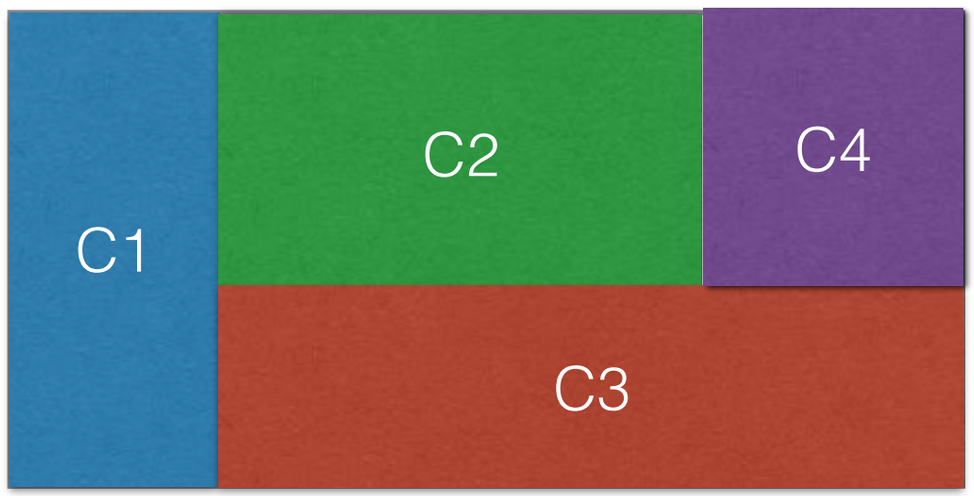
Multi-class Classification with large datasets, in Python, on Spark

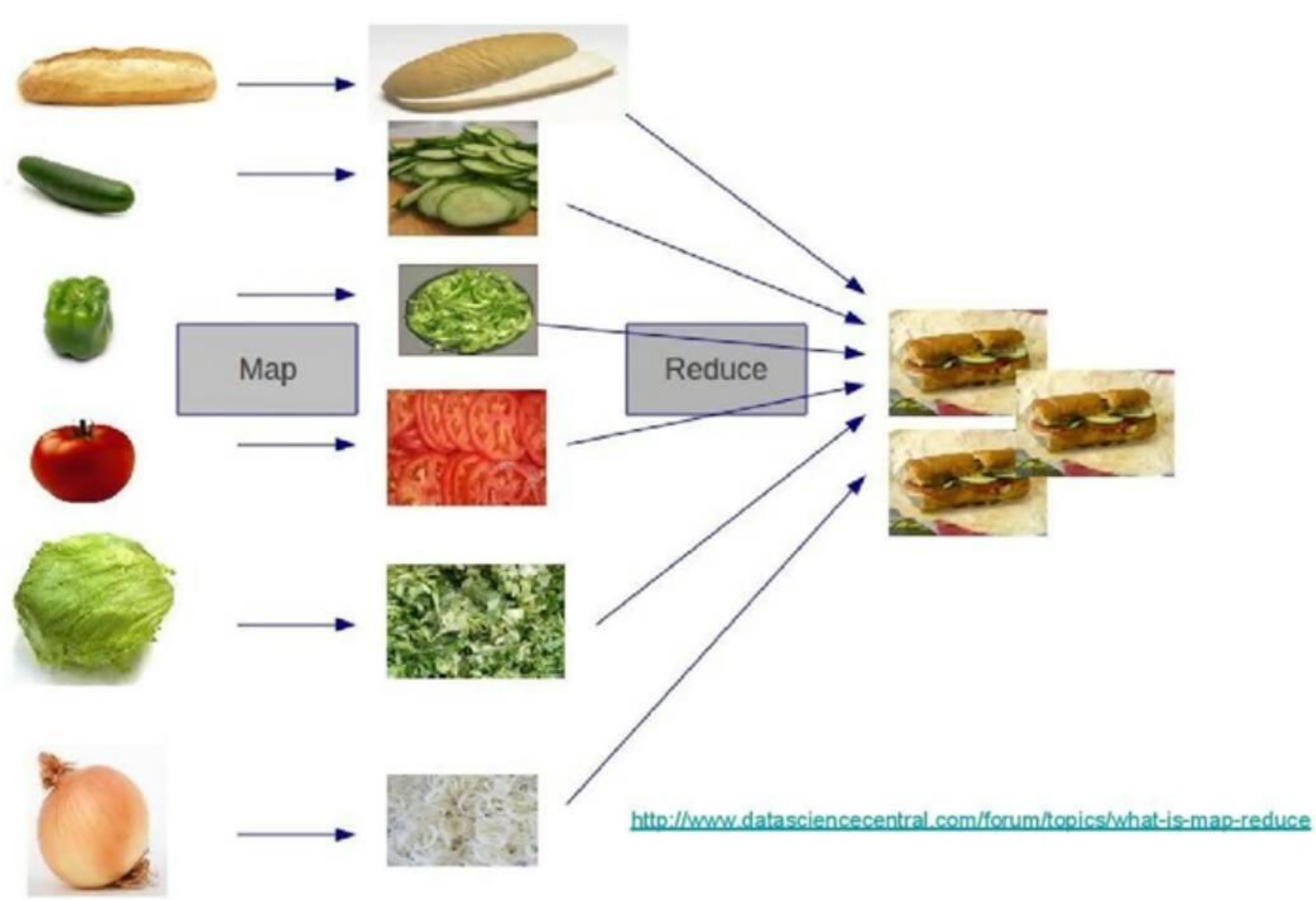


Victor Makarenkov

Agenda

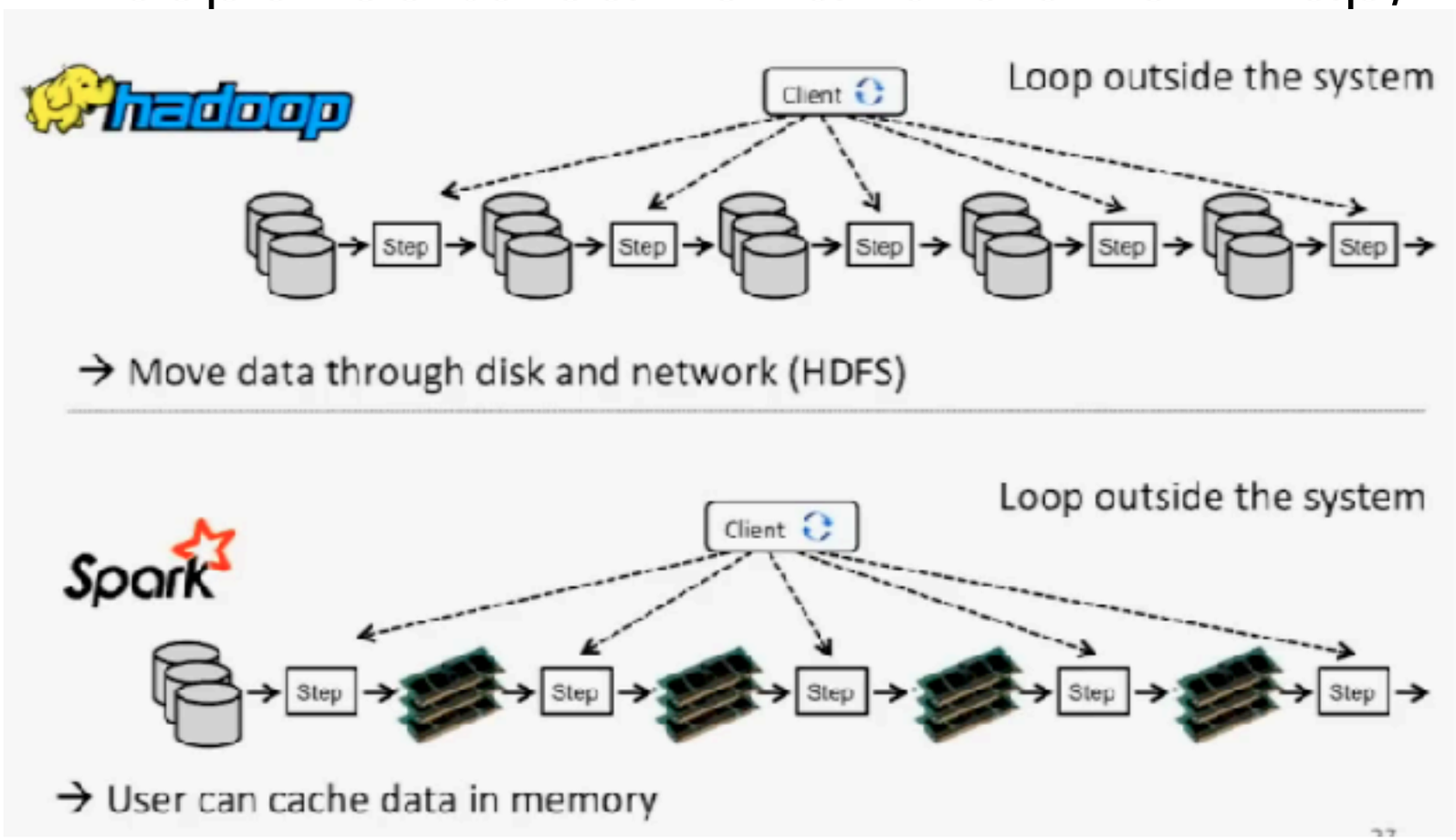
- Spark
- Which models can be learned in a distributed manner more naturally
- Examples: MLLib, PAMAP Dataset
    - Random Forest, Naive Bayes, Logistic Regression
- Do it at home: Databricks, Hortonworks

Map\Reduce


Map
Reduce
http://www.datasciencecentral.com/forum/topics/what-is-map-reduce

Spark

- Open source cluster computing framework
- Central concept in Spark: Resilient Distributed Dataset (RDD)
  - A response to earlier and slower Map/Reduce



Orders of magnitude speedup!

https://www.nextplatform.com/2015/02/22/flink-sparks-next-wave-of-distributed-data-processing/

More benefits

- High efficiency - **computation graphs**, in **memory caching** for multiple complete dataset reads

(MLLib)

Hello World - Word Count

```
text = sc.textFile("myFile.txt") #list of text lines

counts = text.flatMap(lambda line: line.split(" "))
          .map(lambda word: (word, 1))
          .reduceByKey(lambda val1, val2: val1 + val2)
```

RDD

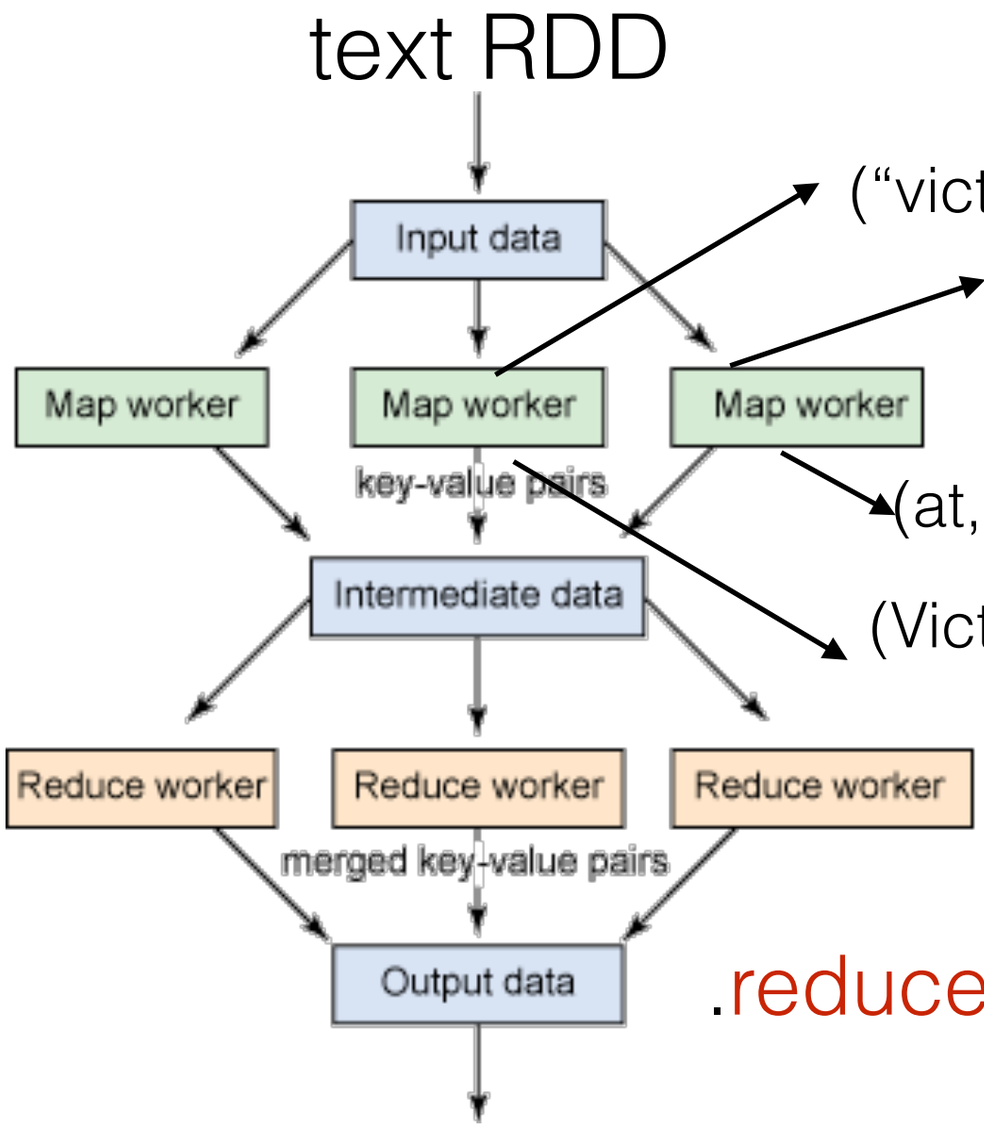
WORD	COUNT
the	65544
a	50000
bla…	x…

Behind the scenes

myfile = Victor presents at PyData Meetup \n
at Taboola offices at ramat gan

text = [“Victor presents at PyData Meetup”, “at Taboola offices at ramat gan”]

text.flatMap(lambda line: line.split(" “))
.map(lambda word: (word, 1))



(“victor”, “presents”, “at”, “PyData”, “Meetup”)

(“at”, “Taboola”, “offices”, “at”, “ramat”, “gan”)

(at,1), (Taboola,1) (offices,1), (at,1),(ramat, 1),(gan, 1)

(Victor,1), (presents,1) (at,1), (PyData,1), (meetup,1)

.reduceByKey(lambda val1, val2: val1 + val2)

(at,1)(at,1)(at,1) => (at,3)

http://www.ibm.com/developerworks/library/l-hadoop-3/

Dataset Representation

- Each algorithm can be fed with labeledPoint RDD
- The **LabeledPoint** class:
  - **LabeledPoint(label, [list of features' values])**
- Each MLLib trainable algorithm uses such RDDs in its training procedures

Distributed Learning

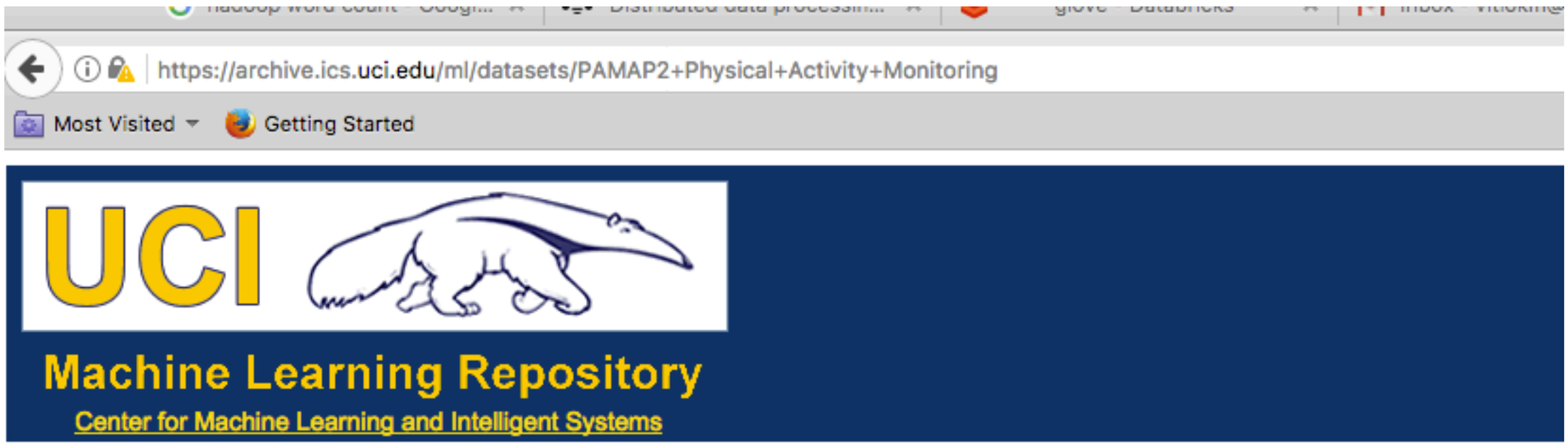
LIKE!

Naive Bayes
knn
k-means
Random Bagging
Random Forest
Linear Regression

hmmm…

Neural Nets
SVM
Logistic Regression
Hidden Markov Models
AdaBoost

PAMAP2 Dataset (1.7 GB)



PAMAP2 Physical Activity Monitoring Data Set

Download: Data Folder, Data Set Description

Abstract: The PAMAP2 Physical Activity Monitoring dataset contains data of 18 different physical activities, performed by 9 subjects wearir

Data Set Characteristics:	Multivariate, Time-Series	**Number of Instances:**	3850505	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	52	**Date Donated**	2012-08-06
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	43147

[1] A. Reiss and D. Stricker. Introducing a New Benchmarked Dataset for Activity Monitoring. The 16th IEEE International Symposium on Wearable Computers (ISWC), 2012.
[2] A. Reiss and D. Stricker. Creating and Benchmarking a New Dataset for Physical Activity Monitoring. The 5th Workshop on Affect and Behaviour Related Assistance (ABRA), 2012.

The PAMAP2 dataset

0	1	2	3-19	20-36	37-53
Time stamp	Label	Heart Rate	IMU-1	IMU-2	IMU-3

1	2 - 4	5 - 7	8-10	10-13	14-17
Temperature	ACC	ACC (invalid)	Gyroscope	Magnetometer	Orientation (invalid)

Classes: Running, sitting, lying, ironing, nordic walking …

18 different classes

Build a model with PAMAP2 Data

- 1) Read the data into into an RDD
- 2) Clean the data, leave only relevant features
- 3) Compute the Average
- 4) Fill missing values
- 5) Train & Evaluate the model

Read, Clean

```
def slice_data(line):
    raw_list = line.split(" ")
    indices = {0,7,8,9,16,17,18,19,24,25,26,33,34,35,36,41,42,43,50,51,52,53}
    sliced_list = [i for j, i in enumerate(raw_list) if j not in indices]
    return sliced_list

lines = lines.map(slice_data)
```

What about side effects?

Accumulator

- Read only by the main driver
- Mutable only by the distributed processors - workers
- Like the *reduce* action and *reduceByKey* transformation it should be **associative** and **commutative**

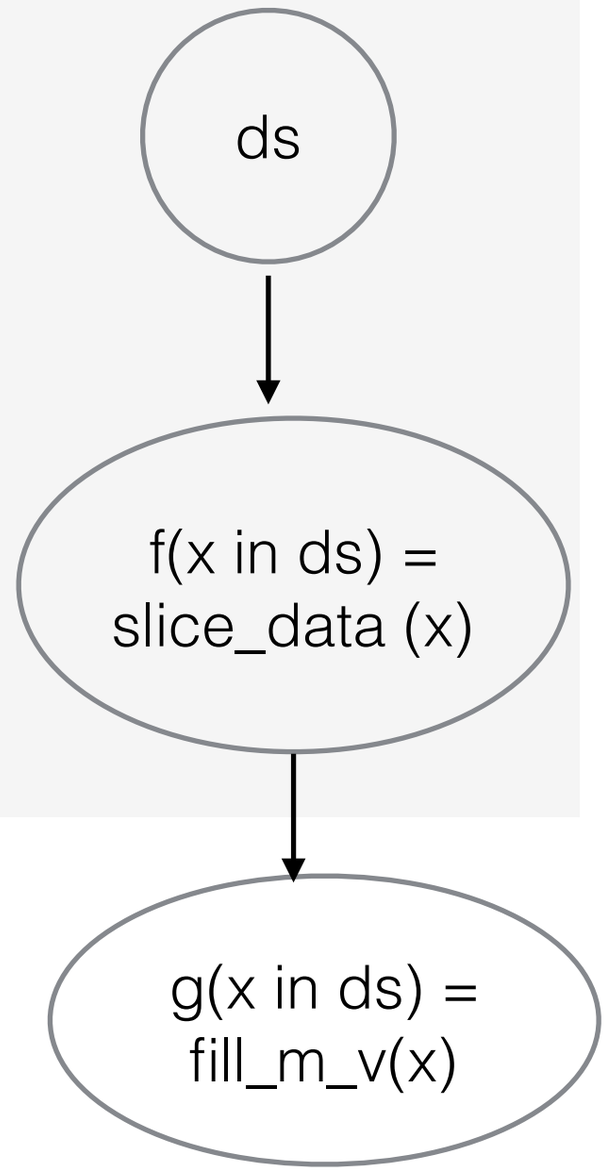
Mean feature values calculation

```
summed_data = [[sc.accumulator(0), sc.accumulator(0)] for y in range(features_number)]

def sum_data(sliced_list):
    for i in range(1, len(sliced_list)):
        if sliced_list[i] != "NaN":
            summed_data[i-1][0] += float(sliced_list[i])
            summed_data[i-1][1] += 1

lines.foreach(sum_data)
```

```
features_means = [0 for i in range (features_number)]

for i in range(0, features_number):
    features_means[i] = summed_data[i][0].value / summed_data[i][1].value
```

Filling Missing Values

```
encoded_labels = {1:0, 2:1, 3:2, 4:3, 5:4, 6:5, 7:6, 9:7, 10:8,
                  11:9, 12:10, 13:11, 16:12, 17:13, 18:14, 19:15, 20:16, 24:17}

def fill_missing_values(sliced_list):
    features = [0 for i in range (features_number)]
    label = int(sliced_list[0])
    for i in range(0, features_number):
        if sliced_list[i+1] != "NaN":
            features[i] = float(sliced_list[i+1])
        else:
            features[i] = features_means[i]

    lp = LabeledPoint(encoded_labels[label], features)
    return lp
```




```
lines = lines.map(slice_data).map(fill_missing_values)
```

Build the model!
MLLib

from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from pyspark.ml.classification import LogisticRegression, OneVsRest
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.tree import RandomForest
from pyspark.mllib.util import MLUtils
from pyspark.mllib.classification import NaiveBayes, NaiveBayesModel
from pyspark.mllib.tree import GradientBoostedTrees, GradientBoostedTreesModel

(I will mention ML library at the end)

Naive Bayes

Default: Multinomial, support Bernoulli

```
# Train a naive Bayes model.
model = NaiveBayes.train(training)
# Make prediction and test accuracy.
predictionAndLabel = test.map(lambda p: (model.predict(p.features), p.label))
err = 1.0 * predictionAndLabel.filter(lambda (p, l): p != l).count() / test.count()
return err
```

-Does not support negative values for features

Random Forest

(training, test) = lines.randomSplit([0.7, 0.3])

```
model = RandomForest.trainClassifier(training, numClasses=18, categoricalFeaturesInfo={},
                                     numTrees=20, featureSubsetStrategy="auto",
                                     impurity='gini', maxDepth=10)
#Evaluate model on test instances and compute test error
predictions = model.predict(test.map(lambda x: x.features))
labelsAndPredictions = test.map(lambda lp: lp.label).zip(predictions)
testErr = labelsAndPredictions.filter(lambda (v, p): v != p).count() / float(test.count())
return testErr
```

Logistic Regression

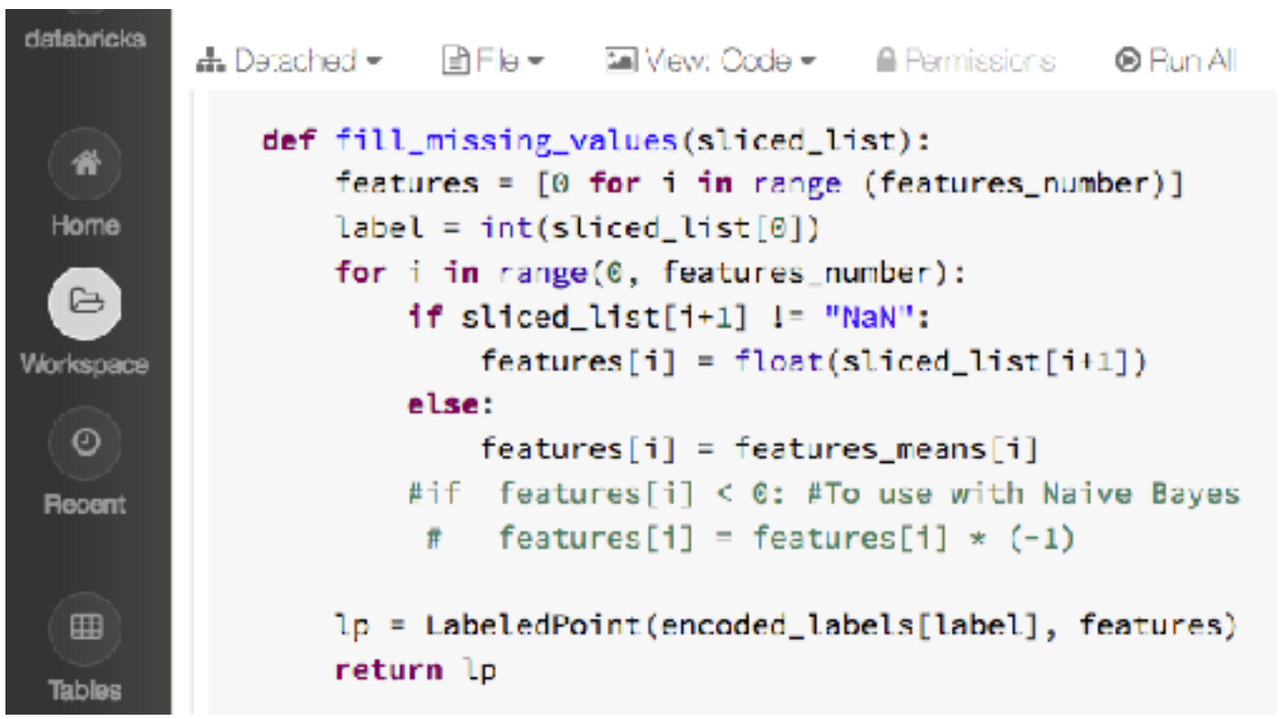
```
model = LogisticRegressionWithLBFGS.train(training, numClasses=18)

# Evaluating the model on training data
labelsAndPreds = test.map(lambda p: (p.label, model.predict(p.features)))
err = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(test.count())
print("Training Error = " + str(ERR))
return err
```

multiclass support is implemented behind the scenes
with one vs. all strategy

Do it at home:

https://github.com/vicmak/Mining-Massive-Datasets

https://community.cloud.databricks.com



A notebook!
Like!

https://hortonworks.com/

HORTONWORKS

Complete image
of a cluster: VM or Virtual Box

Lessons Learned

- Spark: easy development - hard maintenance
  - 1) Python 2) environment setup takes time!
- Be careful to use Hadoop add ons, (i.e. HBase)
- ML and DataFrame, more flexible
- For text processing consider distributed IR solutions: 1) Solr 2) ElasticSearch