databricksProject_03

DSCI 417 – Project 03

Forest Cover Prediction

Sean Graham

Part A: Set up Environment

This part of the project involves importing the required tools from pyspark and creating a SparkSession object.

This cell imports the required tools from pyspark and creates a SparkSession object

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier, LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MulticlassMetrics

spark = SparkSession.builder.getOrCreate()
```

Part B: Load and Explore the Data

This part of the project involves loading and exploring the data.

This cell loads the dataset into a DataFrame.

```
schema_forest_cover = ('elevation INTEGER, aspect INTEGER, slope INTEGER, Horizontal_Distance_To_Hydrology INTEGER, Vertical_Distance_To_Hydrology INTEGER,
Horizontal_Distance_To_Roadways INTEGER, Hillshade_9am INTEGER, Hillshade_Noon INTEGER, Hillshade_3pm INTEGER, Horizontal_Distance_To_Fire_Points INTEGER,
Wilderness_Area STRING, Soil_Type INTEGER, Cover_Type INTEGER')

fc = (
    spark.read
    .option('delimiter', '\t')
    .option('header', True)
    .schema(schema_forest_cover)
    .csv('/FileStore/tables/forest_cover.txt')
)

fc.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)
 |-- Wilderness_Area: string (nullable = true)
 |-- Soil_Type: integer (nullable = true)
 |-- Cover_Type: integer (nullable = true)
```

This cell looks at the first few rows of the DataFrame.

```
fc_columns = fc.columns

fc.select(fc.columns[:6]).show(3)
fc.select(fc.columns[6:]).show(3)
```

```
+---------+------+-----+----------------------------------+-------------------------------+-----------------------------+
|elevation|aspect|slope|Horizontal_Distance_To_Hydrology|Vertical_Distance_To_Hydrology|Horizontal_Distance_To_Roadways|
+---------+------+-----+----------------------------------+-------------------------------+-----------------------------+
|     2596|    51|    3|                               258|                              0|                          510|
|     2590|    56|    2|                               212|                             -6|                          390|
|     2804|   139|    9|                               268|                             65|                         3180|
+---------+------+-----+----------------------------------+-------------------------------+-----------------------------+
only showing top 3 rows

+-------------+--------------+-------------+-----------------------------------+--------------+---------+----------+
|Hillshade_9am|Hillshade_Noon|Hillshade_3pm|Horizontal_Distance_To_Fire_Points|Wilderness_Area|Soil_Type|Cover_Type|
+-------------+--------------+-------------+-----------------------------------+--------------+---------+----------+
|          221|           232|          148|                               6279|          Rawah|       29|         5|
|          220|           235|          151|                               6225|          Rawah|       29|         5|
|          234|           238|          135|                               6121|          Rawah|       12|         2|
+-------------+--------------+-------------+-----------------------------------+--------------+---------+----------+
only showing top 3 rows
```

This cell determines the number of observations in the dataset

```
N = fc.count()
print(N)
```

```
15120
```

We will now determine the proportions of records in each of the two label categories.

```
(
    fc
    .groupBy('Cover_Type')
    .agg(expr('COUNT(*) AS n_type'))
    .withColumn('prop', expr(f'round(n_type/{N}, 4)'))
    .select('Cover_Type', 'prop')
    .show()
)
```

```
+----------+------+
|Cover_Type|  prop|
+----------+------+
|         1|0.1429|
|         6|0.1429|
|         3|0.1429|
|         5|0.1429|
|         4|0.1429|
|         7|0.1429|
|         2|0.1429|
+----------+------+
```

Part C: Preprocessing and Splitting the Data

This part of the project invovles preprocessing and splitting the data.

This cell creates the stages to be used in the preprocessing pipeline.

```
num_features = ['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']
cat_features = ['Wilderness_Area', 'Soil_Type']
ix_features = [c + '_ix' for c in cat_features]
vec_features = [c + '_vec' for c in cat_features]

indexer = StringIndexer(inputCols=cat_features, outputCols=ix_features)

encoder = OneHotEncoder(inputCols=ix_features, outputCols=vec_features, dropLast=False)

assembler_lr = VectorAssembler(inputCols=num_features + vec_features, outputCol='features_lr')

assembler_dt = VectorAssembler(inputCols=num_features + ix_features, outputCol='features_dt')
```

This cell creates a pipeline from the stages above and will apply this to the data.

