PKA-DECISON-TREES

December 12, 2023

[1]:

```
import pyspark
import os
import sys
from pyspark import SparkContext
os.environ['PYSPARK_PYTHON'] = sys.executable
os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

from pyspark.sql import SparkSession
```

[2]:

```
spark = SparkSession.builder.config("spark.driver.memory", "16g").
 ↪appName('chapter_4').getOrCreate()
```

0.0.1 Preparing the Data

[3]:

```
data_without_header = spark.read.option("inferSchema", True)\
                        .option("header", False).csv("data/covtype.data")
data_without_header.printSchema()
```

```
root
 |-- _c0: integer (nullable = true)
 |-- _c1: integer (nullable = true)
 |-- _c2: integer (nullable = true)
 |-- _c3: integer (nullable = true)
 |-- _c4: integer (nullable = true)
 |-- _c5: integer (nullable = true)
 |-- _c6: integer (nullable = true)
 |-- _c7: integer (nullable = true)
 |-- _c8: integer (nullable = true)
 |-- _c9: integer (nullable = true)
 |-- _c10: integer (nullable = true)
 |-- _c11: integer (nullable = true)
 |-- _c12: integer (nullable = true)
 |-- _c13: integer (nullable = true)
 |-- _c14: integer (nullable = true)
 |-- _c15: integer (nullable = true)
 |-- _c16: integer (nullable = true)
 |-- _c17: integer (nullable = true)
 |-- _c18: integer (nullable = true)
|-- _c19: integer (nullable = true)
|-- _c20: integer (nullable = true)
|-- _c21: integer (nullable = true)
|-- _c22: integer (nullable = true)
|-- _c23: integer (nullable = true)
|-- _c24: integer (nullable = true)
|-- _c25: integer (nullable = true)
|-- _c26: integer (nullable = true)
|-- _c27: integer (nullable = true)
|-- _c28: integer (nullable = true)
|-- _c29: integer (nullable = true)
|-- _c30: integer (nullable = true)
|-- _c31: integer (nullable = true)
|-- _c32: integer (nullable = true)
|-- _c33: integer (nullable = true)
|-- _c34: integer (nullable = true)
|-- _c35: integer (nullable = true)
|-- _c36: integer (nullable = true)
|-- _c37: integer (nullable = true)
|-- _c38: integer (nullable = true)
|-- _c39: integer (nullable = true)
|-- _c40: integer (nullable = true)
|-- _c41: integer (nullable = true)
|-- _c42: integer (nullable = true)
|-- _c43: integer (nullable = true)
|-- _c44: integer (nullable = true)
|-- _c45: integer (nullable = true)
|-- _c46: integer (nullable = true)
|-- _c47: integer (nullable = true)
|-- _c48: integer (nullable = true)
|-- _c49: integer (nullable = true)
|-- _c50: integer (nullable = true)
|-- _c51: integer (nullable = true)
|-- _c52: integer (nullable = true)
|-- _c53: integer (nullable = true)
|-- _c54: integer (nullable = true)
```

[4]:

```
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import col

colnames = ["Elevation", "Aspect", "Slope", \
            "Horizontal_Distance_To_Hydrology", \
            "Vertical_Distance_To_Hydrology", "Horizontal_Distance_To_Roadways",␣
↪\
            "Hillshade_9am", "Hillshade_Noon", "Hillshade_3pm", \
            "Horizontal_Distance_To_Fire_Points"] + \
[f"Wilderness_Area_{i}" for i in range(4)] + \
[f"Soil_Type_{i}" for i in range(40)] + \
["Cover_Type"]

data = data_without_header.toDF(*colnames).\
                          withColumn("Cover_Type",
                                    col("Cover_Type").cast(DoubleType()))

data.head()
```

