@AndrzejWasowski
Andrzej Wąsowski

# Advanced Programming

## Case Study: Sentiment Analysis (and not so Big Data)

IT UNIVERSITY OF COPENHAGEN

SOFTWARE QUALITY RESEARCH

© Andrzej Wąsowski, IT University of Copenhagen 

**Credits**

Lecture and slides and the assignment inspired by a presentation of Natalie Schluter, ITU
Some slide design by Marek Rei, University of Cambridge
All errors and incompetent claims by Andrzej Wąsowski, Software Quality Research (SQAURE), ITU

- Sentiment Analysis as a classification problem
- Supervised Learning
- Classifying with a perceptron
- Word embeddings: extracting features from text
- Cross-validation
- All with Scala and Spark examples

# AGENDA

# Sentiment Analysis

✔ *nice and compact to carry!*

✔ *since the camera is small and light, I won't need to carry around those heavy, bulky professional cameras either!*

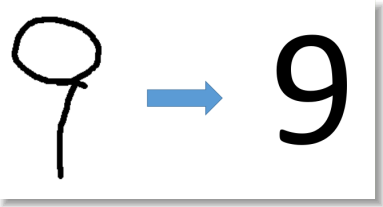
✗ *the camera feels flimsy, is plastic and very light in weight you have to be very delicate in the handling of this camera*

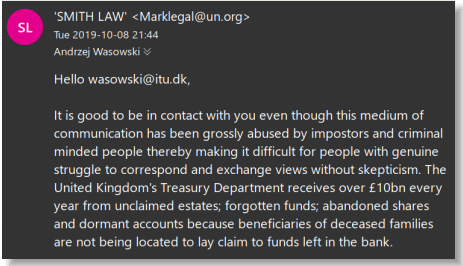
**Widely used** in the industry!
**Not a subject** in this course!

- **Sentiment Analysis:**
  - a technique developed within natural language processing, text analysis, computational linguistics
  - to identify, extract, quantify, and study affective states and subjective information
- AKA **opinion mining** or **emotion AI**, a narrow form of **automatic text summarization**

# Classification Problems

- A **classifier** is a function that maps input data domain ($X$) to one of finitely many categories ($Y$)
- **Examples:** hand writing recognition, spam filtering, traffic sign detection (autonomous driving), weather forecast presentation
- **Question:** What are the input domains and classes/categories below?




# Learning Classifiers

- The **learning problem**: find a function representing the classifier mapping for an input domain $X$ and the class co-domain $Y$
- **Supervised learning**:
  - Given a training set $L = \{(x_i, y_i) \in X \times Y \mid i = 1 \dots N\}$
  - Find an **approximation** of $f : X \mapsto Y$
  - That **agrees with the training set** possibly well, so $f(x_i) = y_i$
  - **Generalizes** to other $x \in X$ possibly well
- The set $L$ is often called a **labeling**, and the $y_i$ values are **labels**
- Labels are often **manually** obtained
- AI is run by people (recall the captcha example from previous slide)
- The set $L$ is often called the **training set**

# Supervised Learning

## Consider the following example

| # | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $y$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 0 | 1 | 1 |

- Our goal is to find a function $f : X \mapsto Y$
- Where $X = \{0, 1\}^4$ and $Y = \{0, 1\}$
- **Question.** What is your guess?
- Let the correct output for $(0, 0, 0, 0)$ be $1$. Does your proposal **generalize**?
- **Key** to **evaluate** any trained classifier on different set than the **training set**
- To learn a classifier automatically we need to **limit the space of possible candidate functions** to a reasonable class

# Amazon labeled data for sentiment analysis

```
{
  "reviewerID": "A2IBPI20UZIR0U",
  "asin": "1384719342",
  "reviewerName": "cassandra tu \"Yeah, well, that's just like, u...",
  "helpful": [0, 0],
  "reviewText": "Not much to write about here, but it does exactly what it's supposed to.
                 filters out the pop sounds. now my recordings are much more crisp. it is
                 one of the lowest prices pop filters on amazon so might as well buy it,
                 they honestly work the same despite their pricing,",
  "overall": 5.0,
  "summary": "good",
  "unixReviewTime": 1393545600,
  "reviewTime": "02 28, 2014"
}
```

- Extract **text data**: summary + reviewText
- Extract "overall" — this is our **labeling for supervised learning**!
- Map $1.0, 2.0 \mapsto 0.0$ (negative), $3.0 \mapsto 1.0$ (neutral), map $4.0, 5.0 \mapsto 2.0$ (positive)
- For practical reasons we add an identifier column

source http://jmcauley.ucsd.edu/data/amazon/links.html

# Loading our labeled data in Scala/Spark

```
1  type ParsedReview    = (Integer, String, Double)
2  def loadReviews (path: String): Dataset[ParsedReview] = spark
3      .read
4      .schema (reviewSchema)
5      .json (path)
6      .rdd
7      .zipWithUniqueId
8      .map[(Integer,String,Double)] { case (row,id) =>
9          (id.toInt, s"${row getString 2} ${row getString 0}", row getDouble 1) }
10     .toDS
11     .withColumnRenamed ("_1", "id" )
12     .withColumnRenamed ("_2", "text")
13     .withColumnRenamed ("_3", "overall")
14     .as[ParsedReview]
```

**Note 1**: Line 6 drops to `rdd` for a missing method (untyped)

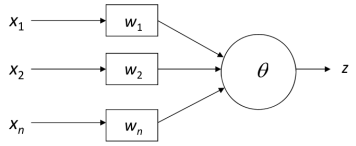
**Note 2**: Line 10 returns to data set (of rows, still weakly typed)

**Note 3**: Line 11–13 reintroduces column names that are always lost by map

**Note 4**: Line 14 reinotroduces a static type. Do this at the boundary of a logical fragment

# Perceptron Node

## Threshold Logic Unit




with the **activation function** $z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$

- **First neural network** learning model in the 1960's
- Rather **loosely inspired** by our neural system
- Simple and limited (single layer models)
- We can **compose** into **multi-layer models** (all the way to Deep learning!)
- In perceptron learning **fix the number and size of the layers** (aka architecture)
- **Train** the **weights** $w_i$ and the **activation threshold** $\theta$ for all neurons in the network
- Input layer: exactly the **size of our input**
- Output layer: 1-hot encoding of the class (so the **number of classes**)

# Multi-Layer Perceptron Classifier in Spark

```
1  import org.apache.spark.ml.classification.MultilayerPerceptronClassifier
2
3  val train  = ... // my training set
4  val trainer = new MultilayerPerceptronClassifier()
5    .setMaxIter(50)
6    .setLayers(Array[Int](50, 5, 4, 3))
7    .setBlockSize(128)
8    .setSeed (1234L)
9
10 val model = trainer fit train
```

**Note 1**: Line 3, this is your data set labeled, ready for training; It should have three columns with titles: id, features, labels

**Note 2**: Line 6, the first layer has to agree with the number of features you have

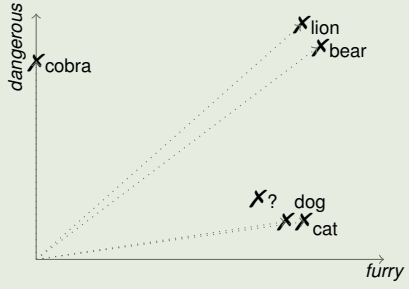
# Word Embeddings with Distributed Vectors

**Goal:** Represent a review text as a numeric vector, so that we can feed it into a perceptron.

**Idea:** Use “features” of concepts to represent meaning

| | *furry* | *dangerous* |
|---|---|---|
| **bear** | 0.90 | 0.85 |
| **cat** | 0.85 | 0.15 |
| **cobra** | 0.00 | 0.95 |
| **lion** | 0.85 | 0.90 |
| **dog** | 0.80 | 0.15 |

```
val bear =Vector (0.90, 0.85)
```




**Key:** If we have a new vector we can infer some information without knowing the word

**Problem 1:** How do we label the entire dictionary?

**Problem 2:** Different features for each concept, or an enormous set of features if unified

Material inspired by presentations of Marek Rei, University of Cambridge

# Distributional Hypothesis

*Words which are similar in meaning occur in similar contexts* (Harris, 1954)

*You shall know a word by the company it keeps* (Firth, 1957)

| | |
|---|---|
| He is reading a **magazine** | I was reading a **newspaper** |
| This **magazine** published my story | The **newspaper** published an article |
| She buys a **magazine** every month | He buys this **newspaper** every day |

- Learning vector representations so that contextually close words are represented by close vectors is a research field in itself (another course ...)
- Fortunately, the GLoVe project at Stanford published a precomputed set of vectors for 400K English words, `https://nlp.stanford.edu/projects/glove/`
- Longer the vector more precise the model (50 . . . 300)
- How to represent a text by a vector (not just a word)?
  One simple way: compute an average vector for words in the GLoVe dictionary, ignore others

# An Example Embedding from GLoVe

```
corythosaurus -0.042672 -0.088106  -0.31724 -0.25209 -0.26851 -0.06615 0.90325  -0.13818 0.3186
0.30621  -0.020125  1.0509 0.40654  -0.10525 -0.12052 -0.54416  -0.03742 0.5367 0.87692 0.040133
-0.20563 -0.45572 0.39592 -0.0070575  0.49928 1.0726  -0.71301 0.50881  0.50365 -0.05629 -1.1315
-0.2297 0.4061  0.52096 -0.481  0.0092775 -0.096609  -0.90419 -0.58088 0.30755 -0.38187 -0.91153
0.11853 -0.1303 4 0.48774 -0.99745 -0.069557 0.24963 0.75791 0.50679
```

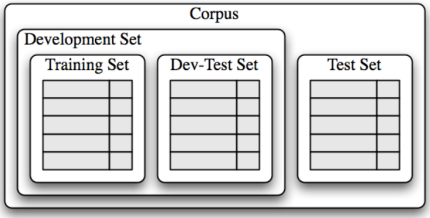
```
type Embedding = (String, List[Double])
def loadGlove (path: String): Dataset[Embedding] = spark
      .read
      .text (path)
      .map  { _ getString 0 split " " }
      .map  (r => (r.head, r.tail.toList.map (_.toDouble)))
      .withColumnRenamed ("_1", "word" )
      .withColumnRenamed ("_2", "vec")
      .as[Embedding]
```

- Now two DataSets share the column ‘word’. Use `join` method to **combine** them
- Still need a single vector for review, not per word. Sum vectors and counters using **reduce**
- Use reduceByKey/groupBy to **sum for each review separately**, not for all together (useless)

# Travel in Time: ADPRO Lecture 1

# Hint! Hint! Hint! Hint!

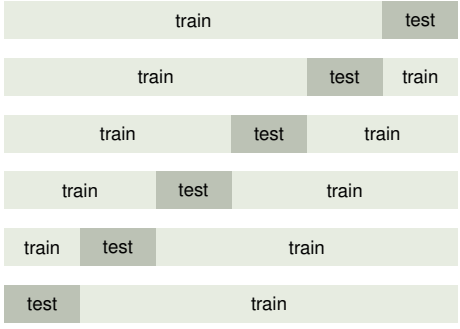
# Train. Test. Evaluate. Measure.




- Development sets:
  - **Train** on one set (“Training Set"),
  - **Test** on another set (“Dev-Test Set”), not seen by training
- Test set:
  - **Evaluate** on yet another set, never used in development;
  - AKA the **holdout** set
- **What to measure?** $\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$

# $k$-Fold Cross-Validation

- **Partition** data into $k$ folds randomly
- For each fold:
  - Choose the fold as a **temporary test** set
  - **Train** on the $k-1$ remaining folds
  - **Measure** accuracy on the test fold
- Report average **accuracy** and **variance/standard-deviation** for $k$ runs
- Figure to the right: **6-fold** validation
- **10-fold** validation is most **common**




```
1 val paramGrid = new ParamGridBuilder().build()
2 val evaluator = new MulticlassClassificationEvaluator().setMetricName("accuracy")
3 val cv = new CrossValidator().setEstimator(trainer) // our MultiLayerPerceptronClassifier
4   .setEvaluator(evaluator)
5   .setEstimatorParamMaps(paramGrid).setParallelism(4).setNumFolds(10);
6 val model = cv.fit(train);
7 val result = model.transform(test) // query the result objects for accuracy
```

- Sentiment Analysis as a classification problem
- Supervised Learning
- Classifying with a perceptron
- Word embeddings: extracting features from text
- Cross-validation
- All with Scala and Spark examples

# AGENDA