Week 3: Supervised learning

- ### Today's lecturer: Dennis Medved

Today's lectures

Overview of machine-learning algorithms. Some linear algebra. How to design a ML pipeline. Programming with MLlib.

Today's schedule

- Overview of machine learning and linear algebra. **<---**
- Exercise in linear algebra.
- Regression and machine learning (ML) pipeline.
- Exercise in linear regression.
- Classification and logistic regression.
- Exercise in logistic regression.

Overview of machine-learning

A definition:

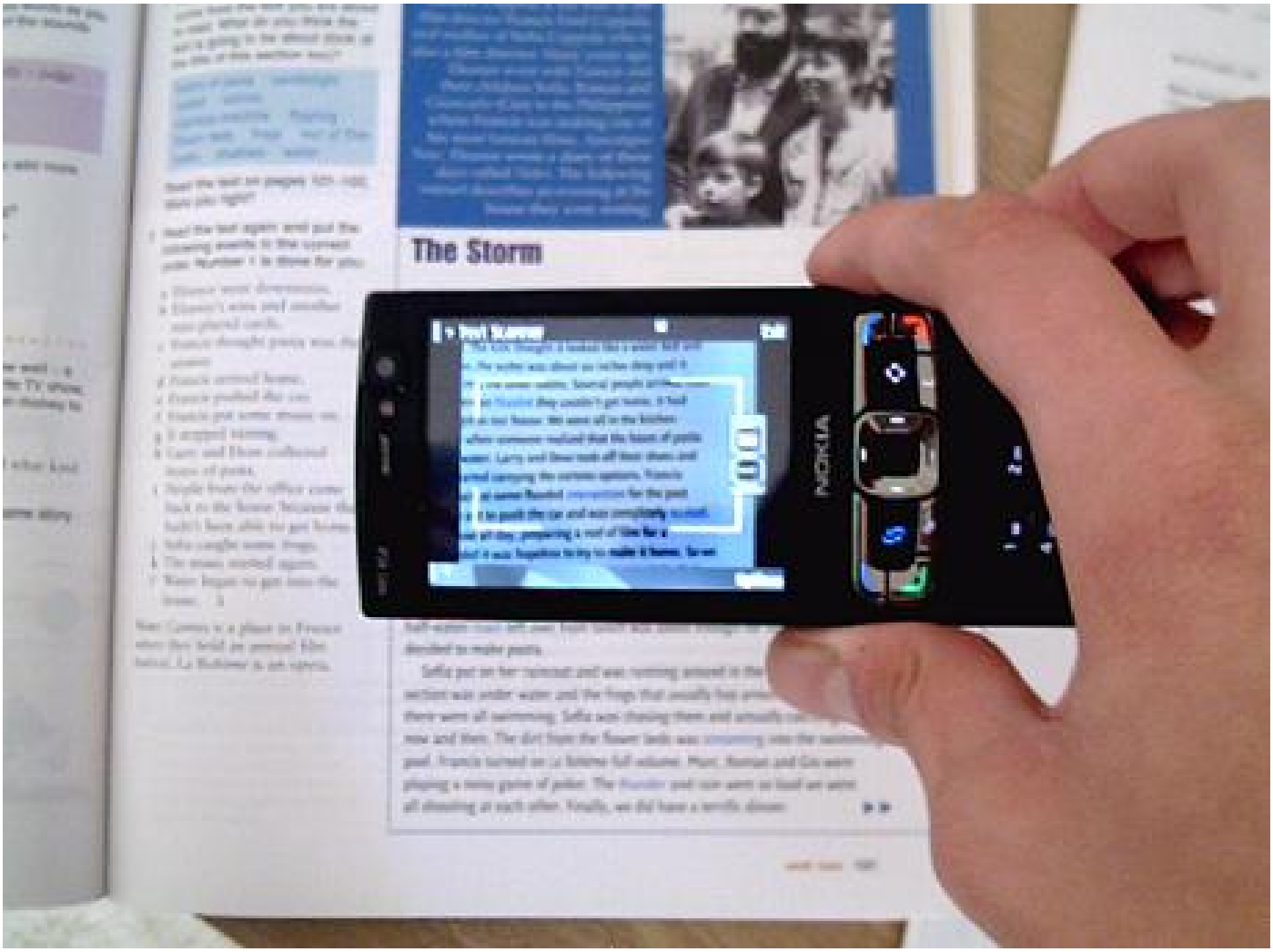
- ### Machine learning is related to statistics.
- ### Cultures and backgrounds are different.
- ### More specifically, explores the construction and study of algorithms that can:
  - ### Learn from data and
  - ### Make predictions on data.

Examples of machine-learning

- Spam filtering

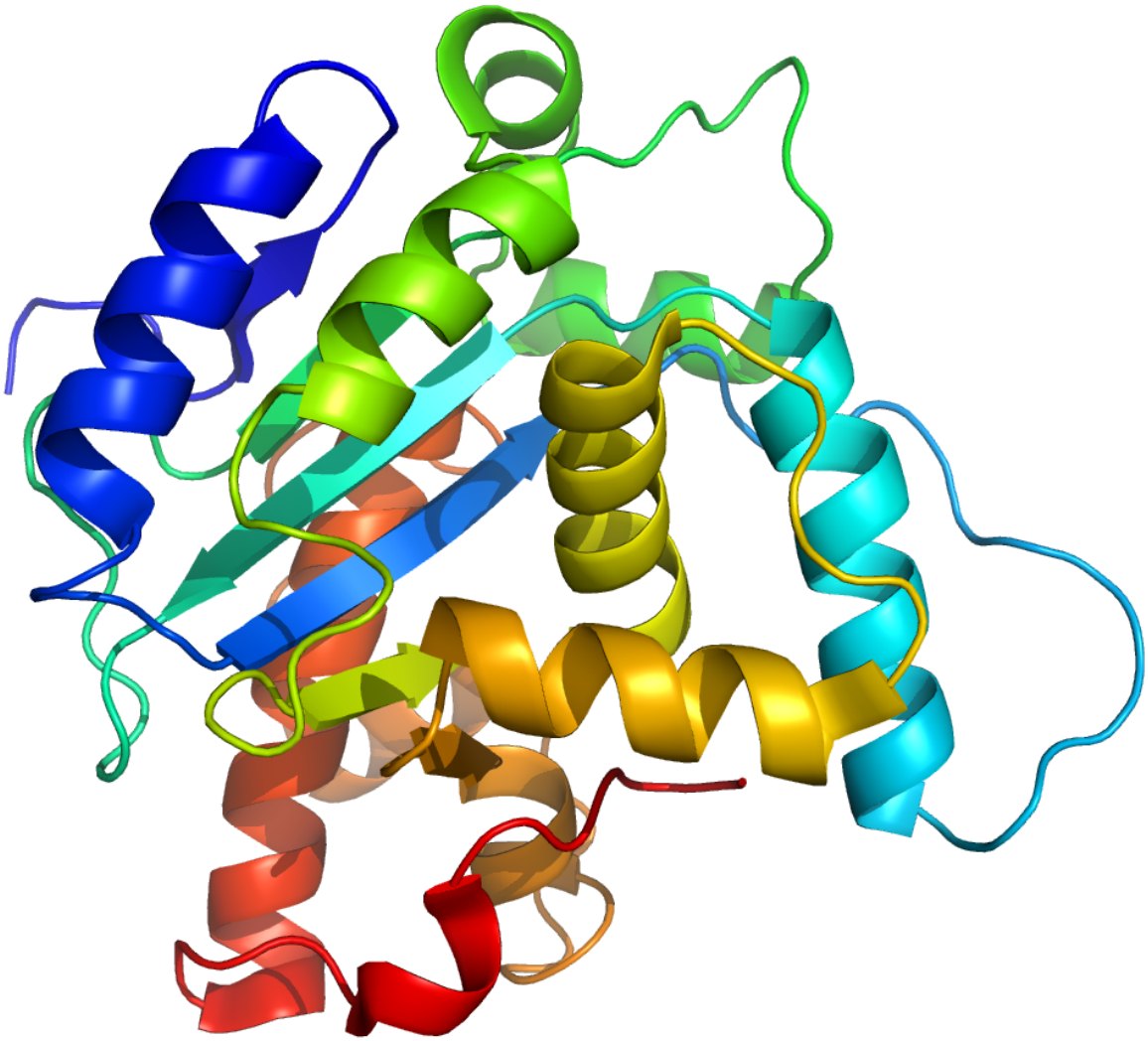
Examples of machine-learning (continued)

- Optical character recognition (OCR)



Examples of machine-learning (continued)

- Protein structure prediction



Examples of machine-learning (continued)

- Playing Jeopardy

Examples of machine-learning (continued)

- Machine vision

Vocabulary

Observations
Examples used for learning or evaluation, e.g. wikipedia articles, scanned characters, and films.

Feature
A feature is an individual measurable property of an observation, represented as a number or category, e.g. length, year, director.

Label
The category or value attached to an observation, e.g. cancer or not cancer, shoe size, score.

Data set example

- Data from Internet Movie DataBase (IMDB)

Title	year	length	director	score
The Shawshank Redemption	1994	142	Frank Darabont	**9.3**
The Godfather	1972	175	Francis Ford Coppola	**9.2**
Saving Christmas	2014	80	Darren Doane	**1.6**
Disaster Movie	2008	87	Jason Friedberg	**1.9**

Models and performance

Model
A function that takes the feature vector as input and produces an output, which predicts the category or a value, e.g. a model produced by logistic regression.

Title	year	length	director	score	
The Shawshank Redemption	1994	142	Frank Darabont	**9.3**	
The Godfather	1972	175	Francis Ford Coppola	**9.2**	
Saving Christmas	2014	80	Darren Doane	**1.6**	
Disaster Movie	2008	87	Jason Friedberg	**1.9**	
Pulp Fiction	1994	154	Quentin Tarantino	**?**	**<--model**

Performance Measure
A measure of of the quality of the predicted label compared to the real label of the observations, e.g. mean absolute value or accuracy. In our case the score (label) is 8.9, the closer, the better.

Performance evaluation framework

Training, validation, and test data
The **data set** consists of observations (together with their labels) used in training and evaluating the model, e.g. a set of potential cancer patients.

The data set is usually split up in the three following subsets:

- **Training data** is used to train the model.
- **Validation data** is used to improve the performance of the model, by evaluating the result and changing the model accordingly.
- **Test data** is used to evaluate the end result, do not use this for model selection.

The distribution is for example: 70% / 15% / 15%.

Cross validation

The whole data set is partitioned into k equal sized subsamples.

Of the k subsamples, a single subsample (in red) is retained as the validation data for testing the model, and the remaining k − 1 samples are used as training data.

The cross-validation process is then repeated k times (the folds). A normal value for k is 5.

Broad learning settings

Supervised learning

Learning a model from observations that are labeled.

- Inferring a function from the labeled training data to the desired output.
- This should be able to generalize to unseen observations.

Title	**year**	**length**	**director**	**labels**	
The Shawshank Redemption	1994	142	Frank Darabont	**9.3**	
The Godfather	1972	175	Francis Ford Coppola	**9.2**	
ABC Africa	2001	83	Abbas Kiarostami	**7.0**	
Saving Christmas	2014	80	Darren Doane	**1.6**	
Pulp Fiction	1994	154	Quentin Tarantino	**?**	**<--model**

Broad learning settings (continued)

Unsupervised learning

Learning a model from observations that are unlabeled.

- Trying to find hidden structures in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution.
- Can be an goal in itself or used as a preprocessing step for a supervised algorithm.

Supervised learning

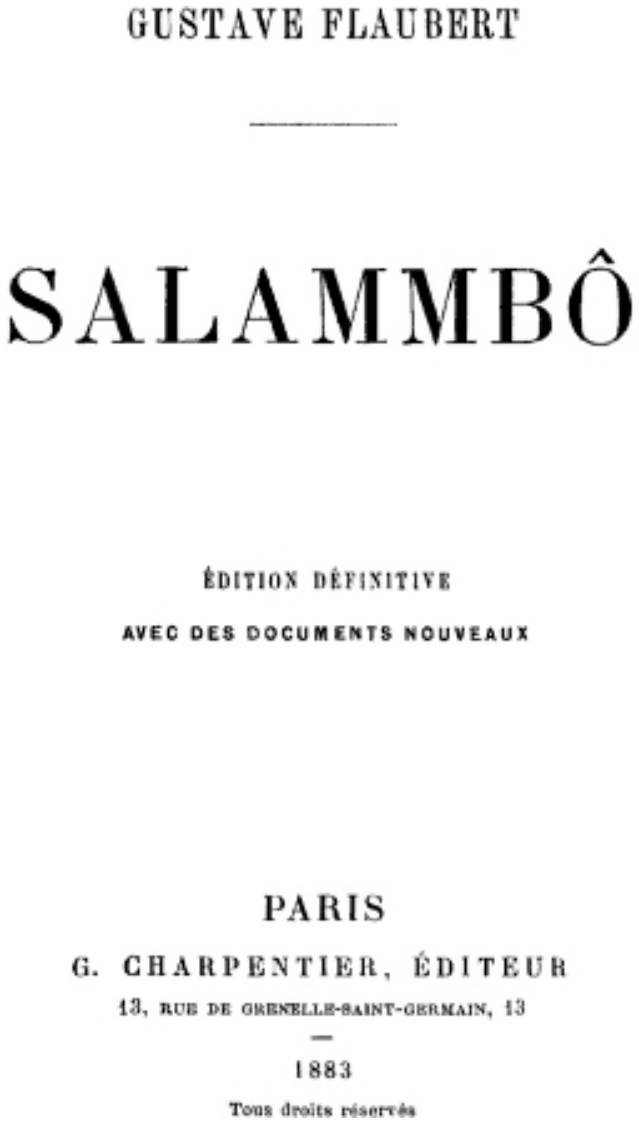
Regression

Outputs a real value from an observation, e.g. the IMDB score.