column overwritten

[4]:

```
Row(Elevation=2596, Aspect=51, Slope=3, Horizontal_Distance_To_Hydrology=258,
Vertical_Distance_To_Hydrology=0, Horizontal_Distance_To_Roadways=510,
Hillshade_9am=221, Hillshade_Noon=232, Hillshade_3pm=148,
Horizontal_Distance_To_Fire_Points=6279, Wilderness_Area_0=1,
Wilderness_Area_1=0, Wilderness_Area_2=0, Wilderness_Area_3=0, Soil_Type_0=0,
Soil_Type_1=0, Soil_Type_2=0, Soil_Type_3=0, Soil_Type_4=0, Soil_Type_5=0,
Soil_Type_6=0, Soil_Type_7=0, Soil_Type_8=0, Soil_Type_9=0, Soil_Type_10=0,
Soil_Type_11=0, Soil_Type_12=0, Soil_Type_13=0, Soil_Type_14=0, Soil_Type_15=0,
Soil_Type_16=0, Soil_Type_17=0, Soil_Type_18=0, Soil_Type_19=0, Soil_Type_20=0,
Soil_Type_21=0, Soil_Type_22=0, Soil_Type_23=0, Soil_Type_24=0, Soil_Type_25=0,
Soil_Type_26=0, Soil_Type_27=0, Soil_Type_28=1, Soil_Type_29=0, Soil_Type_30=0,
Soil_Type_31=0, Soil_Type_32=0, Soil_Type_33=0, Soil_Type_34=0, Soil_Type_35=0,
Soil_Type_36=0, Soil_Type_37=0, Soil_Type_38=0, Soil_Type_39=0, Cover_Type=5.0)
```

0.0.2 Our First Decision Tree

[5]:

```
(train_data, test_data) = data.randomSplit([0.9, 0.1])
train_data.cache()
test_data.cache()
```

[5]:

```
DataFrame[Elevation: int, Aspect: int, Slope: int,
Horizontal_Distance_To_Hydrology: int, Vertical_Distance_To_Hydrology: int,
Horizontal_Distance_To_Roadways: int, Hillshade_9am: int, Hillshade_Noon: int,
Hillshade_3pm: int, Horizontal_Distance_To_Fire_Points: int, Wilderness_Area_0:
int, Wilderness_Area_1: int, Wilderness_Area_2: int, Wilderness_Area_3: int,
Soil_Type_0: int, Soil_Type_1: int, Soil_Type_2: int, Soil_Type_3: int,
Soil_Type_4: int, Soil_Type_5: int, Soil_Type_6: int, Soil_Type_7: int,
Soil_Type_8: int, Soil_Type_9: int, Soil_Type_10: int, Soil_Type_11: int,
Soil_Type_12: int, Soil_Type_13: int, Soil_Type_14: int, Soil_Type_15: int,
Soil_Type_16: int, Soil_Type_17: int, Soil_Type_18: int, Soil_Type_19: int,
Soil_Type_20: int, Soil_Type_21: int, Soil_Type_22: int, Soil_Type_23: int,
Soil_Type_24: int, Soil_Type_25: int, Soil_Type_26: int, Soil_Type_27: int,
Soil_Type_28: int, Soil_Type_29: int, Soil_Type_30: int, Soil_Type_31: int,
Soil_Type_32: int, Soil_Type_33: int, Soil_Type_34: int, Soil_Type_35: int,
Soil_Type_36: int, Soil_Type_37: int, Soil_Type_38: int, Soil_Type_39: int,
Cover_Type: double]
```

columns with 0 or nan not included
kind of sparse matrix

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

input_cols = colnames[:-1]
vector_assembler = VectorAssembler(inputCols=input_cols,
                                   outputCol="featureVector")

assembled_train_data = vector_assembler.transform(train_data)

