TERM PAPER
Submitted by: SUMANA SATPATI

Machine Learning (ML) Pipelines

Analyzing a bike sharing dataset

This term paper demonstrates creating a ML Pipeline to preprocess a dataset, train a Machine Learning model, and make predictions.

Data: The dataset contains bike rental information from 2011 and 2012 in the Capital bikeshare system with additional relevant information such as weather.

Goal: In this term paper, I want to learn to predict bike rental counts (per hour) from information such as day of the week, weather, season, etc. Having good predictions of customer demand allows a business or service to prepare and increase supply as needed.

Approach: Spark ML Pipelines is used here, which help users piece together parts of a workflow such as feature processing and model training. I will also demonstrate model selection (a.k.a. hyperparameter tuning) using Cross Validation in order to fine-tune and improve the ML model.

Load and understand the data

I begin by loading the data, which is stored in Comma-Separated Value (CSV) format. For that, the CSV datasource for Spark is used here, which creates a Spark DataFrame containing the dataset. The data is also cached so that I can read it from disk once.

Data description

From the UCI ML Repository description, we know that the columns have the following meanings.

Feature columns:

- dteday: date
- season: season (1:spring, 2:summer, 3:fall, 4:winter)
- yr: year (0:2011, 1:2012)
- mnth: month (1 to 12)
- hr: hour (0 to 23)

- holiday: whether day is holiday or not
- weekday: day of the week
- workingday: if day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit:
  - 1: Clear, Few clouds, Partly cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)
- atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-16, t_max=+50 (only in hourly scale)
- hum: Normalized humidity. The values are divided to 100 (max)
- windspeed: Normalized wind speed. The values are divided to 67 (max)

Label columns:

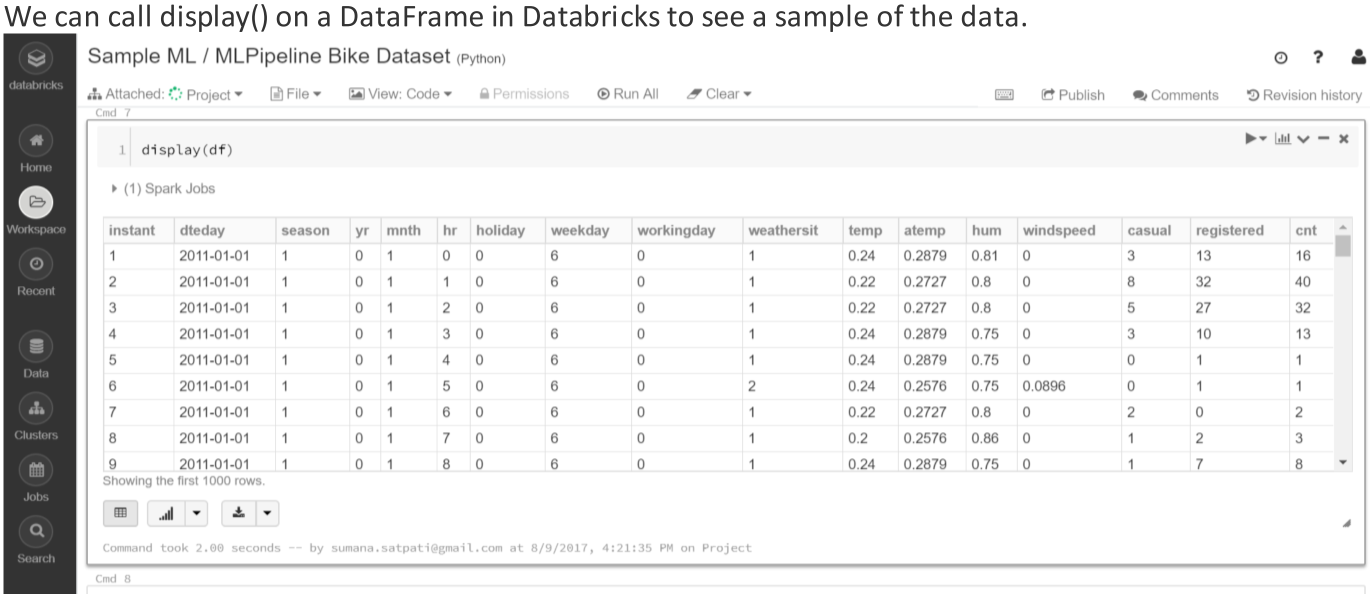
- casual: count of casual users
- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

Extraneous columns:

- instant: record index

For example, the first row is a record of hour 0 on January 1, 2011 and apparently 16 people rented bikes around midnight.

We can call display() on a DataFrame in Databricks to see a sample of the data.



Sample ML / MLPipeline Bike Dataset (Python)

```
display(df)
```

(1) Spark Jobs

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	3	13	16
2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	8	32	40
3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	5	27	32
4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	3	10	13
5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	0	1	1
6	2011-01-01	1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	0	1	1
7	2011-01-01	1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2	0	2
8	2011-01-01	1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	1	2	3
9	2011-01-01	1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	1	7	8

Showing the first 1000 rows.

Command took 2.00 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:21:35 PM on Project

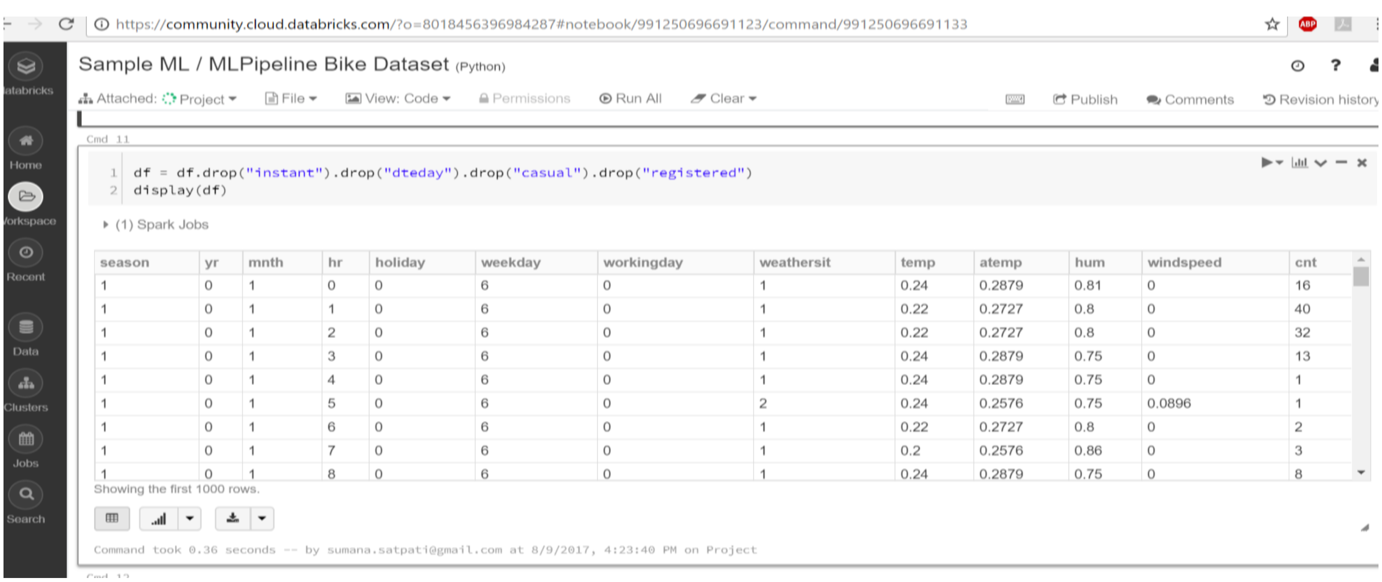
This dataset is nicely prepared for Machine Learning: values such as weekday are already indexed, and all the columns except the date (dteday) are numeric.

Preprocess data

So how the data can be made ready for Machine Learning?
Recall the goal: To learn to predict the count of bike rentals (the cnt column). I have referred to the count as the target "label".
Features: What can be used as features (info describing each row) to predict the cnt label? The rest of the columns can be used here, with a few exceptions:

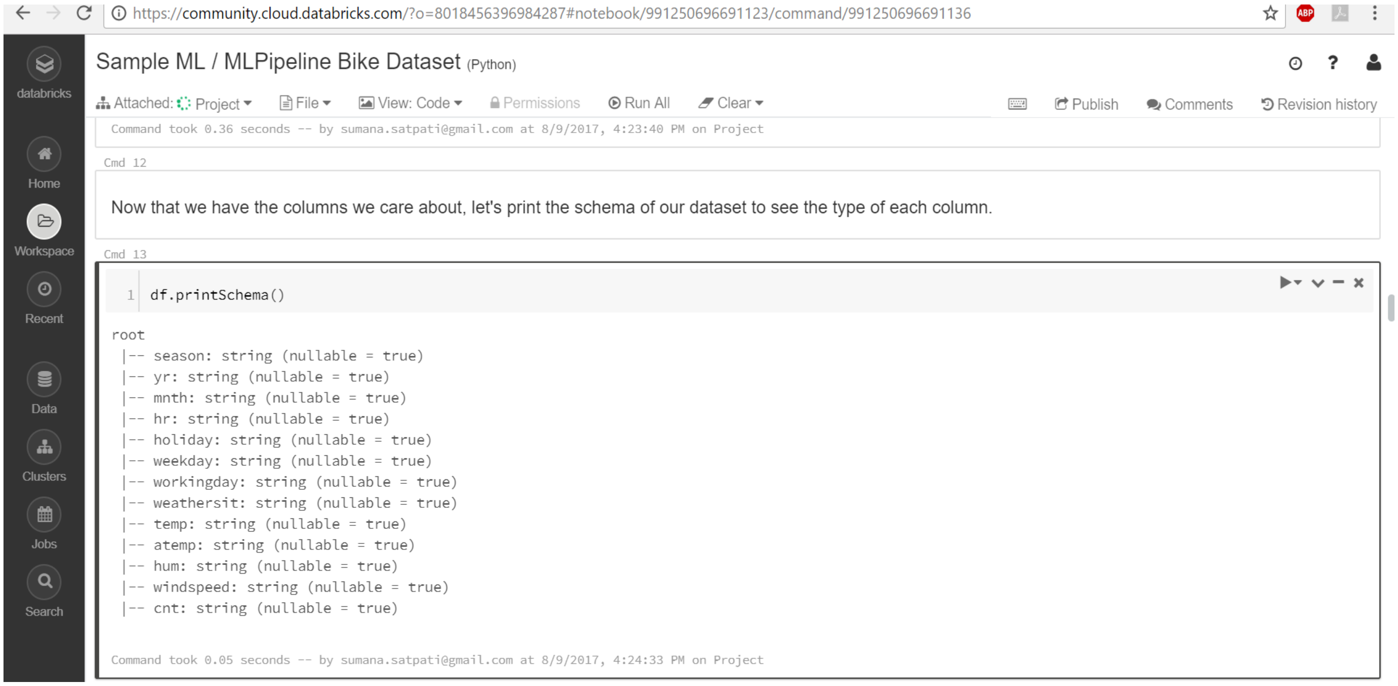
- Some of the columns contain duplicate information. For example, the cnt column I want to predict equals the sum of the casual + registered columns. The casual and registered columns can be removed from the data to make sure I do not use them to predict cnt.
- date column dteday: I could keep it, but it is well-represented by the other date-related columns like season, yr, mnth, and weekday. So, I will discard it.
- row index column instant: This is a useless column and hence removed.

Terminology: *Examples* are rows of the dataset. Each example contains the label to predict, plus features describing it.



season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt
1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	16
1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	40
1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	32
1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	13
1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	1
1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	1
1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2
1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	3
1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	8

Now let's print the schema of the dataset to see the type of each column.



https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691136

Sample ML / MLPipeline Bike Dataset (Python)

Command took 0.36 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:23:40 PM on Project

Cmd 12

Now that we have the columns we care about, let's print the schema of our dataset to see the type of each column.

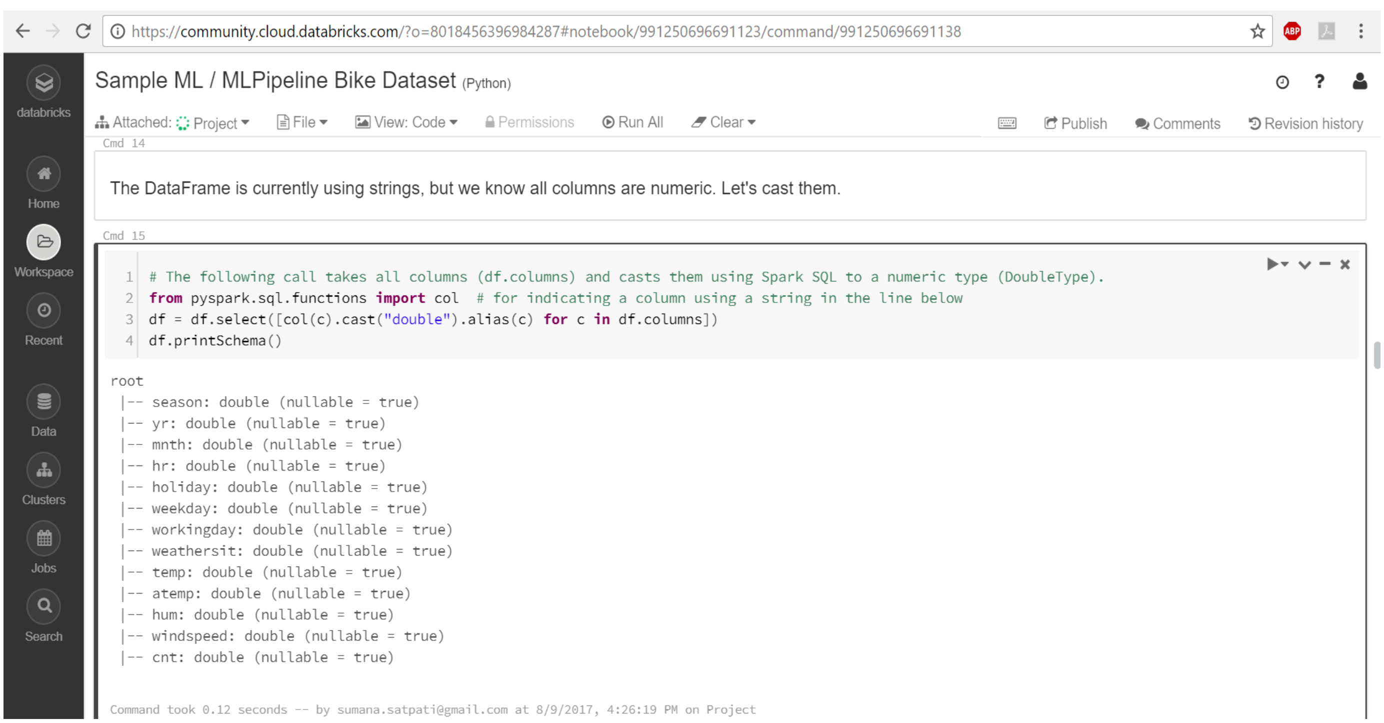
Cmd 13

```
df.printSchema()
```

```
root
 |-- season: string (nullable = true)
 |-- yr: string (nullable = true)
 |-- mnth: string (nullable = true)
 |-- hr: string (nullable = true)
 |-- holiday: string (nullable = true)
 |-- weekday: string (nullable = true)
 |-- workingday: string (nullable = true)
 |-- weathersit: string (nullable = true)
 |-- temp: string (nullable = true)
 |-- atemp: string (nullable = true)
 |-- hum: string (nullable = true)
 |-- windspeed: string (nullable = true)
 |-- cnt: string (nullable = true)
```

Command took 0.05 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:24:33 PM on Project

The DataFrame is currently using strings, but its already known that all columns are numeric. Let's cast them.



https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691138

Sample ML / MLPipeline Bike Dataset (Python)

Cmd 14

The DataFrame is currently using strings, but we know all columns are numeric. Let's cast them.

