

# Анализ больших данных с Apache Spark

## Лекция 2. Spark.ML

Мурашкин Вячеслав  
2017

<https://github.com/a4tunado/lectures-hse-spark/tree/master/express/002>

# Лекция 2. Spark.ML

- Spark DataFrame
- Алгоритмы машинного обучения
- Предобработка данных
- ML Pipelines
- Подбор оптимальных значений гиперпараметров
- Пример: Пайплайн обработки текста
- Пример: Коллаборативная фильтрация
- Задание: Кластеризация


# Spark.ML

- библиотека распределенного машинного обучения
- реализованы алгоритмы регрессии, классификации, кластеризации и коллаборативной фильтрации
- содержит утилиты предобработки данных и подбора оптимальных значений гиперпараметров алгоритмов (кросс валидация)
- основной API: pyspark.ml работает с DataFrame
- pyspark.mllib - работает с RDD, не развивается











# DataFrame и SQL запросы

- `pyspark.sql.`[SparkSession](#)
  - точка входа для работы со SparkSQL
  - содержит ссылку на `SparkContext` (создается при необходимости)
- [DataFrame](#)
  - табличное представление данных
  - набор типизированных записей `pyspark.sql.Row`
  - содержит метаданные об именах и типах полей (схему)
  - является надстройкой над RDD

# DataFrame и SQL запросы

 jupyter sparksql Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

         Code  CellToolbar

```
In [ ]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SQL App').getOrCreate()
sc = spark.sparkContext

In [ ]: !cat ../data/sales_header.csv

In [ ]: def parse_row(line):
# TODO: parse row values
return line.split(',')

sales_rdd = sc.textFile('../data/sales.csv').map(parse_row)

In [ ]: with open('../data/sales_header.csv') as src:
sales_header = src.read().strip().split(',')

sales_df = sales_rdd.toDF(sales_header)
sales_df.registerTempTable('sales')

sales_by_state_df = spark.sql("SELECT SUM(1) FROM sales")
print sales_by_state_df.rdd.collect()

# TODO: print revenue by Country and State
```

# Spark MLlib - Algorithms

- регрессия / классификация
  - [Linear regression](#)
  - [Logistic regression](#)
  - [Random forest](#)
  - [Gradient-boosted tree](#)
- кластеризация
  - [K-means](#)
  - [Latent Dirichlet allocation \(LDA\) \(topic modelling\)](#)
  - [Gaussian Mixture Model \(GMM\)](#)
- коллаборативная фильтрация (рекомендации)
  - [Alternating least squares \(ALS\)](#)

<http://spark.apache.org/docs/latest/ml-classification-regression.html>

# Spark MLlib - Algorithms API

- *Estimator.fit(DataFrame)* - реализует алгоритм обучения, возвращает объект типа *Transformer*
- *Transformer.summary* - информация о результате процесса обучения
- *Transformer.transform(DataFrame)* - возвращает предсказания модели
- *Transformer.save(path)* - сохраняет обученную модель на диск
- *Transformer.load(path)* - загружает сохраненную модель

# Предобработка данных и выделение признаков

- Feature Extractors
  - [TF-IDF](#)
  - [Word2Vec](#)
  - [CountVectorizer](#)
- Feature Transformers
  - [Tokenizer](#)
  - [StopWordsRemover](#)
  - [N-Gram](#)
  - [OneHotEncoder](#)
  - [StandardScaler](#)



# Spark.ML - Pipelines

- позволяет объединять последовательность алгоритмов в единый интерфейс
- алгоритмы делятся на 2 типа:
  - **Transformer** - как правило создает DataFrame с новым столбцом, в котором доступны результаты преобразования
  - **Estimator** - реализует обучение модели, на выходе объект типа Transformer
- предоставляет единый API для задания параметров алгоритмов

# Spark.ML - Тюнинг моделей

- для подбора оптимальных значений гиперпараметров предоставлено два интерфейса: `CrossValidator` и `TrainValidationSplit`
- эти интерфейсы принимают на вход объекты типа:
  - **Estimator** - алгоритм или пайплайн обучения
  - **Evaluator** - метрика оценки качества алгоритма на отложенных данных
    - `RegressionEvaluator`
    - `BinaryClassificationEvaluator`
    - `MulticlassClassificationEvaluator`
- для задания диапазона поиска значений параметров используется утилита `ParamGridBuilder`

<https://github.com/apache/spark/blob/master/docs/mllib-evaluation-metrics.md>

## Пример. Пайплайн обработки текста

Number of reviews: 19411

overall	reviewText
5.0	Good case, solid ...
5.0	This is a terribl...
2.0	I bought this so ...
4.0	works great and c...
4.0	This case is afor...
5.0	This case was by ...
4.0	very good charger...
5.0	Sprint wanted \$30...
5.0	It is Samsung bra...
5.0	good oem car char...