assembled_train_data.select("featureVector").show(truncate = False)
```

```
+--------------------------------------------------------------------------------
----------------------+
|featureVector
|
+--------------------------------------------------------------------------------
----------------------+
|(54,[0,1,2,3,4,5,6,7,8,9,13,15],[1863.0,37.0,17.0,120.0,18.0,90.0,217.0,202.0,1
15.0,769.0,1.0,1.0])  |
|(54,[0,1,2,5,6,7,8,9,13,18],[1874.0,18.0,14.0,90.0,208.0,209.0,135.0,793.0,1.0,
1.0])                 |
|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1879.0,28.0,19.0,30.0,12.0,95.0,209.0,196.0,11
7.0,778.0,1.0,1.0])   |
|(54,[0,1,2,3,4,5,6,7,8,9,13,14],[1889.0,28.0,22.0,150.0,23.0,120.0,205.0,185.0,
108.0,759.0,1.0,1.0]) |
|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1889.0,353.0,30.0,95.0,39.0,67.0,153.0,172.0,1
46.0,600.0,1.0,1.0])  |
|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1896.0,337.0,12.0,30.0,6.0,175.0,195.0,224.0,1
68.0,732.0,1.0,1.0])  |
|(54,[0,1,2,3,4,5,6,7,8,9,13,15],[1898.0,34.0,23.0,175.0,56.0,134.0,210.0,184.0,
99.0,765.0,1.0,1.0])  |
|(54,[0,1,2,3,4,5,6,7,8,9,13,14],[1901.0,311.0,9.0,30.0,2.0,190.0,195.0,234.0,17
9.0,726.0,1.0,1.0])   |
|(54,[0,1,2,3,4,5,6,7,8,9,13,14],[1903.0,5.0,13.0,42.0,4.0,201.0,203.0,214.0,148
.0,708.0,1.0,1.0])    |
|(54,[0,1,2,3,4,5,6,7,8,9,13,16],[1903.0,67.0,16.0,108.0,36.0,120.0,234.0,207.0,
100.0,969.0,1.0,1.0]) |
|(54,[0,1,2,3,4,5,6,7,8,9,13,14],[1904.0,51.0,26.0,67.0,30.0,162.0,222.0,175.0,7
2.0,711.0,1.0,1.0])   |
|(54,[0,1,2,3,4,5,6,7,8,9,13,14],[1905.0,33.0,27.0,90.0,46.0,150.0,204.0,171.0,8
9.0,725.0,1.0,1.0])   |
|(54,[0,1,2,3,4,5,6,7,8,9,13,16],[1905.0,77.0,21.0,90.0,38.0,120.0,241.0,196.0,7
5.0,1025.0,1.0,1.0])  |
|(54,[0,1,2,3,4,5,6,7,8,9,13,15],[1906.0,356.0,20.0,150.0,55.0,120.0,184.0,201.0
,151.0,726.0,1.0,1.0])|
|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1908.0,323.0,32.0,150.0,52.0,120.0,125.0,190.0
,196.0,765.0,1.0,1.0])|
|(54,[0,1,2,3,4,5,6,7,8,9,13,15],[1916.0,24.0,25.0,212.0,74.0,175.0,197.0,177.0,
105.0,789.0,1.0,1.0]) |
|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1916.0,320.0,24.0,190.0,60.0,162.0,151.0,210.0
,195.0,832.0,1.0,1.0])|
|(54,[0,1,2,3,4,5,6,7,8,9,13,23],[1918.0,321.0,28.0,42.0,17.0,85.0,139.0,201.0,1
96.0,402.0,1.0,1.0])  |
|(54,[0,1,2,3,4,5,6,7,8,9,13,14],[1919.0,30.0,22.0,67.0,9.0,256.0,208.0,188.0,10
7.0,661.0,1.0,1.0])   |
|(54,[0,1,2,3,4,5,6,7,8,9,13,18],[1919.0,44.0,26.0,162.0,68.0,150.0,216.0,173.0,
77.0,706.0,1.0,1.0])  |
+--------------------------------------------------------------------------------
----------------------+
only showing top 20 rows
```

[7]:

```python
from pyspark.ml.classification import DecisionTreeClassifier

classifier = DecisionTreeClassifier(seed = 1234, labelCol="Cover_Type",
                                    featuresCol="featureVector",
                                    predictionCol="prediction")

model = classifier.fit(assembled_train_data)
print(model.toDebugString)
```

```
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_978a0adfa76c,
depth=5, numNodes=45, numClasses=8, numFeatures=54
  If (feature 0 <= 3050.5)
   If (feature 0 <= 2502.5)
    If (feature 3 <= 15.0)
     If (feature 12 <= 0.5)
      If (feature 23 <= 0.5)
       Predict: 4.0
      Else (feature 23 > 0.5)
       Predict: 3.0
     Else (feature 12 > 0.5)
      Predict: 6.0
    Else (feature 3 > 15.0)
     If (feature 16 <= 0.5)
      Predict: 3.0
     Else (feature 16 > 0.5)
      If (feature 9 <= 1328.5)
       Predict: 3.0
      Else (feature 9 > 1328.5)
       Predict: 4.0
   Else (feature 0 > 2502.5)
    If (feature 17 <= 0.5)
     If (feature 0 <= 2951.5)
      If (feature 15 <= 0.5)
       Predict: 2.0
      Else (feature 15 > 0.5)
      Predict: 3.0
    Else (feature 0 > 2951.5)
     Predict: 2.0
   Else (feature 17 > 0.5)
    If (feature 0 <= 2703.5)
     Predict: 3.0
    Else (feature 0 > 2703.5)
     If (feature 5 <= 1246.5)
      Predict: 5.0
     Else (feature 5 > 1246.5)
      Predict: 2.0
 Else (feature 0 > 3050.5)
  If (feature 0 <= 3321.5)
   If (feature 7 <= 238.5)
    Predict: 1.0
   Else (feature 7 > 238.5)
    If (feature 3 <= 307.5)
     Predict: 1.0
    Else (feature 3 > 307.5)
     If (feature 0 <= 3207.5)
      Predict: 2.0
     Else (feature 0 > 3207.5)
      Predict: 1.0
  Else (feature 0 > 3321.5)
   If (feature 12 <= 0.5)
    If (feature 3 <= 290.0)
     If (feature 8 <= 185.5)
      Predict: 7.0
     Else (feature 8 > 185.5)
      Predict: 1.0
    Else (feature 3 > 290.0)
     Predict: 1.0
   Else (feature 12 > 0.5)
    If (feature 45 <= 0.5)
     If (feature 44 <= 0.5)
      Predict: 7.0
     Else (feature 44 > 0.5)
      Predict: 1.0
    Else (feature 45 > 0.5)
     If (feature 5 <= 958.0)
      Predict: 7.0
     Else (feature 5 > 958.0)
      Predict: 1.0
```

[8]:

```
import pandas as pd
```

note

```
pd.DataFrame(model.featureImportances.toArray(),
             index=input_cols, columns=['importance']).\
             sort_values(by="importance", ascending=False)
```

```
[8]:                                       importance
     Elevation                               0.832888
     Soil_Type_3                             0.035340
     Soil_Type_1                             0.031061
     Hillshade_Noon                          0.025543
     Horizontal_Distance_To_Hydrology        0.024496
     Soil_Type_31                            0.017925
     Wilderness_Area_2                       0.015337
     Horizontal_Distance_To_Roadways         0.004785
     Soil_Type_2                             0.003540
     Soil_Type_30                            0.003522
     Hillshade_3pm                           0.002455
     Horizontal_Distance_To_Fire_Points      0.002141
     Soil_Type_9                             0.000967
     Soil_Type_27                            0.000000
     Soil_Type_20                            0.000000
     Soil_Type_21                            0.000000
     Soil_Type_22                            0.000000
     Soil_Type_23                            0.000000
     Soil_Type_24                            0.000000
     Soil_Type_25                            0.000000
     Soil_Type_26                            0.000000
     Soil_Type_29                            0.000000
     Soil_Type_28                            0.000000
     Soil_Type_35                            0.000000
     Soil_Type_38                            0.000000
     Soil_Type_37                            0.000000
     Soil_Type_18                            0.000000
     Soil_Type_36                            0.000000
     Soil_Type_32                            0.000000
     Soil_Type_33                            0.000000
     Soil_Type_34                            0.000000
     Soil_Type_19                            0.000000
     Soil_Type_13                            0.000000
     Soil_Type_17                            0.000000
     Soil_Type_5                             0.000000
     Slope                                   0.000000
     Vertical_Distance_To_Hydrology          0.000000
     Hillshade_9am                           0.000000
     Wilderness_Area_0                       0.000000
     Wilderness_Area_1                       0.000000
     Wilderness_Area_3                       0.000000
     Soil_Type_0                             0.000000
```

```
Soil_Type_4                           0.000000
Soil_Type_6                           0.000000
Soil_Type_16                          0.000000
Soil_Type_7                           0.000000
Soil_Type_8                           0.000000
Soil_Type_10                          0.000000
Soil_Type_11                          0.000000
Soil_Type_12                          0.000000
Aspect                                0.000000
Soil_Type_14                          0.000000
Soil_Type_15                          0.000000
Soil_Type_39                          0.000000
```

[9]:

```
predictions = model.transform(assembled_train_data)
predictions.select("Cover_Type", "prediction", "probability").\
            show(10, truncate = False)
```

```
+----------+----------+---------------------------------------------------------
-------------------------------------------------------------------------+
|Cover_Type|prediction|probability
|
+----------+----------+---------------------------------------------------------
-------------------------------------------------------------------------+
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |4.0       |[0.0,0.0,0.04143258426966292,0.23595505617977527,0.615870
7865168539,0.009129213483146067,0.0976123595505618,0.0]                  |
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|6.0       |3.0       |[0.0,3.125488357555868E-
5,0.07016721362712924,0.605875918112205,0.019846851070479763,0.00196905766526019
7,0.30210970464135023,0.0]|
|3.0       |3.0       |[0.0,0.0,0.01900337837837838,0.6680743243243243,0.2440878
3783783783,0.0,0.06883445945945946,0.0]                                  |
+----------+----------+---------------------------------------------------------
-------------------------------------------------------------------------+
only showing top 10 rows
```

[10]:

```python
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(labelCol="Cover_Type",
                                              predictionCol="prediction")

evaluator.setMetricName("accuracy").evaluate(predictions)
evaluator.setMetricName("f1").evaluate(predictions)
```

[10]:

```
0.6890498744680869
```

[11]:

```python
confusion_matrix = predictions.groupBy("Cover_Type").\
  pivot("prediction", range(1,8)).count().\
  na.fill(0.0).\
  orderBy("Cover_Type")

confusion_matrix.show()
```

```
+----------+------+------+-----+----+---+---+-----+
|Cover_Type|     1|     2|    3|   4|  5|  6|    7|
+----------+------+------+-----+----+---+---+-----+
|       1.0|128602| 57179|   96|   0| 29|  7| 4557|
|       2.0| 50565|199538| 3677|  59|367| 59|  542|
|       3.0|     0|  3895|27787| 343| 32|122|    0|
|       4.0|     0|     4| 1328|1122|  0|  0|    0|
|       5.0|     0|  7765|  340|  13|436|  0|    0|
|       6.0|     0|  4371|10623| 139| 11|526|    0|
|       7.0|  8268|    76|    0|   0|  0|  0|10096|
+----------+------+------+-----+----+---+---+-----+
```

[12]:

```python
from pyspark.sql import DataFrame

def class_probabilities(data):
    total = data.count()
    return data.groupBy("Cover_Type").count().\
    orderBy("Cover_Type").\
    select(col("count").cast(DoubleType())).\
    withColumn("count_proportion", col("count")/total).\
    select("count_proportion").collect()
```

```
train_prior_probabilities = class_probabilities(train_data)
test_prior_probabilities = class_probabilities(test_data)

train_prior_probabilities
```

[12]:

```
[Row(count_proportion=0.36448426442953535),
 Row(count_proportion=0.48759984231898257),
 Row(count_proportion=0.06157788179281786),
 Row(count_proportion=0.004695985640311228),
 Row(count_proportion=0.016368973580775163),
 Row(count_proportion=0.029986183774929485),
 Row(count_proportion=0.03528686846264835)]
```

[13]:

```
train_prior_probabilities = [p[0] for p in train_prior_probabilities]
test_prior_probabilities = [p[0] for p in test_prior_probabilities]

sum([train_p * cv_p for train_p, cv_p in zip(train_prior_probabilities,
                                              test_prior_probabilities)])
```

[13]:

```
0.3772120251698258
```

0.0.3 Tuning Decision Trees

[14]:

```
from pyspark.ml import Pipeline

assembler = VectorAssembler(inputCols=input_cols, outputCol="featureVector")
classifier = DecisionTreeClassifier(seed=1234, labelCol="Cover_Type",
                                    featuresCol="featureVector",
                                    predictionCol="prediction")

pipeline = Pipeline(stages=[assembler, classifier])
```

[15]:

```
from pyspark.ml.tuning import ParamGridBuilder

paramGrid = ParamGridBuilder(). \
  addGrid(classifier.impurity, ["gini", "entropy"]). \
  addGrid(classifier.maxDepth, [1, 20]). \
  addGrid(classifier.maxBins, [40, 300]). \
  addGrid(classifier.minInfoGain, [0.0, 0.05]). \
  build()

multiclassEval = MulticlassClassificationEvaluator(). \
  setLabelCol("Cover_Type"). \
  setPredictionCol("prediction"). \
  setMetricName("accuracy")
```

[16]:

```python
from pyspark.ml.tuning import TrainValidationSplit

validator = TrainValidationSplit(seed=1234,
  estimator=pipeline,
  evaluator=multiclassEval,
  estimatorParamMaps=paramGrid,
  trainRatio=0.9)

validator_model = validator.fit(train_data)
```

[17]:

```python
from pprint import pprint

best_model = validator_model.bestModel
pprint(best_model.stages[1].extractParamMap())
```

```
{Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='featuresCol',
doc='features column name.'): 'featureVector',
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='checkpointInterval',
doc='set checkpoint interval (>= 1) or disable checkpoint (-1). E.g. 10 means
that the cache will get checkpointed every 10 iterations. Note: this setting
will be ignored if the checkpoint directory is not set in the SparkContext.'):
10,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='cacheNodeIds',
doc='If false, the algorithm will pass trees to executors to match instances
with nodes. If true, the algorithm will cache node IDs for each instance.
Caching can speed up training of deeper trees. Users can set how often should
the cache be checkpointed or disable it by setting checkpointInterval.'): False,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='probabilityCol',
doc='Column name for predicted class conditional probabilities. Note: Not all
models output well-calibrated probability estimates! These probabilities should
be treated as confidences, not precise probabilities.'): 'probability',
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='labelCol', doc='label
column name.'): 'Cover_Type',
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='rawPredictionCol',
doc='raw prediction (a.k.a. confidence) column name.'): 'rawPrediction',
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInstancesPerNode',
doc='Minimum number of instances each child must have after split. If a split
causes the left or right child to have fewer than minInstancesPerNode, the split
will be discarded as invalid. Should be >= 1.'): 1,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='predictionCol',
doc='prediction column name.'): 'prediction',
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='seed', doc='random
seed.'): 1234,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca',
name='minWeightFractionPerNode', doc='Minimum fraction of the weighted sample
count that each child must have after split. If a split causes the fraction of
the total weight in the left or right child to be less than
minWeightFractionPerNode, the split will be discarded as invalid. Should be in
interval [0.0, 0.5).'): 0.0,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxMemoryInMB',
doc='Maximum memory in MB allocated to histogram aggregation. If too small, then
1 node will be split per iteration, and its aggregates may exceed this size.'):
256,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
 Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='leafCol', doc='Leaf
indices column name. Predicted leaf index of each instance in each tree by
preorder.'): ''}
```

[18]:

```python
validator_model = validator.fit(train_data)

metrics = validator_model.validationMetrics
params = validator_model.getEstimatorParamMaps()
metrics_and_params = list(zip(metrics, params))

metrics_and_params.sort(key=lambda x: x[0], reverse=True)
metrics_and_params
```

[18]:

```
[(0.9145885645281862,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.9094707787851488,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.9051580379904545,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.9046405090950912,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.7230262022962949,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
```

```
0.05}),
 (0.7228345249276418,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05}),
 (0.6729600736041096,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05}),
 (0.6710432999175787,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 20,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05}),
 (0.6360046769277952,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.6360046769277952,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05}),
 (0.6345095934523011,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.6345095934523011,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'gini',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05}),
 (0.49146077322650517,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.49146077322650517,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 40,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05}),
 (0.49032987675145195,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.0}),
 (0.49032987675145195,
  {Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='impurity',
doc='Criterion used for information gain calculation (case-insensitive).
Supported options: entropy, gini'): 'entropy',
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxDepth',
doc='Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1
means 1 internal node + 2 leaf nodes. Must be in range [0, 30].'): 1,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='maxBins', doc='Max
number of bins for discretizing continuous features.  Must be >=2 and >= number
of categories for any categorical feature.'): 300,
   Param(parent='DecisionTreeClassifier_c9beb3f0fdca', name='minInfoGain',
doc='Minimum information gain for a split to be considered at a tree node.'):
0.05})]
```

[19]:

```
metrics.sort(reverse=True)
print(metrics[0])
```

```
0.9145885645281862
```

[20]:

```
multiclassEval.evaluate(best_model.transform(test_data))
```

[20]:

```
0.9098531777268216
```

0.0.4 Categorical Features Revisited

[21]:

```
from pyspark.sql.functions import udf
from pyspark.sql.types import IntegerType

def unencode_one_hot(data):
    wilderness_cols = ['Wilderness_Area_' + str(i) for i in range(4)]
    wilderness_assembler = VectorAssembler().\
                            setInputCols(wilderness_cols).\
                            setOutputCol("wilderness")

    unhot_udf = udf(lambda v: v.toArray().tolist().index(1))

    with_wilderness = wilderness_assembler.transform(data).\
      drop(*wilderness_cols).\
      withColumn("wilderness", unhot_udf(col("wilderness")).cast(IntegerType()))

    soil_cols = ['Soil_Type_' + str(i) for i in range(40)]
    soil_assembler = VectorAssembler().\
                      setInputCols(soil_cols).\
                      setOutputCol("soil")
    with_soil = soil_assembler.\
                transform(with_wilderness).\
                drop(*soil_cols).\
                withColumn("soil", unhot_udf(col("soil")).cast(IntegerType()))

    return with_soil
```

[22]:

```
unenc_train_data = unencode_one_hot(train_data)
unenc_train_data.printSchema()
```

```
root
 |-- Elevation: integer (nullable = true)
 |-- Aspect: integer (nullable = true)
 |-- Slope: integer (nullable = true)
 |-- Horizontal_Distance_To_Hydrology: integer (nullable = true)
 |-- Vertical_Distance_To_Hydrology: integer (nullable = true)
 |-- Horizontal_Distance_To_Roadways: integer (nullable = true)
 |-- Hillshade_9am: integer (nullable = true)
 |-- Hillshade_Noon: integer (nullable = true)
 |-- Hillshade_3pm: integer (nullable = true)
 |-- Horizontal_Distance_To_Fire_Points: integer (nullable = true)
 |-- Cover_Type: double (nullable = true)
 |-- wilderness: integer (nullable = true)
 |-- soil: integer (nullable = true)
```

[23]:

```
unenc_train_data.groupBy('wilderness').count().show()
```

```
+----------+------+
|wilderness| count|
+----------+------+
|         1| 26805|
|         3| 33237|
|         2|227994|
|         0|234538|
+----------+------+
```

[24]:

```
from pyspark.ml.feature import VectorIndexer

cols = unenc_train_data.columns
input_cols = [c for c in cols if c!='Cover_Type']

assembler = VectorAssembler().setInputCols(input_cols).
 ↪setOutputCol("featureVector")

indexer = VectorIndexer().\
  setMaxCategories(40).\
  setInputCol("featureVector").setOutputCol("indexedVector")

classifier = DecisionTreeClassifier().setLabelCol("Cover_Type").\
                                      setFeaturesCol("indexedVector").\
                                      setPredictionCol("prediction")

pipeline = Pipeline().setStages([assembler, indexer, classifier])
```

0.0.5 Random Forests Takes Too Long To Run

[25]:

```
from pyspark.ml.classification import RandomForestClassifier

classifier = RandomForestClassifier(seed=1234, labelCol="Cover_Type",
                                    featuresCol="indexedVector",
                                              predictionCol="prediction")
```

[26]:

```
unenc_train_data.columns
```

[26]:

```
['Elevation',
 'Aspect',
 'Slope',
 'Horizontal_Distance_To_Hydrology',
 'Vertical_Distance_To_Hydrology',
 'Horizontal_Distance_To_Roadways',
 'Hillshade_9am',
 'Hillshade_Noon',
 'Hillshade_3pm',
 'Horizontal_Distance_To_Fire_Points',
 'Cover_Type',
 'wilderness',
 'soil']
```

[27]:

```
######### LONGER TIME ##################################

cols = unenc_train_data.columns
input_cols = [c for c in cols if c!='Cover_Type']

assembler = VectorAssembler().setInputCols(input_cols).
 ↪setOutputCol("featureVector")

indexer = VectorIndexer().\
  setMaxCategories(40).\
  setInputCol("featureVector").setOutputCol("indexedVector")

pipeline = Pipeline().setStages([assembler, indexer, classifier])

paramGrid = ParamGridBuilder(). \
  addGrid(classifier.impurity, ["gini", "entropy"]). \
  addGrid(classifier.maxDepth, [1, 20]). \
  addGrid(classifier.maxBins, [40, 300]). \
  addGrid(classifier.minInfoGain, [0.0, 0.05]). \
  build()

multiclassEval = MulticlassClassificationEvaluator(). \
  setLabelCol("Cover_Type"). \
  setPredictionCol("prediction"). \
  setMetricName("accuracy")

validator = TrainValidationSplit(seed=1234,
  estimator=pipeline,
  evaluator=multiclassEval,
  estimatorParamMaps=paramGrid,
  trainRatio=0.9)

validator_model = validator.fit(unenc_train_data)

best_model = validator_model.bestModel
```

[28]:

```
forest_model = best_model.stages[2]

feature_importance_list = list(zip(input_cols,
                                  forest_model.featureImportances.toArray()))
feature_importance_list.sort(key=lambda x: x[1], reverse=True)

pprint(feature_importance_list)
```

```
[('Elevation', 0.3427225449954387),
 ('soil', 0.1507194164175547),
 ('Horizontal_Distance_To_Roadways', 0.11070316027812355),
 ('Horizontal_Distance_To_Fire_Points', 0.10351207526224093),
 ('wilderness', 0.06922933410510583),
 ('Horizontal_Distance_To_Hydrology', 0.04757941893539708),
 ('Vertical_Distance_To_Hydrology', 0.04244874242868387),
 ('Aspect', 0.03240174066198868),
 ('Hillshade_Noon', 0.02890461357693716),
 ('Hillshade_9am', 0.02807572724134144),
 ('Hillshade_3pm', 0.0243082931922063),
 ('Slope', 0.01939493290498192)]
```

0.0.6 Making Predictions

[29]:

```
unenc_test_data = unencode_one_hot(test_data)

best_model.transform(unenc_test_data.drop("Cover_Type")).\
                    select("prediction").show(1)
```

```
+----------+
|prediction|
+----------+
|       6.0|
+----------+
only showing top 1 row
```