 0.0471, 19: 0.0235, 20: 0.4438, 23: 0.0908, 25: 0.2458, 26: 0.0039,
27: 1.0, 28: 1.0, 29: 24.0, 31: 5424.0, 32: 170.0, 33: 8.0, 34: 0.1529, 35:
0.0791}))]
```

0.4 DesionTreeClassifier

```
[27]: from pyspark.ml.classification import DecisionTreeClassifier
```

运行 DesionTreeClassifier

```
[28]: dt = DecisionTreeClassifier(labelCol="label",  featuresCol="features",
                                 impurity="gini",maxDepth=10, maxBins=14)
```

进行训练，之前创建的 dt 决策树分类，我们可以使用.fit() 方法进行训练，训练结果产生 dt_model 模型，之后可以使用 print 查看产生的模型

```
[29]: dt_model=dt.fit(df3)
      print (dt_model)
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_54cb09dfc8b9) of
depth 10 with 653 nodes
```

进行预测，建立模型后就可以使用.trainform() 进行转换了，转换后会产生预测结果 df4

```
[30]: df4=dt_model.transform(df3)
```

0.5 建立机器学习 Pipeline 流程

```
[31]: from pyspark.ml import Pipeline
      from pyspark.ml.feature import  StringIndexer, OneHotEncoder,VectorAssembler
      from pyspark.ml.classification import DecisionTreeClassifier
```

建立 pipeline

```
[32]: stringIndexer = StringIndexer(inputCol='alchemy_category',
                                    outputCol="alchemy_category_Index")
      encoder = OneHotEncoder(dropLast=False,
                              inputCol='alchemy_category_Index',
                              outputCol="alchemy_category_IndexVec")
      assemblerInputs =['alchemy_category_IndexVec']  + row_df.columns[4:-1]
      assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
      dt = DecisionTreeClassifier(labelCol="label",␣
       ↪featuresCol="features",impurity="gini",
                                                   maxDepth=10, maxBins=14)
```

```
pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler,dt ])
```

建立 pipeline 后，我们还可以使用 getStages() 看到每一个阶段

[33]:
```
pipeline.getStages()
```

[33]:
```
[StringIndexer_bb6df0724112,
 OneHotEncoder_bd0bfc31e99b,
 VectorAssembler_0292b342fb27,
 DecisionTreeClassifier_74c1c147bca2]
```

0.6 使用 pipeline 进行数据处理与训练

[34]:
```
pipelineModel = pipeline.fit(train_df)
```

看训练完成后的决策树模型

[35]:
```
pipelineModel.stages[3]
```

[35]:
```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_74c1c147bca2) of
depth 10 with 623 nodes
```

查看训练完后的决策树模型规则

[36]:
```
print (pipelineModel.stages[3].toDebugString[:1000])
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_74c1c147bca2) of
depth 10 with 623 nodes
  If (feature 31 <= 1805.5)
   If (feature 23 <= 0.035579418)
    If (feature 1 in {1.0})
     If (feature 15 <= 1.317063492)
      If (feature 29 <= 45.5)
       If (feature 29 <= 18.5)
        If (feature 21 <= 1.225E-4)
         Predict: 0.0
        Else (feature 21 > 1.225E-4)
         Predict: 1.0
       Else (feature 29 > 18.5)
```

```
         Predict: 1.0
       Else (feature 29 > 45.5)
        Predict: 0.0
      Else (feature 15 > 1.317063492)
       If (feature 16 <= 0.13066241750000002)
        Predict: 0.0
       Else (feature 16 > 0.13066241750000002)
        If (feature 26 <= 0.110598196)
         Predict: 1.0
        Else (feature 26 > 0.110598196)
         If (feature 14 <= 0.0791547)
          Predict: 0.0
         Else (feature 14 > 0.0791547)
          If (feature 20 <= 0.7448096264999999)
           If (feature 20 <= 0.39076823250000003)
            If (feature 14 <= 0.543944)
             P
```

0.7 使用 pipelineModel 进行预测

使用 pipelineModel 的 transform 方法，传入 test_df 测试数据进行预测

[37]:
```
predicted=pipelineModel.transform(test_df)
```

查看预测后的 Schema，发现新增了 3 个字段

[38]:
```
predicted.columns
```

[38]:
```
['url',
 'alchemy_category',
 'alchemy_category_score',
 'avglinksize',
 'commonlinkratio_1',
 'commonlinkratio_2',
 'commonlinkratio_3',
 'commonlinkratio_4',
 'compression_ratio',
 'embed_ratio',
 'framebased',
 'frameTagRatio',
 'hasDomainLink',
 'html_ratio',
 'image_ratio',
 'is_news',
 'lengthyLinkDomain',
 'linkwordscore',
 'news_front_page',
 'non_markup_alphanum_characters',
 'numberOfLinks',
 'numwords_in_url',
 'parametrizedLinkRatio',
 'spelling_errors_ratio',
 'label',
 'alchemy_category_Index',
 'alchemy_category_IndexVec',
 'features',
 'rawPrediction',
 'probability',
 'prediction']
```

- rawprediction：评估模型准确率时使用
- probability：预测的结果 0 或 1
- prediction：除了知道预测结果，还能知道 0 或 1 的概率

看预测结果 DataFrame

[39]:
```
    predicted.
↪select('url','features','rawprediction','probability','label','prediction').
↪show(10)
```

```
+--------------------+--------------------+-------------+--------------------+--
---+----------+
|                 url|            features|rawprediction|
probability|label|prediction|
+--------------------+--------------------+-------------+--------------------+--
```

```
---+----------+
|http://2oddities…|(36,[7,14,15,16,1…|    [6.0,8.0]|[0.42857142857142…|
1.0|        1.0|
|http://3kidsandus…|(36,[0,15,16,17,1…|    [5.0,85.0]|[0.05555555555555…|
0.0|        1.0|
|http://3kidsandus…|(36,[0,15,16,17,1…|    [35.0,4.0]|[0.89743589743589…|
0.0|        0.0|
|http://6jokes.com…|(36,[3,14,15,16,1…|    [35.0,1.0]|[0.97222222222222…|
0.0|        0.0|
|http://98smile.co…|(36,[5,14,15,16,2…|  [151.0,43.0]|[0.77835051546391…|
0.0|        0.0|
|http://98smile.co…|(36,[4,14,15,16,2…|    [12.0,0.0]|           [1.0,0.0]|
1.0|        0.0|
|http://9gag.com/g…|(36,[0,15,16,17,1…|    [14.0,5.0]|[0.73684210526315…|
0.0|        0.0|
|http://9gag.com/g…|(36,[0,15,16,17,1…|    [14.0,5.0]|[0.73684210526315…|
0.0|        0.0|
|http://9gg.us/hah…|(36,[12,14,15,20,…|    [13.0,0.0]|           [1.0,0.0]|
0.0|        0.0|
|http://9humor.com…|(36,[2,14,15,16,2…|    [48.0,0.0]|           [1.0,0.0]|
0.0|        0.0|
+--------------------+--------------------+-------------+--------------------+--
---+----------+
only showing top 10 rows
```

查看预测结果与概率

```
[40]: predicted.select('probability','prediction') .take(10)
```

```
[40]: [Row(probability=DenseVector([0.4286, 0.5714]), prediction=1.0),
       Row(probability=DenseVector([0.0556, 0.9444]), prediction=1.0),
       Row(probability=DenseVector([0.8974, 0.1026]), prediction=0.0),
       Row(probability=DenseVector([0.9722, 0.0278]), prediction=0.0),
       Row(probability=DenseVector([0.7784, 0.2216]), prediction=0.0),
       Row(probability=DenseVector([1.0, 0.0]), prediction=0.0),
       Row(probability=DenseVector([0.7368, 0.2632]), prediction=0.0),
       Row(probability=DenseVector([0.7368, 0.2632]), prediction=0.0),
       Row(probability=DenseVector([1.0, 0.0]), prediction=0.0),
       Row(probability=DenseVector([1.0, 0.0]), prediction=0.0)]
```

0.8 评估模型的准确率

首先从 pyspark.ml.evaluation 导入 BinaryClassificationEvaluator 模块

[41]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

创建 BinaryClassificationEvaluator，传入下列参数：- rawPredictionCol= “rawPrediction” 之前预测后产生的字段

- labelCol= “label” 标签字段
- metricName= “areaUnderROC” 也就是 AUC

[42]:
```
evaluator = BinaryClassificationEvaluator(
                              rawPredictionCol="rawPrediction",
                              labelCol="label",
                              metricName="areaUnderROC"  )
```

计算 AUC

[43]:
```
predictions =pipelineModel.transform(test_df)
auc= evaluator.evaluate(predictions)
auc
```

[43]:
```
0.5964457619568017
```

使用 TrainValidation 进行训练验证找出最佳模型

从 pyspark.ml.tuning 导入 ParamGridBuilder 与 TrainValidationSplit 模块

[44]:
```
from pyspark.ml.tuning import ParamGridBuilder,TrainValidationSplit
```

设置训练验证的参数，我们使用 ParamGridBuilder 设置 impurity 两个参数值、maxDepth 三个参数值与 maxBins 三个参数值，后续执行训练验证时会执行 233=18 次。

[45]:
```
paramGrid = ParamGridBuilder()\
.addGrid(dt.impurity, [ "gini","entropy"])\
.addGrid(dt.maxDepth, [ 5,10,15])\
.addGrid(dt.maxBins, [10, 15,20])\
.build()
```

创建 TrainValidationSplit，传入下列参数，执行后创建 tvs 变量：

- estimator=dt，之前创建的 DecisionTreeClassifier
- evaluator=evaluator，之前创建的 BinaryClassificationEvaluator
- estimatorParamMaps=paramGrid，之前创建的 ParamGridBuilder
- trainRatio=0.8，训练验证前会先将数据按照 8:2 的比例分成训练数据与验证数据

[46]:
```
tvs = TrainValidationSplit(estimator=dt,evaluator=evaluator,
                           estimatorParamMaps=paramGrid,trainRatio=0.8)
```

建立 tvs_pipeline

[47]:
```
tvs_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, tvs])
```

使用 tvs_pipeline 流程进行训练验证

[48]:
```
tvs_pipelineModel =tvs_pipeline.fit(train_df)
```

查看训练完成的最佳模型

[49]:
```
bestModel=tvs_pipelineModel.stages[3].bestModel
bestModel
```

[49]:
```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_74c1c147bca2) of
depth 15 with 1901 nodes
```

看训练验证完成的最佳模型规则，[:500] 表示只显示前 500 文字

[50]:
```
print (bestModel.toDebugString[:500])
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_74c1c147bca2) of
depth 15 with 1901 nodes
  If (feature 31 <= 1932.0)
   If (feature 31 <= 1221.5)
    If (feature 2 in {1.0})
     If (feature 29 <= 16.5)
```

```
      Predict: 0.0
     Else (feature 29 > 16.5)
      If (feature 33 <= 5.5)
       If (feature 26 <= 0.018264935500000003)
        If (feature 28 <= 0.5)
         If (feature 16 <= 0.529606545)
          If (feature 31 <= 554.5)
           If (feature 25 <= 0.1751179765)
```

评估最佳模型 AUC

[51]:
```
predictions = tvs_pipelineModel.transform(test_df)
auc= evaluator.evaluate(predictions)
auc
```

[51]:
```
0.6502853031051837
```

0.9 使用 crossValidation 交叉验证找出最佳模型

[52]:
```
from pyspark.ml.tuning import CrossValidator
```

建立交叉验证的 CrossValidator

[53]:
```
cv = CrossValidator(estimator=dt, evaluator=evaluator,
                    estimatorParamMaps=paramGrid, numFolds=3)
```

建立交叉验证的 cv_pipeline

[54]:
```
cv_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, cv])
```

用 cv_pipeline 流程进行交叉验证

[55]:
```
cv_pipelineModel = cv_pipeline.fit(train_df)
```

查看交叉验证完成的最佳模型

[56]:
```
bestModel=cv_pipelineModel.stages[3].bestModel
bestModel
```