# Пример. Пайплайн обработки текста

"It works great. Doesn't heat up like crazy like the other ones..."

Tokenizer

["It", "works", "great", "Doesn't", "heat", "up", "like", "crazy", "like",  
"the", "others", "ones"]

StopWordsRemover

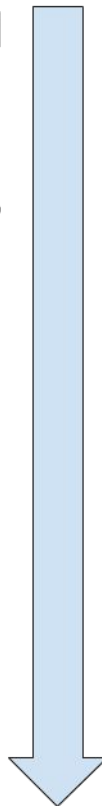
["works", "great", "heat", "up", "like", "crazy", "like", "ones"]

Ngram

[("works", "great"), ("great", "heat"), ("heat", "up"), ("up", "like"),  
("like", "crazy"), ("crazy", "like"), ("like", "ones")]

HashingTF

[0, 1, 0, ... 1, 1, 0]



# Пример. Пайплайн обработки текста

Term	Index
John	1
likes	2
to	3
watch	4
movies	5
Mary	6
too	7
also	8
football	9

- *John likes to watch movies.*
- *Mary likes movies too.*
- *John also likes football.*

John	likes	to	watch	movies	Mary	too	also	football
1	1	1	1	1	0	0	0	0
0	1	0	0	1	1	1	0	0
1	1	0	0	0	0	0	1	1

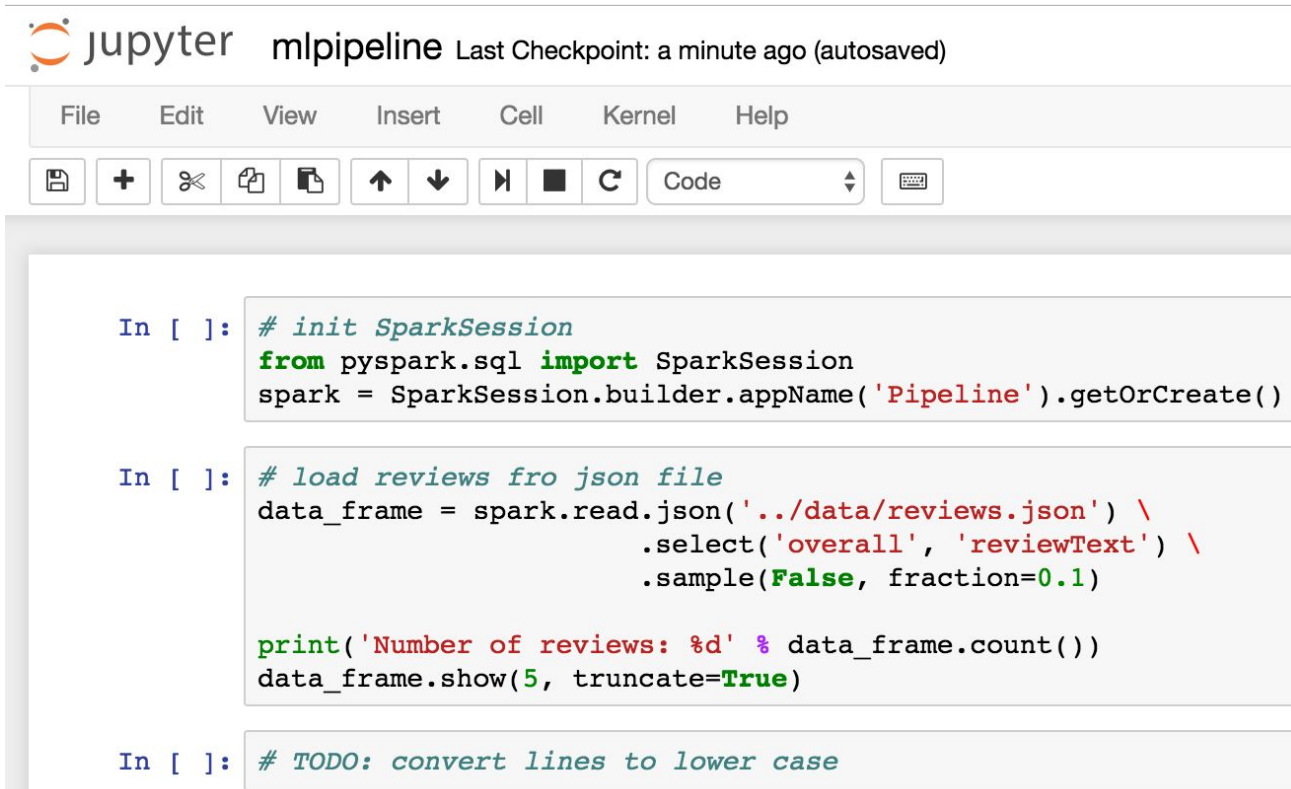
## Пример. Пайплайн обработки текста

```
function hashing_vectorizer(features : array of string, N : integer):  
    x := new vector[N]  
    for f in features:  
        h := hash(f)  
        x[h mod N] += 1  
    return x
```