- The output is continous (the score, a real number).
- Evaluation defined by the closeness on the real values.

Salammbô

- Salammbô is a historical novel by Gustave Flaubert.


GUSTAVE FLAUBERT

SALAMMBÔ

ÉDITION DÉFINITIVE

AVEC DES DOCUMENTS NOUVEAUX

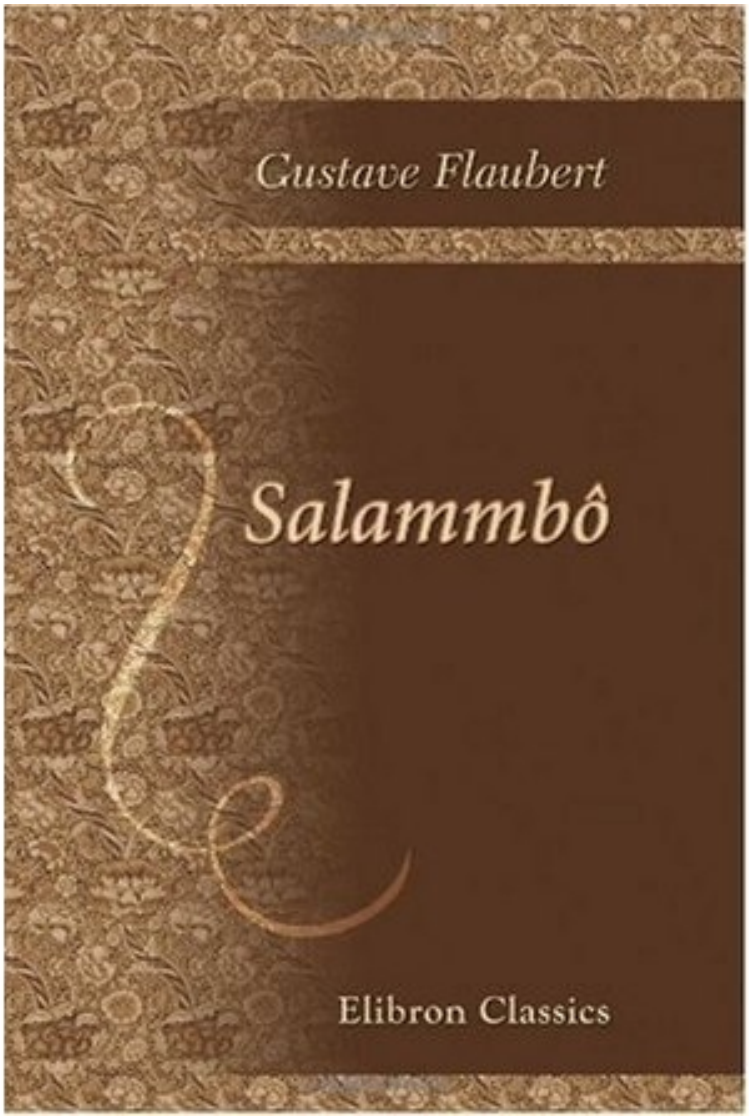
PARIS

G. CHARPENTIER, ÉDITEUR

13, RUE DE GRENELLE-SAINT-GERMAIN, 13

1883

Tous droits réservés



Salammbô (continued)

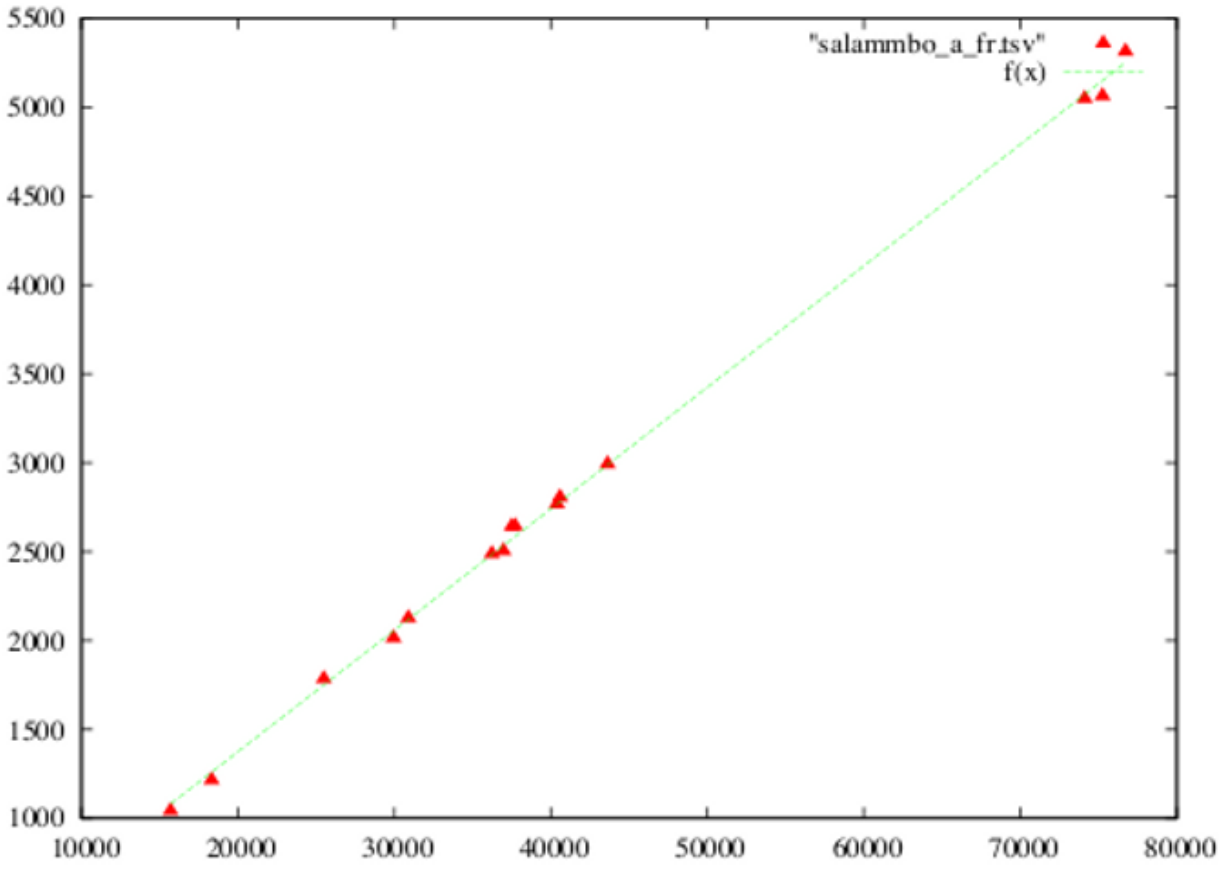
- Original in French, but there exists an English translation.

Chapter	French # chars	# A	English # chars	# A
1	36,961	2,503	35,680	2,217
2	43,621	2,992	42,514	2,761
3	15,694	1,042	15,162	990
4	36,231	2,487	35,298	2,274
5	29,945	2,014	29,800	1,865
6	40,588	2,805	40,255	2,606
7	75,255	5,062	74,532	4,805
8	37,709	2,643	37,464	2,396
9	30,899	2,126	31,030	1,993
10	25,486	1,784	24,843	1,627
11	37,497	2,641	36,172	2,375
12	40,398	2,766	39,552	2,560
13	74,105	5,047	72,545	4,597
14	76,725	5,312	75,352	4,871
15	18,317	1,215	18,031	1,119

Example of regression

Regression on French version: given the number of characters, predict the number of A:s

- **# chars** on x-axis
- **# A** on Y-axis:



Supervised learning (continued)

Classification

Outputs a class or category for each observation, e.g:

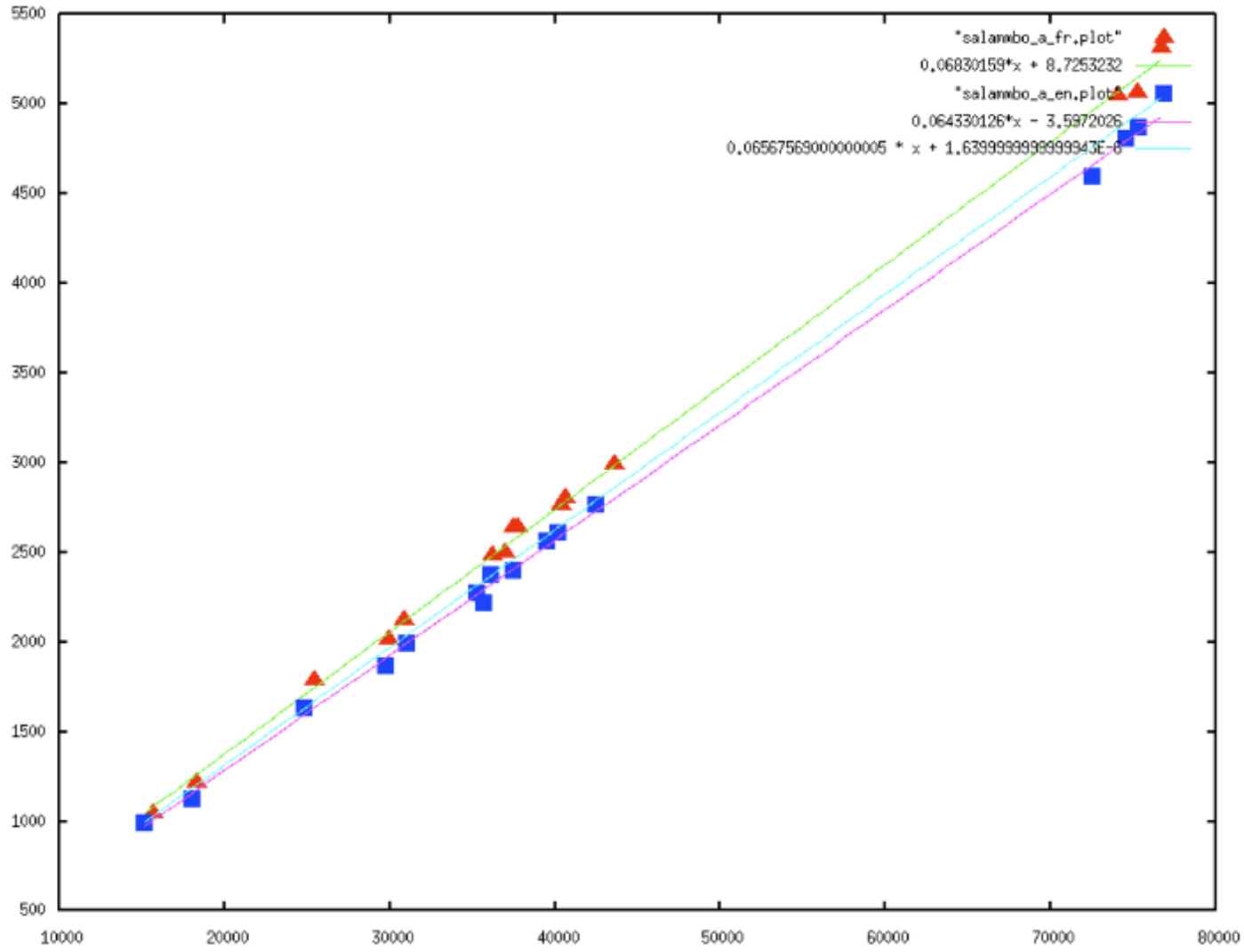
- **French** or
- **English**

depending on the number of characters and A:s.

- The output is discrete.
- Closeness can not usually be defined on categories.

Example of classification

Classifying the language in Salammbô:
French is the green line, English the purple:



Here the classifier is a straight line.

Example of classification (continued)

Binary classification, class: 1 = **French**, 0 = **English**:

Chapter	French # chars	# A	Class	English # chars	# A	Class
1	36,961	2,503	1	35,680	2,217	0
2	43,621	2,992	1	42,514	2,761	0
3	15,694	1,042	1	15,162	990	0
4	36,231	2,487	1	35,298	2,274	0
5	29,945	2,014	1	29,800	1,865	0
6	40,588	2,805	1	40,255	2,606	0
7	75,255	5,062	1	74,532	4,805	0
8	37,709	2,643	1	37,464	2,396	0
9	30,899	2,126	1	31,030	1,993	0
10	25,486	1,784	1	24,843	1,627	0
11	37,497	2,641	1	36,172	2,375	0
12	40,398	2,766	1	39,552	2,560	0
13	74,105	5,047	1	72,545	4,597	0
14	76,725	5,312	1	75,352	4,871	0
15	18,317	1,215	1	18,031	1,119	0

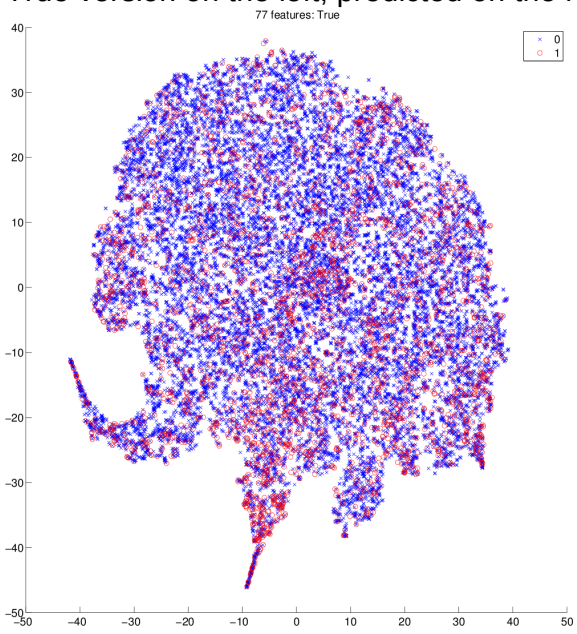
Unsupervised learning

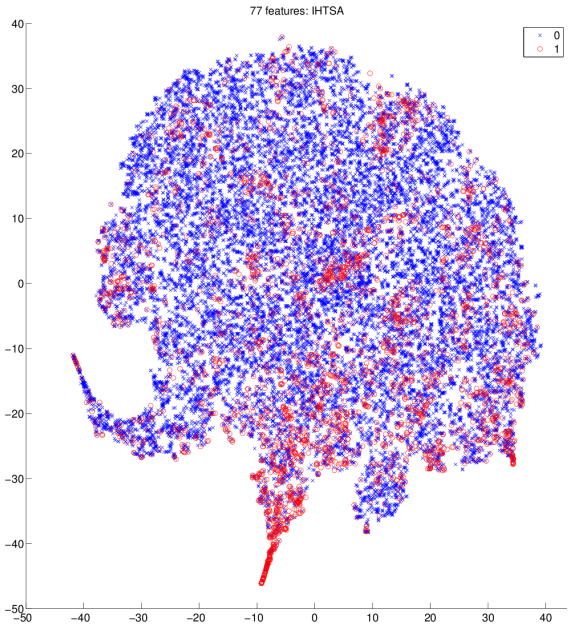
Dimensionality reduction

- The task to transforms the data in a higher dimensional space to a space of fewer dimensions
- Enables us to visualize the data in 2D.

Example of dimensionality reduction

- 77 dimensions reduced to 2
- Blue points are patients that are alive, red are dead
- True version on the left, predicted on the right:





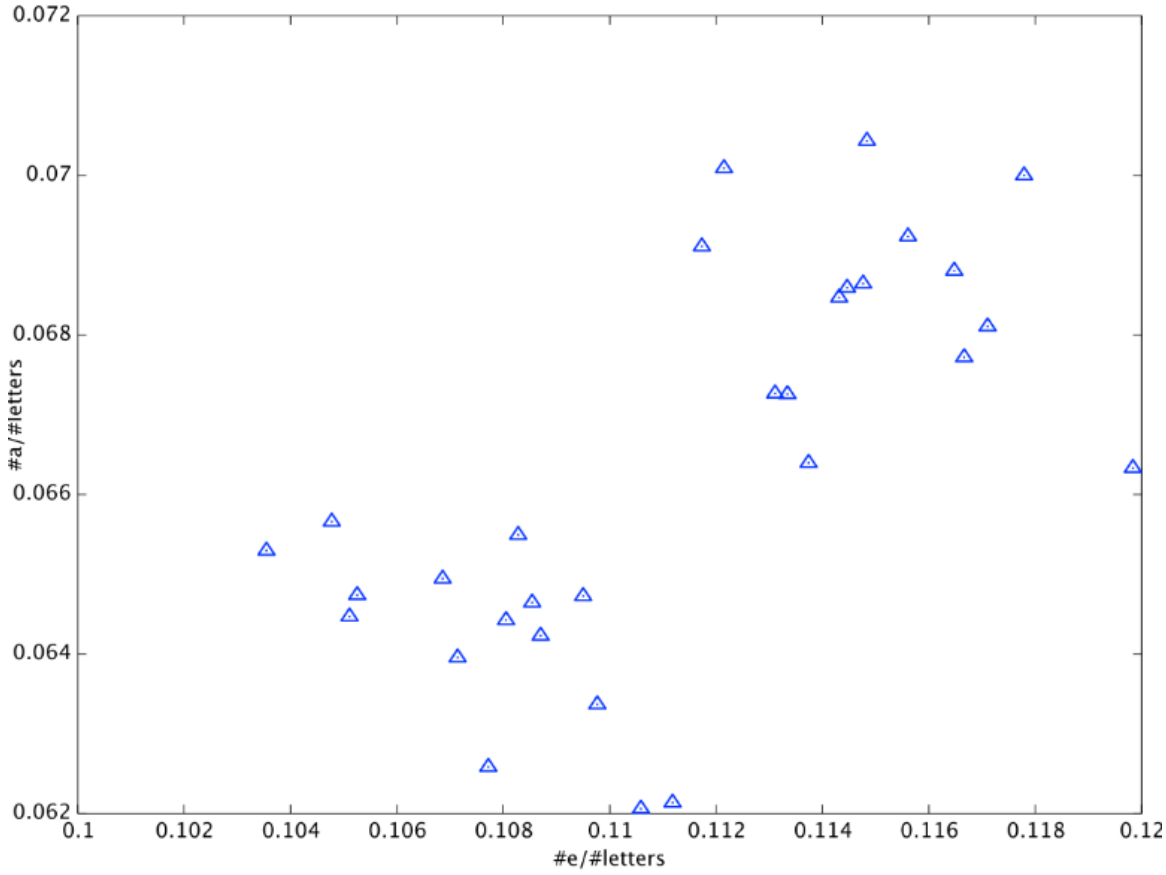
Unsupervised learning (continued)

Clustering

- The task to group observations in clusters
- The observations in a cluster are more "similar" to each other than to those in other clusters
- E.g. clustering different customer types together.

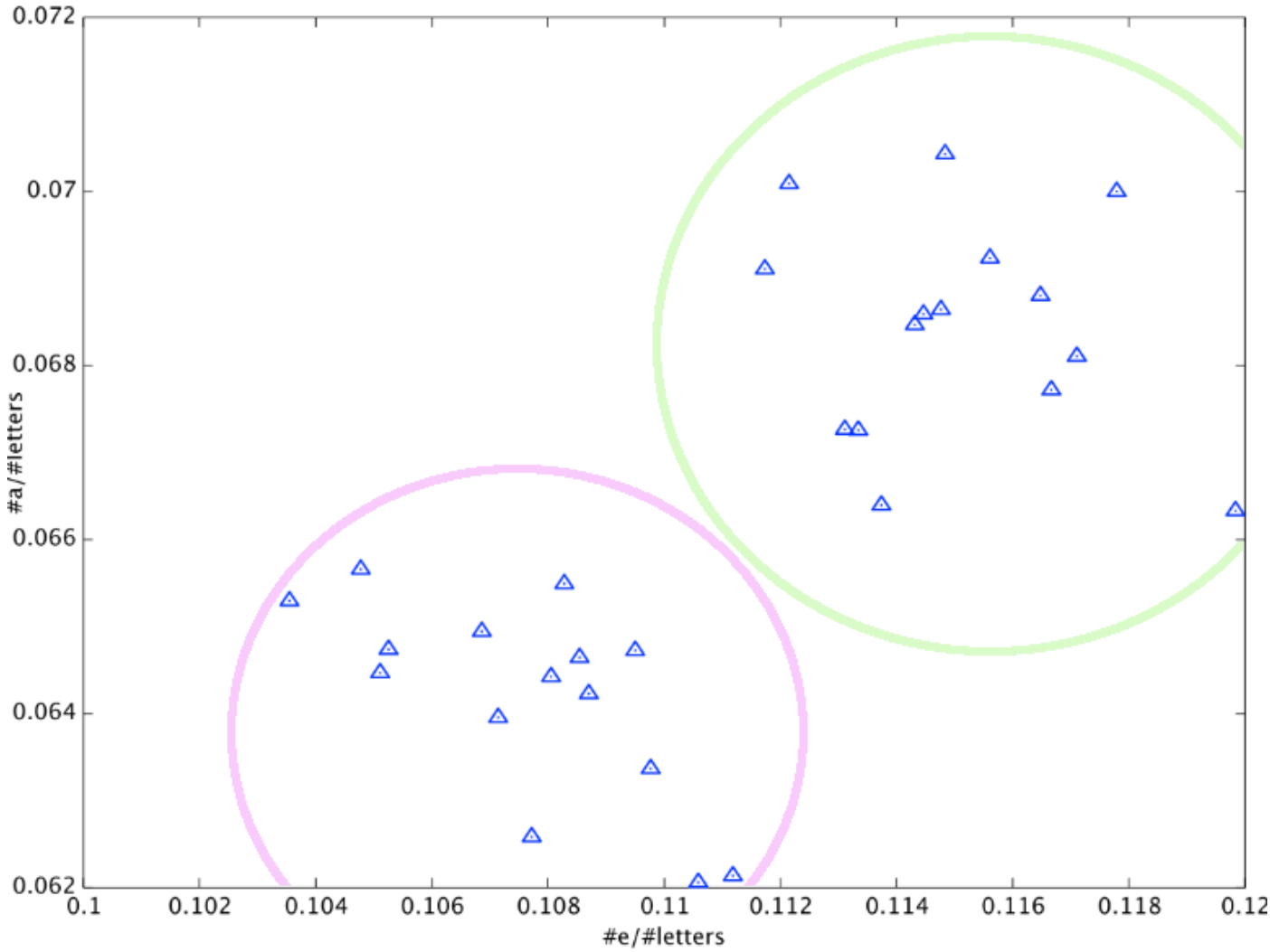
Example of clustering

- Clustering of Salammbô:
- Relative frequency:
  - E:s on X-axis
  - A:s on Y-axis



Example of clustering (continued)

French is the green circle, **English** the purple:



Following lectures

In the two following lectures, I will go through:

- The two different types of supervised learning.
- How to construct a machine-learning pipeline.
- How to use MLlib, the Spark machine learning library.

Linear algebra

Matrices

We represent data sets using matrices.

A matrix is a rectangular array of numbers, where in our case:

- The rows are the observations, in the example below the movies Shawshank Redemption and the Godfather.
- The columns are the features: year, length, number of Oscars, first weekend box office.

$$\mathbf{A} = \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix}$$

- Matrices are symbolized using bold upper-case letters, e.g. $\mathbf{A}$ in the example above.
- An $n \times m$ matrix has n rows and m columns, e.g. $\mathbf{A}$ is 2×5 matrix.

Matrices (continued)

$$\mathbf{A} = \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \end{bmatrix}$$

- Entries are denoted using lower-case letter with two subscript indices, $a_{i,j}$ corresponds to the entry at the i-th row and j-th column, e.g. $a_{2,4} = 302393$.

Vectors

A vector is a special case of a matrix with several rows and one column or one row and several columns. It is a 1 dimensional data structure (here representing class and scores).

$$\mathbf{w} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} 9.3 \\ 9.2 \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

- Vectors are denoted by bold lower-case letters, e.g. $\mathbf{v}$ in the example above.
- Entries are symbolized by lower-case letters with one subscript index. For instance, a_i corresponds to the entry at the i-th row.

NumPy

NumPy is the standard Python library to perform numerical computing. It adds:

- Multi-dimensional arrays and matrices, to store the data sets.
- A large library of high-level mathematical functions to operate on these arrays.
- It is used by MLlib.

In [14]:

```
# The convention is to import NumPy as the alias np
import numpy as np
```

Creating arrays with NumPy

- Numpy's array class is called **ndarray**; it is also known by the alias **array**.
- Ndarray is a multidimensional array of fixed-size that contains numerical elements of one type, e.g. floats or integers.
- Creating an array from a python list of numbers results into a one-dimensional array, which is, for our purposes, equivalent to a vector.
- You can also use the following class to represent matrices: **np.matrix()**.

Ndarray example

- To create a matrix object, either pass the constructor a two-dimensional ndarray, or a list of lists to the function, or a string e.g. '1 2; 3 4'.

$$\mathbf{A} = \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \mathbf{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Utilizing the **np.array()** to create the above matrix and vector:

In [15]:

```
A = np.array([[1994,142,7,7273274],
              [ 1972,175,3,302393 ]])
v = np.array([1,0])

print(str(A))
print(str(v))
```

```
[[   1994     142       7 7273274]
 [   1972     175       3  302393]]
[1 0]
```

Norm

- Norm defines the length or magnitude of a vector.
- In many cases we need to know the magnitude of the observations.
- In the IMDB data set, a succesful movie will have greater magnitude.
- P-norms are usually used.

$$\|\mathbf{v}\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p}$$

- The p-norm of a vector $\mathbf{v}$ is denoted by $\|\mathbf{v}\|_p$.

Euclidian norm

- The Euclidian norm is the special case of the p-norm when p = 2.
- Capture the intuitive notion of length of a vector.

$$\|\mathbf{v}\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p} \Rightarrow \|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x_1^2 + \cdots + x_n^2}$$

Norm example

$$\mathbf{A} = \begin{bmatrix} 1994 & 142 & 7 & 727327 \end{bmatrix} \quad \|\mathbf{v}\|_2 = \sqrt{1994^2 + 142^2 + 7^2 + 727327^2} \approx 7273274$$

NumPy has built in function to calculate the Euclidian norm of a vector, **np.linalg.norm(v)**:

In [16]:

```
v = A[0] #[1994,142,7,7273274]
norm = np.linalg.norm(v)
print(norm)
```

```
7273274.27472
```

Infinity norm

- The infinity norm is the special case of the p-norm when $p = \infty$.

$$\|\mathbf{v}\|_\infty = \max(|x_i|)$$

- Property of the norms p-norms: $\|\mathbf{v}\|_\infty \leq \|\mathbf{v}\|_2$

Normalizing

- In the IMDB data set, the box office feature outweighs the number of Oscars for example.

$$\mathbf{A} = \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix}$$

- The idea is to normalize the features so they become more homogeneous.
- This usually improves the prediction result and speeds up the training of the model.

Normalizing (continued)

- Normalizing a feature vector, i.e. making the feature vector have unit length, i.e. length = 1 in the norm:

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \qquad \mathbf{v}' = \frac{\mathbf{v}}{\|\mathbf{v}\|_p} \qquad \|\mathbf{v}'\| = 1$$

Normalizing

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \qquad \mathbf{v}' = \frac{\mathbf{v}}{\|\mathbf{v}\|_p} \qquad \|\mathbf{v}'\| = 1$$

In [17]:

```
print(v) # Shawshank Redemption
print(norm)
vNorm = v/norm
print(vNorm)
norm = np.linalg.norm(vNorm)
print(norm)
```

```
[   1994     142       7 7273274]
7273274.27472
[  2.74154380e-04   1.95235316e-05   9.62427613e-07   9.99999962e-01
]
1.0
```

Transpose

Tranpose is an operation on matrices that swaps the row for the columns.

$$\begin{bmatrix} 1994 & 1972 \\ 142 & 175 \\ 7 & 3 \\ 727327 & 302393 \end{bmatrix}^T \Rightarrow \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix}$$

- Tranpose is denoted by a superscript upper-case T, e.g. $\mathbf{A}^T$.
- Because the rows are swapped with the columns the following is true: $\mathbf{A}_{i,j} = A^T_{j,i}$

Transpose (continued)

$$\begin{bmatrix} 1 & 0 \end{bmatrix}^T \Rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Transpose is a self-inverse: $(\mathbf{A}^T)^T = \mathbf{A}$.
- If $\mathbf{A}$ is an $n \times m$ matrix then $\mathbf{A}^T$ is an $m \times n$ matrix.

Transpose example

$$\begin{bmatrix} 1994 & 1972 \\ 142 & 175 \\ 7 & 3 \\ 727327 & 302393 \end{bmatrix}^T \Rightarrow \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix}$$

To transpose a matrix you can use either **np.matrix.transpose()** or **.T** on the matrix object:

In [18]:

```
A = np.matrix([[1994,1972],[142,175],[7,3],[727327,302393]])
At = A.T #Identical to np.matrix.transpose(A)
print(A)
print(At)
```

```
[[  1994   1972]
 [   142    175]
 [     7      3]
 [727327 302393]]
[[  1994    142      7 727327]
 [  1972    175      3 302393]]
```

Scalar product

- Gives an idea of the similarity of the vectors.
- Is also known as dot product or inner product.
- The scalar product is symbolized by the operator $\cdot$, e.g. $\mathbf{v} \cdot \mathbf{u}$.
- Only defined on vectors, of the same length.
- The result is a scalar.
- The operations are done element-wise, e.g. $\mathbf{v} \cdot \mathbf{u} = k$ then $\sum v_i \times u_i = k$

Example scalar product

$$\begin{bmatrix} 1972 \\ 175 \\ 3 \\ 302393 \end{bmatrix} \cdot \begin{bmatrix} 2001 \\ 83 \\ 0 \\ 559 \end{bmatrix} = 1972 \times 2001 + 175 \times 83 + 3 \times 0 + 302393 \times 559 = 172998184$$

- You can use either:
- **np.dot()** or
- **np.array.dot()**
- The order of the arrays does not matter: np.dot(x, y), np.dot(y, x), x.dot(y), or y.dot(x).

In [19]:

```
u = np.array([1994,142,7,727327]) # Shawshank redemption
v = np.array([1972,175,3,302393]) # The Godfather
w = np.array([2001,83,0,559]) # ABC Africa
print(u.dot(v))
print(v.dot(w))
```

```
219942550550
172998184
```

Scalar product (continued)

- A geometric interpretation of dot product is the product between:
  - the length of the vectors (norm) and
  - the cosine of the angle between the vectors (θ).

$$\mathbf{v} \cdot \mathbf{u} = \| \mathbf{v} \| \| \mathbf{u} \| \cos\theta$$

Cosine similarity

- Cosine similarity is defined as the cosine between the angle of the vectors (θ).
- Useful as a similarity measure, e.g. in natural language processing to decide closeness of two sentences.

$$cosine\_similarity(u, v) = \cos\theta$$

Example cosine similarity

- If both vectors are normalized, i.e. they have length = 1, then the scalar product is equal to the cosine similarity.

$$\mathbf{v} \cdot \mathbf{u} = 1 \times 1 \times \cos\theta$$

In [20]:

```
from sklearn.metrics.pairwise import cosine_similarity
cosSim = cosine_similarity(v,w)
print(np.dot(v,w))
print(np.linalg.norm(v)*np.linalg.norm(w)*cosSim[0][0])
print("Similarity between The Godfather and ABC Africa:")
print(cosSim[0][0])
print("Similarity between The Godfather and Shawshank Redemption:")
print(cosine_similarity(u,v)[0][0])
```

```
172998184
172998184.0
Similarity between The Godfather and ABC Africa:
0.275137160447
Similarity between The Godfather and Shawshank Redemption:
0.99999278348
```

Example vector space model

- The vector space model is a representation of text, also called a bag of words.
- Starts from the collection of all the words.

The two sentences:

- Chrysler plans new investments in latin america.
- Chrysler plans major investments in mexico.

Are represented this way:

america	chrysler	in	investments	latin	major	mexico	new	plans
1	1	1	1	1	0	0	1	1
0	1	1	1	0	1	1	0	1

Example vector space model (continued)

Using the vector space model and cosine similarity we can compute the closeness of two sentences:

In [21]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfIdf = TfidfVectorizer(use_idf=False, norm=None).fit_transform(
    ["chrysler plans new investments in latin america",
     "chrysler plans major investments in mexico",])
print(tfIdf.A)
print(cosine_similarity(tfIdf[0],tfIdf[1])[0][0])
```

```
[[ 1.  1.  1.  1.  1.  0.  0.  1.  1.]
 [ 0.  1.  1.  1.  0.  1.  1.  0.  1.]]
0.617213399848
```

Scalar matrix multiplication

$$2 \times \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix}$$

$$11 \times \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}$$

- A scalar is a real number.
- The operation is done element-wise, e.g. $k \times \mathbf{A} = \mathbf{C}$ then $k \times a_{i,j} = kc_{i,j}$.

Scalar matrix multiplication

$$2 \times \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} 2 \times 1 & 2 \times 6 \\ 2 \times 4 & 2 \times 8 \end{bmatrix}$$

$$11 \times \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \times 2 \\ 11 \times 3 \\ 11 \times 5 \end{bmatrix}$$

- A scalar is a real number.
- The operation is done element-wise, e.g. $k \times \mathbf{A} = \mathbf{C}$ then $k \times a_{i,j} = kc_{i,j}$.

Scalar matrix multiplication

$$2 \times \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} 2 \times 1 & 2 \times 6 \\ 2 \times 4 & 2 \times 8 \end{bmatrix} = \begin{bmatrix} 2 & 12 \\ 8 & 16 \end{bmatrix}$$

$$11 \times \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \times 2 \\ 11 \times 3 \\ 11 \times 5 \end{bmatrix} = \begin{bmatrix} 22 \\ 33 \\ 55 \end{bmatrix}$$

- A scalar is a real number.
- The operation is done element-wise, e.g. $k \times \mathbf{A} = \mathbf{C}$ then $k \times a_{i,j} = kc_{i,j}$.

Scalar matrix multiplication example

$$2 \times \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix} \qquad 11 \times \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}$$

In [22]:

```
A = np.matrix([[1,6],[4,8]])
print(2*A)
B = np.matrix([2,3,5])
print(11*B)
```

```
[[ 2 12]
 [ 8 16]]
[[22 33 55]]
```

Matrix-vector multiplication

- Matrix-vector multiplication is used by linear regression
- The model is represented by a vector of weights
- The observations multiplied by the weights produce the score:

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 \\ -0.0001 \\ 0.1 \\ 0.0000001 \end{bmatrix} \approx \begin{bmatrix} \ \\ \ \end{bmatrix}$$

- The length of the vector need to be the same as n, if $\mathbf{A}$ is a $m \times n$ matrix.
- The result is a vector of length m.
- The i-th entry of the result vector is the scalar product of row i in A and v, e.g. $w_i = A^T{}_i \cdot \mathbf{v}$

Matrix-vector multiplication

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 \\ -0.0001 \\ 0.1 \\ 0.0000001 \end{bmatrix} \approx \begin{bmatrix} 9.4 \end{bmatrix}$$

$$1994 \times 0.004 - 142 \times 0.0001 + 7 \times 0.1 + 727327 \times 0.0000001 = 9.4$$

- The length of the vector need to be the same as n, if $\mathbf{A}$ is a $m \times n$ matrix.
- The result is a vector of length m.
- The i-th entry of the result vector is the scalar product of row i in A and v, e.g. $w_i = A^T{}_i \cdot \mathbf{v}$

Matrix-vector multiplication

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 \\ -0.0001 \\ 0.1 \\ 0.0000001 \end{bmatrix} \approx \begin{bmatrix} 9.4 \\ 8.2 \end{bmatrix}$$

$$1972 \times 0.004 - 175 \times 0.0001 + 3 \times 0.1 + 302393 \times 0.0000001 = 8.2$$

- The length of the vector need to be the same as n, if $\mathbf{A}$ is a $m \times n$ matrix.
- The result is a vector of length m.
- The i-th entry of the result vector is the scalar product of row i in A and v, e.g. $w_i = A^T{}_i \cdot \mathbf{v}$

Matrix-vector multiplication example

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 \\ -0.0001 \\ 0.1 \\ 0.0000001 \end{bmatrix} \approx \begin{bmatrix} 9.4 \\ 8.2 \end{bmatrix}$$

Instead of element-wise multiplication, as is the case for ndarray, the operator ***, using the **matrix** class, does matrix multiplication.

In [23]:

```
A = np.array([[1994,142,7,7273274], [ 1972,175,3,302393 ]])
v = np.matrix('0.004;-0.0001;0.1;0.0000001')
print(A*v)
```

```
[[ 9.3891274]
 [ 8.2007393]]
```

Matrix-matrix multiplication

- $\mathbf{A} \times \mathbf{B} = \mathbf{C}$
- The $c_{i,j}$ entry is the dot product of the i-th row in $\mathbf{A}$ and the j-th column in $\mathbf{B}$
- Matrix-matrix multiplication is only defined on matrices $\mathbf{A}$ and $\mathbf{B}$, where the number of columns in $\mathbf{A}$ equals the number of rows in $\mathbf{B}$.
- If $\mathbf{A}$ is $m \times n$ and $\mathbf{B}$ is $n \times p$ then $\mathbf{C}$ is $m \times p$

Matrix-matrix multiplication (continued)

An example of matrix-matrix multiplication:

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 & 1 \\ -0.0001 & 2 \\ 0.1 & 3 \\ 0.0000001 & 4 \end{bmatrix} \approx \begin{bmatrix} 9.4 & \\ 8.2 & \end{bmatrix}$$

Matrix-matrix multiplication (continued)

An example of matrix-matrix multiplication:

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 & 1 \\ -0.0001 & 2 \\ 0.1 & 3 \\ 0.0000001 & 4 \end{bmatrix} \approx \begin{bmatrix} 9.4 & \\ 8.2 & \end{bmatrix}$$

$$1994 \times 1 - 142 \times 2 + 7 \times 3 + 727327 \times 4 = 9.4$$

Matrix-matrix multiplication (continued)

An example of matrix-matrix multiplication:

$$\begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0.004 & 1 \\ -0.0001 & 2 \\ 0.1 & 3 \\ 0.0000001 & 4 \end{bmatrix} \approx \begin{bmatrix} 9.4 & 29095395 \\ 8.2 & 1211903 \end{bmatrix}$$

$$1972 \times 1 - 175 \times 2 + 3 \times 3 + 302393 \times 4 = 8.2$$

In [24]:

```
A = np.matrix([[1994,142,7,7273274], [ 1972,175,3,302393 ]])
B = np.matrix('0.004,1;-0.0001,2;0.1,3;0.0000001,4')
print(A*B)
```

```
[[  9.38912740e+00   2.90953950e+07]
 [  8.20073930e+00   1.21190300e+06]]
```

Computational complexity of matrix multiplication

- Using the definition of matrix multiplication gives an naive algorithm that takes time on the order of n^3 to multiply two $n \times n$ matrices, $\mathcal{O}(n^3)$.
- Fastest practical algorithm is the Strassen algorithm, which only uses 7 multiplications for multiplying two 2×2 matrices, resulting in $\mathcal{O}(n^{\log_2 7}) \approx \mathcal{O}(n^{2.8})$, not a big difference.
- Fastest theoretical is an optimized version of the Coppersmith–Winograd algorithm, with a complexity of $\mathcal{O}(n^{2.373})$, but with a huge constant making it infeasible for matrices with reasonable sizes.

Identity matrix

$$\mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- An identity matrix is square ($n \times n$) with ones on the main diagonal and zeros elsewhere.
- Is usually denoted by $\mathbf{I}_n$ where n is the dimension.
- The identity matrix is the multiplicative identity for matrix multiplication, i.e. $\mathbf{I}_n\mathbf{A} = \mathbf{A}\mathbf{I}_n = \mathbf{A}$ for any matrix $\mathbf{A}$.

Identity matrix (continued)

To utilize the identity matrix in NumPy, use **np.identity(n)**:

In [25]:

```
I3 = np.identity(3)
A = np.matrix([[2,3,5],[8,12,20],[12,18,30]])
print(I3)
print(A)
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
[[ 2  3  5]
 [ 8 12 20]
 [12 18 30]]
```

In [26]:

```
print(I3*A)
print(A*I3)
```

```
[[  2.   3.   5.]
 [  8.  12.  20.]
 [ 12.  18.  30.]]
[[  2.   3.   5.]
 [  8.  12.  20.]
 [ 12.  18.  30.]]
```

Inverse matrix

- Inverse of a matrix $\mathbf{A}$ is denoted $\mathbf{A}^{-1}$.
- Inverses of matrices are only defined on square matrices and not all matrices are invertible.
- The inverse of matrix is the multiplicative inverse, i.e. $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$.
- Inverting a matrix has the same computational complexity as matrix multiplication.

Example inverse matrix

To calculate the inverse of a matrix you can use **np.linalg.inv()** or **.I** on the matrix object:

In [27]:

```
A = np.matrix([[2,3,4],[8,12,20],[12,19,30]])
Ainv = np.linalg.inv(A) #Identical to A.I
print(A)
print(Ainv)
```

```
[[ 2  3  4]
 [ 8 12 20]
 [12 19 30]]
[[ 2.5   1.75 -1.5 ]
 [-0.   -1.5   1.  ]
 [-1.    0.25 -0.  ]]
```

Should be equal to the identity matrix (within some numerical rounding error):

In [28]:

```
print((A*Ainv).round(13))
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

Summary

- Machine learning
  - Supervised
  - Unsupervised
- Linear algebra
  - Matrix
  - Vector
  - Scalar product
  - Matrix multiplication

First exercise

Will involve vector and matrix math, the NumPy Python package:

- #### 1. Math checkup Where you will do some of the math by hand.
- #### 2. NumPy and Spark linear algebra You will do some exercise using the NumPy package.

Extra slides

Euclidian norm (continued)

$$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x_1^2 + \cdots + x_n^2}$$

- If the size of the vector n = 1 (scalar), you get the absolute value:

$$\|x\|_2 = \sqrt{x^2} = |x|$$

Scalar product again

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \Rightarrow \begin{bmatrix} A_{1,1} \\ A_{2,1} \\ A_{3,1} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

- Scalar product is a special case of matrix-vector multiplication, where the number of rows is one.
- $\mathbf{A} \times \mathbf{v} = \mathbf{w} \Rightarrow \mathbf{A}^T \cdot \mathbf{v}$ if A is $1 \times n$

Addition and subtraction

- Two matrices must have an equal number of rows and columns to be added or subtracted, e.g. $2 \times 2 + 2 \times 2$.
- The resulting matrix has the same dimensions as the operands.
- The operations are done element-wise, e.g. $\mathbf{A} + \mathbf{B} = \mathbf{C}$ then $a_{i,j} + b_{i,j} = c_{i,j}$

Addition and subtraction (continued)

Example of addition:

$$\begin{bmatrix} 2 & 5 \\ 3 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix}$$

Example of subtraction:

$$\begin{bmatrix} 5 \\ 7 \end{bmatrix} - \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Addition and subtraction (continued)

Example of addition:

$$\begin{bmatrix} 2 & 5 \\ 3 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} 2+1 & 5+6 \\ 3+4 & 7+8 \end{bmatrix}$$

Example of subtraction:

$$\begin{bmatrix} 5 \\ 7 \end{bmatrix} - \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 5-1 \\ 7-4 \end{bmatrix}$$

Addition and subtraction (continued)

Example of addition:

$$\begin{bmatrix} 2 & 5 \\ 3 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 6 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} 2+1 & 5+6 \\ 3+4 & 7+8 \end{bmatrix} = \begin{bmatrix} 3 & 11 \\ 7 & 15 \end{bmatrix}$$

Example of subtraction:

$$\begin{bmatrix} 5 \\ 7 \end{bmatrix} - \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 5-1 \\ 7-4 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

In [29]:

```
A = np.matrix([[2,5],[3,7]])
B = np.matrix([[1,6],[4,8]])
print(A+B)
C = np.array([5,7])
D = np.array([1,4])
print(C-D)
```

```
[[ 3 11]
 [ 7 15]]
[4 3]
```

Matrix-matrix multiplication (continued)

Some properties of matrix multiplication:

- Is an associative operation, rearranging the parentheses in such an expression will not change its value, e.g. $\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}$
- Is not a commutative operation, the order of the operands matter, e.g. generally $\mathbf{AB} \neq \mathbf{BA}$
- The transpose of $\mathbf{AB}$ is the tranpose of $\mathbf{B}$ multiplied by the transpose of $\mathbf{A}$, e.g. $\mathbf{AB}^T = \mathbf{B}^T\mathbf{A}^T$

Outer product

- The operation is symbolized by $\otimes$, e.g. $\mathbf{v} \otimes \mathbf{w} = \mathbf{A}$
- Is defined on two vectors and the result is a matrix.
- Is a special case of matrix multiplication: $\mathbf{v} \otimes \mathbf{w} = \mathbf{v}\mathbf{w}^T = \mathbf{A}$, i.e. $a_{i,j} = \mathbf{v}_i\mathbf{w}_j$

Outer product (continued)

An example of an outer product:

$$\begin{bmatrix} 1 \\ 4 \\ 6 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 4 \\ 6 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 5 \end{bmatrix}$$

Outer product (continued)

An example of an outer product:

$$\begin{bmatrix}1\\4\\6\end{bmatrix} \otimes \begin{bmatrix}2\\3\\5\end{bmatrix} \Rightarrow \begin{bmatrix}1\\4\\6\end{bmatrix} \times \begin{bmatrix}2 & 3 & 5\end{bmatrix} = \begin{bmatrix}1\times 2 & 1\times 3 & 1\times 5\\4\times 2 & 4\times 3 & 4\times 5\\6\times 2 & 6\times 3 & 6\times 5\end{bmatrix}$$

Outer product (continued)

An example of an outer product:

$$\begin{bmatrix}1\\4\\6\end{bmatrix} \otimes \begin{bmatrix}2\\3\\5\end{bmatrix} \Rightarrow \begin{bmatrix}1\\4\\6\end{bmatrix} \times \begin{bmatrix}2 & 3 & 5\end{bmatrix} = \begin{bmatrix}1\times 2 & 1\times 3 & 1\times 5\\4\times 2 & 4\times 3 & 4\times 5\\6\times 2 & 6\times 3 & 6\times 5\end{bmatrix} = \begin{bmatrix}2 & 3 & 15\\8 & 12 & 20\\12 & 18 & 30\end{bmatrix}$$

Outer product (continued)

An example of an outer product:

$$\begin{bmatrix}1\\4\\6\end{bmatrix} \otimes \begin{bmatrix}2\\3\\5\end{bmatrix} = \begin{bmatrix}2 & 3 & 15\\8 & 12 & 20\\12 & 18 & 30\end{bmatrix}$$

To calculate the outer product using NumPy, use the function np.outer($\mathbf{u}$,$\mathbf{v}$):

In [30]:

```
u = np.array([1,4,6])
v = np.array([2,3,5])
A = np.outer(u,v)
print(A)
```

```
[[ 2  3  5]
 [ 8 12 20]
 [12 18 30]]
```

Supervised learning and regression

Today's schedule

- Overview of machine-learning and linear algebra.
- Exercise in linear algebra.
- Regression and ML pipeline. **<---**
- Exercise in linear regression.
- Classification and logistic regression.
- Exercise in logistic regression.

Regression

Regression

To learn a model that predicts a continuous label, i.e. a numeric value, from observations (features).

Examples of values that can be predicted:

- Age of a person from his/her shoe size, length, etc.
- Price of a car from manufacturer, country of purchase, etc.
- IMDb score of a movie from year, director, etc.

Linear regression

Each observation is represented by a label y and a feature vector $\mathbf{x}$:

$$y \in \mathrm{R} \qquad \mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_i \end{bmatrix}$$

We assume that there exists a linear mapping between the features and the label:

$$y \approx \hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_i x_i$$

Linear regression (continued)

If we extend the feature vector with a constant $x_0 = 1$, we get the following:

$$\mathbf{x}^T = \begin{bmatrix} 1 & x_1 & x_2 & x_3 & \dots & x_i \end{bmatrix}$$

And let the weights be the vector $\mathbf{w}$:

$$\mathbf{w}^T = \begin{bmatrix} w_0 & w_1 & w_2 & w_3 & \dots & w_i \end{bmatrix}$$

And then we can write the linear mapping as the following scalar product:

$$y \approx \hat{y} = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

Linear regression (continued)

The model ($\mathbf{w}$) enables us to predict the IMDb score from the observations ($\mathbf{x}$):

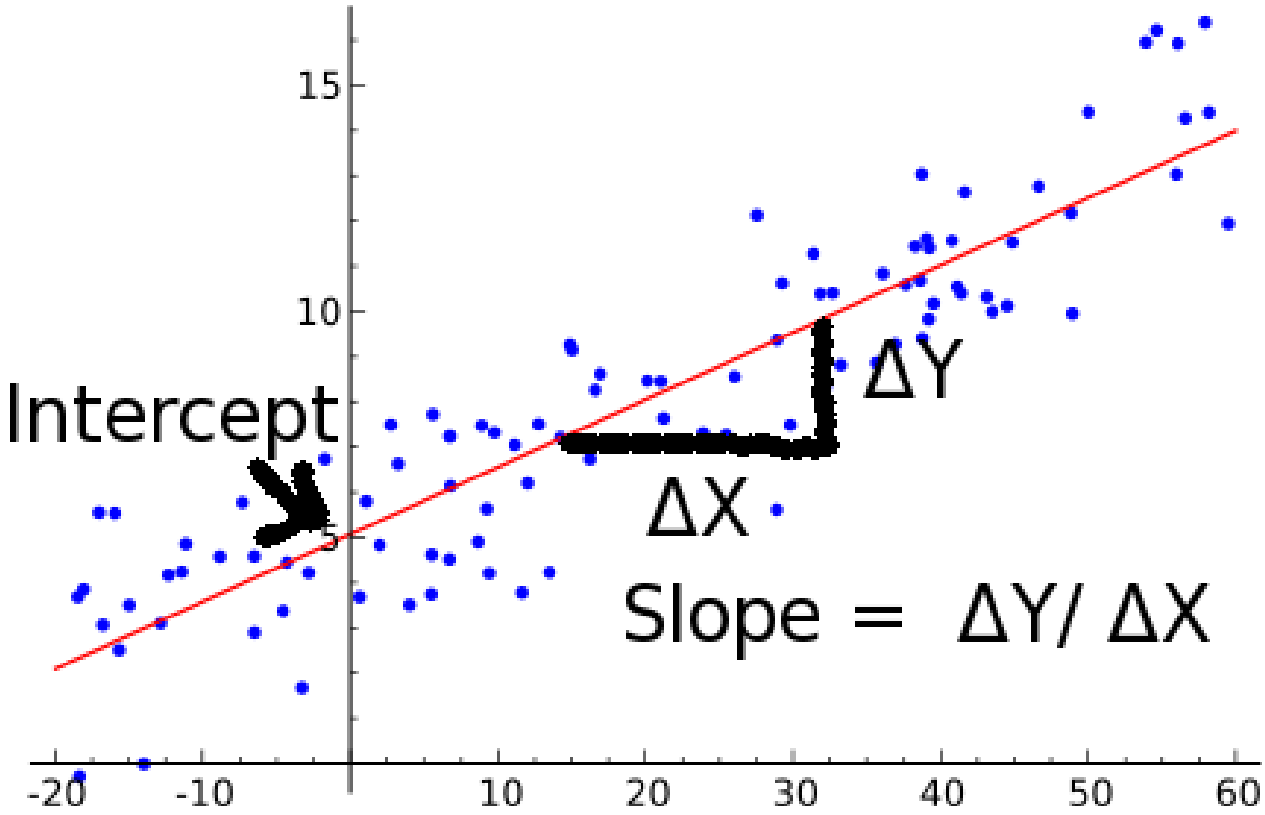
$$\begin{bmatrix} 1 & 1994 & 142 & 7 & 727327 \\ 1 & 1972 & 175 & 3 & 302393 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0.004 \\ -0.0001 \\ 0.1 \\ 0.0000001 \end{bmatrix} \approx \begin{bmatrix} 9.4 \\ 8.2 \end{bmatrix}$$

Linear mapping

Reasons why we would use a linear mapping:

- It is simple to understand.
- It has low computational complexity.
- Works for many tasks.
- It is possible to introduce non-linearities with feature transformation.

Linear regression with one feature



Trying to fit following line to the observations:

$y \approx \hat{y} = w_0 + w_1 x_1$

where w_0 is the intercept or offset and w_1 is the slope or gradient of the line.

Representing an observation: regression

MLlib's represents a data point with the class **LabeledPoint(label, features)**, which consists of the following:

- A **label**, represented by a real value.
- A list of **features**.

In [15]:

```
LabeledPoint(9.3, [1994,142,7,727327]) #Shawshank Redemption
```

Out[15]:

```
LabeledPoint(9.3, [1994.0,142.0,7.0,727327.0])
```

Representing an observation (continued)

- The feature vector can be inputed in one of the following formats:
  - A list.
  - NumPy array.
  - SparseVector, if we have many zeros.
  - DenseVector, a NumPy compatible MLib representation.

Sparse representation

- Encodes only the nonzero elements.
- Efficient storage if it has got many zeros.
- Consists of:
  - the length
  - indices of nonzero elements
  - the values

In [23]:

```
vecA = [ 0,  2,  1,  0,  1,  3,  0,  2,  0,  0]
vecB = [ 0,  0,  0,  1,  1,  2,  1,  2,  0,  0]

vecASparse = (10,[1,2,4,5,7],[2.0,1.0,1.0,3.0,2.0])
vecBSparse = (10,[3,4,5,6,7],[1.0,1.0,2.0,1.0,2.0])
```

Representing an observation: classification

In the case of a classification, the label is a class:

- 0 (negative) and 1 (positive) for binary.
- 0,1,2,... for multiclass.

In [16]:

```
LabeledPoint(1, [36961, 2503]) #Salammbô first chapter French
```

Out[16]:

```
LabeledPoint(1.0, [36961.0,2503.0])
```

Example of LabeledPoint

In [17]:

```
from pyspark.mllib.regression import LabeledPoint
import numpy as np
#Create a 10000x4 matrix with random numbers
randomMatrix = np.random.rand(10000, 4)
print(randomMatrix[0])
print
obs = sc.parallelize(randomMatrix)
#Take first number as label, rest as feature vector
obs = obs.map(lambda x: LabeledPoint(x[0], x[1:]))
print(obs.take(1)[0].label, obs.take(1)[0].features)
```

```
[ 0.5735422   0.90549853  0.10395045  0.5981861 ]

(0.5735421994807254, DenseVector([0.9055, 0.104, 0.5982]))
```

Evaluating regression

We can measure the closeness between the labels (y_i) and the predictions ($\hat{y}_i$) with the following performance measures:

- Mean absolute error (MAE):

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Evaluating regression (continued)

- Mean squared error (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Has nice mathematical properties, as we soon will see.

Evaluating regression (continued)

- Root mean squared error (RMSE):

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

which has the same unit as the quantity being estimated, e.g. IMDb score instead of IMDb score^2. It is usually slightly higher than MAE.

Evaluating using MLlib

In MLlib you can use the **RegressionMetrics** class to calculate the previously mentioned performance measures. It takes a RDD of (prediction, label) pairs as an argument.

The following variables are than available from the class:

- meanAbsoluteError
- meanSquaredError
- rootMeanSquaredError

An example will follow after we introduce LinearRegressionWithSGD.

Linear least squares

Let us first define some mathematical objects:

- n is the number of observations.
- d is the number of features.
- $\mathbf{X} \in \mathrm{R}^{n \times d}$ Matrix containing the observations.
- $\mathbf{y} \in \mathrm{R}^{n}$ The labels.
- $\mathbf{w} \in \mathrm{R}^{d}$ The weights of the model.
- $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} \in \mathrm{R}^{n}$ The predictions.

Linear least squares (continued)

We want to learn a model $\mathbf{w}$, that tries to minimize the MSE:

$$\min_w \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \min_w \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}\mathbf{x}_i)^2 \Rightarrow \min_w \| \mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

It has the following closed form solution, if the inverse exist:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Ridge regression

- We want the model to generalize well, i.e. predict unseen data correctly.
  - Least squares tries to minimize the error on the training data and it could overfit.
- A simpler model may be less prone to overfitting.
  - A model with smaller weights is simpler.

Ridge regression (continued)

We can introduce a term representing the model complexity, represented by the square of the norm of the weights:

$$\min_{w} \; \| \mathbf{y} - \mathbf{X}\mathbf{w} \|_2^2 + \lambda \; \| \mathbf{w} \|_2^2$$

Where λ is a tunable parameter, to trade off model complexity with training error.

It has the following closed form solution, if the inverse exists:

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_d)^{-1}\mathbf{X}^T\mathbf{y}$$

Optimizing linear regression

- We saw a closed form solution that minimizes $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$
  - If the number of features is large, this solution is unusable.
  - We need to iteratively solve the problem.
- Gradient descent is an iterative way to optimize the function.

Least squares with gradient descent

The iterative solution is given by this update formula:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \sum_{i=j}^{n} (\mathbf{w}_i \mathbf{x}_j - y_j)\mathbf{x}_j$$

It has the parameter α (step size), which decides how big the jump for each iteration is.

(see "Extra slides: gradient descent" for more information)

Choosing the α (step size)

If you choose an alpha (step size) that is:

- Smaller than needed, you will converge slowly.
- Larger than needed, it is possible to overshoot the target and even diverge.

Choosing the α (continued)

- It is possible to change the α during the descent, so that it will decrease with the number of iterations.
  - This will make the jumps smaller the closer you come to the solution.

$$\alpha_i = \frac{\alpha}{n\sqrt{i}}$$

In the equation:

- α is a constant.
- n is the number of observations.
- i is the number of iterations.

Linear regression with stochastic gradient descent (SGD)

- The MLib implementation of stochastic gradient descent uses a simple (distributed) sampling of the data examples.
- Calculating the true gradient, would require access to the full data set, instead we use a fraction of the data.
- For each iteration, a computation of the sum of the partial results is performed.

Linear regression with Stochastic Gradient Descent (continued)

The advantages of SGD are:

- Efficiency.
- Can handle large amount of features and observations.

The disadvantages are:

- Requires several hyperparameters, e.g. step size or regularization.
- Sensitive to feature scaling.

SGD with MLlib

The class that trains linear regression with stochastic gradient descent is called **LinearRegressionWithSGD**.

The class has the method **train()**, which returns a LinearRegressionModel, the method has the following arguments:

- data – The training data, an RDD of LabeledPoint.
- Optional:
  - iterations – The number of iterations.
  - step – The step parameter used in SGD.
  - miniBatchFraction – Fraction of data to be used for each iteration.
  - initialWeights – The initial weights.

SGD with MLlib (continued)

- Optional arguments continued:
  - regParam – The regularizer parameter.
  - regType – The type of regularizer used for training our model.
    - “l1” for using L1 regularization (lasso).
    - “l2” for using L2 regularization (ridge).
    - None for no regularization.
  - intercept – Boolean parameter, to use an intercept.
  - validateData – Boolean parameter, to validate data.

SGD example

In the following example I will do the following:

- Reuse the observations that was created in a previous example.
  - It contains 10000 observations with a label and 3 features.
  - The numbers are all uniformly distributed between 0 and 1.
- Split the data set into a 75% training set and 25% validation set.

SGD example

- Train a LinearRegressionWithSGD model with the standard values for the arguments.
  - It should create a model that is no better than randomly guessing a number between 0 and 1.
- Predict the labels for the validation set using the model.
- Calculate some performance measures on the predicted labels.
  - The mean absolute error should be $\approx$ 0.25 (because of the distribution of the numbers).

SGD example (continued)

In [20]:

```
from pyspark.mllib.regression import LinearRegressionWithSGD
from pyspark.mllib.evaluation import RegressionMetrics

obsTrain, obsVal = obs.randomSplit([0.75,0.25],0)
SGDmodel = LinearRegressionWithSGD.train(obsTrain,
                                         iterations=100,
                                         step=1.0,
                                         miniBatchFraction=1.0,
                                         initialWeights=None,
                                         regParam=0.01,
                                         regType="l2",
                                         intercept=True,
                                         validateData=True)

labelPred = obsVal.map(lambda lp: (float(SGDmodel.
                                         predict(lp.features))
                                   ,lp.label))

metrics = RegressionMetrics(labelPred)
print(SGDmodel.intercept,SGDmodel.weights)
print("MAE: "+str(metrics.meanAbsoluteError))
print("MSE: "+str(metrics.meanSquaredError))
print("RMSE: "+str(metrics.rootMeanSquaredError))
```

```
(0.5625502514661913, DenseVector([-0.0346, -0.045, -0.0335]))
MAE: 0.2499296327
MSE: 0.0834748610986
RMSE: 0.288920163884
```

A supervised learning pipeline

Supervised learning pipeline

- Starts out as raw data.

Raw data

- You need to collect the data somehow, e.g:
  - Record which ads an user clicks on.
  - Crawl the IMDb website.
  - Get patient data from a registry.
- Real world data tends to be "dirty".

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.

Preprocessing data

- A tip is to do an ocular inspection (on samples) of the data set before starting the preprocessing procedure.
- You often see if something obivious stands out on the data, e.g. if a variable is coded wrong or if a feature has many missing values.

Then you may want to do any of the following steps:

- Formatting the data.
- Removing corrupt data.
- Handle missing values:
  - Remove observations with missing data.
  - Impute the missing value.
- Converting the data, e.g. Yes -> 1, No -> 0.
- Sampling the data.

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.
- Feature extraction.

Extracting features

- Creating the possible features (feature engineering) that are going to be used to train the model.
- This can be done as a part of the preprocessing step or on the fly by the program.

Some ways of creating the features:

- Using the data as it is.
- Changing the unit of the data, e.g. creatinine from mg/dl to µmol/l.
- Using one-hot encoding to represent categorical data.
- Using bag-of-words to represent text.

Extracting features (continued)

- Creating categorical data from real valued, e.g angle (°) to direction (N,S,W,E).
- Using mathematical functions, e.g. $\sqrt{x}$ or $\sin(x)$.
- Combining features:
  - Creating polynomial features, e.g. x^2 or xy.
  - Using binary operators, e.g. AND or XOR.
  - Using any combination of mathematical functions.

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.
- Feature extraction.
- Transformation of features.

Scaling and normalizing

- The idea is to scale/standardize/normalize the features so they become more homogeneous. This may:
  - Improve the prediction result.
  - Speed up the training of the model.

- In the IMDB data set, the box office feature outweights the number of Oscars for example.

$$\mathbf{A} = \begin{bmatrix} 1994 & 142 & 7 & 727327 \\ 1972 & 175 & 3 & 302393 \end{bmatrix}$$

- This means that the norm of an observation will likely to be equal to this feature.
- The idea is to normalize the features so they become more homogeneous.

Scaling and normalizing (continued)

- **Rescaling** a feature, i.e. scaling the feature to the interval [0,1]:

$$x' = \frac{x - \min(x)}{\max x - \min x}$$

- **Standardizing** a feature, i.e. that the feature has zero-mean and unit-variance:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Scaling and normalizing (continued)

- **Normalizing** a feature vector, i.e. making the feature vector have unit length in the norm (usually the euclidan norm, i.e p = 2):

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \qquad \mathbf{v}' = \frac{\mathbf{v}}{\|\mathbf{v}\|_p} \qquad \|\mathbf{v}'\| = 1$$

Scaling and normalizing with MLlib

MLlib has two classes representing feature standardization and normalizing: **StandardScaler** and **Normalizer**.

- The StandardScaler takes an RDD[Vector] as input, using the **fit()** and the **transform()** methods, learns the summary statistics, and then returns a standardized data set.

$$x' = \frac{x - \bar{x}}{\sigma}$$

- The Normalizer takes a float p and then transforms individual feature vectors to have unit p-norm.

$$\|\mathbf{v}\|_p = (\sum_{i=1}^{n} |x_i|^p)^{1/p} \qquad \mathbf{v}' = \frac{\mathbf{v}}{\|\mathbf{v}\|_p} \qquad \|\mathbf{v}'\| = 1$$

Scaling and normalizing example

In [19]:

```
from pyspark.mllib.feature import StandardScaler
from pyspark.mllib.feature import Normalizer

features = sc.parallelize([[666,1337,1789],[1066,21,1],[1,3,5]])

scaler = StandardScaler().fit(features)
scaledFeatures = scaler.transform(features)
print(scaledFeatures.collect())
print
norm = Normalizer(float('inf'))
normScaledFeatures = norm.transform(scaledFeatures)
print(normScaledFeatures.collect())
```

```
[DenseVector([1.238, 1.7476, 1.735]), DenseVector([1.9815, 0.0274,
0.001]), DenseVector([0.0019, 0.0039, 0.0048])]

[DenseVector([0.7084, 1.0, 0.9928]), DenseVector([1.0, 0.0139, 0.00
05]), DenseVector([0.3834, 0.8087, 1.0])]
```

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.
- Feature extraction.
- Transformation of features.
- Supervised learning.

Learning a model

Train a model on your training set, e.g. using any of these algorithms:

- Regression.
  - Linear (lasso/ridge).
  - Nonlinear.
  - Regression trees.
- Classification (will go through in next lecture).

Learning a model (continued)

- Both.
  - Random forest.
  - Artificial neural network (ANN).
  - Support vector machine (SVM).

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.
- Feature extraction.
- Transformation of features.
- Supervised learning.
- Evaluation.

Evaluation

- Evaluate on your validation set, and then test set, if you created one.
- Possibly use cross validation, instead of a validation set.
- Either way you need a performance meassure/metric, e.g. using any of these:
  - Mean absolute error (MAE).
  - Mean squared error (MSE).
  - Root mean squared error (RMSE).

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.
  - Feature extraction.
  - Transformation of features.
  - Supervised learning.
  - Evaluation.
- Possible reiteration (of the above steps).

Reiteration/optimization

You usually want to reiterate through some of the previously mentioned steps, to be able to improve the prediction of unseen examples, which we estimate by predicting the validation set and then applying an appropriate evaluation metric.

Changes that may improve the result includes:

- The features being used (the feature set).
- The scaling and normalization of features.
- The algorithm to create the model.
- Hyperparameters of the model.

Tuning of the hyperparameters

For example:

- LinearRegressionWithSGD has a number of hyperparameters, e.g. the number of iterations and the step size.
- These parameters can sometimes greatly influence the quality of the model.
- You want to optimize the model with the best choice of these arguments.
- Cross-validation is often used to estimate this performance.

Tuning of the hyperparameters (continued)

Grid search is an exhaustive search, which may be computationally infeasible, if the parameters are somewhat independent of each other, it is possible to tune on at a time:

- Optimizing one parameter at a time.
  - Choose reasonable starting values for the parameters.
  - Tune one parameter.
  - Evaluate on validation data.
  - Fix the best parameter value.
  - May need to reiterate for some parameters.

Supervised learning pipeline (continued)

- Starts out as raw data.
- Preprocessing data.
  - Feature extraction.
  - Transformation of features.
  - Supervised learning.
  - Evaluation.
- Possible reiteration (of the above steps).
- Prediction.

Prediction

Once the model is ready you can predict:

- I.e. apply the model to an unseen observation.
- E.g. predicting if a patient has cancer or not.
- You may want to reserve a test set from the original data, to be able to estimate the generalization of the model.

Summary

- Regression
  - Evaluation
  - Gradient descent
  - SGD with MLlib
- Supervised learning pipeline
  - Extract features
  - Scaling and normalizing
  - Learning a model
  - Evaluate / reiterate

Second exercise

Will involve linear regression with stochastic gradient descent and with MLlib.

This exercise will be divided into three parts:

- #### 1. Importing and preparing the data
- #### 2. Creating a baseline benchmark
- #### 3. Utilizing MLlib

Parkinsons Telemonitoring Data Set

- A data set from the UCI Machine Learning Repository.
- Unified Parkinson's Disease Rating Scale (UPDRS)
  - Measures a person's progress in the disease in a scale from 0 to 176.
- Has about 6000 observations and 26 features, which are numerical and taken from voice samples.
- Your task is to correctly predict the UDPRS-score based on these voice measures.
- The performance metric is the mean absolute error (MAE).

Extra slides: gradient descent (they got missing images)

Convex function

A convex function is U-shaped, which means that there only exist one local minimum and it is the global minimum.

Non-convex function

A non convex function may have several local minima, which are not the the global minimum.

Convextivity

Fortunately Least Squares, Ridge Regression and Logistic Regression are all convex minimization problems.

Gradient descent (continued)

- First pick an arbitrary starting point.
- Iterate:
  - Decide a descent direction.
  - Pick a step size.
  - Update.
- Until stoping criterion is satisfied.

Deciding the descent direction

Because it is a convex function, the direction of descent is the negtive slope.

This gives the following update rule:

$$w_{i+1} = w_i - \alpha_i \frac{df}{dw}(w_i)$$

Least squares with gradient descent

Using least squares as the function gives the following update rule:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \frac{df}{dw}(\mathbf{w}_i) = \mathbf{w}_i - \alpha_i \frac{df}{dw}(\sum_{j=1}^{n} (\mathbf{w}_i \mathbf{x}_j - \hat{y}_j)^2) \Rightarrow$$

Using the chain rule gives the following formula:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \sum_{i=j}^{n} (\mathbf{w}_i \mathbf{x}_j - \hat{y}_j)\mathbf{x}_j$$

Evaluation (continued)

- Classification.
  - Confusion matrix.
    - Accuracy.
    - Precision, recall, and F1.
  - Receiver operator characteristic (ROC).
    - Area under ROC (AUROC).

Tuning of the hyperparameters (continued)

- Grid search
  - Define and discretize the search space (linear or log scale), e.g:
    - iter $\in \{10, 50, 100, 500, 1000\}$
    - $\alpha \in \{1, 2, 3, 4, 5\}$
  - Test all n-tuples in the cartesian set of the parameters, e.g:
    - para $\in$ iter $\times$ $\alpha = \{(1, 10), (1, 50), \ldots\}$
    - |para| = 25

    - Grows quickly with the number of parameters.
- Evaluate on the validation data.

Classification and logistic regression

Today's schedule

- Overview of machine-learning and linear algebra.
- Exercise in linear algebra.
- Regression and ML pipeline.
- Exercise in linear regression.
- Classification and logistic regression. **<---**
- Exercise in logistic regression.

Classification

To learn a model that predicts a categorical label, i.e. a discrete value, which may be binary (1 or 0), or multiclass (1,2,3,...) from observations (features).

Examples of values that can be predicted:

- An email as either spam or not spam, from the subject, body of the mail, etc.
- A patient as having cancer or not cancer, from the age, blood values, etc.
- A customer as having low, medium, or high interest, from click patterns, buying patterns, etc.

Features

Types of data

The raw data may contain several types of information, for example:

- Numeric, e.g. age or blood creatinine value.
- Binary, e.g. gender or apple/pc user.
- Categorical, e.g. country or car brand.
  - Ordinal, e.g. customer interest (low, medium, high)
- Text, e.g. body of an email or wikipedia article.

Ordinal variables

For ordinal variables, there exists an ordering of the categories, e.g. high > low, but usually no measure of closeness.

We can represent the variable as a single numerical feature, e.g. low = 1, medium = 2, and high = 3

- Then we introduce a degree of closeness, which may or may not be desirable.
- Or we can use one-hot encoding (a way to represent nominal/categorical data).

Categorical variables

For categorical variables, there is no intrinsic ordering or measure of closeness, you can not say that Germany > France.

We could represent the variable as a single numerical feature, e.g. Germany = 0, France = 1, Belgium = 2

- Then we introduce both an ordering and a degree of closeness, which usually is not desirable.
- Better to use one-hot encoding!

One-hot encoding (OHE)

- Also called dummy encoding.
- Create a vector of binary features, where one of them is equal to 1 and rest is 0
  - Where the index in the vector represents the category value.
- Like in C bit field encoding.
- This does not introduce spurious relationships between the categories.
- Exist in MLib from version 1.4 as **OneHotEncoder**.

For example:
{Germany = 0, France = 1, Belgium = 2} →
Germany = [1,0,0]
France = [0,1,0]
Belgium = [0,0,1]

Example of OHE

```
In [1]:
```

```
{'Belgium': 2, 'Sweden': 3, 'Germany': 0, 'Russia': 4, 'France':
 1}
Sweden = [ 0.  0.  0.  1.  0.]
France = [ 0.  1.  0.  0.  0.]
```

Text data

- A way to represent text data is to use the bag-of-words representation:
  - First create a dictionary of the words similar to OHE encoding.
  - Instead of binary values for the indices, we store a number corresponding to the frequency of the word in that observation.

- This usually results in quite sparse feature vector
  - The number of unique words in the English language is over 1 million, although all words are not used in any corpus.

Example of bag of words

- Text 1: John likes to watch movies. Mary likes movies too.
- Text 2: John also likes to watch football games

{John = 0, likes = 1, to = 2, watch = 3, movies = 4, also = 5, football = 6, games = 7, Mary = 8, too = 9} →

- Encoding of text 1: [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
- Encoding of text 2: [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

The element values are called the term frequency (TF).

Feature hashing

- Use a hash function to calculate the index of the token, modulus the length of the desired vector.
  - This is less memory and computationally expensive then creating a dictionary.
- If the hash function is a reasonable one, the distribution of tokens should be somewhat uniform.
- This can reduce the dimension of the feature vector considerably.
  - Which possibly could create collisions, which may lower the quality of the model.

HashingTF

- In Mllib there exist a class, called **HashingTF**
  - It represents feature hashing and the counting of the frequencies of the hashes.
  - The class takes as argument the number of features.
  - It has got the method **transform()**, that takes the input dataset and return an RDD of SparseVector:s.

Example of feature hashing using TF

In [2]:

```
(10,[1,2,4,5,7],[2.0,1.0,1.0,3.0,2.0])
(10,[3,4,5,6,7],[1.0,1.0,2.0,1.0,2.0])
[ 0.  2.  1.  0.  1.  3.  0.  2.  0.  0.]
[ 0.  0.  0.  1.  1.  2.  1.  2.  0.  0.]
```

Inverse document frequency (IDF)

- Term frequency (TF) is used to measure the importance.
- It is easy to over-emphasize terms that appear very often, e.g. “a”, “the”, and “of”.
- If a term appears very often across the corpus, it means it does not carry special information.
- Inverse document frequency (IDF) is a numerical measure of how much information a term provides.
- IDF is defined as the inverse of the number of documents that contain the term.
- The TF multiplied by the IDF is often called TF-IDF.

TF-IDF

- In Mllib there exist a class, called **IDF**, that represents the TD-IDF model.
  - It needs to do a pass through the data first to calculate the IDF.
  - Then you can use the method **transform(HashingTF)**, that multiplies the TF with the IDF.

Example of IDF

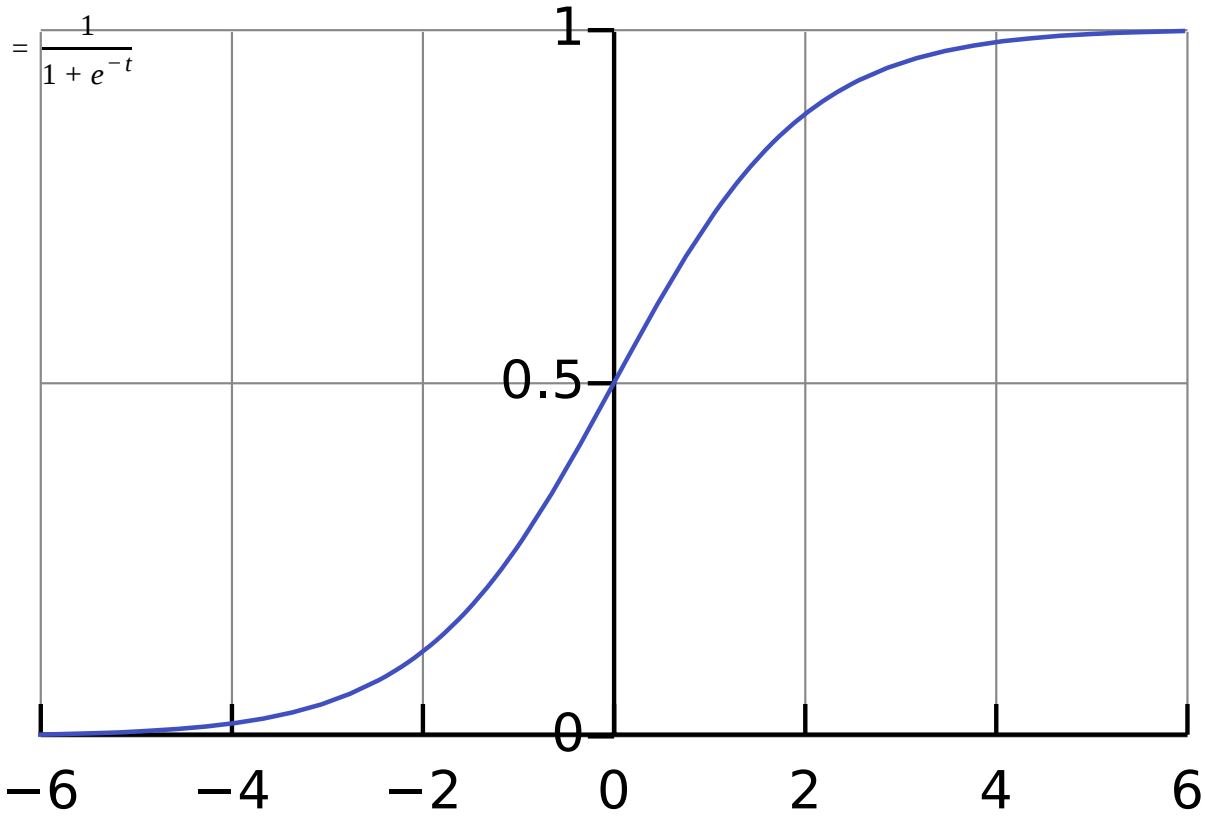
In [4]:

```
(10,[1,2,4,5,7],[2.0,1.0,1.0,3.0,2.0])
(10,[3,4,5,6,7],[1.0,1.0,2.0,1.0,2.0])
(10,[1,2,4,5,7],[0.810930216216,0.405465108108,0.0,0.0,0.0])
(10,[3,4,5,6,7],[0.405465108108,0.0,0.0,0.405465108108,0.0])
```

Logistic regression

The following equation is the logistic function:

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



Logistic regression (continued)

If t is a linear combination of weights, i.e. $t = \mathbf{w} \cdot \mathbf{x}$ we get the following equation:

$$\sigma(t) = \frac{1}{1 + e^{w_0 + w_1 x_1 + \ldots + w_i x_i}}$$

$\sigma(t)$ is between 0 and 1 for every t, for positive infinity it is equal to 1 and for negative it is 0. It is therefore interpretable as a probability.

Logistic regression (continued)

To go from a probability to a binary classification we use a threshold, often 0.5.

- If it is over the treshold we classify it as positive.

$$\sigma(t) \geq 0.5 \Rightarrow 1$$

- If it is under we classify it as negative.

$$\sigma(t) < 0.5 \Rightarrow 0$$

Logistic regression with limited-memory BFGS (L-BFGS)

There exists a SGD variant of logistic regression in MLlib also, but the LBFGS tends to be both faster and use less memory.

It is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using a limited amount of computer memory.

L-BFGS with MLlib

The class that trains logistic regression with L-BFGS is called LogisticRegressionWithLBFGS. The class has the method train, which returns a LogisticRegressionModel, the method has the following arguments:

- data – The training data, an RDD of LabeledPoint.
- Optional:
  - iterations – The maximum number of iterations.
  - initialWeights – The initial weights.
  - corrections – The number of corrections used in the LBFGS update.
  - tolerance – The convergence tolerance of iterations for L-BFGS

L-BFGS with MLlib (continued)

- Optional continued:
  - regParam – The regularizer parameter.
  - regType – The type of regularizer used for training our model.
    - “l1” for using L1 regularization (lasso).
    - “l2” for using L2 regularization (ridge).
    - None for no regularization.
  - intercept – Boolean parameter, to use an intercept.
  - validateData – Boolean parameter, to validate data.
  - numClasses – The number of classes a label can take.

Example of L-BFGS

In [5]:

```
[-0.258464021679,0.740603840877,-0.671843280475,0.805617017835,-
0.469895541298,0.134604891419,-0.407452394167,-1.27835829141,0.5
72797907614]
0.170672270554
```

Classification metrics

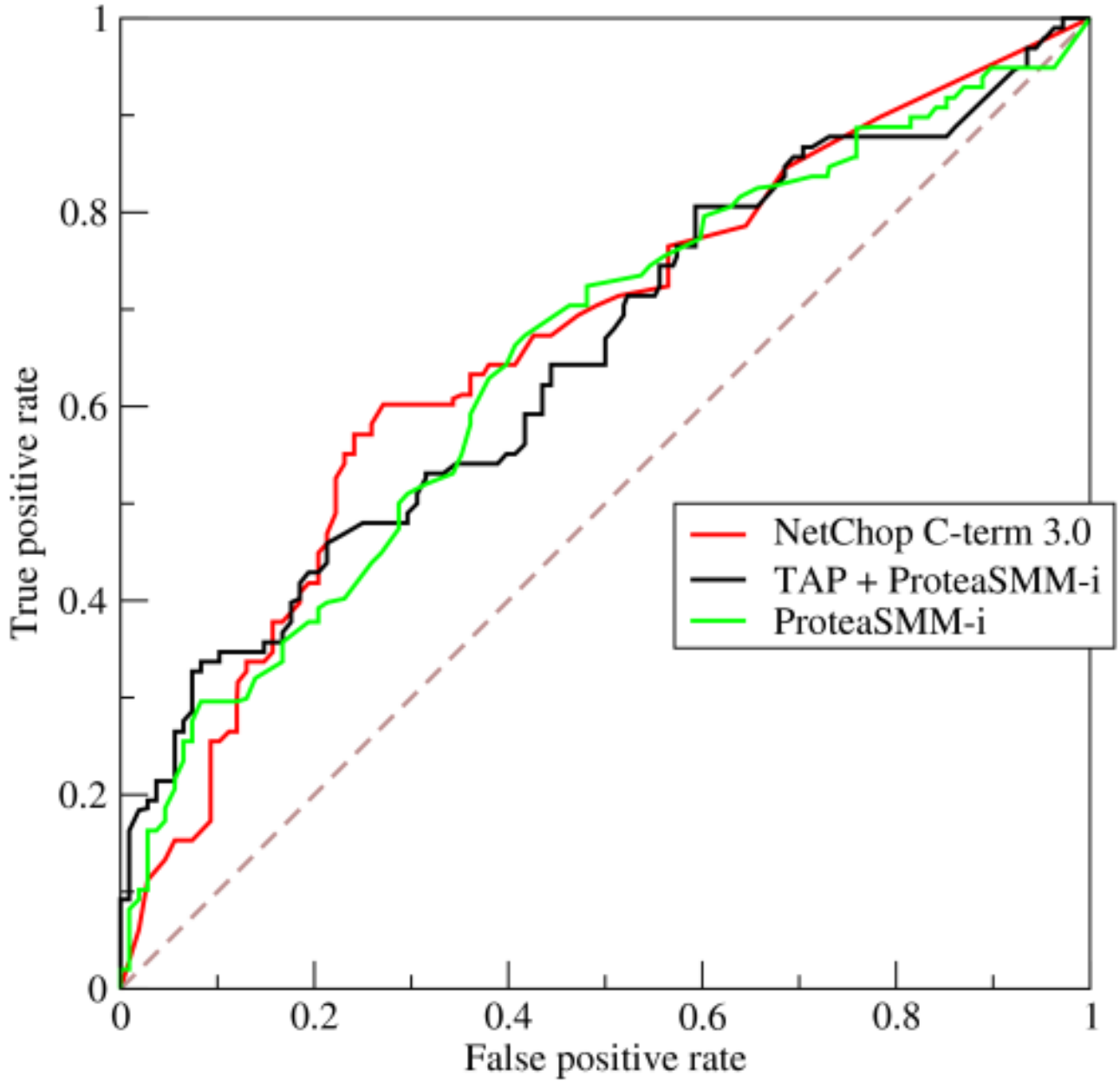
	p' (Predicted)	n' (Predicted)
P (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

Classification metrics (continued)

The following metrics can be defined from the confusion matrix:

- Accuracy.
- Precision, recall.
  - F1, the harmonic mean between them.
  - Area under Precision/Recall curve.
- And a few other metrics.

Receiver operating characteristic (ROC)



ROC (continued)

The ROC is a graph over true positive rate/false positive rate (can be defined from the confusion matrix).

The area under the ROC graph can be interpeted as the probability that as randomly chosen positive example > negative example.

- A model that assigns random labels to observations has an AUROC of 0.5.
- A perfect model has an AUROC of 1.0
- A reasonable model should therefore have an AUROC between 0.5 and 1.0

BinaryClassificationMetrics

In MLlib there exist a class called BinaryClassificationMetrics, that takes a pair RDD of predicted labels and true labels, and then has the following methods:

- areaUnderPR(): Computes the area under the precision-recall curve.
- areaUnderROC(): Computes the area under the ROC curve.

Before using it you need call clearThreshold() on the model to remove the classification and get the raw probability.

Example of AUROC

In [7]:

```
[(0.43377292697214964, 1.0), (0.30489754918354856, 0.0), (0.4111
3216061927377, 1.0), (0.4474745011380102, 0.0), (0.2708764900478
493, 0.0), (0.3322889798311969, 1.0), (0.6380750346426529, 1.0)]

The AUROC of random features: 0.538194444444

The AUPR of random features: 0.608066661347
```

Cross-validation

The whole data set is partitioned into k equal sized subsamples.

Of the k subsamples, a single subsample (in red) is retained as the validation data for testing the model, and the remaining k − 1 samples are used as training data.

The cross-validation process is then repeated k times (the folds). A normal value for k is 5.

Third exercise

Will involve logistic regression with numerical and text based features with MLlib.

This exercise will be divided into three parts:

- #### 1. Importing and preparing the data
- #### 2. Logistic regression
- #### 3. Evaluating the results

The Kaggle data set

- Kaggle is a webpage where you can upload competitions in ML, with monetary awards.
- Teams can do a certain amount of submissions, which the result is reflectd on a leaderboard.
- We have choosen the StumbleUpon competition as the basis for the third exercise.
  - Which is a binary classification task, having 26 features and about 7k observations.
  - The task is to classify webpages as having long lasting appeal (evergreen) to the users or not (ephemeral).
  - The performance measure is the area under the ROC curve (AUROC).

The Kaggle data set (continued)

Imputation of variables

If you use real world data, and in this exercise, it is likely that you will encounter missing feature values, often represented in the data as:

- "" (the empty string)
- ?
- NA

There can be missing values, because in the real world, for various reasons there is not always complete information available, e.g:

- some value was not recorded at collection time
- is not applicable to that data point

Imputation of variables (continued)

Ways of handling observations with missing data

- Casewise deletion: delete the observations that has missing values.
  - This effectively reduces the size of the data set.
  - If the missing values are random, no bias is introduced in the model.
  - However missing values rarely tend to be completely random.

Imputation of variables (continued)

- Mean/mode imputation: replace the missing value with the mean for real valued features and mode for categorical.
  - Has the property that the sample mean for that variable is unchanged.
  - But this can severely distort the distribution for this variable.
  - Distorts relationships between variables by “pulling” estimates of the correlation toward zero.

This is the one you will use in the exercise.

Imputation of variables (continued)

- Hot-deck imputation: replacing the missing value from a uniform distribution of the non-missing values.
  - Has the property that the the distribution for that variable is unchanged.

- Using a model: replace the missing values based on a model created on the other variables.
  - Works good if the variable is correlated with the other variables.

Extra slides

Exhaustive cross-validation

Leave-one-out cross-validation (LOCV) involves using one observation as the validation set and the remaining observations as the training set.

This is repeated on all ways to cut the original sample on a validation set of one observation and a training set. And then calculating the mean validation metric over all partitions.

So as soon as the number of observations is quite big it becomes infeasible to calculate.

k-fold cross-validation

The original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 samples are used as training data.

The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data.

A normal value for k is 10. When k = the number of observations, the k-fold cross-validation is exactly the leave-one-out cross-validation.