```
[56]: DecisionTreeClassificationModel (uid=DecisionTreeClassifier_74c1c147bca2) of
      depth 15 with 1321 nodes
```

评估最佳模型 AUC

```
[57]: predictions = cv_pipelineModel.transform(test_df)
      auc= evaluator.evaluate(predictions)
      auc
```

```
[57]: 0.6428303564592135
```

1 使用随机森林 RandomForestClassifier 分类器

使用随机森林 RandomForestClassifier 分类器

创建 RandomForestClassifier 变量 rf，传入参数与决策树类似，只是多了 numTrees 参数（设置决策森林中有多少决策树，这里设为 10）

```
[58]: from pyspark.ml.classification import RandomForestClassifier

      rf =RandomForestClassifier(labelCol="label",
                                 featuresCol="features",numTrees=10)

      rfpipeline = Pipeline(stages=[stringIndexer,encoder ,assembler,rf ])
```

评估 RandomForestClassifier 的准确度

rfpipeline.fit 传入 train_df 进行训练，再用 rftvs_pipelineModel.transform 传入 test_df 进行评估，我们可以看到 AUC 约为 0.72，比之前使用决策树的准确度明显增加

```
[59]: rfpipelineModel = rfpipeline.fit(train_df)
      rfpredicted=rfpipelineModel.transform(test_df)
      evaluator.evaluate(rfpredicted)
```

```
[59]: 0.7290033794365983
```

使用 RandomForestClassifier TrainValidation 找出最佳模型

```
[60]: from pyspark.ml.tuning import ParamGridBuilder, TrainValidationSplit
      from pyspark.ml.evaluation import BinaryClassificationEvaluator
      from pyspark.ml.classification import RandomForestClassifier

      paramGrid = ParamGridBuilder()\
      .addGrid(rf.impurity, [ "gini","entropy"])\
      .addGrid(rf.maxDepth, [ 5,10,15])\
      .addGrid(rf.maxBins, [10, 15,20])\
      .addGrid(rf.numTrees, [10, 20,30])\
      .build()

      rftvs = TrainValidationSplit(estimator=rf, evaluator=evaluator,
                                       estimatorParamMaps=paramGrid, trainRatio=0.8)

      rftvs_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, rftvs])
      rftvs_pipelineModel =rftvs_pipeline.fit(train_df)
      rftvspredictions = rftvs_pipelineModel.transform(test_df)
      auc= evaluator.evaluate(rftvspredictions)
      auc
```

[60]:
```
0.7504950822410787
```

使用 crossValidation 找出最佳模型

[ ]:
```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline

rfcv = CrossValidator(estimator=rf, evaluator=evaluator,
                          estimatorParamMaps=paramGrid, numFolds=3)

rfcv_pipeline = Pipeline(stages=[stringIndexer,encoder ,assembler, rfcv])
rfcv_pipelineModel = rfcv_pipeline.fit(train_df)
```

使用最佳模型进行预测

[ ]:
```
rfcvpredictions = rfcv_pipelineModel.transform(test_df)
```

显示使用最佳模型进行预测结果

```
[ ]: DescDict = {
               0: "暂时性网页 (ephemeral)",
               1: "长青网页 (evergreen)"
          }
     for data in rfcvpredictions .select('url','prediction').take(5):
         print (" 网址:   " +str(data[0])+"\n" +\
                      "            ==> 预测:"+ str(data[1])+ \
                      " 说明:"+DescDict[data[1]] +"\n")
```

计算最佳模型 AUC

```
[ ]: auc= evaluator.evaluate(rfcvpredictions)
     auc
```