```
pipeline = Pipeline(stages=[indexer, encoder, assembler_lr, assembler_dt]).fit(fc)
fc_proc = pipeline.transform(fc)
fc_proc.persist()

fc_proc.select(['features_dt', 'Cover_Type']).show(5, truncate=False)
```

```
+-----------------------------------------------------------------------+----------+
|features_dt                                                            |Cover_Type|
+-----------------------------------------------------------------------+----------+
|[2596.0,51.0,3.0,258.0,0.0,510.0,221.0,232.0,148.0,6279.0,2.0,1.0]     |5         |
|[2590.0,56.0,2.0,212.0,-6.0,390.0,220.0,235.0,151.0,6225.0,2.0,1.0]    |5         |
|[2804.0,139.0,9.0,268.0,65.0,3180.0,234.0,238.0,135.0,6121.0,2.0,20.0] |2         |
|[2785.0,155.0,18.0,242.0,118.0,3090.0,238.0,238.0,122.0,6211.0,2.0,6.0]|2         |
|[2595.0,45.0,2.0,153.0,-1.0,391.0,220.0,234.0,150.0,6172.0,2.0,1.0]    |5         |
+-----------------------------------------------------------------------+----------+
only showing top 5 rows
```

This cell creates training and test sets.

```
splits = fc_proc.randomSplit([0.8, 0.2], seed=1)
train = splits[0]
test = splits[1]

train.persist()
print('Training Observations:', train.count())
print('Testing Observations: ', test.count())
```

```
Training Observations: 12118
Testing Observations:  3002
```

Part D: Hyperparameters for Logistic Regression

This part of the project involves finding hyperparameters for logistic regression.

This cell creates an accuracy evaluator to be used when performing hyperparameter tuning.

```
accuracy_eval = MulticlassClassificationEvaluator(predictionCol='prediction', labelCol='Cover_Type', metricName='accuracy')
```

This cell uses grid search and cross-validation to perform hyperparameter tuning for logistic regression.

```
logreg = LogisticRegression(featuresCol='features_lr', labelCol='Cover_Type')

param_grid = (ParamGridBuilder()
              .addGrid(logreg.regParam, [0.00001, 0.0001, 0.001, 0.01, 0.1])
              .addGrid(logreg.elasticNetParam, [0, 0.5, 1])
             ).build()

cv = CrossValidator(estimator=logreg, estimatorParamMaps=param_grid, evaluator=accuracy_eval, numFolds=5, seed=1, parallelism=8)

cv_model = cv.fit(train)

MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
```

This cell identifies the optimal model found by the grid search algorithm.

```
lr_model = cv_model.bestModel

regParam = lr_model.getRegParam()
enetParam = lr_model.getElasticNetParam()

print('Max CV Score:  ', round(max(cv_model.avgMetrics),4))
print('Optimal Lambda:', regParam)
print('Optimal Alpha: ', enetParam)

Max CV Score:   0.6739
Optimal Lambda: 1e-05
Optimal Alpha:  0.0
```

This cell generates a plot to display the results of the cross-validation.

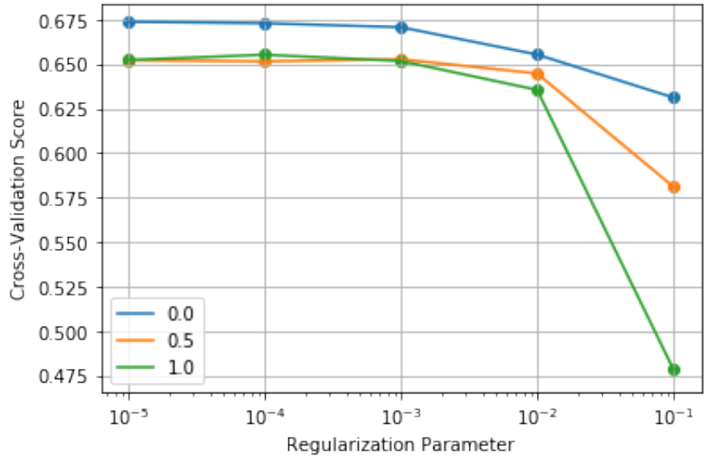
```
model_params = cv_model.getEstimatorParamMaps()

lr_cv_summary_list = []
for param_set, acc in zip(model_params, cv_model.avgMetrics):
    new_set = list(param_set.values()) + [acc]
    lr_cv_summary_list.append(new_set)

cv_summary = pd.DataFrame(lr_cv_summary_list, columns=['reg_param', 'enet_param', 'acc'])

for en in cv_summary.enet_param.unique():
    sel = cv_summary.enet_param == en
    plt.plot(cv_summary.reg_param[sel], cv_summary.acc[sel], label=en)
    plt.scatter(cv_summary.reg_param[sel], cv_summary.acc[sel])

plt.legend()
plt.xscale('log')
plt.grid()
plt.xlabel('Regularization Parameter')
plt.ylabel('Cross-Validation Score')
plt.show()
```



Part E: Hyperparameter Tuning for Decision Trees

This part of the project involves finding hyperparameters for decision trees.

This cell uses grid search and cross-validation to perform hyperparameter tuning for decision trees.

```
dtree = DecisionTreeClassifier(featuresCol='features_dt', labelCol='Cover_Type', seed=1, maxBins=38)

param_grid = (ParamGridBuilder()
              .addGrid(dtree.maxDepth, range(2,26,2))
              .addGrid(dtree.minInstancesPerNode, [1, 2, 4])
             ).build()

cv = CrossValidator(estimator=dtree, estimatorParamMaps=param_grid, numFolds=5, evaluator=accuracy_eval, seed=1, parallelism=6)

cv_model = cv.fit(train)

MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
```

This cell identifies the optimal model found by the grid search algorithm.

```
dt_model = cv_model.bestModel
maxDepth = dt_model.getMaxDepth()
minInstancesPerNode = dt_model.getMinInstancesPerNode()

print('Max CV Score:   ', round(max(cv_model.avgMetrics),4))
print('Optimal Depth:  ', maxDepth)
print('Optimal MinInst:', minInstancesPerNode)
```

```
Max CV Score:    0.7775
Optimal Depth:   16
Optimal MinInst: 1
```

This cell generates a plot to display the results of the cross-validation.

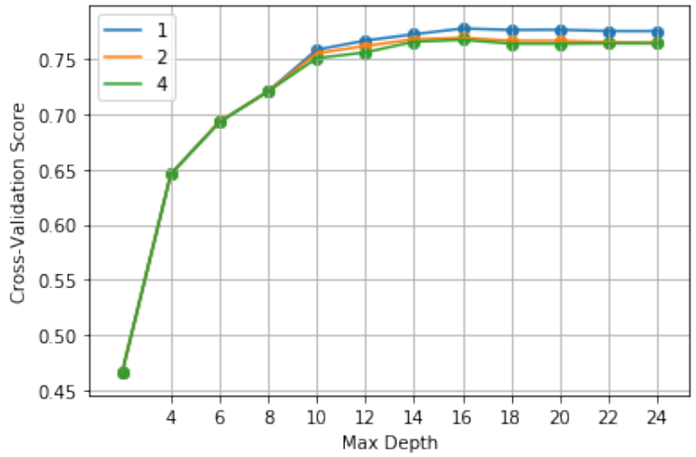
```
model_params = cv_model.getEstimatorParamMaps()

dt_cv_summary_list = []
for param_set, acc in zip(model_params, cv_model.avgMetrics):
    new_set = list(param_set.values()) + [acc]
    dt_cv_summary_list.append(new_set)

cv_summary = pd.DataFrame(dt_cv_summary_list, columns=['maxDepth', 'minInst', 'acc'])

for mi in cv_summary.minInst.unique():
    sel = cv_summary.minInst == mi
    plt.plot(cv_summary.maxDepth[sel], cv_summary.acc[sel], label=mi)
    plt.scatter(cv_summary.maxDepth[sel], cv_summary.acc[sel])

plt.legend()
plt.grid()
plt.xticks(range(4,26,2))
plt.xlabel('Max Depth')
plt.ylabel('Cross-Validation Score')
plt.show()
```



This cell displays the feature importance for the each of the features used in our optimal decision tree model.