Cmd 15

```
# The following call takes all columns (df.columns) and casts them using Spark SQL to a numeric type (DoubleType).
from pyspark.sql.functions import col  # for indicating a column using a string in the line below
df = df.select([col(c).cast("double").alias(c) for c in df.columns])
df.printSchema()
```

```
root
 |-- season: double (nullable = true)
 |-- yr: double (nullable = true)
 |-- mnth: double (nullable = true)
 |-- hr: double (nullable = true)
 |-- holiday: double (nullable = true)
 |-- weekday: double (nullable = true)
 |-- workingday: double (nullable = true)
 |-- weathersit: double (nullable = true)
 |-- temp: double (nullable = true)
 |-- atemp: double (nullable = true)
 |-- hum: double (nullable = true)
 |-- windspeed: double (nullable = true)
 |-- cnt: double (nullable = true)
```

Command took 0.12 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:26:19 PM on Project

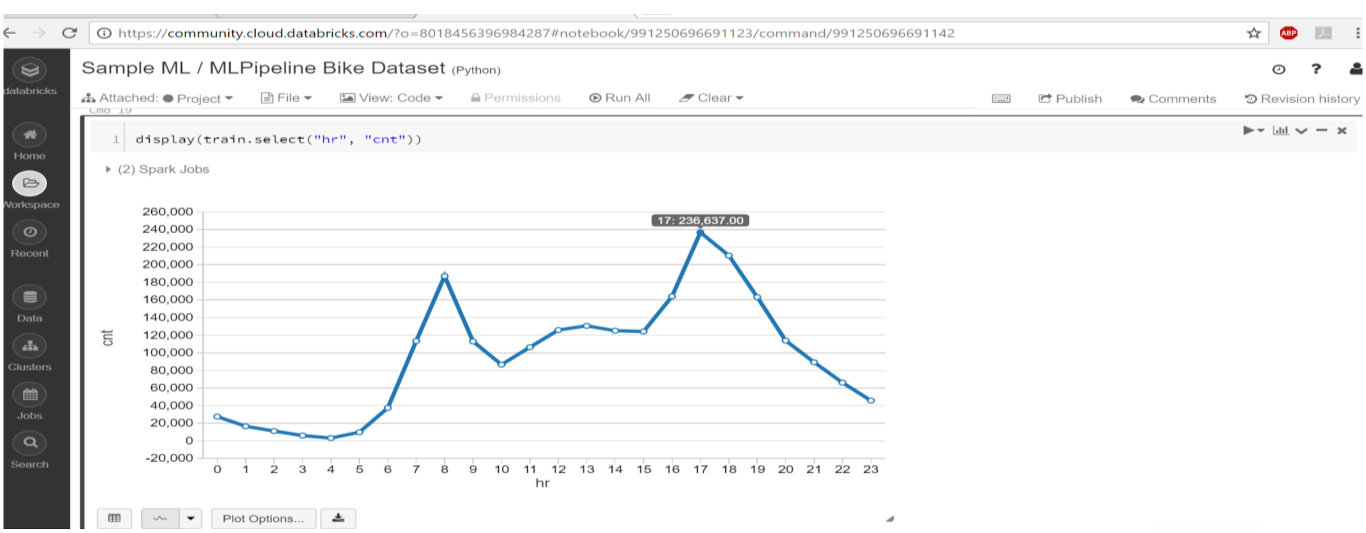
Split data into training and test sets

Now in the final data preparation step, I will split the dataset into separate training and test sets. I can train and tune the model as much as I like on the training set, as long as I do not look at the test set. After I have a good model (based on the training set), I can validate it on the held-out test set in order to know with high confidence how well the model will make predictions on future (unseen) data.

Visualize the data

Now that the features are preprocessed and a training dataset is prepared, I can visualize the data to get a sense of whether the features are meaningful.

Calling display() on a DataFrame in Databricks and clicking the plot icon below the table will let me draw and pivot various plots. In the below plot, bike rental counts versus hour of the day are compared. As one might expect, rentals are low during the night, and they peak in the morning (8am) and in the early evening (6pm). This indicates the hr feature is useful and can help predict the label cnt.



Train a Machine Learning Pipeline

Now that I have understood the data and prepared it as a DataFrame with numeric values, let's learn a ML model to predict bike sharing rentals in the future. Most ML algorithms expect to predict a single "label" column (cnt for this dataset) using a single "features" column of feature vectors. For each row in the data, the feature vector should describe what are known: weather, day of the week, etc., and the label should be what I want to predict (cnt).

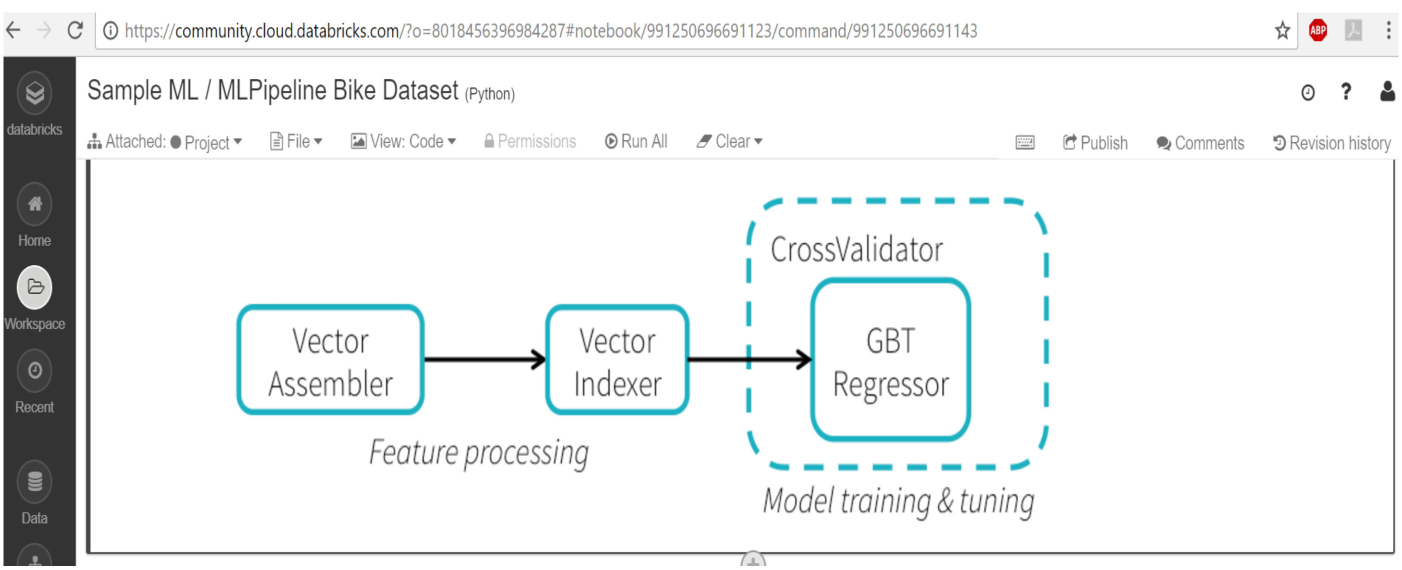
I will put together a simple Pipeline with the following stages:

VectorAssembler: Assemble the feature columns into a feature vector.

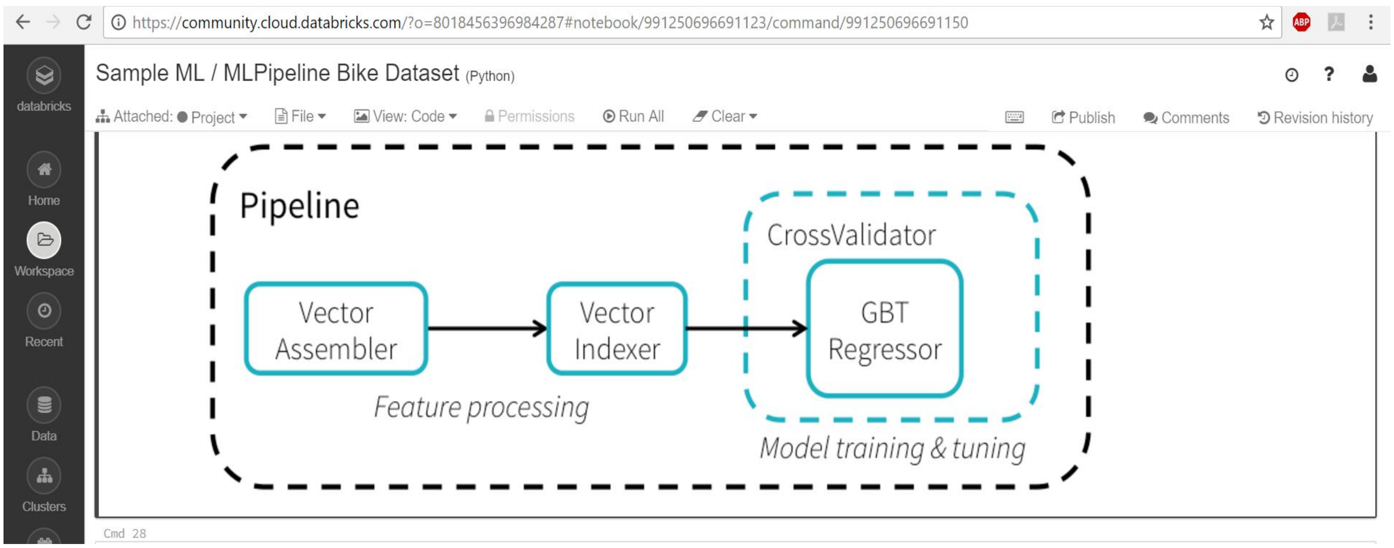
VectorIndexer: Identify columns which should be treated as categorical. This is done heuristically, identifying any column with a small number of distinct values as being categorical. This will be the yr (2 values), season (4 values), holiday (2 values), workingday (2 values), and weathersit (4 values).

GBTRegressor: This will use the Gradient-Boosted Trees (GBT) algorithm to learn how to predict rental counts from the feature vectors.

CrossValidator: The GBT algorithm has several hyperparameters, and tuning them to the data can improve accuracy. I will do this tuning using Spark's Cross Validation framework, which automatically tests a grid of hyperparameters and chooses the best.



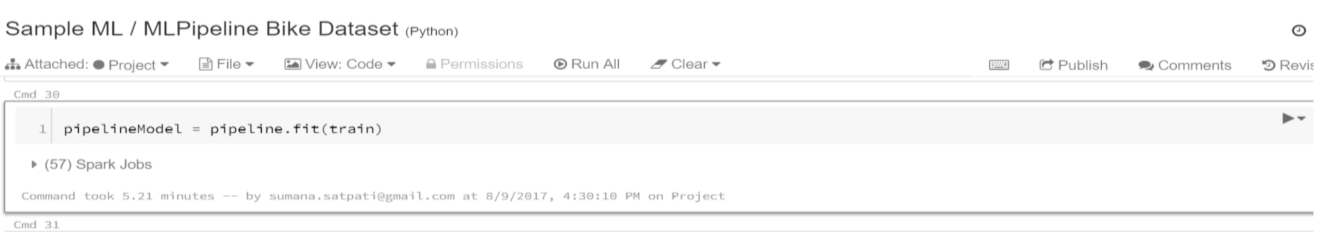
- Firstly, the feature processing stages of the Pipeline are defined:
  - Assemble feature columns into a feature vector.
  - Identify categorical features, and index them.
- Secondly, the model training stage of the Pipeline is defined. GBTRegressor takes feature vectors and labels as input and learns to predict labels of new examples.
- Thirdly, the model training stage is wrapped within a CrossValidator stage. CrossValidator knows how to call the GBT algorithm with different hyperparameter settings. It will train multiple models and choose the best one, based on minimizing some metric. In this example, the metric is Root Mean Squared Error (RMSE).
- Finally, the feature processing and model training stages are tied together into a single Pipeline.



Train the Pipeline!

Now that the workflow is set up, I can train the Pipeline in a single call. Calling fit() will run feature processing, model tuning, and training in a single call. I can get back a fitted Pipeline with the best model found.

- For each random sample of data in Cross Validation,
  - For each setting of the hyperparameters,
    - CrossValidator is training a separate GBT ensemble which contains many Decision Trees.

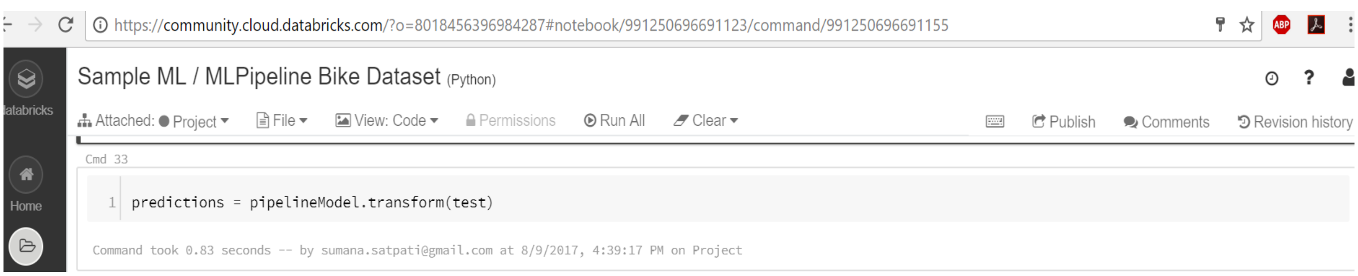


Make predictions, and evaluate results

The final step will be to use the fitted model to make predictions on new data. I will use the held-out test set, but this model could be used to make predictions on completely new data. For example, if some features data were created based on weather predictions for the next week, I could predict bike rentals expected during the next week.

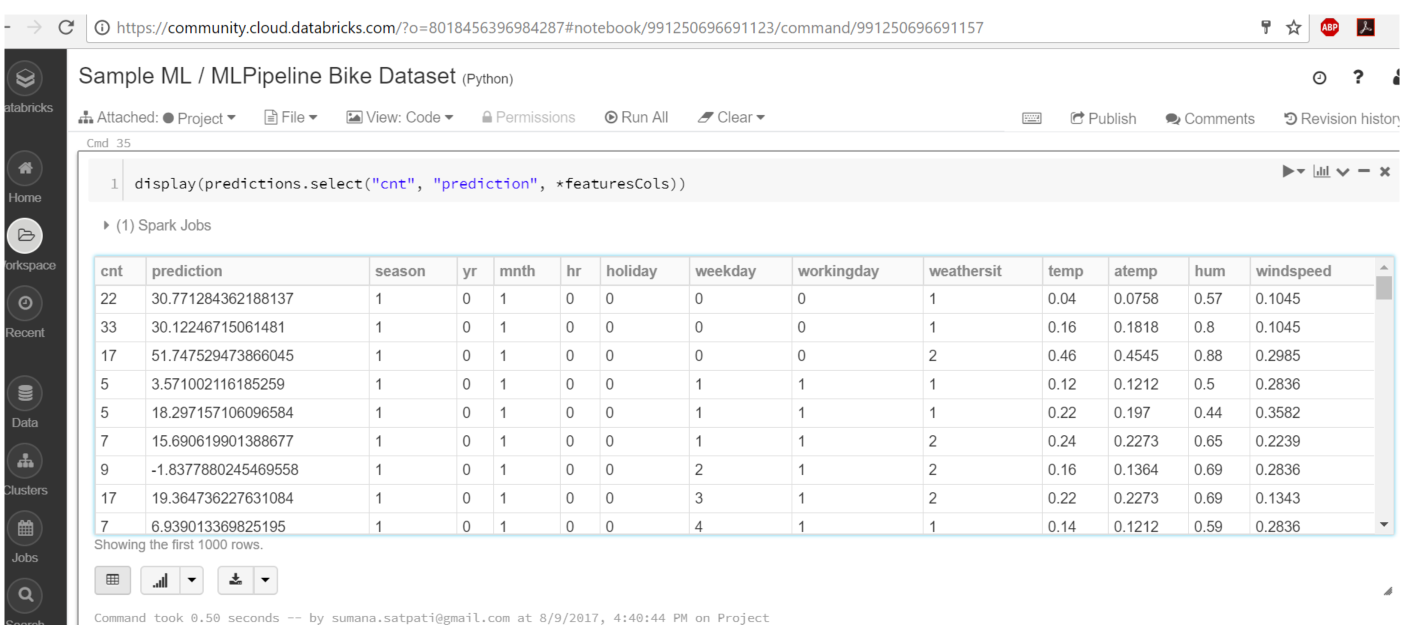
I will also evaluate the predictions. Computing evaluation metrics is important for understanding the quality of predictions, as well as for comparing models and tuning parameters.