# Пример. Пайплайн обработки текста

```
uint32_t jenkins_one_at_a_time_hash(const uint8_t* key, size_t length) {  
    size_t i = 0;  
    uint32_t hash = 0;  
    while (i != length) {  
        hash += key[i++];  
        hash += hash << 10;  
        hash ^= hash >> 6;  
    }  
    hash += hash << 3;  
    hash ^= hash >> 11;  
    hash += hash << 15;  
    return hash;  
}
```

# Пример. Пайплайн обработки текста



The image shows a Jupyter Notebook interface with the title "mlpipeline" and a status message "Last Checkpoint: a minute ago (autosaved)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and a dropdown menu currently set to "Code". The notebook contains three code cells. The first cell initializes a SparkSession. The second cell loads reviews from a JSON file, selects specific columns, and samples the data. The third cell is a TODO comment for converting lines to lower case.

```
In [ ]: # init SparkSession
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Pipeline').getOrCreate()

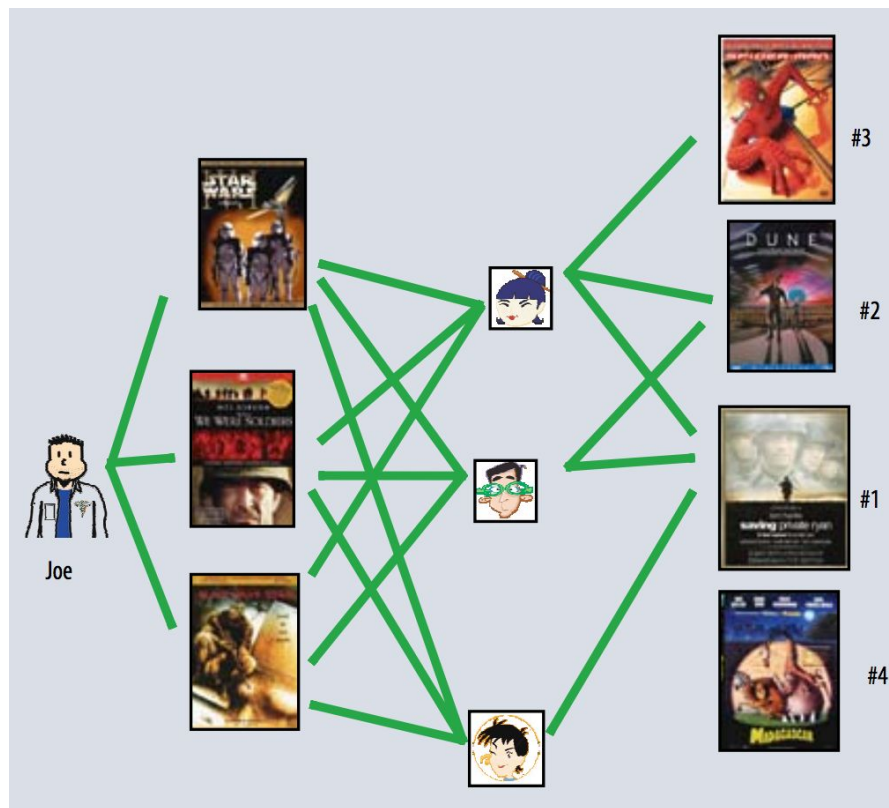
In [ ]: # load reviews fro json file
data_frame = spark.read.json('../data/reviews.json') \
    .select('overall', 'reviewText') \
    .sample(False, fraction=0.1)

print('Number of reviews: %d' % data_frame.count())
data_frame.show(5, truncate=True)

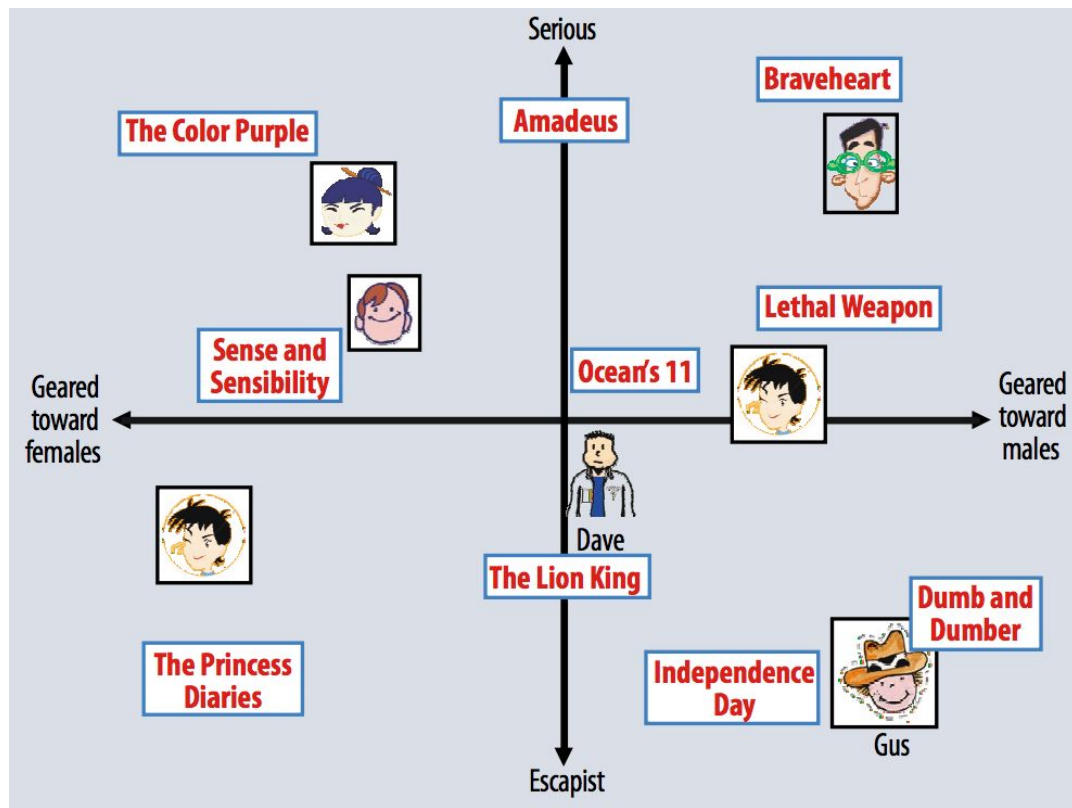
In [ ]: # TODO: convert lines to lower case
```



# Пример. Рекомендации



# Пример. Рекомендации



## Пример. Рекомендации

$$\min_{q^*, p^*} \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2)$$

# Пример: Кластеризация. Исходные данные

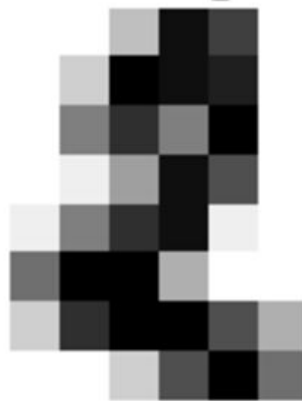
Training: 0



Training: 1



Training: 2



Training: 3



# Полезные материалы

- Machine Learning Library (MLlib) Guide  
<http://spark.apache.org/docs/latest/ml-guide.html>
- pyspark.ml package  
<http://spark.apache.org/docs/latest/api/python/pyspark.ml.html>
- Evaluation Metrics - RDD-based API  
<https://github.com/apache/spark/blob/master/docs/mllib-evaluation-metrics.md>
- Feature hashing  
[https://en.wikipedia.org/wiki/Feature\\_hashing](https://en.wikipedia.org/wiki/Feature_hashing)
- Matrix factorization techniques for recommender systems  
<https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf>