STEVENS INSTITUTE OF TECHNOLOGY

EE627A: Machine Learning with PySpark and MLlib

Rensheng Wang,
https://sit.instructure.com/courses/33244
November 12, 2019

Spark and PySpark

- Apache Spark, once a component of the Hadoop ecosystem, is now becoming the big-data platform of choice for enterprises.
- It is a powerful open source engine that provides real-time stream processing, interactive processing, graph processing, in-memory processing as well as batch processing with very fast speed, ease of use and standard interface.
- The use of PySpark is to write Spark apps in Python.
- PySpark is the python shell of spark. Spark is a computing engine, possibly taking charge or distributing files to nodes, collect them and return them.

Exploring The Data

- We built a Logistic Regression example in Python and it is related to direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict whether the client will subscribe (Yes/No) to a term deposit.
- Sample code:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('ml-bank').getOrCreate()
df = spark.read.csv('bank.csv', header = True, inferSchema = True)
df.printSchema()
```

```
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
```

Exploring The Data

- Input variables:

  age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, poutcome.

- Output variable:

  deposit

- Have a peek of the first five observations.

  Pandas data frame is prettier than Spark DataFrame.show().

```
import pandas as pd
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

Exploring The Data

```
import pandas as pd
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

	0	1	2	3	4
age	59	56	41	55	54
job	admin.	admin.	technician	services	admin.
marital	married	married	married	married	married
education	secondary	secondary	secondary	secondary	tertiary
default	no	no	no	no	no
balance	2343	45	1270	2476	184
housing	yes	no	yes	yes	no
loan	no	no	no	no	no
contact	unknown	unknown	unknown	unknown	unknown
day	5	5	5	5	5
month	may	may	may	may	may
duration	1042	1467	1389	579	673
campaign	1	1	1	1	2
pdays	-1	-1	-1	-1	-1
previous	0	0	0	0	0
poutcome	unknown	unknown	unknown	unknown	unknown
deposit	yes	yes	yes	yes	yes

Exploring The Data

- Check if the two classes balanced:

```
import pandas as pd
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

	deposit	count
0	no	5873
1	yes	5289

- Summary statistics for numeric variables

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int']
df.select(numeric_features).describe().toPandas().transpose()
```

Exploring The Data

- Summary statistics for numeric variables

```
numeric_features = [t[0] for t in df.dtypes if t[1] == 'int']
df.select(numeric_features).describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
age	11162	41.231947679627304	11.913369192215518	18	95
balance	11162	1528.5385235620856	3225.413325946149	-6847	81204
day	11162	15.658036194230425	8.420739541006462	1	31
duration	11162	371.99381831213043	347.12838571630687	2	3881
campaign	11162	2.508421429851281	2.7220771816614824	1	63
pdays	11162	51.33040673714388	108.75828197197717	-1	854
previous	11162	0.8325568894463358	2.292007218670508	0	58

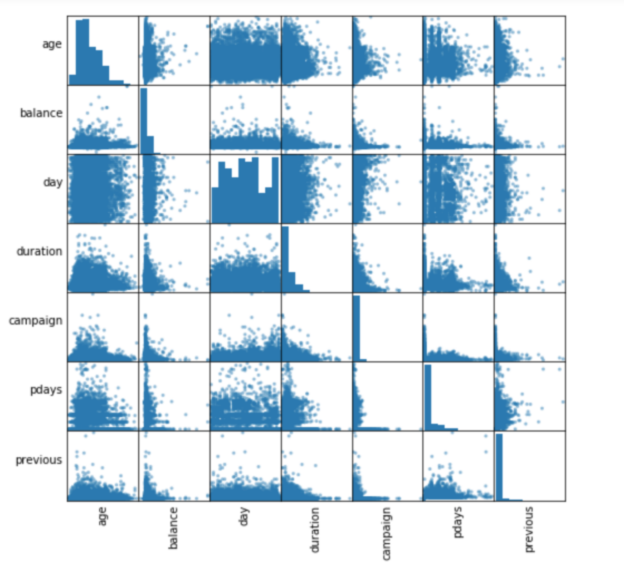
Correlation

- Correlations between independent variables.

```
numeric_data = df.select(numeric_features).toPandas()
axs = pd.scatter_matrix(numeric_data, figsize=(8, 8));
n = len(numeric_data.columns)
for i in range(n):
v = axs[i, 0]
v.yaxis.label.set_rotation(0)
v.yaxis.label.set_ha('right')
v.set_yticks(())
h = axs[n-1, i]
h.xaxis.label.set_rotation(90)
h.set_xticks(())
```

Correlation

- Correlations between independent variables.



Clean Features

- Its obvious that there arent highly correlated numeric variables.
- Therefore, we will keep all of them for the model. However, day and month columns are not really useful, we will remove these two columns.

```
df = df.select('age', 'job', 'marital', 'education', 'default',
'balance', 'housing', 'loan', 'contact', 'duration', 'campaign',
'pdays', 'previous', 'poutcome', 'deposit')
cols = df.columns
df.printSchema()
```

Clean Features

- df.printSchema()

```
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
```

Preparing Data for Machine Learning

- The process includes Category Indexing, One-Hot Encoding and VectorAssembler a feature transformer that merges multiple columns into a vector column.

```
from pyspark.ml.feature import OneHotEncoderEstimator,
StringIndexer, VectorAssembler
categoricalColumns = ['job', 'marital', 'education', 'default',
'housing', 'loan', 'contact', 'poutcome']
stages = []
for categoricalCol in categoricalColumns:

stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol =
categoricalCol + 'Index')
encoder=OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()],
outputCols=[categoricalCol + "classVec"])
stages += [stringIndexer, encoder]
```

Preparing Data for Machine Learning

```
label_stringIdx = StringIndexer(inputCol = 'deposit', outputCol =
'label')
stages += [label_stringIdx]
numericCols = ['age', 'balance', 'duration', 'campaign', 'pdays',
'previous']
assemblerInputs = [c + "classVec" for c in categoricalColumns] +
numericCols
assembler = VectorAssembler(inputCols=assemblerInputs,
outputCol="features")
stages += [assembler]
```

- The above code are taken from databricks official site and it indexes each categorical column using the StringIndexer, then converts the indexed categories into one-hot encoded variables.
- The resulting output has the binary vectors appended to the end of each row. We use the StringIndexer again to encode our labels to label indices.
- We use the VectorAssembler to combine all the feature columns into a single vector column.

Pipeline

- We use Pipeline to chain multiple Transformers and Estimators together to specify our machine learning workflow.
- A Pipelines stages are specified as an ordered array.

```
from pyspark.ml import Pipeline
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['label', 'features'] + cols
df = df.select(selectedCols)
df.printSchema()
```

Pipeline

- A Pipelines stages are specified as an ordered array.

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
```

Pipeline

```
pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

	0	1	2	3	4
label	1	1	1	1	1
features	(0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...	(0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
age	59	56	41	55	54
job	admin.	admin.	technician	services	admin.
marital	married	married	married	married	married
education	secondary	secondary	secondary	secondary	tertiary
default	no	no	no	no	no
balance	2343	45	1270	2476	184
housing	yes	no	yes	yes	no
loan	no	no	no	no	no
contact	unknown	unknown	unknown	unknown	unknown
duration	1042	1467	1389	579	673
campaign	1	1	1	1	2
pdays	-1	-1	-1	-1	-1
previous	0	0	0	0	0
poutcome	unknown	unknown	unknown	unknown	unknown
deposit	yes	yes	yes	yes	yes

Pipeline

- As we might already know, maximizing As you can see, we now have features column and label column.
- Randomly split data into train and test sets, and set seed for reproducibility.
- Python code:

```
train, test = df.randomSplit([0.7, 0.3], seed = 2018)
print("Training Dataset Count:  " + str(train.count()))
print("Test Dataset Count:  " + str(test.count()))
```

Training Dataset Count: 7764

Test Dataset Count: 3398

Logistic Regression Model

- Use logistic regression from the pySpark machine learning library:

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol =
'label', maxIter=10)
lrModel = lr.fit(train)
```

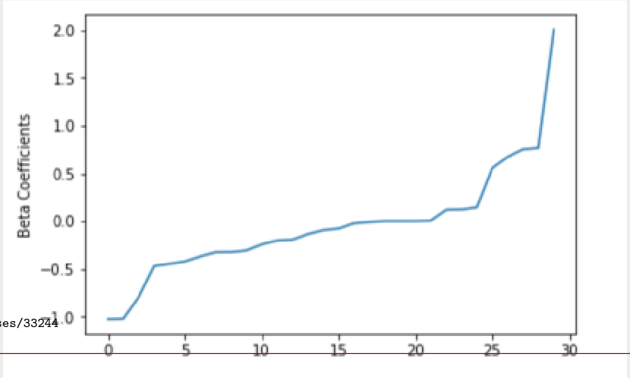
- We can obtain the coefficients by using LogisticRegressionModels attributes.

```
import matplotlib.pyplot as plt
import numpy as np
beta = np.sort(lrModel.coefficients)
plt.plot(beta)
plt.ylabel('Beta Coefficients')
plt.show()
```

Logistic Regression Model

- We can obtain the coefficients by using LogisticRegressionModels attributes.

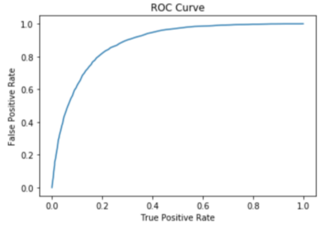
```
import matplotlib.pyplot as plt
import numpy as np
beta = np.sort(lrModel.coefficients)
plt.plot(beta)
plt.ylabel('Beta Coefficients')
plt.show()
```



ROC & AUC

- Summarize the model over the training set, we can also obtain the receiver-operating characteristic and areaUnderROC.

```
trainingSummary = lrModel.summary
roc = trainingSummary.roc.toPandas(d)
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```




```
Training set areaUnderROC: 0.8849092421146739
```

Prediction on Test

- Make predictions on the test set.

```
predictions = lrModel.transform(test)
predictions.select('age', 'job', 'label', 'rawPrediction',
'prediction', 'probability').show(10)
```

age	job	label	rawPrediction	prediction	probability
37	management	0.0	[1.19871810716723...	0.0	[0.76829666339830...
40	management	0.0	[2.20534940465796...	0.0	[0.90072886169926...
53	management	0.0	[1.02590348276690...	0.0	[0.73612093009497...
32	management	0.0	[1.25795481657702...	0.0	[0.77867383994058...
54	management	0.0	[1.33232096924268...	0.0	[0.79122429116078...
40	management	0.0	[1.57095096412779...	0.0	[0.82791913346617...
56	management	0.0	[3.06095963426752...	0.0	[0.95525333386804...
50	management	0.0	[-0.8102603273804...	1.0	[0.30783502428597...
47	management	0.0	[0.67024288891379...	0.0	[0.66155754396054...
44	management	0.0	[1.29756265761715...	0.0	[0.78542449653716...

only showing top 10 rows

Prediction on Test

- Evaluate our Logistic Regression model.

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(predictions))
```

- Test Area Under ROC 0.8858

Decision Tree Classifier

- Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multi-class classification, do not require feature scaling, and are able to capture non-linearities and feature interactions.

```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label',
maxDepth = 3)
dtModel = dt.fit(train)
predictions = dtModel.transform(test)
predictions.select('age', 'job', 'label', 'rawPrediction', 'prediction',
'probability').show(10)
```

Decision Tree Classifier

- Decision Tree:

age	job	label	rawPrediction	prediction	probability
37	management	0.0	[1130.0,387.0]	0.0	[0.74489123269611...
40	management	0.0	[1333.0,86.0]	0.0	[0.93939393939393...
53	management	0.0	[1130.0,387.0]	0.0	[0.74489123269611...
32	management	0.0	[1130.0,387.0]	0.0	[0.74489123269611...
54	management	0.0	[1333.0,86.0]	0.0	[0.93939393939393...
40	management	0.0	[373.0,30.0]	0.0	[0.92555831265508...
56	management	0.0	[1333.0,86.0]	0.0	[0.93939393939393...
50	management	0.0	[788.0,1230.0]	1.0	[0.39048562933597...
47	management	0.0	[788.0,1230.0]	1.0	[0.39048562933597...
44	management	0.0	[1130.0,387.0]	0.0	[0.74489123269611...

only showing top 10 rows

Evaluate Decision Tree Classifier

- `evaluator = BinaryClassificationEvaluator()`
  `print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, evaluator.metricName: "areaUnderROC")))`
- Test Area Under ROC: 0.78072
- One simple decision tree performed poorly because it is too weak given the range of different features.
- The prediction accuracy of decision trees can be improved by Ensemble methods, such as Random Forest and Gradient-Boosted Tree.

Random Forest Classifier

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
rfModel = rf.fit(train)
predictions = rfModel.transform(test)
predictions.select('age', 'job', 'label', 'rawPrediction', 'prediction',
'probability').show(10)
```

```
+---+----------+-----+--------------------+----------+--------------------+
|age|       job|label|       rawPrediction|prediction|         probability|
+---+----------+-----+--------------------+----------+--------------------+
| 37|management|  0.0|[14.0335043427458...|       0.0|[0.70167521713729...|
| 40|management|  0.0|[15.5696428848403...|       0.0|[0.77848214424201...|
| 53|management|  0.0|[14.0604626680319...|       0.0|[0.70302313340159...|
| 32|management|  0.0|[14.9300431640802...|       0.0|[0.74650215820401...|
| 54|management|  0.0|[14.4483343905905...|       0.0|[0.72241671952952...|
| 40|management|  0.0|[15.0798534256099...|       0.0|[0.75399267128049...|
| 56|management|  0.0|[18.2682164556330...|       0.0|[0.91341082278165...|
| 50|management|  0.0|[5.99972486043756...|       1.0|[0.29998624302187...|
| 47|management|  0.0|[10.8585049161417...|       0.0|[0.54292524580708...|
| 44|management|  0.0|[10.6040194546245...|       0.0|[0.53020097273122...|
+---+----------+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

Evaluate Random Forest Classifier

- `evaluator = BinaryClassificationEvaluator()`
  `print("Test Area Under ROC: " + str(evaluator.evaluate(predictions, evaluator.metricName: "areaUnderROC")))`
- Test Area Under ROC: 0.8846

Gradient-Boosted Tree Classifier

- ```
  from pyspark.ml.classification import GBTClassifier
  gbt = GBTClassifier(maxIter=10)
  gbtModel = gbt.fit(train)
  predictions = gbtModel.transform(test)
  predictions.select('age', 'job', 'label', 'rawPrediction', 'prediction',
  'probability').show(10)
  ```

```
+---+----------+-----+--------------------+----------+--------------------+
|age|       job|label|       rawPrediction|prediction|         probability|
+---+----------+-----+--------------------+----------+--------------------+
| 37|management|  0.0|[0.57808138910181...|       0.0|[0.76063477260811...|
| 40|management|  0.0|[1.37467582901950...|       0.0|[0.93987672346171...|
| 53|management|  0.0|[-0.0012929624008...|       1.0|[0.49935351915983...|
| 32|management|  0.0|[0.61900313605401...|       0.0|[0.77521678642033...|
| 54|management|  0.0|[0.98157815641818...|       0.0|[0.87687413211579...|
| 40|management|  0.0|[0.96138354833170...|       0.0|[0.87244668327834...|
| 56|management|  0.0|[1.39120025731353...|       0.0|[0.94171733839668...|
| 50|management|  0.0|[-0.6141629093446...|       1.0|[0.22647458093662...|
| 47|management|  0.0|[-0.0439971283470...|       1.0|[0.47801561939801...|
| 44|management|  0.0|[0.26452511568224...|       0.0|[0.62926156628314...|
+---+----------+-----+--------------------+----------+--------------------+
only showing top 10 rows
```
```

Evaluate Gradient-Boosted Tree Classifier

- ```
  evaluator = BinaryClassificationEvaluator()
  print("Test Area Under ROC: " + str(evaluator.evaluate(predictions,
  evaluator.metricName:  "areaUnderROC")))
  ```
- Test Area Under ROC: 0.89407
- Gradient-Boosted Tree achieved the best results, we will try tuning this model with the `ParamGridBuilder` and the `CrossValidator`.
- Before that we can use `explainParams()` to print a list of all params and their definitions to understand what params available for tuning.
  ```

Explain Params in Gradient-Boosted Tree Classifier

- `print(gbt.explainParams())`

```
cacheNodeIds: If false, the algorithm will pass trees to executors to match instances with nodes. If true, the algori
thm will cache node IDs for each instance. Caching can speed up training of deeper trees. Users can set how often sho
uld the cache be checkpointed or disable it by setting checkpointInterval. (default: False)
checkpointInterval: 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 Sp
arkContext. (default: 10)
featuresCol: features column name. (default: features)
labelCol: label column name. (default: label)
lossType: Loss function which GBT tries to minimize (case-insensitive). Supported options: logistic (default: logisti
c)
maxBins: Max number of bins for discretizing continuous features.  Must be >=2 and >= number of categories for any ca
tegorical feature. (default: 32)
maxDepth: Maximum depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf n
odes. (default: 5)
maxIter: max number of iterations (>= 0). (default: 20, current: 10)
maxMemoryInMB: Maximum memory in MB allocated to histogram aggregation. If too small, then 1 node will be split per i
teration, and its aggregates may exceed this size. (default: 256)
minInfoGain: Minimum information gain for a split to be considered at a tree node. (default: 0.0)
minInstancesPerNode: Minimum number of instances each child must have after split. If a split causes the left or righ
t child to have fewer than minInstancesPerNode, the split will be discarded as invalid. Should be >= 1. (default: 1)
predictionCol: prediction column name. (default: prediction)
seed: random seed. (default: -7674267899484452859)
stepSize: Step size (a.k.a. learning rate) in interval (0, 1] for shrinking the contribution of each estimator. (defa
ult: 0.1)
subsamplingRate: Fraction of the training data used for learning each decision tree, in range (0, 1]. (default: 1.0)
```

Cross Validator in Gradient-Boosted Tree Classifier

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
paramGrid = (ParamGridBuilder()
.addGrid(gbt.maxDepth, [2, 4, 6])
.addGrid(gbt.maxBins, [20, 60])
.addGrid(gbt.maxIter, [10, 20])
.build())
cv = CrossValidator(estimator=gbt, estimatorParamMaps=paramGrid,
evaluator=evaluator, numFolds=5)
# Run cross validations.  This can take about 6 to 10 minutes since it
is training over 20 trees!
cvModel = cv.fit(train)
predictions = cvModel.transform(test)
evaluator.evaluate(predictions)
```