Calling transform() on a new dataset passes that data through feature processing and uses the fitted model to make predictions. I get back a DataFrame with a new column predictions (as well as intermediate results such as the rawFeatures column from feature processing).



The columns displayed are listed below:

- cnt: the true count of bike rentals
- prediction: the predicted count of bike rentals
- feature columns: the original (human-readable) feature columns

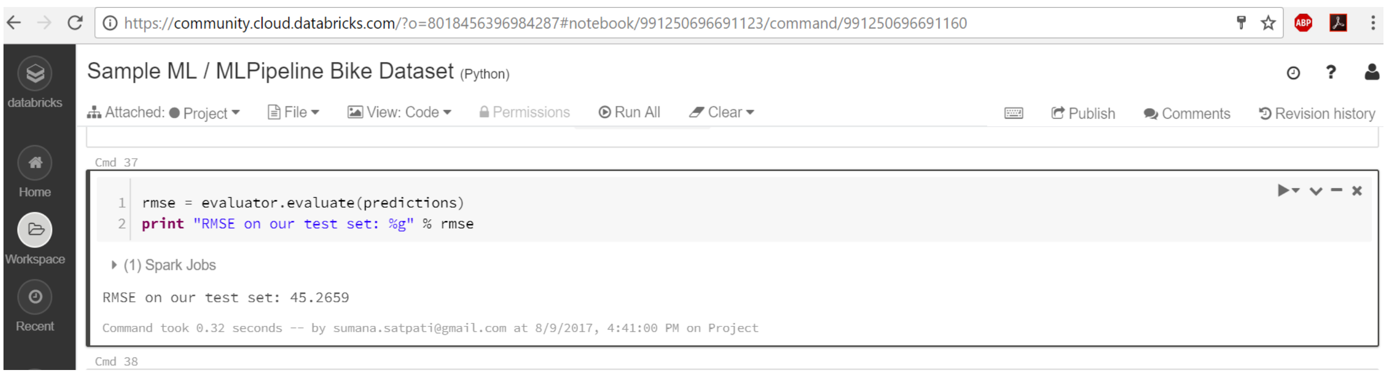


cnt	prediction	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
22	30.771284362188137	1	0	1	0	0	0	0	1	0.04	0.0758	0.57	0.1045
33	30.12246715061481	1	0	1	0	0	0	0	1	0.16	0.1818	0.8	0.1045
17	51.747529473866045	1	0	1	0	0	0	0	2	0.46	0.4545	0.88	0.2985
5	3.571002116185259	1	0	1	0	0	1	1	1	0.12	0.1212	0.5	0.2836
5	18.297157106096584	1	0	1	0	0	1	1	1	0.22	0.197	0.44	0.3582
7	15.690619901388677	1	0	1	0	0	1	1	2	0.24	0.2273	0.65	0.2239
9	-1.8377880245469558	1	0	1	0	0	2	1	2	0.16	0.1364	0.69	0.2836
17	19.364736227631084	1	0	1	0	0	3	1	2	0.22	0.2273	0.69	0.1343
7	6.939013369825195	1	0	1	0	0	4	1	1	0.14	0.1212	0.59	0.2836

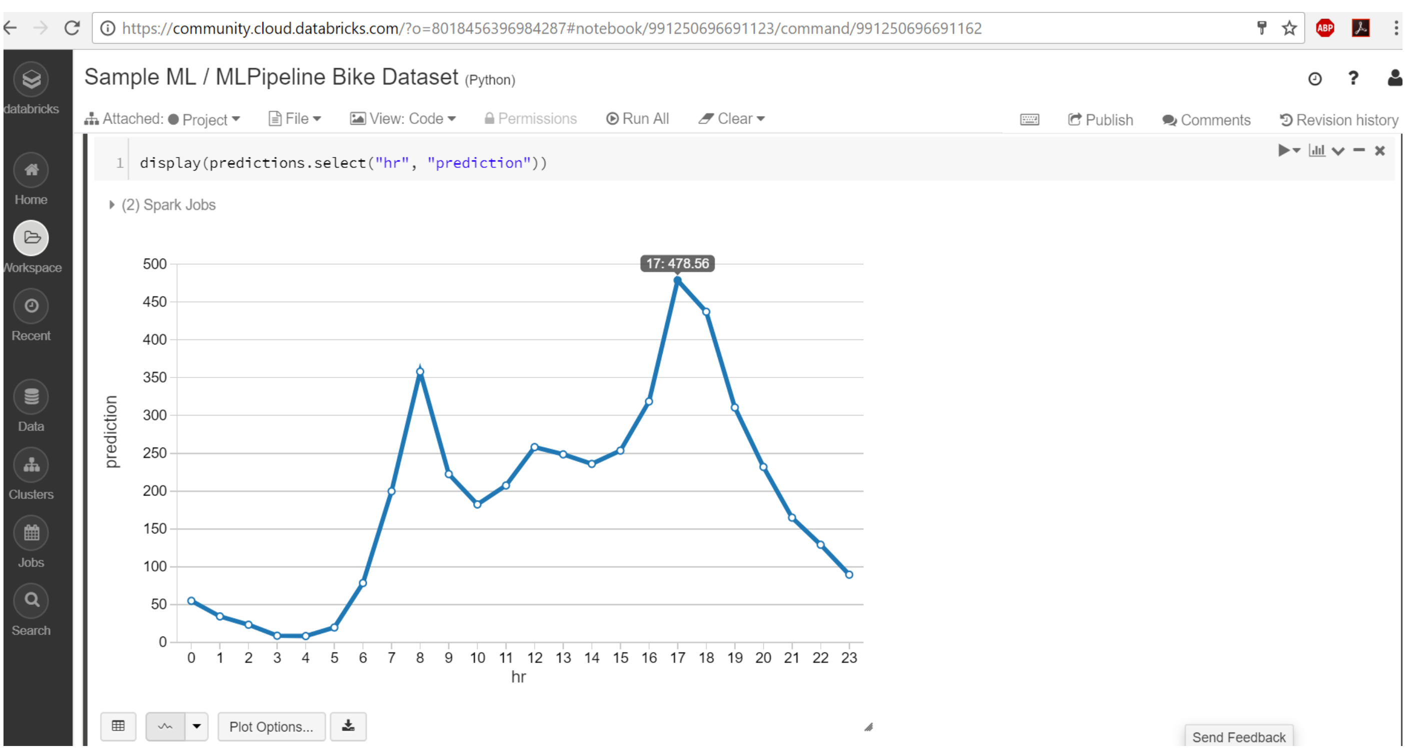
These results are not perfect, but there is a correlation between the counts and predictions. However, there is room to improve.

Here, I can give two tips on understanding the results:

(1) Metrics: Manually viewing the predictions gives intuition about accuracy, but it can be useful to have a more concrete metric. Below, an evaluation metric is computed which tells us how well the model makes predictions on all the data. In this case (for RMSE), lower is better. This metric does not mean much on its own, but it can be used to compare different models. (This is what CrossValidator does internally.)



(2) Visualization: Plotting predictions vs. features can help make sure that the model "understands" the input features and is using them properly to make predictions. Below, the model predictions are correlated with the hour of the day, just like the true labels were.



Appendix:

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691127

databricks Home Workspace Recent Data Clusters Jobs

Sample ML / MLPipeline Bike Dataset (Python)

Attached: Project | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Revision history

```python
# We use the sqlContext.read method to read the data and set a few options:
#  'format': specifies the Spark CSV data source
#  'header': set to true to indicate that the first line of the CSV data file is a header
# The file is called 'hour.csv'.
if sc.version >= '2.0':
  # Spark 2.0+ includes CSV as a native Spark SQL datasource.
  df = sqlContext.read.format('csv').option("header", 'true').load("/databricks-datasets/bikeSharing/data-001/hour.csv")
else:
  # Earlier Spark versions can use the Spark CSV package
  df = sqlContext.read.format('com.databricks.spark.csv').option("header", 'true').load("/databricks-datasets/bikeSharing/data-001/hour.csv")
# Calling cache on the DataFrame will make sure we persist it in memory the first time it is used
# The following uses will be able to read from memory, instead of re-reading the data from disk
df.cache()
```

(1) Spark Jobs

```
Out[1]: DataFrame[instant: string, dteday: string, season: string, yr: string, mnth: string, hr: string, holiday: string, weekday: string, workingday: string, weathersit: string, temp: string, atemp: string, hum: string, windspeed: string, casual: string, registered: string, cnt: string]
```

Command took 3.78 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:20:43 PM on Project

databricks Home Workspace Recent Data Clusters Jobs Search

Sample ML / MLPipeline Bike Dataset (Python)

Attached: Project | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Revision history

Cmd 7

```python
display(df)
```

(1) Spark Jobs

instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	3	13	16
2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	8	32	40
3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	5	27	32
4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	3	10	13
5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	0	1	1
6	2011-01-01	1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	0	1	1
7	2011-01-01	1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2	0	2
8	2011-01-01	1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	1	2	3
9	2011-01-01	1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	1	7	8

Showing the first 1000 rows.

Command took 2.00 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:21:35 PM on Project

Cmd 8

Sample ML / MLPipeline Bike Dataset (Python)

Cmd 9

```
print "Our dataset has %d rows." % df.count()
```

(1) Spark Jobs

```
Our dataset has 17379 rows.
```

Command took 0.53 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:22:09 PM on Project

Cmd 10

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691133

Sample ML / MLPipeline Bike Dataset (Python)

Cmd 11

```
df = df.drop("instant").drop("dteday").drop("casual").drop("registered")
display(df)
```

(1) Spark Jobs

season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	cnt
1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	16
1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	40
1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	32
1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	13
1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	1
1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	1
1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2
1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	3
1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	8

Showing the first 1000 rows.

Command took 0.36 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:23:40 PM on Project

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691136

Sample ML / MLPipeline Bike Dataset (Python)

Command took 0.36 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:23:40 PM on Project

Cmd 12

Now that we have the columns we care about, let's print the schema of our dataset to see the type of each column.

Cmd 13

```
df.printSchema()
```

```
root
 |-- season: string (nullable = true)
 |-- yr: string (nullable = true)
 |-- mnth: string (nullable = true)
 |-- hr: string (nullable = true)
 |-- holiday: string (nullable = true)
 |-- weekday: string (nullable = true)
 |-- workingday: string (nullable = true)
 |-- weathersit: string (nullable = true)
 |-- temp: string (nullable = true)
 |-- atemp: string (nullable = true)
 |-- hum: string (nullable = true)
 |-- windspeed: string (nullable = true)
 |-- cnt: string (nullable = true)
```

Command took 0.05 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:24:33 PM on Project

TERM PAPER
Submitted by: SUMANA SATPATI

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691138

Sample ML / MLPipeline Bike Dataset (Python)

Cmd 14

The DataFrame is currently using strings, but we know all columns are numeric. Let's cast them.

Cmd 15

```python
# The following call takes all columns (df.columns) and casts them using Spark SQL to a numeric type (DoubleType).
from pyspark.sql.functions import col  # for indicating a column using a string in the line below
df = df.select([col(c).cast("double").alias(c) for c in df.columns])
df.printSchema()
```

```
root
 |-- season: double (nullable = true)
 |-- yr: double (nullable = true)
 |-- mnth: double (nullable = true)
 |-- hr: double (nullable = true)
 |-- holiday: double (nullable = true)
 |-- weekday: double (nullable = true)
 |-- workingday: double (nullable = true)
 |-- weathersit: double (nullable = true)
 |-- temp: double (nullable = true)
 |-- atemp: double (nullable = true)
 |-- hum: double (nullable = true)
 |-- windspeed: double (nullable = true)
 |-- cnt: double (nullable = true)
```

Command took 0.12 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:26:19 PM on Project

Sample ML / MLPipeline Bike Dataset (Python)

```python
# Split the dataset randomly into 70% for training and 30% for testing.
train, test = df.randomSplit([0.7, 0.3])
print "We have %d training examples and %d test examples." % (train.count(), test.count())
```

(2) Spark Jobs

We have 12243 training examples and 5136 test examples.

Command took 1.21 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:26:43 PM on Project

Cmd 18

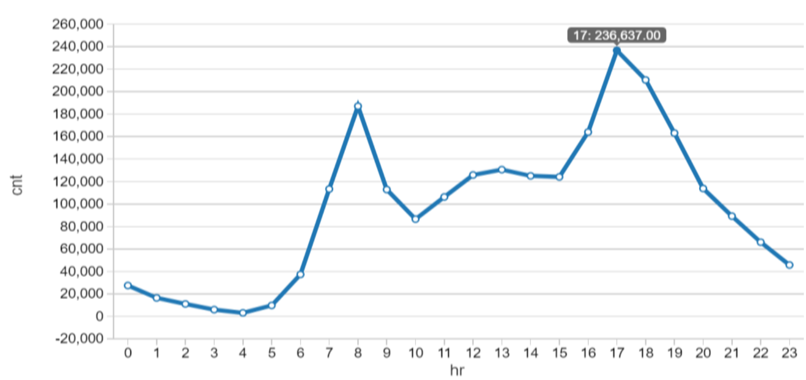
https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691142

Sample ML / MLPipeline Bike Dataset (Python)

