INSIGHT

Spark MLlib

Apache Spark's scalable machine learning library

Disciplina de Garimpagem de Dados [06/12/2017]

INSIGHT

DEPARTAMENTO DE COMPUTAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

AGENDA

1. Overview
2. Basic Statistics
3. ML Pipelines
4. Feature Extraction and Transformation
5. Evaluation Metrics
6. Hands-on

OVERVIEW

OVERVIEW

APACHE Spark MLlib

- Spark's machine learning library
- Make **practical** machine learning **scalable** and **easy**
- Provides several tools
    - **ML Algoritms**, clustering, classification, regression and collaborative filtering
    - **Feature** extraction, transformation, dimensionality reduction and selection
    - **Pipelines** for constructing, evaluating and tuning
    - **Persistence** for saving and load algorithms, models and pipelines
    - **Utilities** such as linear algebra, statistics and data handling

OVERVIEW

- **Classification**
    - Linear and Logistic regression
    - SVM, Naive Bayes, Decision Tree, others
- **Clustering**
    - K-Means
    - Gaussian Mixture Model GMM
    - Power Iteration Clustering PIC
    - Latent Dirichlet Allocation LDA
- **Recommender Systems**
    - Collaborative Filtering - Alternating Least Square ALS

BASIC STATISTICS

BASIC STATISTICS

- **Correlation**
  - Pearson and Spearman implementations
- **Hypothesis testing**
  - Powerful tool in statistics to determine whether a result is statistically significant, whether this result occurred by chance or not
  - Pearson’s Chi-squared

ML PIPELINES

ML PIPELINES

- Uniform set of high-level APIs built on top of **DataFrames**
- Standardize APIS for ML algorithms to make it easier to combine multiple algorithms
- Inspired by the **scikit-learn** project
    - Pearson and Spearman implementations

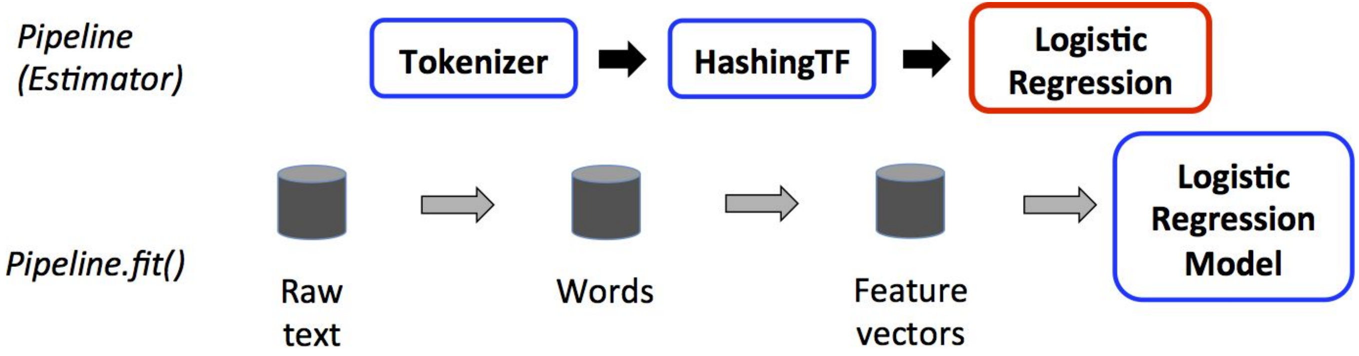
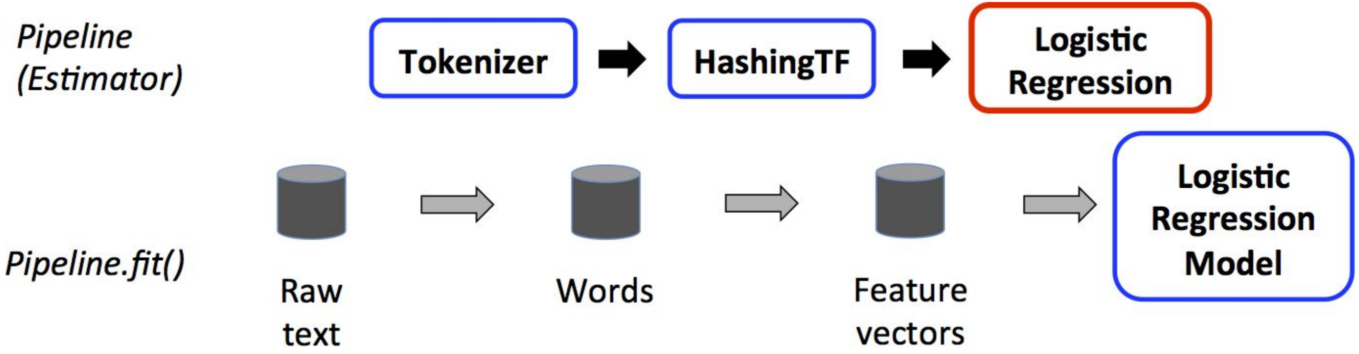


Figure from https://spark.apache.org/docs/latest/ml-pipeline.html

ML PIPELINES

- **Main concepts**
    - **DATAFRAME** ML dataset
    - **TRANSFORMER** An algorithm which can transform one **DataFrame** into another **DataFrame**. E.g. ML Model, DataFrame with Feature -> DataFrame with predictions
    - **ESTIMATOR** An algorithm which can be fit on a DataFrame to produce a Transformer. E.g. a **learning algorithm** which trains on a **DataFrame** and produces a **model**
    - **PIPELINE** Chains multiple Transformers and Estimators
    - **PARAMETER** Common API for specifying parameters

ML PIPELINES




```
1.   # Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF, and
     lr.
2.   tokenizer = Tokenizer(inputCol="text", outputCol="words")
3.   hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
4.   lr = LogisticRegression(maxIter=10, regParam=0.001)
5.   pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

EXTRACTING, TRANSFORMING AND SELECTING FEATURES

FEATURE EXTRACTORS

- Extract features from "raw" data
  - **TF-IDF** (Term frequency-inverse document frequency) feature vectorization widely used in text mining to determine the importance of a term to a document in the corpus
  - In MLlib, **TF** and **IDF** are separated to make them flexible

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

FEATURE EXTRACTORS

- **TF** *HashingTF* and *CountVectorizer*
- **IDF** Estimator which is fit on a dataset and produces *IDFModel*

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j

df_i = number of documents containing i

N = total number of documents

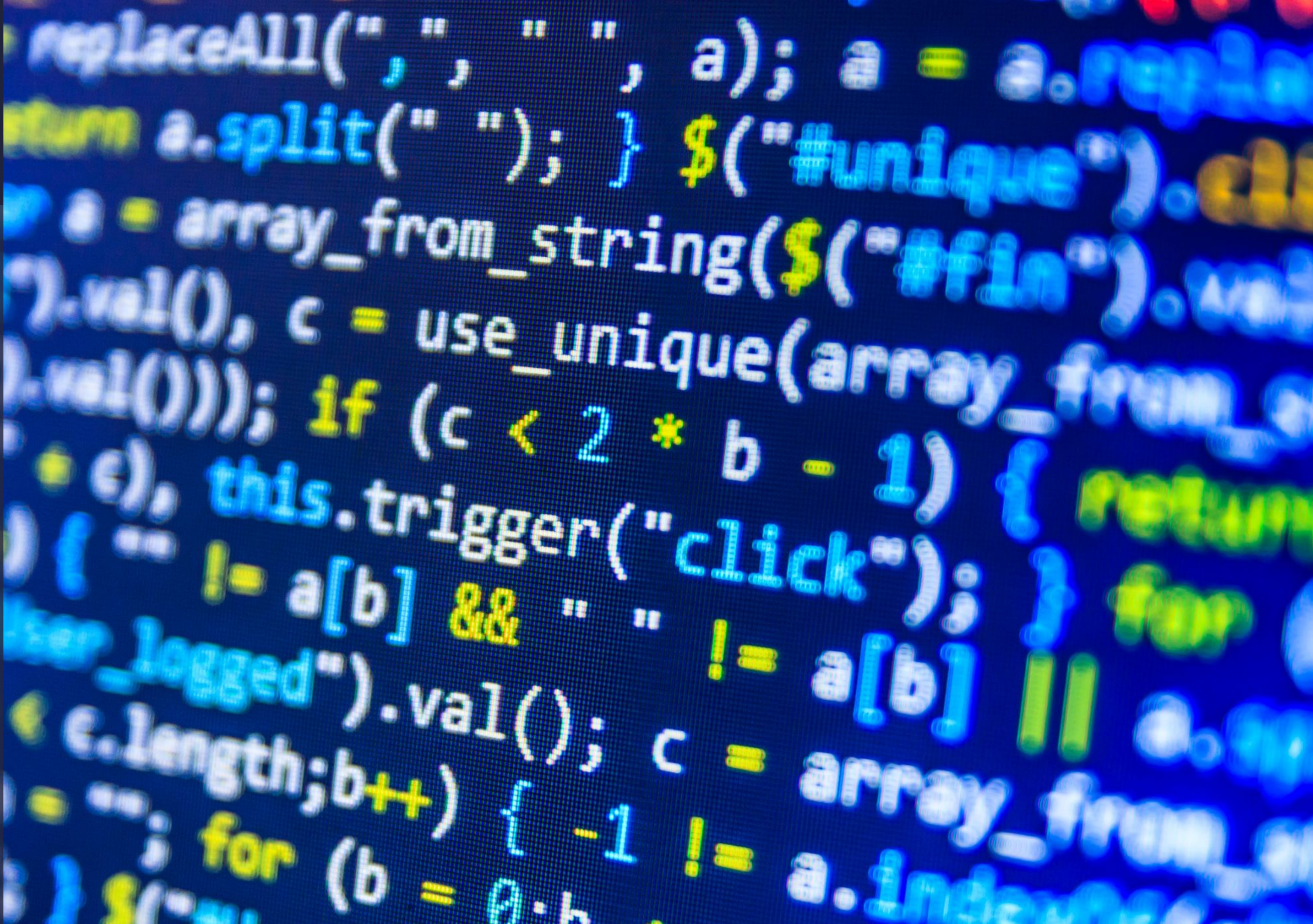
FEATURE EXTRACTORS

- **Word2Vec**
  - Computes distributed vector representation of words
  - Similar words are close in the vector space

```
from pyspark.mllib.feature import Word2Vec

input = sc.textFile("text_by_line").map(lambda row: row.split(" "))

word2vec = Word2Vec()
model = word2vec.fit(input)

synonyms = model.findSynonyms('china', 40)

for word, cosine_distance in synonyms:
    print("{}: {}".format(word, cosine_distance))
```

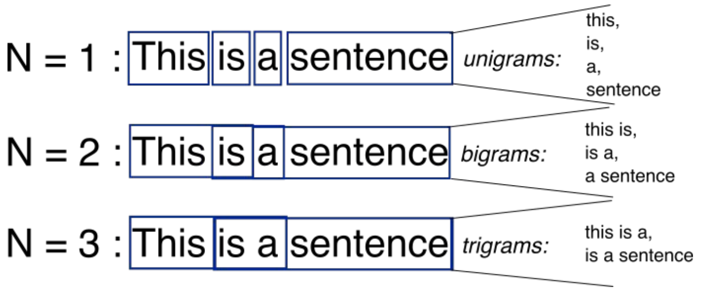
[1] https://en.wikipedia.org/wiki/Word2vec, https://code.google.com/archive/p/word2vec/, https://www.quora.com/How-does-word2vec-work

HANDS-ON ML PIPELINES



FEATURE TRANSFORMERS

- **Tokenizer**
  - Takes text (sentences) and breaks it into terms (words)
- **RegexTokenizer**
  - Advanced tokenization based on regular expression matching
- **StopWordsRemover**
- **n-gram**
  - Contigous sequence of *n* tokens from a given sequence of text or speech



FEATURE TRANSFORMERS

- **Binarizer**
  - Threshold numerical features to binary (0/1) features
- **PCA**
  - Principal Component Analysis
  - Trans a model to project vectors to a low-dimensional space

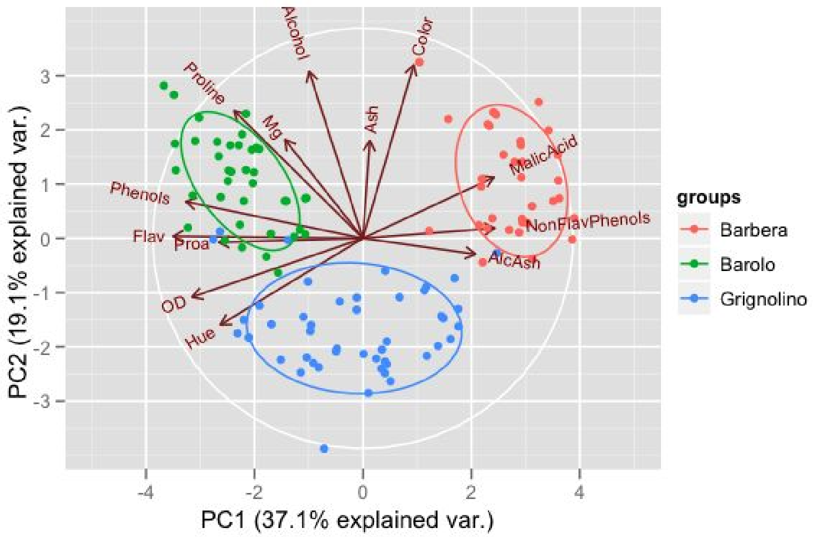


Figure from https://stats.stackexchange.com/questions/7860/visualizing-a-million-pca-edition

FEATURE TRANSFORMERS

- **StringIndexer** encodes a string column of labels to a column of label indices
- **IndexToString** maps a column of label indices back to a column containing the original labels as strings

```
 id | categoryIndex | originalCategory
----|---------------|-----------------
 0  | 0.0           | a
 1  | 2.0           | b
 2  | 1.0           | c
 3  | 0.0           | a
 4  | 0.0           | a
 5  | 1.0           | c
```

FEATURE TRANSFORMERS

- **Normalizer** transforms a dataset of Vector rows, normalizing each Vector *p-norm* or *Lˆp-norm*
- **StandardScaler**
- **MinMaxScaler**
- **MaxAbsScaler**
- Check out more TRANSFORMERS in https://spark.apache.org/docs/latest/ml-features.html

MODEL SELECTION AND TUNING

MODEL SECTION AND TUNING

- **Model Selection**
    - Using data to find the best model or parameters for a given task
    - Also called *tuning*
    - It can be done for **Estimators** or for entire **Pipeline**
- **Supported tools**
    - TrainValidationSplit
    - CrossValidator
    - ParamGridBuilder - to help construct the parameter grid

CROSS-VALIDATION

- Splits the dataset into a set of folds used as separate training and test datasets
- For example: k = 3 folds
  - 3 (training, test) datasets pairs
  - ⅔ for training and ⅓ for testing

```
paramGrid = ParamGridBuilder() \
    .addGrid(hashingTF.numFeatures, [10, 100, 1000]) \
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=2)  # use 3+ folds in practice
```

TRAIN-VALIDATION SPLIT

- Evaluate each combination of parameters once
- Less expensive than CrossValidator
- May produce unreliable results when the training dataset is not sufficiently large
- Create single (training, test) using the *trainRatio*

```
paramGrid = ParamGridBuilder()\
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .addGrid(lr.fitIntercept, [False, True])\
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])\
    .build()

# In this case the estimator is simply the linear regression.
# A TrainValidationSplit requires an Estimator, a set of Estimator ParamMaps, and an Evaluator.
tvs = TrainValidationSplit(estimator=lr,
                           estimatorParamMaps=paramGrid,
                           evaluator=RegressionEvaluator(),
                           # 80% of the data will be used for training, 20% for validation.
                           trainRatio=0.8)
```

HANDS-ON