

Mining massive databases

session 2

Diego Saez-Trumper

Data Science - UCU

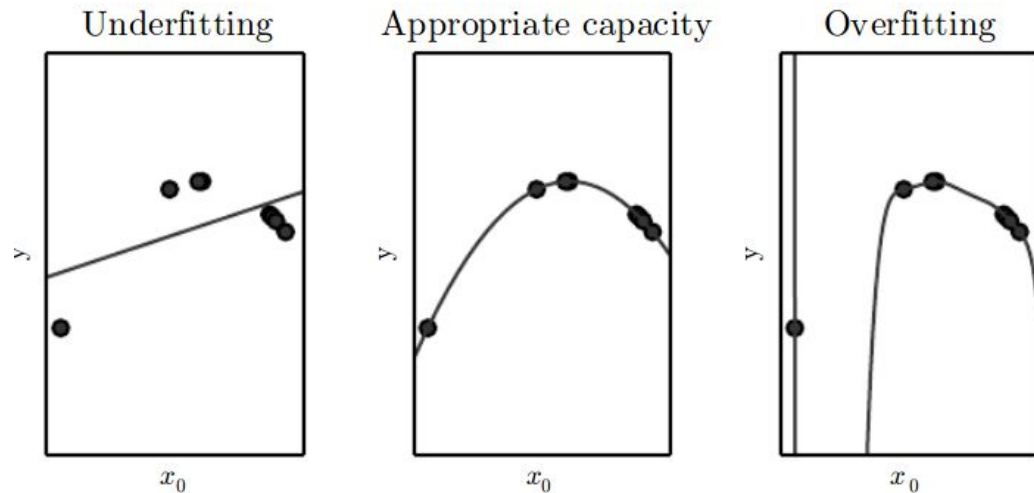
Quick Review

- The task, T
 - Find similar items
 - Classification
 - Classification with missing inputs
 - Regression
 - Machine Translate
 - Structured output
 - Outlier detection
 - Etc, etc.
- The performance measure, P
 - Error rate
 - MSE
 - F1
- The Experience, E
 - The data



Quick Review

- Overfitting
- Underfitting
- Capacity



Source: <http://www.deeplearningbook.org>

Real world applications

- Recommender Systems
- Fraud detection
- Anomaly detection
- Ads selection
- User modeling
- Etc.

Text Mining

- Data mining using text as input
- Why text mining?
 - Text is sequence of symbols that combined can represent high level ideas
 - Many items can be represented as text (e.g. content, products, entities)
 - It is easy to interpret and generalize (good for pedagogic purposes)
 - Most of the techniques used in text mining can be extended to other domains

"There is no outside-text"
J. Derrida



Text representation

Key concepts and approaches:

- Bag of Words
 - Represents documents as bag of unsorted words.
- N-Gram models (word-based)
 - Represents documents as bag of unsorted tuples of n-words.
- TF-IDF
 - Vector model to represent the importance of word in a document.
- Word Embeddings
 - Group of models based on NN, that allows represent words in a vectorial space, making possible perform [mathematical operations with text](#).

Text Mining Pipeline

1. Parse text:
 - Get the interesting content, i.e. remove noise
2. Remove punctuation and other characters
 - “This, is the end.\n” -> “This is the end”
3. Put everything in lowercase (or upper)
 - “This is the end” -> “this is the end”
4. Stemming and/or lemmatization:
 - “Cars” to “car”
5. Remove stopwords:
 - “This is the end” -> “end”

BBC News Sport Weather Shop Earth Travel More Search


SPORT FOOTBALL

Home Football Formula 1 Cricket Rugby U Tennis Golf Athletics Cycling All Sport

African Football > Results Fixtures Tables Live Scores All Teams Leagues & Cups

The draw for group stage of the Confederation Cup is completed

26 April 2017 | Football



DR Congo's TP Mazembe will defend their Confederation Cup title after being knocked out of the African Champions League for a second successive season

TP Mazembe of DR Congo face trips to South Africa, Guinea and Gabon as they bid to defend their Confederation Cup title.

The five-time African champions have been draw in a pool with SuperSport United, Horoya and Mounana in the expanded group stage.

Three-time winners CS Sfaxien of Tunisia will face South Africa's Platinum Stars, debutants Mbabane Swallows of Swaziland and Mouloudia Alger from Algeria.

Moroccan side FUS Rabat, who won the tournament in 2010, face two sides making the debuts in the group stages in Rivers United of Nigeria and Uganda's KCCA.

The group is completed by Tunisia's Club Africain, who were crowned continental champions in 1991 and runners-up in this competition 20 years later.

Top Stories

- England all-rounder Woakes to have scan on side strain - Morgan 5h | Cricket
- Murray into French Open third round 8h | Tennis 54
- Chelsea paid £150.8m by Premier League - how much did your club receive? 5h | Football

Related to this story

- Swallows grab famous win over AC Leopards 15 Apr | Football
- Holders TP Mazembe through to group phase 16 Apr | Football

Football notifications, social media and more

Get latest scores and headlines sent straight to your phone, sign-up to our newsletter and learn where to find us online.

Text mining and Python

- Python is a great language for dealing with text
- Built-in functions for the *str* type:
 - `.lower` “Hello Students”.`lower()` => “hello students”
 - `.split` “this is a test of split”.`split()` => [“this”, “is”, “a”, “test”, “of”, “split”]
 - `.join` “-”.`join([“a”, “b”, “c”])` => a-b-c
 - `.strip` “end of line \n”.`strip()` => “end of line”
 - `.replace` “this the end”.`replace(“end”, “start”)` => “this is the start”
- Powerful regex built-in module

Python Tools for Text Mining / NLP

- Natural Language ToolKit ([NLTK](#))
 - Resources and tools for Natural Language processing in multiple languages.
 - Tools include: tokenization, stemming, tagging, parsing, among other things.
- [Spacy](#):
 - Object-oriented approach for NLP tasks
- BeautifulSoup ([BS4](#))
 - Parser for html and XML document
- [Gensim](#)
 - Includes an interesting set of topic modeling algorithms including: LDA, LSI, TFIDF, and efficient word2vec implementation.
- Python-[boilerpipe](#)
 - A wrapper for (java) boilerpipe, a tool to extract relevant content from a webpage.
- [Torchtext](#)
 - Pytorch library for text processing
- [Huggingface](#)
 - Set of tools and pretrained ML models, including several NLP related tools
 - Provides [Spark integration](#).
- [PySpark: Spark-NLP](#)

Spark MLLIB

[MLlib](#) is Spark's machine learning library with some basic (and some more sophisticated) algorithms and tools for feature engineering.

Interesting ML-LIB text mining related tools for pyspark:

- Feature extractors:
 - TF-IDF
 - word2Vec
- Feature transformers
 - Tokenizer
 - StopWordsRemover

Spark ML-LIB / TF-IDF example in pyspark

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer

sentenceData = spark.createDataFrame([
    ("doc1", "This is a Data Science Course"),
    ("doc2", "Data Science is cool, but bit complex"),
    ("doc3", "I love complexity, but not in 0()")
], ["label", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)

hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
featurizedData = hashingTF.transform(wordsData)

idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
results = idfModel.transform(featurizedData)
results.select("label", "features").collect()
```

Real World Problems

Example: News aggregation

Goal: Collect all the articles (news) coming from a list 100 world-wide news providers, and produce a daily report of the main “stories” of that day, with their respective articles, in human-friendly (journalist-friendly) format.

Data Crawling: How to collect the data?

Data Cleaning: How to get the title and content of the news, removing noise?

ML Task 1: Unsupervised cluster with unknown number of clusters

ML Task 2: How to represent the clusters in human-readable format?

Example: News aggregation (I)

Goal: Collect all the articles (news) coming from a list 100 world-wide news providers, and produce a daily report of the main “stories” of that day, with their respective articles, in human-friendly (journalist-friendly) format.

Data Crawling: Twitter API + RSS + urllib

Data Cleaning: Number of Words + Link Density (Boilerpipe)

Example: News aggregation (II)

ML Task 1: Unsupervised cluster with unknown number of clusters

- How many articles per day? ~3K
- Represent each article as TF-IDF vector
- Compute cosine similarity between **all pairs**
- Set a minimum threshold similarity S , and create a graph with all edges with weight $> S$
- Compute k-core ($k=2$)
 - Each connected component is a story!
 - K-core also helps to remove drifts!

Pros: High precision and recall

Cons: ?

ML Task 2: Take the highest value of the multiplication of TF-IDF values in each cluster, to retrieve the keywords

Challenge I

- Take the airbnb dataset and from the column “name” (description) do the following:
 - Tokenize (remove punctuation and split by word), you can do it in pure python or using ml-lib tokenizer
 - Remove stopwords using ML-LIB [stopwordsremover](#), and store in a new column called “CleanTokens”
 - But we don’t have a stopwordsremover for all language and contexts.
 - Create your own list of stopwords from this text (think: what is a stopwords?)
 - Remove stopwords again, and store in column “MyCleanTokens”
 - Perform TFIDF in a new column called “VectorSpace”

You have 25 minutes



Homework (Optional)

- In a new column('word2vec'), repeat the procedure using word2vec instead of TF-IDF.
 - <https://spark.apache.org/docs/2.2.0/mllib-feature-extraction.html#word2vec>

Example II: Recommender System (Cold start)

Goal: Consider a big online-retailer company, realising up to 100K products a day (in batches). When a user X clicks on a (new) product, recommend similar items based on that product. For each product you have just name and few lines of descriptions. Update the system every 12 hours

Constraints:

- You don't have information about the user
- Products are new (no history information)
- You have time constraints

Locality-sensitive hashing (LSH)

“One general approach to LSH is to “hash” items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are. We then consider any pair that hashed to the same bucket for any of the hashings to be a candidate pair”.

Source: Mining of Massive Datasets

Jure Leskovec, Anand Rajaraman, Jeff Ullman

- Generate random hyperplanes: h_1, h_2, h_3
- Hashcode each data point point in those hyperplanes
- Compare points with the same hashcode
- Repeat with different hyperplanes

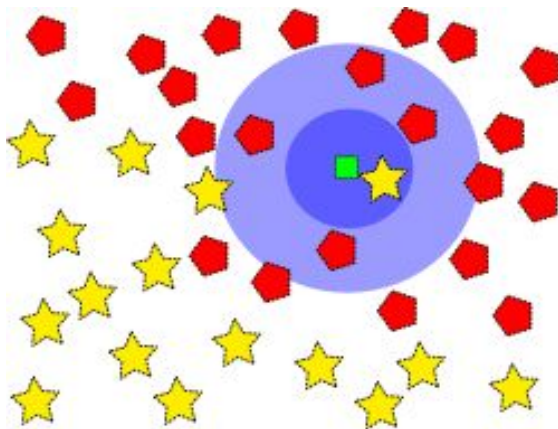
Reading: [Chapter 3, Mining of Massive Datasets](#)

Video: [V. Lavrenko University of Edinburgh](#),

Example II: Recommender System (Cold start) (II)

ML TASK:

- Nearest neighbor search [NNS] (?)
 - Approximate Nearest Neighbor (ANN)
 - **Locality-sensitive hashing (LSH)**



Locality-sensitive hashing (LSH) in Python

- [ANNOY \(Spotify\)](#)
- [Sklearn.neighbors.LSHForest](#)
- [BucketedRandomProjectionLSH](#) (experimental)

Sklearn.neighbors.LSHForest

```
from sklearn.neighbors import LSHForest

#Potential neighbors
candidates = [[0.1, 0.1, 0], [1, 1, 1], [0, 1, 0], [0.8, 0.8, .8], [0, 0,1]]
#Points which we want to find neighbors
queryPoints = [[1, 1, 0], [0.5, .5, .5] ]

#Fix random state for compare results
lshf = LSHForest(random_state=4)

#Train the model
lshf.fit(candidates)

#Search for neighbors
distances, indices = lshf.kneighbors(queryPoints, n_neighbors=2)
print distances
print indices
```

```
[[ 2.22044605e-16  1.83503419e-01]
 [ 0.00000000e+00  0.00000000e+00]]
[[0 1]
 [1 3]]
```

Results

- Are your results correct?
- How to know?
 - You need a **ground truth**: given that we have an small subset we can get exact solution
 - Use the [NearestNeighbors module](#) to get exact solution
 - Do you remember the second slide?!
 - We need a performance measure, for example:
 - precision: how many of top-5 elements in lsh results are in the ground truth
 - Ranking: Compute the kendall-tau comparison between lsh and ground truth ranking

Challenge II

- Take the first 500 flats in the list
 - `Mysample = df.limit(500)`
- Find the 3 nearest neighbors for each element in that subset (candidates and query points are within the sample of 500)

You have 6 minutes



Hyperparameter tuning strategies

- The results of any ML algorithm are closely related with the parameters selected.
- Some possible strategies:
 - **Grid search:** At certain granularity, try all the combinations of possible of parameters.
 - **Random Search:** Randomized search over parameters, where each setting is sampled from a distribution over possible parameter values.
- In both cases different parameters setups are tested using cross-fold validation.

Hyperparameter tuning in scikit-learn

- Scikit learn offers both grid and randomize search functionalities parameter tuning.
 - [GridsearchCV](#)
 - [RandomizedSearchCV](#)
- Both are optimized for parallelizing their job, trying to be as faster as possible

GridsearchCV Example

```
#import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

#create some dummie data
X, y = make_classification(n_samples=300, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)

#init your model
clf = RandomForestClassifier(random_state=0)

#define your grid of parameters
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4],
    'min_samples_split': [8, 10],
    'n_estimators': [100, 200]
}

#fit de model
grid_search = GridSearchCV(estimator = clf, param_grid = param_grid,
                            scoring='accuracy')

parameters_best = grid_search.fit(X,y)
```

```
print(parameters_best.best_params_)
```

```
{'bootstrap': True, 'max_depth': 80, 'max_features': 2, 'min_samples_leaf': 3, 'min_
samples_split': 8, 'n_estimators': 100}
```

GridsearchCV Spark Boosted

The [joblib-spark](#) module allows to use distributed systems to boost our grid search, just by adding few lines of code

```
#import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
#create some dummie data
X, y = make_classification(n_samples=300, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)

#init your model
clf = RandomForestClassifier(random_state=0)

#define your grid of parameters
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4],
    'min_samples_split': [8, 10],
    'n_estimators': [100, 200]
}

#fit de model
grid_search = GridSearchCV(estimator = clf, param_grid = param_grid)

#lets add some spark
import pyspark
spark = pyspark.SparkContext('local[*]')
from sklearn.utils import parallel_backend
from joblibspark import register_spark

register_spark() # register spark backend

with parallel_backend('spark',n_jobs=3):
    grid_search = GridSearchCV(estimator = clf, param_grid = param_grid,
                               scoring='accuracy')

parameters_best = grid_search.fit(X,y)
print(parameters_best.best_params_)
```

```
{'bootstrap': True, 'max_depth': 80, 'max_features': 2, 'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 100}
```

Challenge III: Homework

- **Homework:**

- Repeat the LSH experiment for the full **Barcelona** dataset, and develop and **spark** boosted methodology to have an efficient parameter tuning process.
- Do the same with the field “Title” with the September-2024 top-1000 Articles in Ukrainian Wikipedia. You can use the [Wikimedia Pageview Tool](#) or the [Pageviews API](#)
- Your report should include:
 - The accuracy of your model with at least 4 combinations of parameters
 - The computation time spent in the parameter tuning procedure (specify also the characteristics of the machine(s) that you have used)
 - What happen if we tune the parameters using Airbnb and the run with those parameters in The Wikipedia Dataset? What is the difference in accuracy with the parameters tuned directly in Wikipedia data?

Bibliography:

- Machine Learning
 - Chapter 5 of <http://www.deeplearningbook.org/>
 - Chapter 3 of <http://www.mmds.org/>
- Comparison between exact and approximate kNN in scikit-learn:
 - [A DEMONSTRATION OF THE USAGE OF LOCALITY SENSITIVE HASHING FOREST IN APPROXIMATE NEAREST NEIGHBOR SEARCH](#)
- [Word2Vec paper](#)
- [Fasttext paper](#)