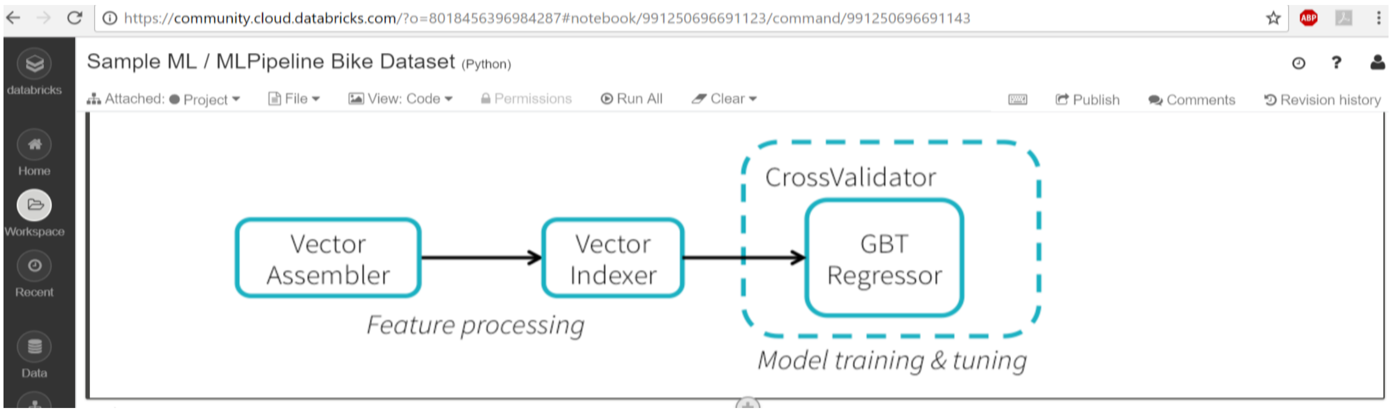
Cmd 19

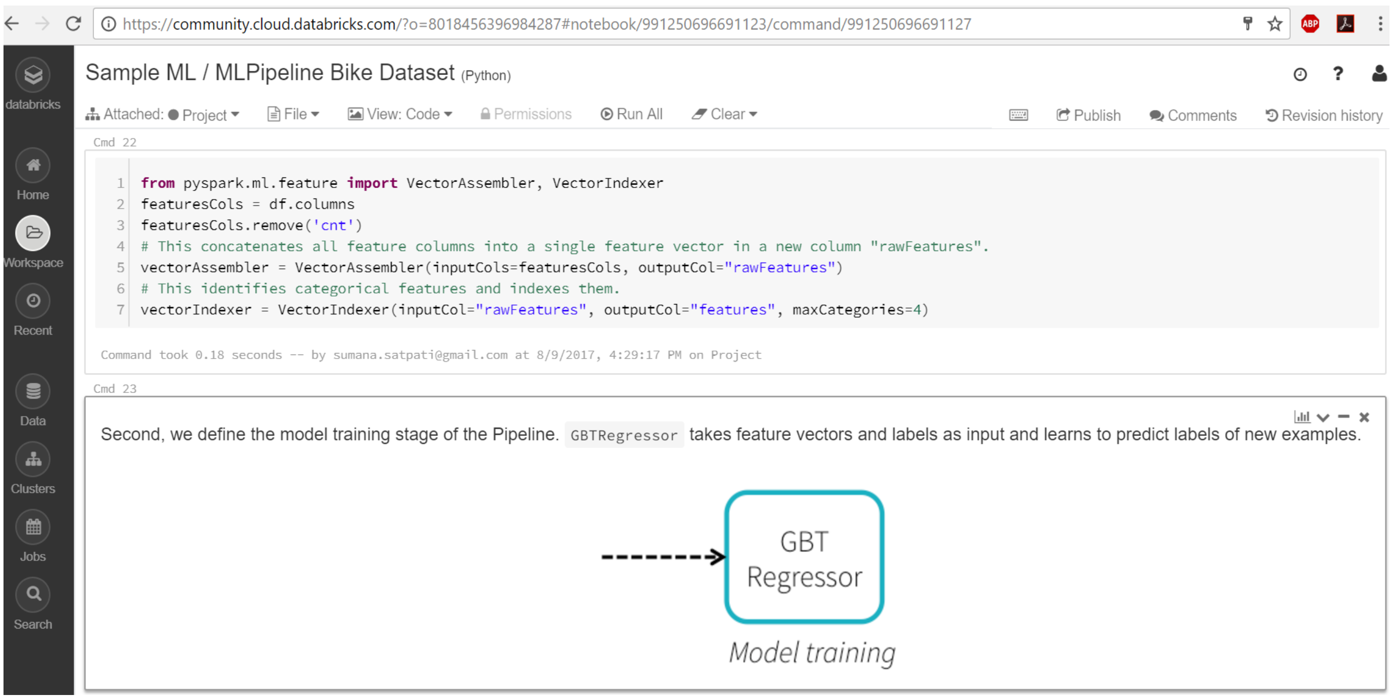
```python
display(train.select("hr", "cnt"))
```

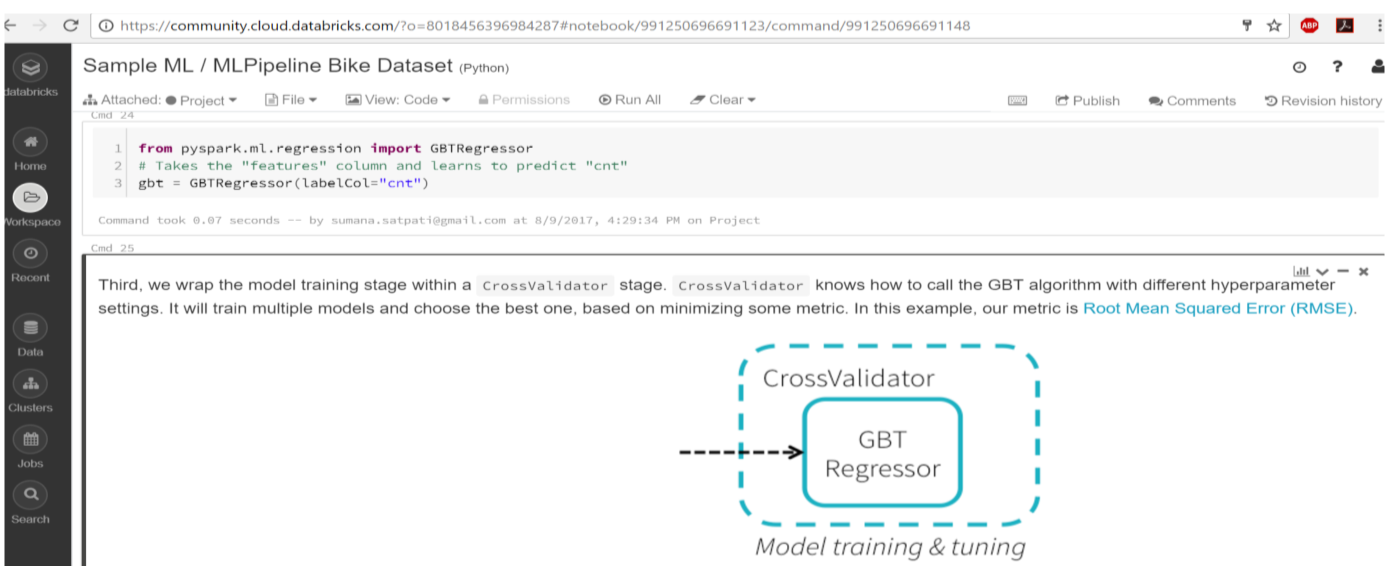
(2) Spark Jobs



Plot Options...


https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691143
Sample ML / MLPipeline Bike Dataset (Python)
Attached: Project File View: Code Permissions Run All Clear Publish Comments Revision history
databricks
Home
Workspace
Recent
Data
CrossValidator
Vector Assembler
Vector Indexer
GBT Regressor
Feature processing
Model training & tuning


https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691127
Sample ML / MLPipeline Bike Dataset (Python)
Attached: Project File View: Code Permissions Run All Clear Publish Comments Revision history
databricks
Home
Workspace
Recent
Data
Clusters
Jobs
Search
Cmd 22
from pyspark.ml.feature import VectorAssembler, VectorIndexer
featuresCols = df.columns
featuresCols.remove('cnt')
# This concatenates all feature columns into a single feature vector in a new column "rawFeatures".
vectorAssembler = VectorAssembler(inputCols=featuresCols, outputCol="rawFeatures")
# This identifies categorical features and indexes them.
vectorIndexer = VectorIndexer(inputCol="rawFeatures", outputCol="features", maxCategories=4)
Command took 0.18 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:29:17 PM on Project
Cmd 23
Second, we define the model training stage of the Pipeline. GBTRegressor takes feature vectors and labels as input and learns to predict labels of new examples.
GBT Regressor
Model training


https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691148
Sample ML / MLPipeline Bike Dataset (Python)
Attached: Project File View: Code Permissions Run All Clear Publish Comments Revision history
databricks
Home
Workspace
Recent
Data
Clusters
Jobs
Search
Cmd 24
from pyspark.ml.regression import GBTRegressor
# Takes the "features" column and learns to predict "cnt"
gbt = GBTRegressor(labelCol="cnt")
Command took 0.07 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:29:34 PM on Project
Cmd 25
Third, we wrap the model training stage within a CrossValidator stage. CrossValidator knows how to call the GBT algorithm with different hyperparameter settings. It will train multiple models and choose the best one, based on minimizing some metric. In this example, our metric is Root Mean Squared Error (RMSE).
CrossValidator
GBT Regressor
Model training & tuning

TERM PAPER
Submitted by: SUMANA SATPATI

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691148

Sample ML / MLPipeline Bike Dataset (Python)

Cmd 26

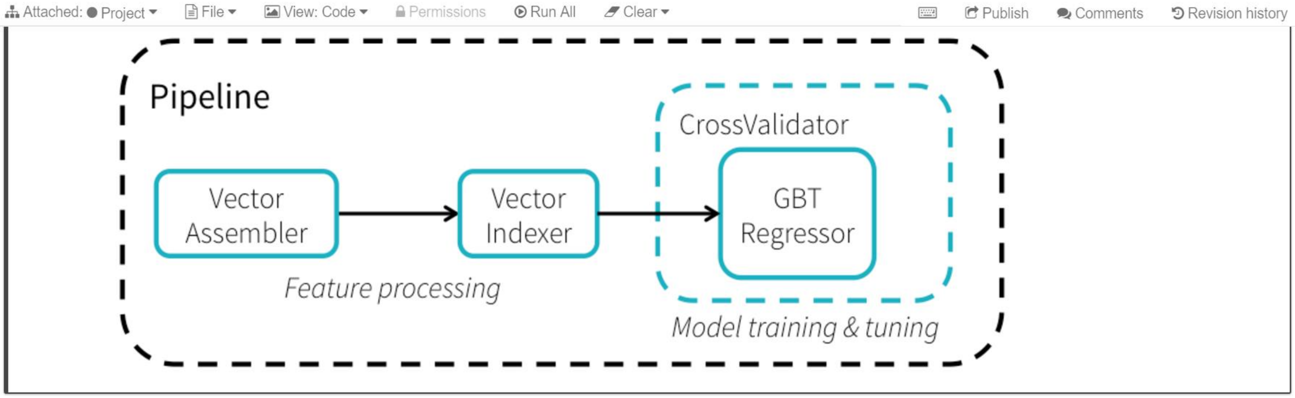
```python
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
# Define a grid of hyperparameters to test:
#  - maxDepth: max depth of each decision tree in the GBT ensemble
#  - maxIter: iterations, i.e., number of trees in each GBT ensemble
# In this example notebook, we keep these values small.  In practice, to get the highest accuracy, you would likely want to try deeper trees (10 or higher) and more trees in the ensemble (>100).
paramGrid = ParamGridBuilder()\
  .addGrid(gbt.maxDepth, [2, 5])\
  .addGrid(gbt.maxIter, [10, 100])\
  .build()
# We define an evaluation metric.  This tells CrossValidator how well we are doing by comparing the true labels with predictions.
evaluator = RegressionEvaluator(metricName="rmse", labelCol=gbt.getLabelCol(), predictionCol=gbt.getPredictionCol())
# Declare the CrossValidator, which runs model tuning for us.
cv = CrossValidator(estimator=gbt, evaluator=evaluator, estimatorParamMaps=paramGrid)
```

Command took 0.07 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:29:49 PM on Project

Cmd 27

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691150

Sample ML / MLPipeline Bike Dataset (Python)



Cmd 28

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691150

Sample ML / MLPipeline Bike Dataset (Python)

Cmd 28

```python
from pyspark.ml import Pipeline
pipeline = Pipeline(stages=[vectorAssembler, vectorIndexer, cv])
```

Command took 0.07 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:29:57 PM on Project

Sample ML / MLPipeline Bike Dataset (Python)

Attached: Project | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Revis

Cmd 30

```
pipelineModel = pipeline.fit(train)
```

(57) Spark Jobs

Command took 5.21 minutes -- by sumana.satpati@gmail.com at 8/9/2017, 4:30:10 PM on Project

Cmd 31

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691155

Sample ML / MLPipeline Bike Dataset (Python)

Attached: Project | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Revision history

Cmd 33

```
predictions = pipelineModel.transform(test)
```

Command took 0.83 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:39:17 PM on Project

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691157

Sample ML / MLPipeline Bike Dataset (Python)

Attached: Project | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Revision history

Cmd 35

```
display(predictions.select("cnt", "prediction", *featuresCols))
```

(1) Spark Jobs

cnt	prediction	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed
22	30.771284362188137	1	0	1	0	0	0	0	1	0.04	0.0758	0.57	0.1045
33	30.12246715061481	1	0	1	0	0	0	0	1	0.16	0.1818	0.8	0.1045
17	51.747529473866045	1	0	1	0	0	0	0	2	0.46	0.4545	0.88	0.2985
5	3.571002116185259	1	0	1	0	0	1	1	1	0.12	0.1212	0.5	0.2836
5	18.297157106096584	1	0	1	0	0	1	1	1	0.22	0.197	0.44	0.3582
7	15.690619901388677	1	0	1	0	0	1	1	2	0.24	0.2273	0.65	0.2239
9	-1.8377880245469558	1	0	1	0	0	2	1	2	0.16	0.1364	0.69	0.2836
17	19.364736227631084	1	0	1	0	0	3	1	2	0.22	0.2273	0.69	0.1343
7	6.939013369825195	1	0	1	0	0	4	1	1	0.14	0.1212	0.59	0.2836

Showing the first 1000 rows.

Command took 0.50 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:40:44 PM on Project

https://community.cloud.databricks.com/?o=8018456396984287#notebook/991250696691123/command/991250696691160

Sample ML / MLPipeline Bike Dataset (Python)

Attached: Project | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Revision history

Cmd 37

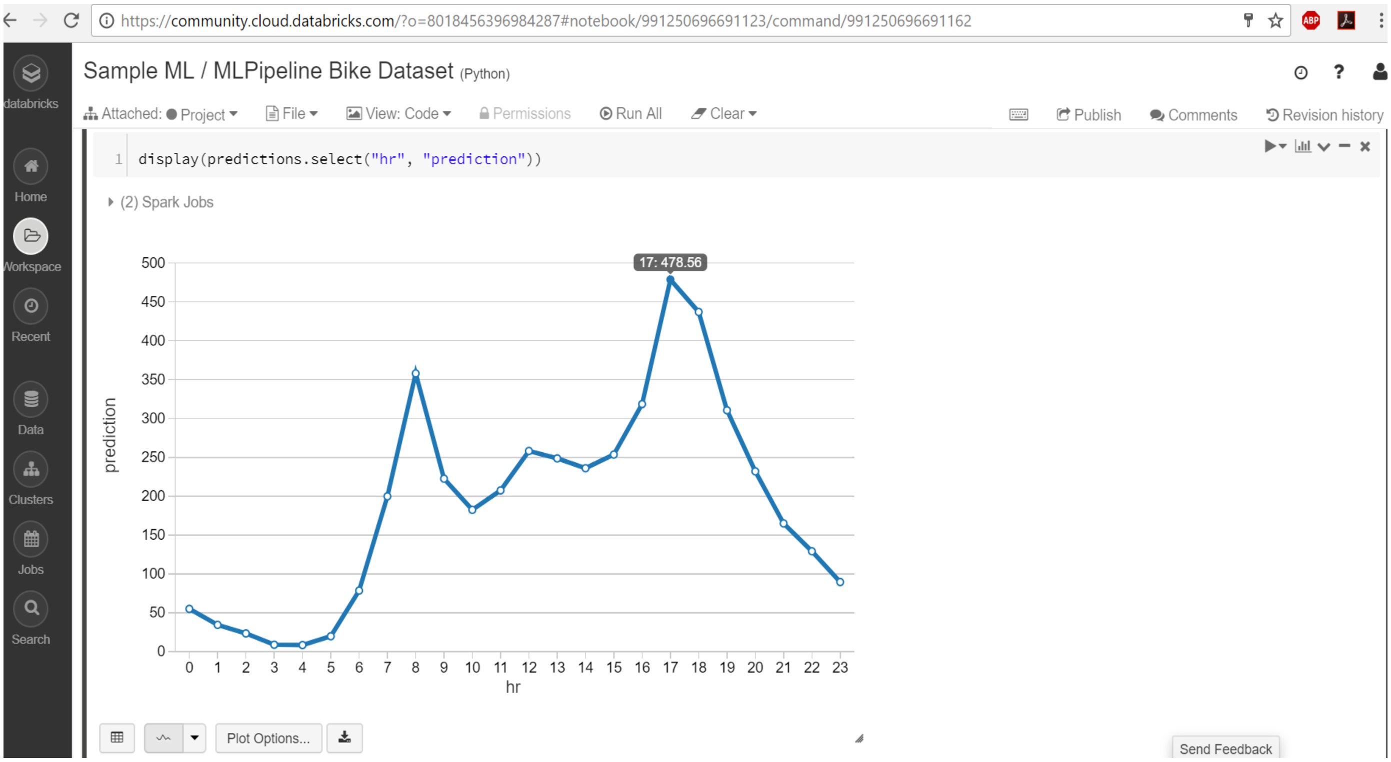
```
rmse = evaluator.evaluate(predictions)
print "RMSE on our test set: %g" % rmse
```

(1) Spark Jobs

RMSE on our test set: 45.2659

Command took 0.32 seconds -- by sumana.satpati@gmail.com at 8/9/2017, 4:41:00 PM on Project

Cmd 38



Reference:

https://docs.cloud.databricks.com/docs/latest/sample_applications/Sample%20ML/MLPipeline%20Bike%20Dataset.html

http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset