PySpark ML Models Illustrated

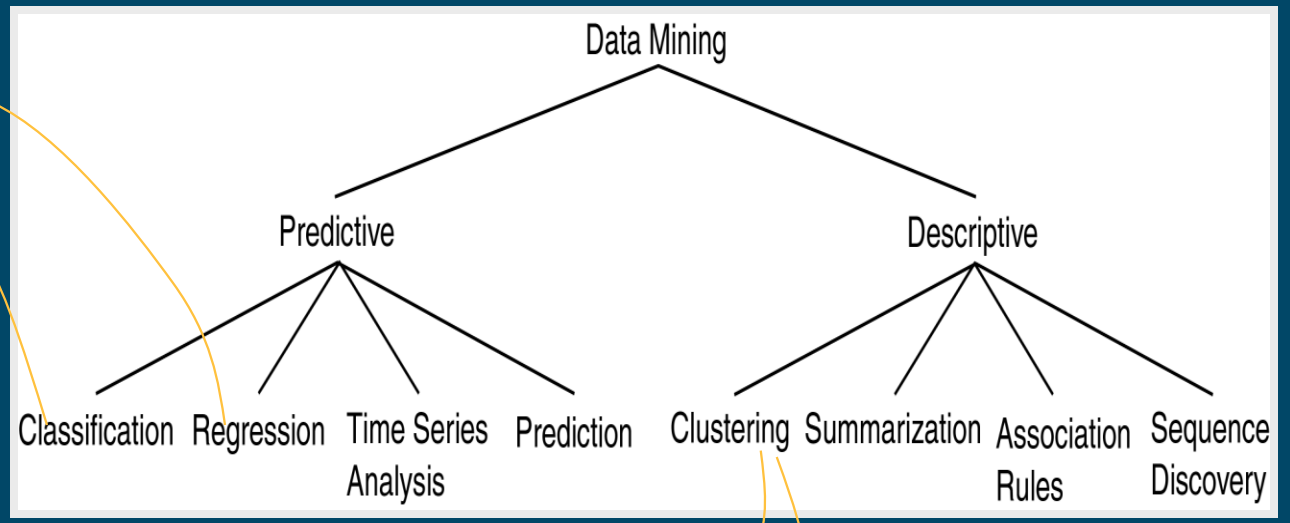
Demonstrations of some common ML models

ML Models

- Most have a common interface and usage
  - Features must be present in a DataFrame
  - The model is instantiated and its parameters set
  - Call model fit()
  - View the predictions

Data mining models

- Linear regression
- Random forest classifier
- K-means clustering
- LDA clustering



Classification

Classification

- Predict a label that has a finite set of possible values
  - binary classification, has two labels values
  - multiclass classification, has more than two labels values
  - multilabel classification, multiple labels can be predicted
- Scalability of Spark MLlib Models

Model	Features count	Training examples	Output classes
Logistic regression	1 to 10 million	No limit	Features x Classes < 10 million
Decision trees	1,000s	No limit	Features x Classes < 10,000s
Random forest	10,000s	No limit	Features x Classes < 100,000s
Gradient-boosted trees	1,000s	No limit	Features x Classes < 10,000s

Logistic Regression

- Parameters
  - family
    - multinomial (two or more distinct labels)
    - binary (only two distinct labels)
  - elasticNetParam
    - A mix of L1 and L2 regularization, from [0,1]
  - maxIter
    - Maximum number of iterations
  - etc.

Running it

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression()
print lr.explainParams() # see all parameters
lrModel = lr.fit(bInput)
```

Print results

```
print lrModel.coefficients
print lrModel.intercept
```

Print model evaluation

```
summary = lrModel.summary
print summary.areaUnderROC
summary.roc.show()
summary.pr.show()
```

Decision Trees

- Decision trees are like decision trees used in management
  - Random Forest and Gradient-Boosted Trees train more than one DT
  - Gradient-boosted trees, each tree makes a weighted prediction
  - Parameters for these models are similar
- Parameters
  - maxDepth
    - It can be helpful to specify a max depth in order to avoid overfitting to the dataset
      - in the extreme, every row ends up as its own leaf node
      - default is 5

```
from pyspark.ml.classification import RandomForestClassifier
rfClassifier = RandomForestClassifier()
print rfClassifier.explainParams()
trainedModel = rfClassifier.fit(bInput)
```