```
features = num_features + cat_features
pd.DataFrame({'feature':features, 'importance':dt_model.featureImportances})
```

Out[16]:

	feature	importance
0	elevation	0.394260
1	aspect	0.048116
2	slope	0.026453
3	Horizontal_Distance_To_Hydrology	0.078278
4	Vertical_Distance_To_Hydrology	0.036748
5	Horizontal_Distance_To_Roadways	0.070875
6	Hillshade_9am	0.025840
7	Hillshade_Noon	0.023199
8	Hillshade_3pm	0.017314
9	Horizontal_Distance_To_Fire_Points	0.079365
10	Wilderness_Area	0.057567
11	Soil_Type	0.141985

Part F: Identifying and Evaluating the Final Model

This part of the project involves identifying and evaluating the final model.

The model dt_model seems to perform better on out-of-sample data; its cross-validation score is 0.7775, whereas the cross-validation score for lr_model is only 0.6739.

This cell will use the optimal model to generate predictions for the test set.

```
test_pred = dt_model.transform(test)
test_pred.select('probability', 'prediction', 'Cover_Type').show(10, truncate=False)
```

```
+--------------------------------------------------------------------------------+----------+----------+
|probability                                                                     |prediction|Cover_Type|
+--------------------------------------------------------------------------------+----------+----------+
|[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0]                                               |6.0       |6         |
|[0.0,0.0,0.0273972602739726,0.3150684931506849,0.0,0.0,0.6575342465753424,0.0]|6.0       |6         |
|[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0]                                               |6.0       |6         |
|[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0]                                               |3.0       |3         |
|[0.0,0.0,0.0,0.09375,0.0,0.0,0.90625,0.0]                                       |6.0       |6         |
|[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0]                                               |3.0       |3         |
|[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0]                                               |3.0       |6         |
|[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0]                                               |3.0       |3         |
|[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0]                                               |6.0       |3         |
|[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0]                                               |6.0       |6         |
+--------------------------------------------------------------------------------+----------+----------+
only showing top 10 rows
```

This cell will use the testpredictions to evaluate the final model's performance on out-of-sample data.

```
pred_and_labels = test_pred.rdd.map(lambda x:(x['prediction'],float(x['Cover_Type'])))

metrics = MulticlassMetrics(pred_and_labels)
print('Test Set Accuracy:', round(metrics.accuracy, 4))
```

```
Test Set Accuracy: 0.7818
```

This cell will display the confusion matrix for the test data.

```
labels = [1,2,3,4,5,6,7]

cm = metrics.confusionMatrix().toArray().astype(int)
pd.DataFrame(data=cm, columns=labels, index=labels)
```

Out[19]:

	1	2	3	4	5	6	7
1	311	90	2	0	13	5	36
2	101	220	12	0	58	11	3
3	1	10	306	23	13	78	0
4	0	1	22	423	0	25	0
5	9	26	3	0	364	4	0
6	1	4	58	13	5	319	0
7	19	7	0	0	2	0	404

Observations in the test set with Cover Type 2 were misclassified by the model as Cover Type 1 a total of 101 times. This was the most common type of misclassification in the test set.

This cell displays the precision and recall for all 7 label classes.

```
print('Label   Precision   Recall')
print('-------------------------')
for i, lab in enumerate(labels):
    print(f'{lab:<8}{metrics.precision(i+1):<12.4f}{metrics.recall(i+1):.4f}')
```

```
Label   Precision   Recall
```

```
--------------------------
1       0.7036      0.6805
2       0.6145      0.5432
3       0.7593      0.7100
4       0.9216      0.8981
5       0.8000      0.8966
6       0.7217      0.7975
7       0.9120      0.9352
```

1. Cover type 7 is most likely to be correctly classified; it has the highest recall value and a high precision value, and the two values are not far apart form each other.
2. Cover type 2 is most likely to be misclassified; it has the lowest precision and recall values.
3. Cover type 5 has the greatest difference between its precision (the probability that a prediction of Type 5 will actually be correct) and its recall (the probability that an observation of Type 5 will be correctly identified).