Data science and deep learning on Spark with 1/10th of the code

Roope Astala, Sudarshan Ragunathan
Microsoft Corporation

Agenda

- User story: Snow Leopard Conservation

- Introducing Microsoft Machine Learning Library for Apache Spark
  - Vision: Productivity, Scalability, State-of-Art Algorithms, Open Source
  - Deep Learning and Image Processing
  - Text Analytics

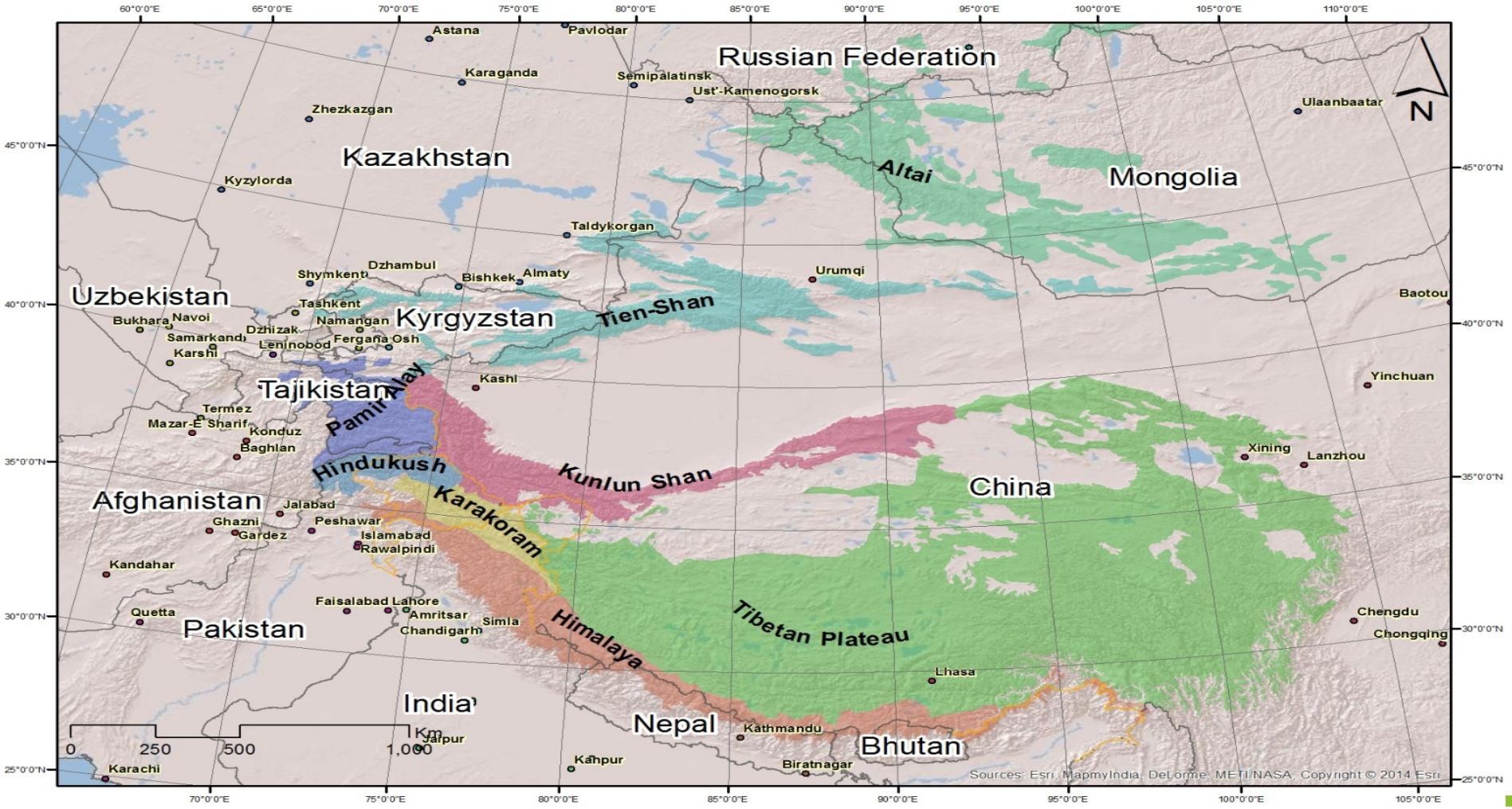
Disclaimer: Any roadmap items are subject to change without notice.


Snow
Leopard
Trust
Rhetick Sengupta
President, Board of Directors

Snow leopards

- 3,900-6,500 individuals left in the wild
- Variety of Threats
  - Poaching
  - Retribution killing
  - Loss of prey
  - Loss of habitat (mining)
- Little known about their ecology, behavior, movement patterns, survival rates
- More data required to influence survival

Range spread across 1.5 million km^2



Camera trapping since 2009

- 1,700 sq km
- 42 camera traps
- 8,490 trap nights
- 4 primary sampling periods
- 56 secondary sampling periods
- ~1.3 mil images

Camera Trap Images

Manually classifying images averages 300 hours per survey

Automated Image Classification Benefits

Short term

- Thousands of hours of researcher and volunteer time saved
- Resources redeployed to science and conservation vs image sorting
- Much more accurate data on range and population

Long term

- Population numbers that can be accurately monitored
- Influence governments on protected areas
- Enhance community based conservation programs
- Predict threats before they happen

How can you help?

We need more camera surveys!

- 1,700 sq km surveyed of 1,500,000
- $500 will buy an additional camera
- $5,000 will fund a researcher
- Any amount helps

Contact me directly at rhetick@hotmail.com or donate online.

www.snowleopard.org

SPARK SUMMIT 2017

Microsoft Machine Learning Library for Apache Spark (MMLSpark)

GitHub Repo: https://github.com/Azure/mmlspark

Get started now using Docker image:

```
docker run -it -p 8888:8888 -e ACCEPT_EULA=yes microsoft/mmlspark
```

Navigate to http://localhost:8888 to view example Jupyter notebooks

Challenges when using Spark for ML

- User needs to write lots of “ceremonial” code to prepare features for ML algorithms.
  - Coerce types and data layout to that what’s expected by learner
  - Use different conventions for different learners
- Lack of domain-specific libraries: computer vision or text analytics…
- Limited capabilities for model evaluation & model management

Vision of Microsoft Machine Learning Library for Apache Spark

- Enable you to solve ML problems dealing with large amounts of data.
- Make you as a professional data scientist more productive on Spark, so you can focus on ML problems, not software engineering problems.
- Provide cutting edge ML algorithms on Spark: deep learning, computer vision, text analytics…
- Open Source at GitHub

Example: Hello MMLSpark

Predict income from US Census data

Using Plain PySpark

```
In [1]: import numpy as np
        import pandas as pd
        import pyspark.ml.feature
        import pyspark.ml
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
172	application_1495458128606_0006	pyspark3	idle	Link	Link	✓

```
SparkSession available as 'spark'.
```

```
In [22]: dataFile = "AdultCensusIncome.csv"
         import os, urllib
         if not os.path.isfile(dataFile):
             urllib.request.urlretrieve("https://amldockerdatasets.azureedge.net/"+dataFile, dataFile)
         data = spark.createDataFrame(pd.read_csv(dataFile, dtype={" hours-per-week": np.float64}))
         data = data.select([" education", " marital-status", " hours-per-week", " income"])
         data.show(5)
```

```
+----------+------------------+--------------+--------+
| education|    marital-status|hours-per-week|  income|
+----------+------------------+--------------+--------+
| Bachelors|     Never-married|          40.0|  <=50K|
| Bachelors|Married-civ-spouse|          13.0|  <=50K|
|   HS-grad|          Divorced|          40.0|  <=50K|
|      11th|Married-civ-spouse|          40.0|  <=50K|
| Bachelors|Married-civ-spouse|          40.0|  <=50K|
+----------+------------------+--------------+--------+
only showing top 5 rows
```

SPARK SUMMIT 2017

In [13]:
```
## Keep track of feature column names - unnecessary in azureml
feature_cols = data.columns[:-1]
```

In [23]:
```
## Manual encoding of categoricals - unnecessary in azureml
categorical_cols = [' education',' marital-status']
for col in categorical_cols:
    tok = pyspark.ml.feature.Tokenizer(inputCol=col, outputCol=col+'Tok')
    data = tok.transform(data).drop(col)
    fh = pyspark.ml.feature.HashingTF(numFeatures=256,inputCol=col+'Tok',outputCol=col+"Index")
    data = fh.transform(data).drop(col+'Tok')
    data = data.withColumnRenamed(col+"Index",col)
data.show(10)
```

```
+--------------+-------+--------------------+--------------------+
| hours-per-week| income|           education|      marital-status|
+--------------+-------+--------------------+--------------------+
|          40.0|  <=50K|(256,[92,132],[1....|(256,[92,152],[1....|
|          13.0|  <=50K|(256,[92,132],[1....|(256,[92,205],[1....|
|          40.0|  <=50K|(256,[92,172],[1....|(256,[92,252],[1....|
|          40.0|  <=50K|(256,[92,250],[1....|(256,[92,205],[1....|
|          40.0|  <=50K|(256,[92,132],[1....|(256,[92,205],[1....|
|          40.0|  <=50K|(256,[4,92],[1.0,...|(256,[92,205],[1....|
|          16.0|  <=50K|(256,[92,137],[1....|(256,[67,92],[1.0...|
|          45.0|   >50K|(256,[92,172],[1....|(256,[92,205],[1....|
|          50.0|   >50K|(256,[4,92],[1.0,...|(256,[92,152],[1....|
|          40.0|   >50K|(256,[92,132],[1....|(256,[92,205],[1....|
+--------------+-------+--------------------+--------------------+
only showing top 10 rows
```

In [24]:
```
## Manual vector assembly to formate expected by learner - unnecessary in azureml
assembler = pyspark.ml.feature.VectorAssembler(inputCols=feature_cols,outputCol='features')
data = assembler.transform(data)
```

In [25]:
```
## Convert label to format expected by learner - unnecessary in azureml
from pyspark.sql.functions import *
data = data.select(['features',' income'])
labelMapper = udf(lambda income: 0.0 if income == ' <=50K' else 1.0)
data= data.withColumn('label',labelMapper(data[' income']).cast(pyspark.sql.types.DoubleType())).drop(' income')
data.show(10)
```

```
+--------------------+-----+
|            features|label|
+--------------------+-----+
|(513,[92,132,348,...|  0.0|
|(513,[92,132,348,...|  0.0|
|(513,[92,172,348,...|  0.0|
|(513,[92,250,348,...|  0.0|
|(513,[92,132,348,...|  0.0|
|(513,[4,92,348,46...|  0.0|
|(513,[92,137,323,...|  0.0|
|(513,[92,172,348,...|  1.0|
|(513,[4,92,348,40...|  1.0|
|(513,[92,132,348,...|  1.0|
+--------------------+-----+
only showing top 10 rows
```

In [26]:
```
train,test = data.randomSplit([0.75,0.25],seed=123)
```

In [27]:
```
from pyspark.ml.classification import LogisticRegression
lr = pyspark.ml.classification.LogisticRegression()
model = lr.fit(train)
```

In [28]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
prediction = model.transform(test)
evaluator = pyspark.ml.evaluation.BinaryClassificationEvaluator()
accuracy = evaluator.evaluate(prediction)
print(accuracy)
model.write().overwrite().save(model_path+'AdultCensus.LogisticRegression')
```

```
0.8635034300369923
```

In [24]:
```
## Manua
assemble
data = a
```

In [23]:
```
## Manua
categori
for col
    tok
    data
    fh =
    data
    data
data.sho
```

```
+-------
| hours-
+-------
|
|
|
|
|
|
|
|
|
+-------
only sho
```

In [13]:
```
## Keep
feature_
```

In [ ]:

SPARK SUMMIT 2017

Using MMLSpark

```
from mmlspark import TrainClassifier, ComputeModelStatistics
from pyspark.ml.classification import LogisticRegression
model = TrainClassifier(model=LogisticRegression(), labelCol=" income").fit(train)
prediction = model.transform(test)
metrics = ComputeModelStatistics().transform(prediction)
```

Algorithms

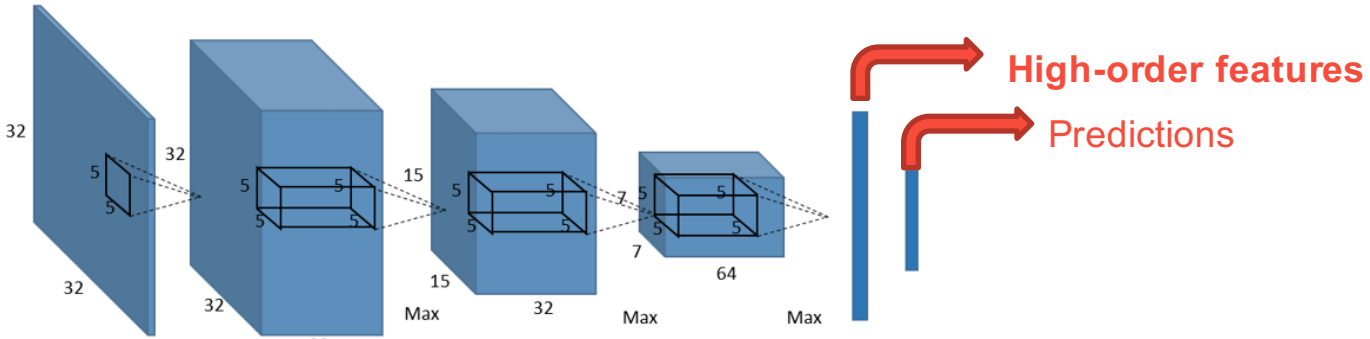
- Deep learning through Microsoft Cognitive Toolkit (CNTK)
  - Scale-out DNN featurization and scoring. Take an existing DNN model or train locally on big GPU machine, and deploy it to Spark cluster to score large data.
  - Scale-up training on a GPU VM. Preprocess large data on Spark cluster workers and feed to GPU to train the DNN.
- Scale-out algorithms for “traditional” ML through SparkML

Design Principles

- Run on every platform and on language supported by Spark
- Follow *SparkML pipeline* model for composability. MML-Spark consists of Estimators and Transforms that can be combined with existing SparkML components into pipelines.
- Use *SparkML DataFrames* as common format. You can use existing capabilities in Spark for reading in the data to your model.
- *Consistent handling* of different datatypes – text, categoricals, images – for different algorithms. No need for low-level type coercion, encoding or vector assembly.

Deep Neural Net Featurization

- Basic idea: Interior layers of pre-trained DNN models have high-order information about features



- Using "headless" pre-trained DNNs allows us to extract really good set of features from images that can in turn be used to train more "traditional" models like random forests, SVM, logistic regression, etc.
  - Pre-trained DNNs are typically state-of-the-art models on datasets like ImageNet, MSCoco or CIFAR, for example ResNet (Microsoft), GoogLeNet (Google), Inception (Google), VGG, etc.
- *Transfer learning* enables us to train effective models where we don't have enough data, computational power or domain expertise to train a new DNN *from scratch*
- Performance ***scales with executors***

DNN Featurization using MML-Spark

```
cntkModel = CNTKModel().setInputCol("images").
     setOutputCol("features").setModelLocation(resnetModel).
     setOutputNode("z.x")

featurizedImages = cntkModel.transform(imagesWithLabels).
     select(['labels','features'])

model = TrainClassifier(model=LogisticRegression(),labelCol="labels").
     fit(featurizedImages)
```

The DNN featurization is incorporated as SparkML pipeline stage. The evaluation happens directly on JVM from Scala: no Python UDF overhead!

Image Processing Transforms

DNNs are often picky about their input data shape and normalization.

We provide bindings to OpenCV image ingress and processing operations, exposed as SparkML PipelineStages:

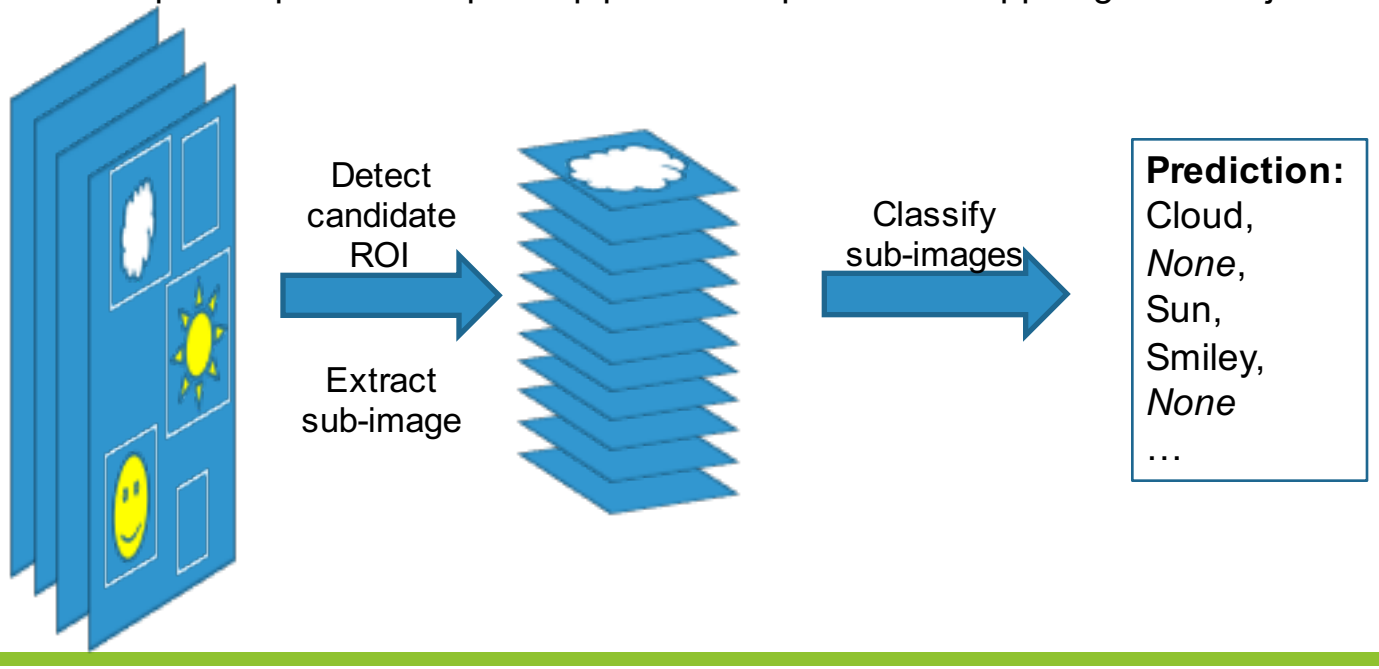
```
images = spark.readImages(IMAGE_PATH, recursive = True, sampleRatio = 0.1)

tr = ImageTransform().setOutputCol("transformed")
    .resize(height = 200, width = 200)
    .crop(0, 0, height = 180, width = 180)

smallImages = tr.transform(images).select("transformed")
```

Image Pipeline for Object Detection and Classification

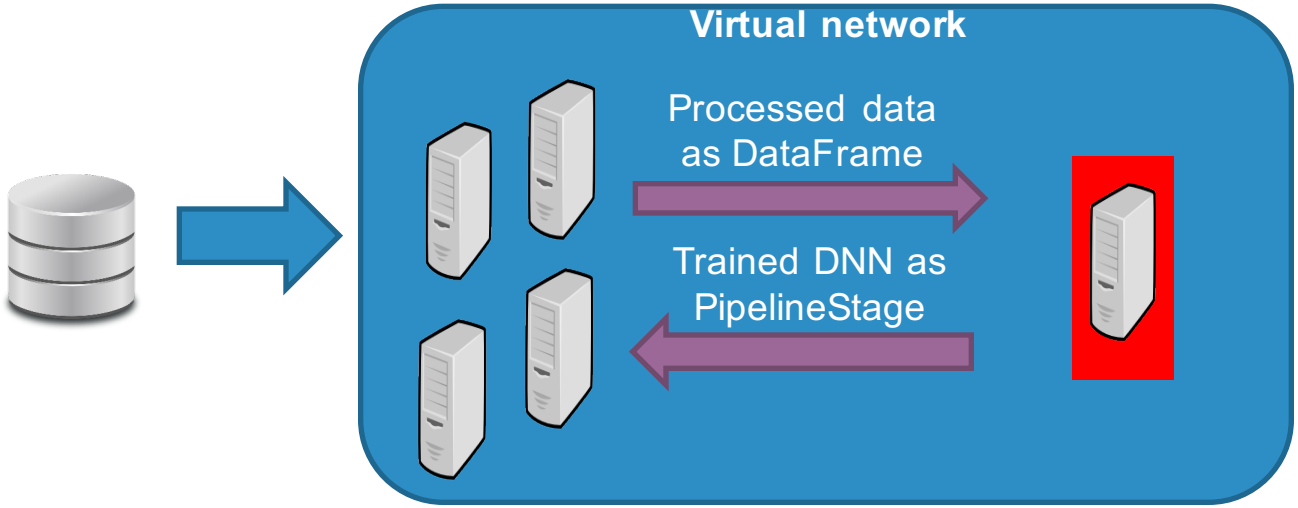
- In real data, objects are often not framed and 1 per image. There can be many candidate sub-images.
- You could extract candidate sub-images using flatMaps and UDFs, but this gets cumbersome quickly.
- We plan to provide simplified pipeline components to support generic object detection workflow.



Training of DNNs on GPU node

- GPUs are very powerful for training DNNs. However, running an entire cluster of GPUs is often too expensive and unnecessary.
- Instead, load and prep large data on CPU Spark cluster, then feed the prepped data to GPU node on virtual network for training. Once DNN is trained, broadcast the model to CPU nodes for evaluation.

```
learner = CNTKLearner(brainScript=brainscriptText, dataTransfer='hdfs-mount',
   gpuMachines='my-gpu-vm', workingDir='file:/tmp/').fit(trainData)
predictions = learner.setOutputNode('z').transform(testData)
```



Text Analytics

- Goal: provide one-step text featurization capability that lets you take free-form text columns and turn them into feature vectors for ML algorithms.
  - Tokenization, stop-word removal, case normalization
  - N-Gram creation, feature hashing, IDF weighing.

- Future: multi-language support, more advanced text preprocessing and DNN featurization capabilities.

Example: Text analytics for document classification by sentiment

```
from pyspark.ml import Pipeline
from mmlspark import TextFeaturizer, SelectColumns, TrainClassifier
from pyspark.ml.classification import LogisticRegression

textFeaturizer = TextFeaturizer(inputCol="text", outputCol = "features",
                                useStopWordsRemover=True, useIDF=True, minDocFreq=5,
                                numFeatures=2**16)

columnSelector = SelectColumns(cols=["features","label"])

classifier = TrainClassifier(model = LogisticRegression(), labelCol='label')

textPipeline = Pipeline(stages= [textFeaturizer,columnSelector,classifier])
```

Easy to get started

GitHub Repo: https://github.com/Azure/mmlspark

Get started now using Docker image:

```
docker run -it -p 8888:8888 -e ACCEPT_EULA=yes microsoft/mmlspark
```

Navigate to http://localhost:8888 for example Jupyter notebooks

Spark package installable for generic Spark 2.1 clusters

Script Action installation on Azure HDInsight cluster