```
from pyspark.ml.classification import GBTClassifier
gbtClassifier = GBTClassifier()
print gbtClassifier.explainParams()
trainedModel = gbtClassifier.fit(bInput)
```

ml.classification RandomForestClassifier

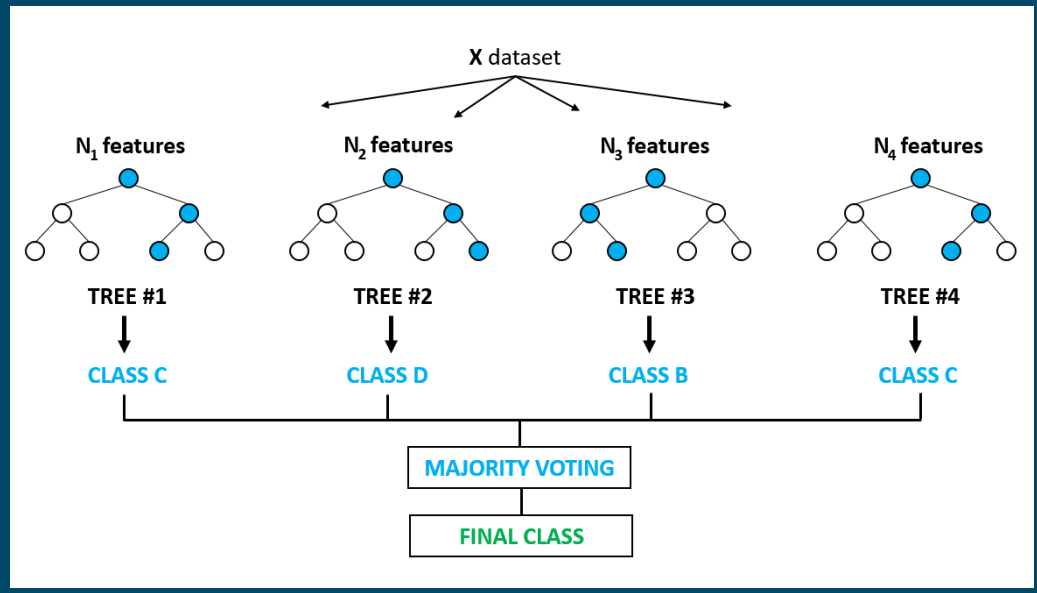
- Purpose
  - From the labeled input features, compute a tree of propositions over the features that correct labels the data
- Random forests are ensembles of decision trees that combine many decision trees in order to reduce the risk of overfitting
- Example

```
# The model
rf = RandomForestClassifier()
# The ML pipeline
pipeline = Pipeline(stages=[stringIndexer_label, stringIndexer_prof, stringIndexer_ms,
stringIndexer_gender, vectorAssembler_features, rf, labelConverter])
```

Features

Random forest

- Different features for different decision trees
- These trees make different predictions
- Majority voting determines the final classification
  - These are independent models, which are combined (called bagging)



Evaluation metrics

- Evaluators provide a metric for success for a model
- Use the evaluator to select the best model
- An evaluator is typically used in a pipeline, where we can automate a grid search of all combinations of model parameters and transformers to see which ones perform the best

Regression

Regression examples

- Linear regression

```
from pyspark.ml.regression import LinearRegression
lr = LinearRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8)
print lr.explainParams()
lrModel = lr.fit(df)
```

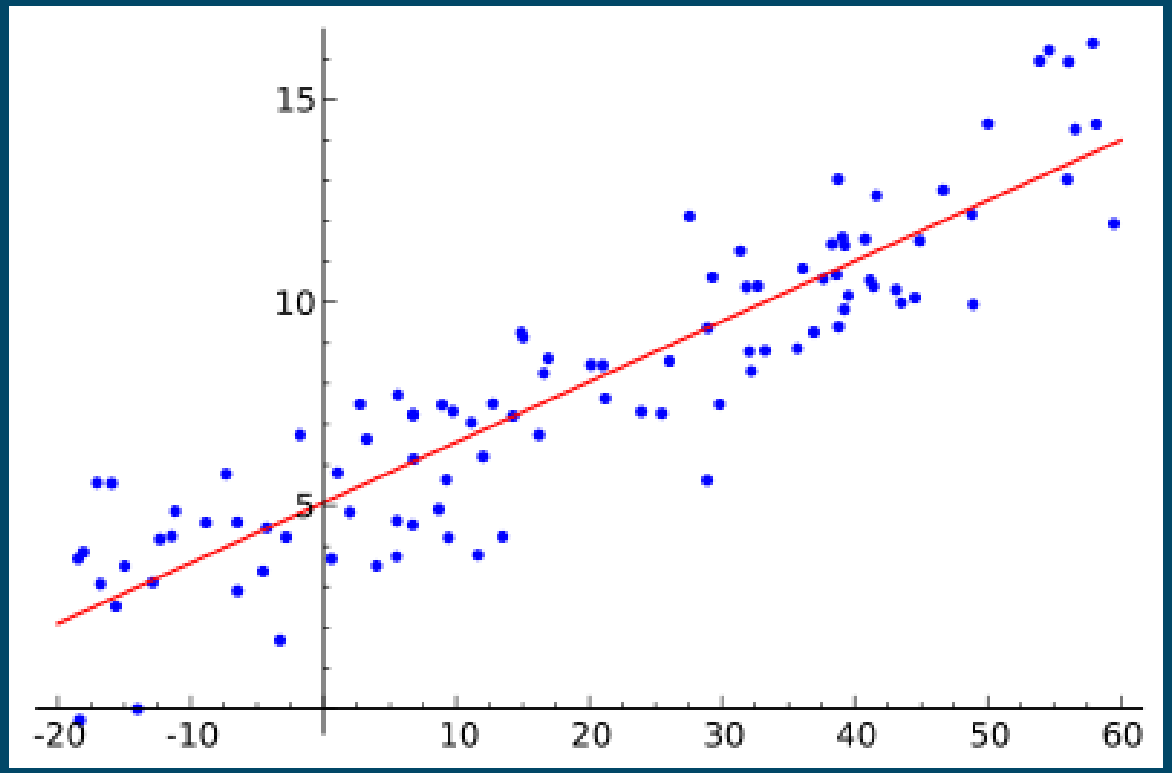
```
summary = lrModel.summary
summary.residuals.show()
print summary.totalIterations
print summary.objectiveHistory
print summary.rootMeanSquaredError
print summary.r2
```

- Generalized Linear Regression
  - allows you to select the expected noise distribution from a variety of families
  - Gaussian, logistic, poisson, and gamma.

```
from pyspark.ml.regression import GeneralizedLinearRegression
glr = GeneralizedLinearRegression()\
  .setFamily("gaussian")\
  .setLink("identity")\
  .setMaxIter(10)\
  .setRegParam(0.3)\
  .setLinkPredictionCol("linkOut")
print glr.explainParams()
glrModel = glr.fit(df)
```

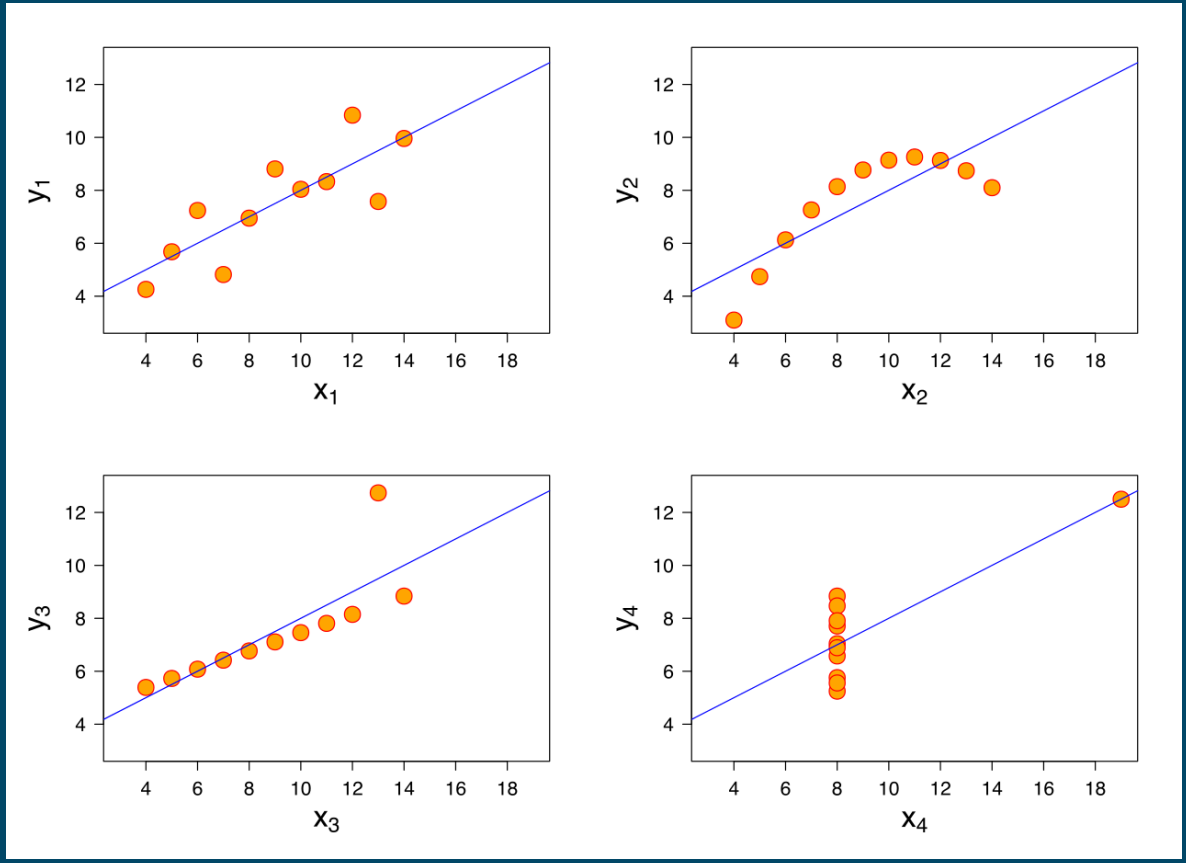
Linear Regression

We want to find the “best” line (linear function y=f(X)) to explain the data.



R^2-values

We can **always** fit a linear model to any dataset, but how do we know if there is a **real linear relationship**?



R^2-values

Approach: Measure how much the total "noise" (variance) is reduced when we include the line as an offset.

R-squared: a suitable measure. Let $\hat{y} = X\,\hat{\beta}$ be a predicted value, and $\bar{y}$ be the sample mean. Then the R-squared value is

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

And can be described as the fraction of the total variance not explained by the model.

$R^2 = 0$: bad model. No evidence of a linear relationship.

$R^2 = 1$: good model. The line perfectly fits the data.

Regularization with Secondary Criteria

While secondary criteria can be measured after the model is built, its too late then to affect the model.

Using secondary criteria **during** the optimization process is called **"regularization".**

Examples:

- **L1 regularization** adds a term to the measure being optimized which is the **sum of absolute value of model coefficients**.
- **L2 regularization** adds a term to the measure being optimized which is the **sum of squares of model coefficients**.
- Considering these can result in reducing or dropping features/coefficients

1. Pipeline Python & Spark Regression (Python)

s6.1 | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Runs

Cmd 1

Two Example Regressions: Python & PySpark

- Based on prior example (which did not have pipline)
- Using the same data
- Both using Pipeline

Cmd 2

Housing data

Cmd 3

```
1 %sh
2 wget https://raw.githubusercontent.com/bcbarsness/machine-learning/master/USA_Housing.csv
```

```
--2020-03-30 20:53:51--  https://raw.githubusercontent.com/bcbarsness/machine-learning/master/USA_Housing.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.52.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.52.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 726209 (709K) [text/plain]
Saving to: 'USA_Housing.csv.1'

     0K .......... .......... .......... .......... ..........  7% 3.50M 0s
    50K .......... .......... .......... .......... .......... 14% 7.02M 0s
   100K .......... .......... .......... .......... .......... 21% 44.7M 0s
   150K .......... .......... .......... .......... .......... 28% 8.15M 0s
   200K .......... .......... .......... .......... .......... 35%  190M 0s
   250K .......... .......... .......... .......... .......... 42% 54.2M 0s
   300K .......... .......... .......... .......... .......... 49%  174M 0s
   350K .......... .......... .......... .......... .......... 56%  169M 0s
   400K .......... .......... .......... .......... .......... 63% 8.83M 0s
```

Recommendations

Recommendation example

- Alternating Least Squares (ALS)
  - Uses collaborative filtering, which makes recommendations based on which items users interacted with in the past
- Recommendations
  - By User
  - By Item

```
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
ratings = spark.read.text("/data/sample_movielens_ratings.txt")\
 .rdd.toDF()\
 .selectExpr("split(value , '::') as col")\
 .selectExpr(
  "cast(col[0] as int) as userId",
  "cast(col[1] as int) as movieId",
  "cast(col[2] as float) as rating",
  "cast(col[3] as long) as timestamp")
training, test = ratings.randomSplit([0.8, 0.2])
als = ALS()\
 .setMaxIter(5)\
 .setRegParam(0.01)\
 .setUserCol("userId")\
 .setItemCol("movieId")\
 .setRatingCol("rating")
print als.explainParams()
```

```
alsModel = als.fit(training)
predictions = alsModel.transform(test)
```

```
alsModel.recommendForAllUsers(10)\
 .selectExpr("userId", "explode(recommendations)").show()
alsModel.recommendForAllItems(10)\
 .selectExpr("movieId", "explode(recommendations)").show()
```

Alternating Least Squares (ALS) on DataBricks

5. Recommendation (1) (Python)

s5.5 python 2 | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Runs

Cmd 1

Ch. 28 Recommenation

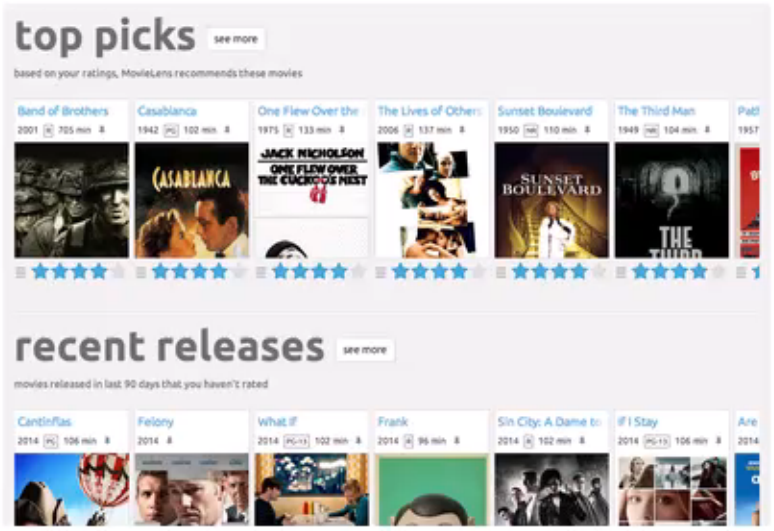
Cmd 2

Warning: Needs to run in Python 2.x

- Use DataBricks environment 5.5

Cmd 3

MovieLens dataset



Cmd 4

```
%fs
ls /databricks-datasets/definitive-guide/data/
```

Unsupervised learning

Clustering

- Unsupervised learning discovers patterns or derive a concise representation of the underlying structure of a given dataset
  - Use cases
    - Finding anomalies
    - Topic modeling
- Scalability of Spark MLlib Models

Model	Statistical recommendation	Computation limits	Training examples
k-means	50 to 100 maximum	Features x clusters < 10 million	No limit
Bisecting *k*-means	50 to 100 maximum	Features x clusters < 10 million	No limit
GMM	50 to 100 maximum	Features x clusters < 10 million	No limit
LDA	An interpretable number	1,000s of topics	No limit

K-means clustering

- A user-specified number of clusters (*k*) are randomly assigned to different points in the dataset.
- The unassigned points are then "assigned" to a cluster based on their proximity (measured in Euclidean distance) to the previously assigned point.
- Once this assignment happens, the center of this cluster (called the centroid) is computed, and the process repeats.

```
from pyspark.ml.clustering import KMeans
km = KMeans().setK(5)
print km.explainParams()
kmModel = km.fit(sales)
```

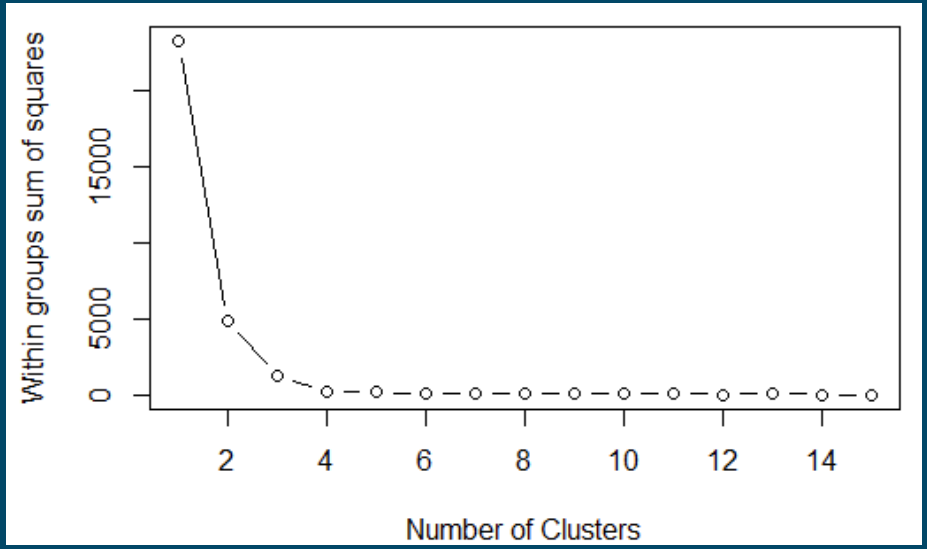
ml.clustering KMeans

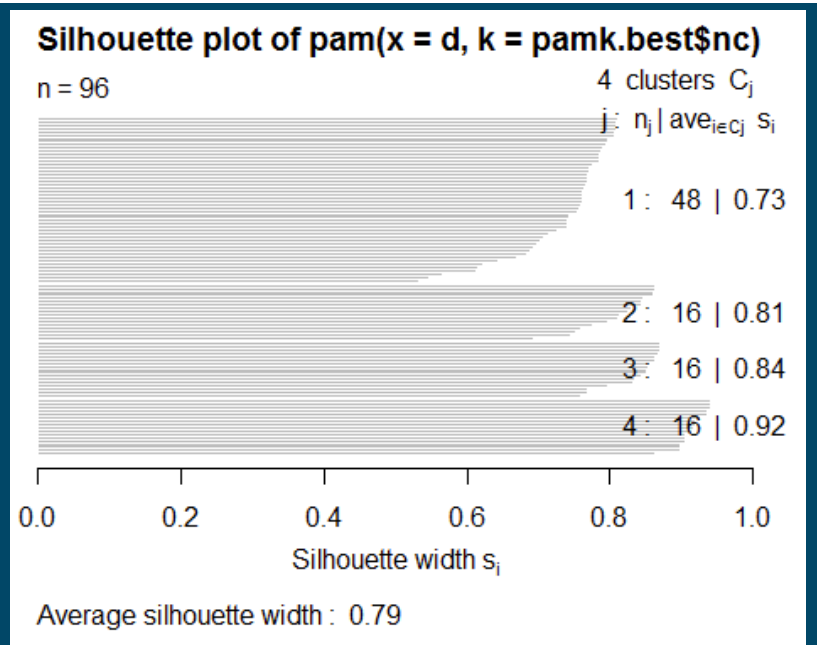
- Purpose
  - group objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense) to each other than to those in other groups (clusters).
- k-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters.

```
cols = ['Session_Connection_Time',
 'Bytes Transferred',
 'Kali_Trace_Used',
 'Servers_Corrupted',
 'Pages_Corrupted',
 'WPM_Typing_Speed']
assembler = VectorAssembler(inputCols=cols, outputCol='features')

scaler = StandardScaler(inputCol='features', outputCol='scaledFeatures')

k_means_2 = KMeans(featuresCol='scaledFeatures', k=2)
k_means_3 = KMeans(featuresCol='scaledFeatures', k=3)

pipeline_k2 = Pipeline(stages=[assembler, scaler, k_means_2])
pipeline_k3 = Pipeline(stages=[assembler, scaler, k_means_3])
```

K-means requires selecting a *k*

- Elbow method
  - Choose a number of clusters so that adding another cluster doesn't give much better modeling of the data.
- Silhouette
  - The silhouette measures how close each point in one cluster is to points in the neighboring clusters
    - 0.71-1.0; A strong structure has been found
    - 0.51-0.70; A reasonable structure has been found
    - 0.26-0.50; The structure is weak and could be artificial. Try additional methods of data analysis.
    - < 0.25; No substantial structure has been found





https://stackoverflow.com/questions/15376075/cluster-analysis-in-r-determine-the-optimal-number-of-clusters

K-means on Databricks

5. K-means (Python)

s5, p2 | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Runs

Cmd 1

K-means example

- Dervied from: https://medium.com/tensorist/using-k-means-to-analyse-hacking-attacks-81957c492c93
- Data to determine how many people tried to hack into a network

Cmd 2

Warning: Needs to run in Python 2.x

- To move to Python 3.x, simple changes to use of zip for knee analysis

Cmd 3

Setting

- From the above link

RhinoTech is a large technology firm that has been recently hacked. Luckily, the forensic engineers at the company have been able to grab metadata about each session used by the hackers to connect to RhinoTech servers. This data includes information such as session time, locations, words-per-minute typing speed, etc.

You have been informed that there are three potential hackers that perpetrated the attack.

The RhinoTech forensic team are certain that the first two hackers were involved, but they want to know whether the third hacker was involved as well. One last key piece of information you've been given by the forensic engineers is that the hackers trade off attacks equally.

For example, imagine there were 100 attack instances. If 2 hackers were involved, then each hacker would have about 50 attacks. but in the case of 3 hackers, each would have only 33 attacks.

Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA)

- A hierarchical clustering model typically used to perform topic modelling on text documents
- It interprets each document as having a variable number of contributions from multiple input topics
- Input text data must be converted to a number format

```
from pyspark.ml.feature import Tokenizer, CountVectorizer
tkn = Tokenizer().setInputCol("Description").setOutputCol("DescOut")
tokenized = tkn.transform(sales.drop("features"))
cv = CountVectorizer()\
  .setInputCol("DescOut")\
  .setOutputCol("features")\
  .setVocabSize(500)\
  .setMinTF(0)\
  .setMinDF(0)\
  .setBinary(True)
cvFitted = cv.fit(tokenized)
prepped = cvFitted.transform(tokenized)
```

```
from pyspark.ml.clustering import LDA
lda = LDA().setK(10).setMaxIter(5)
print lda.explainParams()
model = lda.fit(prepped)
```

ml.clustering LDA

- Purpose
  - Cluster objects
- A generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar
  - if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's presence is attributable to one of the document's topics.
- Example

```
lda = LDA(k=10,maxIter=5)
# Pipeline
pipeline = Pipeline(stages=[tokenizer, stopWrds, countVector, lda])
```

https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

Topics from reviews

LDA on Databricks

4. LDA Topics (Python)

s5.5 python 2 | File | View: Code | Permissions | Run All | Clear | Publish | Comments | Runs

Cmd 1

Topic Analysis (of reviews) using LDA from spark ml

- For ecommerce review data
  - Run LDA for topic analysis
  - Graph the results References
- https://medium.com/@connectwithghosh/topic-modelling-with-latent-dirichlet-allocation-lda-in-pyspark-2cb3ebd5678e
- https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews

Cmd 2

Warning: Needs to run in Python 2.x

Cmd 3

```
%sh
wget -O reviews.csv 'https://raw.githubusercontent.com/AFAgarap/ecommerce-reviews-analysis/master/Womens%20Clothing%20E-Commerce%20Reviews.csv'
```

```
--2020-03-31 15:44:09--  https://raw.githubusercontent.com/AFAgarap/ecommerce-reviews-analysis/master/Womens%20Clothing%20E-Commerce%20Reviews.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.52.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.52.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8483448 (8.1M) [text/plain]
Saving to: ‘reviews.csv’

     0K .......... .......... .......... .......... ..........  0% 3.97M 2s
    50K .......... .......... .......... .......... ..........  1% 8.12M 2s
   100K .......... .......... .......... .......... ..........  1% 24.1M 1s
   150K .......... .......... .......... .......... ..........  2% 31.4M 1s
```

Supplemental information for LDA

FYI Supplemental details, to understand the GUI

LDA has two important hyperparameters (alpha and beta) that are used to create the topics. See this link for a brief summary.) The pyLDAvis interface includes a lambda parameter that can be used to explore term rankings within a topic. The interface is demonstrated and described here. The following is a summary.

The relevance of a term to a given topic is displayed in the bar chart to the right when a topic is selected. Note that the terms are ranked from most important to least. These ranks change with a change in λ.

The *relevance* of a term to a given topic is determined by a weight parameter, $0 \leq \lambda \leq 1$, according to this equation: $\lambda \log(p(\text{term} \mid \text{topic})) + (1 - \lambda) \log(p(\text{term} \mid \text{topic})/p(\text{term}))$

Notice that as λ approaches 1, the second part of the equation becomes 0, leaving only $1 * \log(p(\text{term} \mid \text{topic}))$.

Conversely, as λ approaches 0, the first part of the equation becomes 0, leaving the second part of the equation, $\log(p(\text{term} \mid \text{topic})/p(\text{term}))$.

FYI Supplemental details, to understand the GUI

Thus, λ balances the two probabilities. The following table summarizes these terms (dropping the log)

Description	Equation element	λ
Part 1 **Probability of term found in topic**	p(term | topic)	When λ = 1
Part 2 **Probability of term in topic, divided by probability of the term in all documents**	p(term | topic) / p(term)	When λ = 0

FYI Supplemental details, to understand the GUI

When λ approaches 1, the relevance becomes simply the probability of the term in the topic. However, as λ approaches 0, the probability of the term in all documents becomes prominent. When a topic is selected, the terms are ranked at the right side of the interface. The red bars represent the frequency of a term in a given topic, (proportional to p(term | topic)), and the blue bars represent a term's frequency across the entire corpus, (proportional to p(term)).
Changing the value of λ adjusts the term rankings -- small values of λ (near 0) highlight potentially rare, relatively exclusive terms for the selected topic, and large values of λ (near 1) highlight frequent, but not necessarily exclusive, terms for the selected topic.
A user study in their paper suggested that setting λ near 0.6 aids users in topic interpretation, although this is likely to vary across topics and data sets (hence the tool allows you to adjust λ).

FYI Supplemental details, to understand the GUI

LDA estimates

- The probabilistic topic model estimated by LDA consists of two tables (matrices).
  - The first table describes the chance of selecting a particular part (word) when sampling a particular topic (category).
  - The second table describes the chance of selecting a particular topic when sampling a particular document or composite.

FYI Supplemental details, to understand the GUI

Harry potter

	harry	hermione	malfoy	magic	wand	robe	spell
Document 0	3	0	0	1	2	0	0
Document 1	0	3	0	1	0	2	0
Document 2	0	0	3	1	0	0	2
Document 3	2	2	2	0	0	0	0

- Table 1: Term in topic frequencies

	Topic 0	Topic 1	Topic 2	Topic 3
harry	0.709	0.001	0.001	0.003
hermione	0.001	0.709	0.001	0.003
malfoy	0.001	0.001	0.709	0.003
magic	0.001	0.001	0.001	0.980
wand	0.284	0.001	0.001	0.003
robe	0.001	0.284	0.001	0.003
spell	0.001	0.001	0.284	0.003

- Table 2: Topic in document frequencies

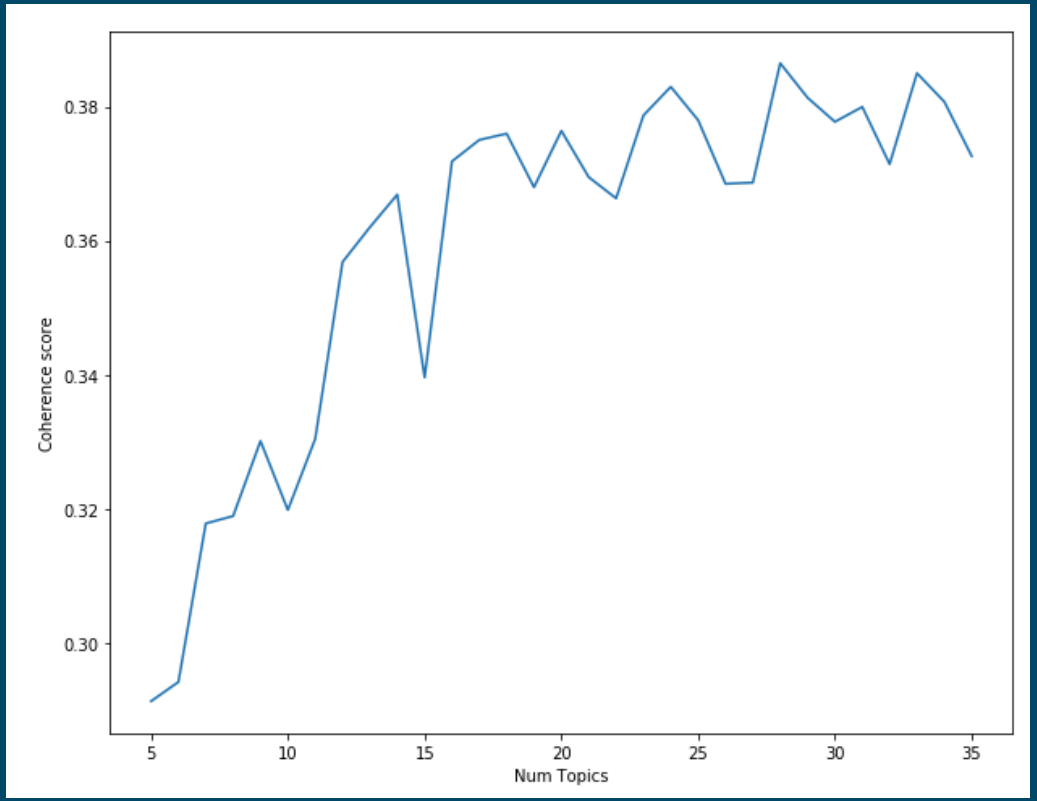
	Topic 0	Topic 1	Topic 2	Topic 3
Document 0	0.727	0.045	0.045	0.182
Document 1	0.045	0.727	0.045	0.182
Document 2	0.045	0.045	0.727	0.182
Document 3	0.318	0.318	0.318	0.045

https://towardsdatascience.com/basic-nlp-on-the-te

FYI Supplemental details, to understand the GUI

LDA's k

- Number of topics is a parameter
- Select similar to k-means
  - Try different k
  - PySpark
    - Use EMLDAOptimizer
    - E.g. LDA optimized
  - Gensim
    - Measure with coherence score



Important to remember

- Modeling in Spark ML lib
  - Features must be present in a DataFrame
  - The model is instantiated and its parameters set
  - Call model fit() and then transform()
  - View the predictions