ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Text classification with (mostly) linear models

2024–25

Ion Androutsopoulos

http://www.aueb.gr/users/ion/

These slides are partly based on material from the books:

- *Speech and Language Processing* by D. Jurafsky and J.H. Martin, 2nd edition, Pearson Education, 2009,
- *Artificial Intelligence – A Modern Approach* by S. Russel and P. Norvig, 2nd edition, Prentice Hall, 2003,
- *Machine Learning* by T. Mitchell, McGraw-Hill, 1997,
- Neural Network Methods for Natural Language Processing by Y. Goldberg, Morgan & Claypool, 2017.
- *Foundations of Statistical Natural Language Processing* by C.D. Manning and H. Schutze, MIT Press, 1999,

and material from the Machine Learning course of A. Ng at Stanford University.

Contents

- Representing texts as **bags of words**. Boolean and **TF-IDF** features.
- **Feature selection** using **information gain**.
- Text classification with ***k* nearest neighbors** and **Naive Bayes**.
- Evaluating classifiers: **precision, recall, F1, AUC**.
- **Semi-supervised** classification with **Expectation Maximization**.
- Obtaining **word embeddings** from **PMI** scores, using **SVD-based dimensionality reduction**.
- **Word** and **text clustering** with ***k*-means**.
- **Linear** and **logistic regression**, (stochastic) **gradient descent**.
- **Lexicon-based features**. Constructing and using **sentiment lexica**.
- **Support Vector Machines** (SVMs) and **kernels**.
- **Practical advice** and **diagnostics** for text classification with supervised machine learning.

Example: spam filters

our highly successful multi – national company gives you an exclusive business that generates an extra weekly income of up to $ 600 or more … anyone can easily make money … if you wish to be removed from our list …

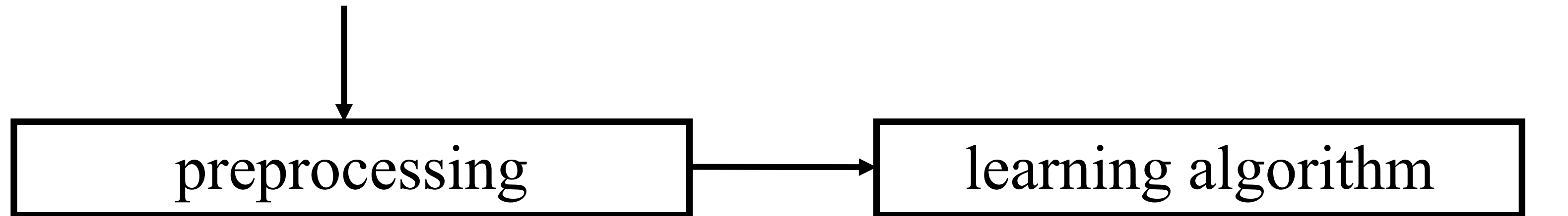
call for papers 9 th european workshop on natural language generation … is a subfield of natural language processing that generates texts in human languages from non-linguistic data or knowledge … for the systems to be successful …

- **Classifying messages** in **two classes.**
  - Spam ($C = 1$), ham ($C = 0$).
- More generally, ***n* classes**.
  - Financial news, politics, sports news (**possibly overlapping**).
  - Positive, negative, neutral, conflict **sentiment** (e.g., of tweets).

Spam filtering with supervised ML

examples of positive and negative
messages + correct classes

training phase



classification
function

incoming message

decision: positive or negative

use (or testing) phase

Text preprocessing

our highly successful multi – national company gives you an exclusive business that generates an extra weekly income of up to $ 600 or more … anyone can easily make money … if you wish to be removed from our list …

call for papers 9 th european workshop on natural language generation … is a subfield of natural language processing that generates texts in human languages from non-linguistic data or knowledge … for the systems to be successful …

< money:1, language:0, natural:0, $:1, adult:0, call:0, exclusive:1, **successful:1**, removed:1, **generates:1**, …>

< money:0, language:1, natural:1, $:0, adult:0, call:1, exclusive:0, **successful:1**, removed:0, **generates:1**, …>

- Alternatively the **features** may be **word (or *n*-gram) frequencies**, **TF-IDF** scores, **non-textual** information (e.g., attachments, colors).

Representing texts as bags of words

- **Boolean** vectors (contain 0, 1 values): **which words** of a vocabulary **occur or not** in the text.
- **Term frequency** (***TF***) vectors: **how frequent** each vocabulary word is in the text.
  - Possibly **divided by the number of tokens** of the text.
  - Or taking the **logarithm** of the frequency.
- ***TF-IDF*** vectors: **for each vocabulary word w_i**, the vector contains its $TF_i \cdot IDF_i$ **score**:

$$IDF_i = \log(\frac{Ndoc}{DF_i})$$

Number of documents in a corpus. (← $Ndoc$)

Number of corpus documents containing w_i. (← DF_i)

  - IDF_i (inverse document frequency) shows **how rare w_i** is **in the language** (not the particular text). Common words are not informative in some tasks (e.g., information retrieval).

Feature selection

- **For which words** (or *n*-grams or ...) should we **include features** in the vectors that represent the texts?
  - 1st step: **discard** words (or *n*-grams) that **do not occur** at least ***k* times** in the **training data** (e.g., $k = 3$).
  - Usually **thousands of words remain** (many more *n*-grams).
  - **Large feature sets → efficiency** and **over-fitting** problems.
  - Alternative 1st step: **discard** words (*n*-grams) **with low TF-IDF** (computing both TF and IDF on the **entire training set**).
  - But features (words, *n*-grams) with **high TF-IDF may not always help** discriminate the classes we are interested in.
- **How useful** is **each candidate feature** X?
  - **How much information** does **knowing the value of** X offer?
  - How much is our **uncertainty** about the **random variable** C (class) **reduced** if we know the value of X?

Information Gain (for discrete features)

- Entropy of C **if we learn that X = 1**:

$$H(C \mid X=1) = -\sum_{c} P(C=c \mid X=1) \cdot \log_2 P(C=c \mid X=1)$$

- Entropy of C **if we learn that X = 0**:

$$H(C \mid X=0) = -\sum_{c} P(C=c \mid X=0) \cdot \log_2 P(C=c \mid X=0)$$

- **Information Gain** (IG):

Expected decrease of $H(C)$, if we learn the value of X.

$$IG(C,X) = IG(X,C) = H(C) - \sum_{x} P(X=x) \cdot H(C \mid X=x)$$

expected value

for every possible value of X

Feature selection with IG

- **Compute** the information gain ***IG*(*C*, *X*)** of each **candidate feature *X***.
  – For example, X may show if the word "money" occurs ($X = 1$) in the text or not ($X = 0$).
  – Probabilities are estimated from training messages (e.g., with Laplace smoothing).
- **Select the features** with the ***m*** **highest *IG*(*C*, *X*)** scores.
  – For spam filtering, $m = 1000$ works reasonably well.
- **Represent the texts** as BOW vectors of ***m*** **dimensions**.
  – E.g., $\langle X_1, X_2, X_3, \ldots, X_m \rangle = \langle 0, 1, 1, \ldots, 0 \rangle$
- Other **similar** feature selection **measures** exist (e.g., χ^2).
- We can also use **SVD** to get **fewer *new* features**.

Example of IG-selected Boolean features

Word of X_i	$P(X_i=1)$	$P(X_i=1 \mid C=0)$	$P(X_i=1 \mid C=1)$
!	0.484105	0.216129	0.828157
$	0.257947	0.040322	0.538302
language	0.247956	0.440322	0.002070
money	0.163487	0.001612	0.372670
remove	0.146230	0.001612	0.333333
free	0.309718	0.104838	0.573498
university	0.219800	0.374193	0.022774

Word embeddings of business terms

(produced with word2vec, here projected to 2D using UMAP)

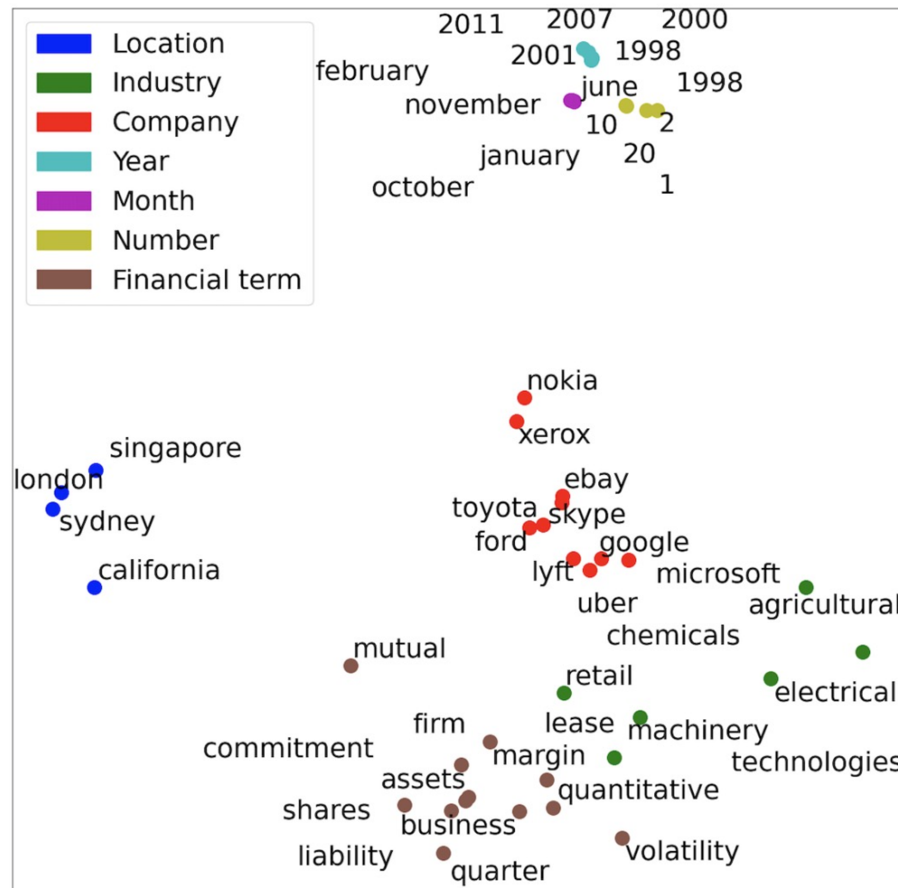
Vectors (points) in **2D**:

y

∘ ⟨2,4⟩

∘ ⟨3,2⟩

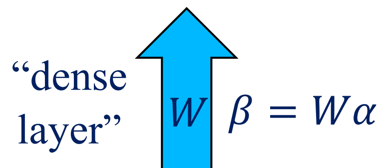
x



Word embeddings are vectors (points), e.g., in a **300D** space.

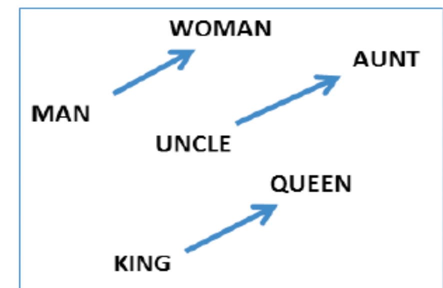
They capture **relatedness**, **analogy**, …

2D vector β



$\beta = W\alpha$

300D vector α



Large image from Loukas et al., "EDGAR-CORPUS: Billions of Tokens Make The World Go Round", EcoNLP workshop, EMNLP 2021 (https://aclanthology.org/2021.econlp-1.2/). Small image from Mikolov et al., "Linguistic Regularities in Continuous Space Word Representations". NAACL 2013 (https://aclanthology.org/N13-1090/). For a quick intro to **UMAP** (and **t-SNE**) check: https://www.youtube.com/watch?v=6BPl81wGGP8.

Word sense embeddings

(produced by a method that produces **dense, sense-specific** word embeddings, then **projected** to **2 dimensions**)

Most words have **multiple senses**. **Word embeddings** (a single point per word) end up being in the **middle of the points** that would correspond to their multiple **senses**.

In a **financial discussion**, we would like "**bank**" to move **closer** to its **financial sense**, **away** from its **river sense**. We'll address this problem later (e.g., with RNNs)

Image source: http://www.socher.org/uploads/Main/MultipleVectorWordEmbedding.png
Huan et al. 2012, "Improving Word Representations via Global Context and Multiple Word Prototypes".

Word embeddings from PMI scores

- Represent **each <u>word</u>** as a **vector** (“**word embedding**”).
  - Here the **vector** shows **how often the word co-occurs** with **every other word** of a vocabulary.

$$\overrightarrow{pilot} = \langle PMI(pilot, air), PMI(pilot, tree), PMI(pilot, door) \ldots \rangle$$

  - The **co-occurrence scores** in the vector are often Pointwise Mutual Information (***PMI***) scores:

$$PMI(w, w_i) = \log \frac{P(w, w_i)}{P(w) \cdot P(w_i)}$$

How likely is it for ***w*** to **co-occur** with the ***i-th*** **vocabulary word**?

Improved, normalized PMI definitions also exist.

  - To “**co-occur**” may mean in the **same sentence**, or in a **window of *n* words**, or **connected with dependencies** (produced by a dependency parser) etc.
- We can **use SVD** to obtain **embeddings** of **fewer dimensions**.
- **Word similarity = similarity of word embeddings** (e.g., cosine).

Dimensionality reduction with SVD

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ x_{2,1} & \cdots & x_{2,n} \\ \vdots & \cdots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix}$$

Each instance (row) is initially a vector of n original features.

$$Z = \begin{bmatrix} z_{1,1} & \cdots & z_{1,k} \\ z_{2,1} & \cdots & z_{2,k} \\ \vdots & \cdots & \vdots \\ z_{m,1} & \cdots & z_{m,k} \end{bmatrix}$$

New form of the instances. Each instance (row) is now a vector of $k < n$ new features.

- **Diagonalization** with **SVD**: $X = UDV$
  - D rect. diagonal. Values on diagonal: **singular values** in **decreasing order**.
  - U $(m \times m)$, V $(n \times n)$ orthonormal (rows/columns unit-length & orthogonal).
- **Approximation** of X: $\tilde{X} = \tilde{U}\tilde{D}\tilde{V}$
  - $\tilde{D}$ diagonal $(k \times k)$. Values on the diagonal: ***k* largest singular values** of D.
  - $\tilde{U}$ $(m \times k)$ and $\tilde{V}$ $(k \times n)$. Hence $\tilde{X}$ is still $m \times n$.
- **Approximation** of $\tilde{X}$: $Z = \tilde{U}\tilde{D}$
  - Z is $(m \times k)$ with $k < n$. **Dot-products** between rows of $\tilde{X}$ **preserved**.
  - In practice, we use $Z = \tilde{U}\sqrt{\tilde{D}}$ or $Z = \tilde{U}$. See Goldberg for more details.

Embeddings of biomedical terms

Table 1 Closest words to the 30 most frequent words of the BioASQ question answering task, using the cosine similarity of the dense vectors to measure proximity. Relevant (closely related) words are shown in bold, possibly relevant in normal font, and irrelevant (or misspelled) words in strikeout.

protein	proteins	a-anchoring	pka-anchoring
thyroid	thyroidal	nonthyroid	hyperfunctioning
associated	correlated	related	correlates
hormone	gh	luetinizing	fshluteinizing
human	murine	mouse	~~immortalized~~
used	utilized	employed	applied
genes	gene	paralogs	operons
treatment	therapy	treatments	treating
disease	diseases	disease-like	~~mmrn1rs6532197~~
gene	genes	pseudogene	gene-encoding
heart	cardiac	chf	congestive
role	roles	~~plays~~	~~play~~
affect	alter	modify	impair
dna	dnas	bisulfite-treated	polymerase-mediated
histone	histones	h4k16	h4
involved	implicated	participates	regulating
list	lists	listing	to-do
proteins	protein	polypeptides	hsp70s
known	~~yet~~	~~presently~~	well-known
patients	outpatients	subjects	whom
present	this	~~aimed~~	~~our~~
cancer	cancers	crc	~~caner~~
receptor	receptors	hmc5	5-nonyloxytryptamine
regulate	modulate	regulates	orchestrate
cell	cells	cancer-cell	sw1710
coding	5-noncoding	5-untranslated	3-noncoding
inhibitors	inhibitor	small-molecule	atp-competing
many	several	some	numerous
related	linked	associated	relate
cardiomyopathy	cardiomyopathies	myocardiopathy	dcm

See http://bioasq.org/news/bioasq-releases-continuous-space-word-vectors-obtained-applying-word2vec-pubmed-abstracts

Centroids of word embeddings

- We can represent **each text *T*** (word sequence) $w_1, w_2, \ldots w_d$ as the **centroid** of its **word embeddings**:

$$\vec{T} = \frac{1}{d}\sum_{i=1}^{d} \vec{w}_i = \frac{\sum_{j=1}^{|V|} \vec{w}_j \cdot TF(w_j, T)}{\sum_{j=1}^{|V|} TF(w_j, T)}$$

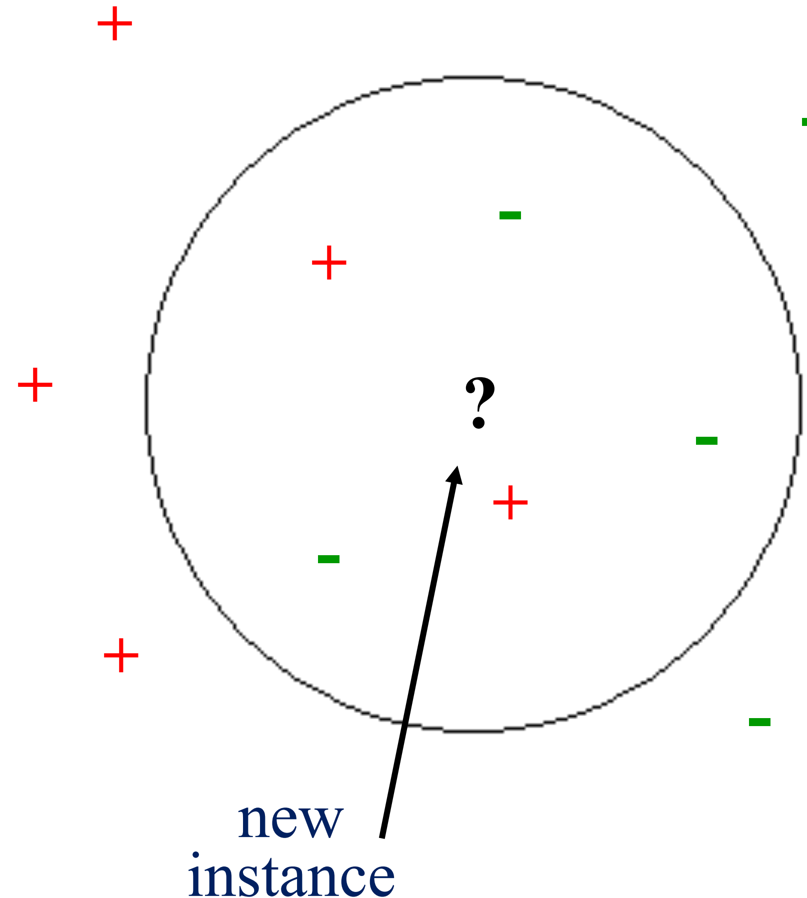
- Or (better) taking into account the ***IDF* scores** of the words:

$$\vec{T} = \frac{\sum_{j=1}^{|V|} \vec{w}_j \cdot TF(w_j, T) \cdot IDF(w_j)}{\sum_{j=1}^{|V|} TF(w_j, T) \cdot IDF(w_j)}$$

- We can classify texts by **classifying their centroids**.
- Better (but more complex) methods to **encode each text as a dense vector** exist (e.g., using **deep neural nets**).

k nearest neighbors (*k*-NN)

- During **training**, simply **store** the vectors of the **training examples**.
- To classify an **unseen instance**, find the **closest *k* training instances** (e.g., $k = 5$) and classify in their **majority class**.
- **Distance weighting** (better): each **neighbor** has a **vote** whose **weight decreases** (e.g., $\propto \frac{1}{d}$ or $\frac{1}{d^2}$) as the **distance** (d) from the instance being classified **increases**.
- In **regression** problems (real-valued responses), return the (weighted) **average** value of the ***k* neighbors**.



Examples of distance measures

- **Euclidian** distance:

$$d(\vec{x}_i, \vec{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + \cdots + (x_{im} - x_{jm})^2}$$

- **Cosine similarity**:

$$sim_{cos}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \cdot \|\vec{y}\|}$$

- For **Boolean** feature vectors, **how many of the features** of the two vectors **are different**:

$$d(\vec{x}_i, \vec{x}_j) = \sum_{r=1}^{m} 1\{x_{ir} \neq x_{jr}\}$$

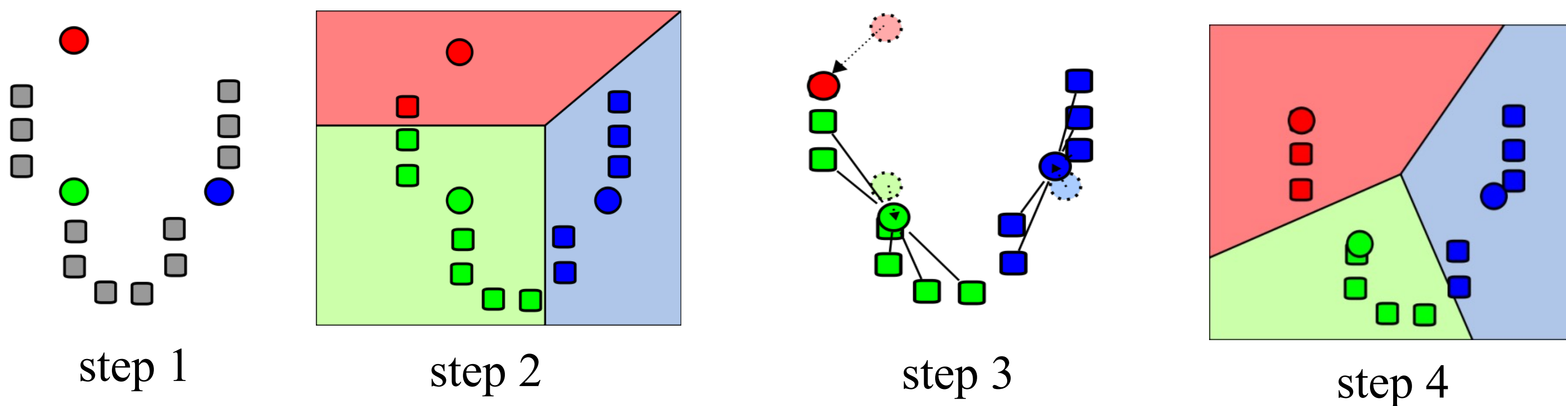
- **Feature (attribute) weighting**: the differences are **weighted** depending on the importance of the features:

$$d(\vec{x}_i, \vec{x}_j) = \sum_{r=1}^{m} IG(C, X_r) \cdot 1\{x_{ir} \neq x_{jr}\}$$

Pros and cons of *k*-NN

- **Computationally cheap** during **training**.
  - We simply store the training instances (lazy).
- But **expensive** in terms of **memory**.
  - We store all the training instances.
- **Computationally expensive** when **classifying**.
  - In the simplest form of *k*-NN, we need to **compute** the **distance** of each **unseen instance** to **every training instance**.
  - There are **approximations** that **reduce this cost** considerably.
  - See http://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces/ and https://github.com/spotify/annoy.
- It can approximate **any separating hyperplane**.
  - Some other algorithms can only learn **linear separators**.
- **Non parametric** learning method:
  - We do not learn the values of a fixed number of parameters.

Clustering with *k*-means



- Start with *k* **random centroids** (one per desired cluster).
- Place **each instance** into the **cluster** with the **closest centroid**.
- **Re-compute** the **centroids**. **Repeat** until convergence.

- **Unsupervised learning**. Can be made **semi-supervised** (how?).
- Produces **hard clusters**, unlike EM's $P(C = i|\vec{X})$.
- Tries to minimize the **sum of the distances** of the **instances** to the **centroids** of their clusters.
- May find a **local minimum**. Sensitive to the initial random centroids. **Restart** several times with **different initial random centroids**.

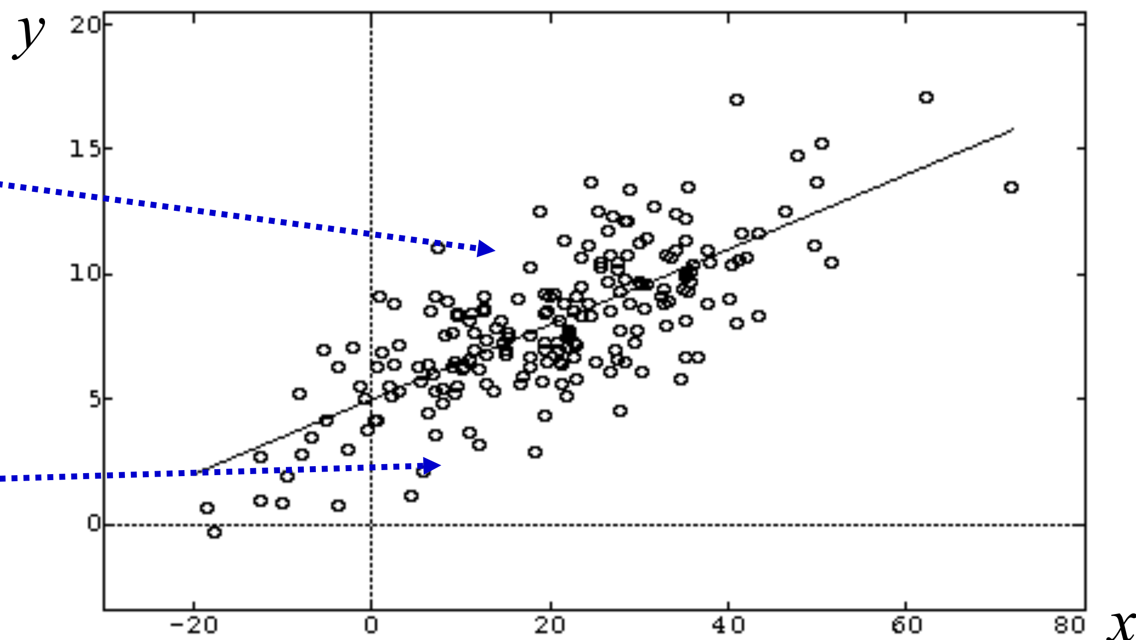
Images from: http://en.wikipedia.org/wiki/K-means_clustering

Clustering documents and/or words

- We can **cluster documents** (e.g., their *TF-IDF* vectors).
  - For example, hoping to get a **view** of their **topics**.
  - More elaborate **topic modeling** methods (e.g., **LDA**) exist.
- We can **cluster word embeddings**.
  - For example, to **replace words** by their **clusters** in **BOW** representations of documents (fewer features).

Linear regression

For points above the line of $f(x)$: $y > w_1 x + w_0$.



For points below the line of $f(x)$: $y < w_1 x + w_0$.

Image source: http://en.wikipedia.org/wiki/Linear_regression

- We try to **learn $f(x)$** from a **sample** (dots).
  - Example: $f(x)$ may show **how fluent** a sentence, which has been assigned a **fluency score x** by a language model, will **actually** be considered by **native speakers** of the language.
- We only consider **linear hypotheses** (functions):

$$y = f_{w_1, w_0}(x) = w_1 x + w_0$$

- Hence, we **search** for the **best w_1, w_0**.

Linear regression – continued

- If we have **two features** x_1, x_2, our linear hypotheses are **planes** in the 3D space:

$$y = f_{w_2, w_1, w_0}(x_1, x_2) = w_2 x_2 + w_1 x_1 + w_0$$

- If we have ***n*** **features** $x_1, \ldots, x_n$, our linear hypotheses are **hyper-planes** in a space of $n + 1$ dimensions:

$$y = f_{w_n, \ldots, w_0}(x_1, \ldots, x_n) = w_n x_n + \ldots w_1 x_1 + w_0$$

$$= \sum_{l=0}^{n} w_l x_l = \langle w_0, w_1, \ldots, w_n \rangle \cdot \langle x_0, x_1, \ldots x_n \rangle$$

$$f_{\vec{w}}(\vec{x}) = \vec{w} \cdot \vec{x} = W^T X$$

For simplicity, always: $x_0 = 1$.

Treating each vector as a single-column matrix.

- We search for the **best** $\vec{w}$.

Squared error loss

- The **search space** contains **all the possible $\vec{w}$**.
- To **assess each $\vec{w}$**, we may use the **squared error loss**:

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

$\vec{x}^{(i)}$ is the ***i*-th** the **training example**,

$y^{(i)}$ is the **correct** (desired) **output** for $\vec{x}^{(i)}$,

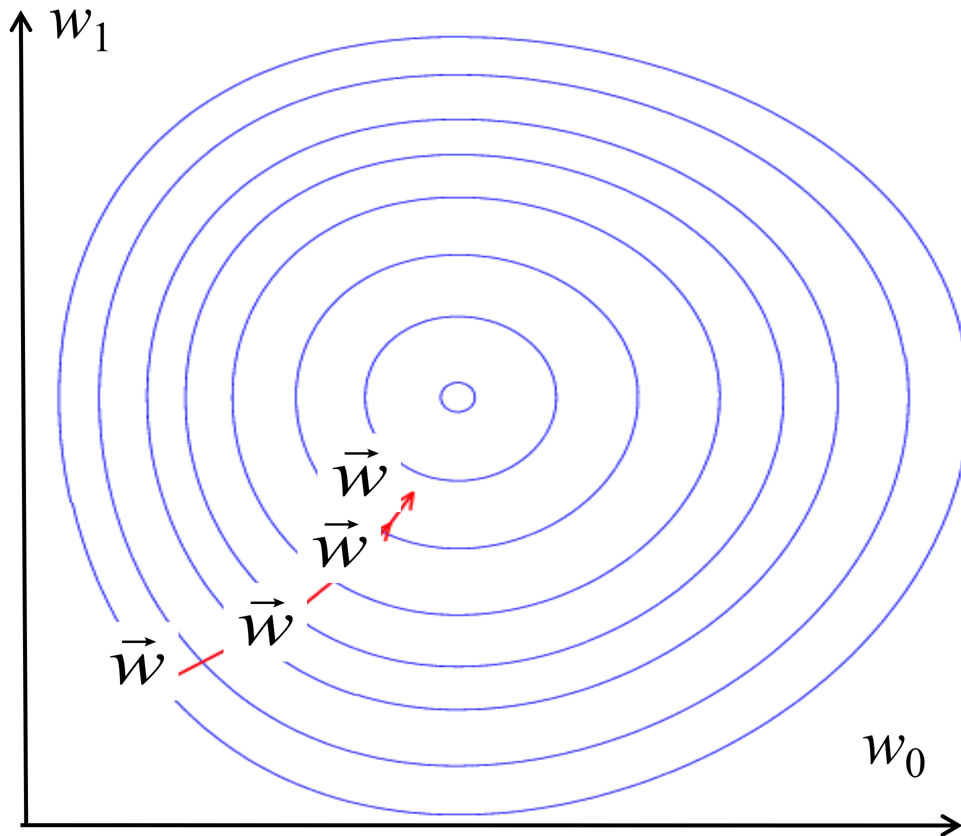
m is the **number** of **training examples**.

- "Least squares linear regression":
  - Regression (not classification), with linear hypotheses.
  - Minimizing the squared error loss.

The search space

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

The curve of $E(\vec{w})$ is **convex**, hence it has **no local minima**.



We **search** for the **weights $\vec{w}$** for which $\boldsymbol{E(\vec{w})}$, i.e., the **total squared error** on the **training data**, is **minimum**.

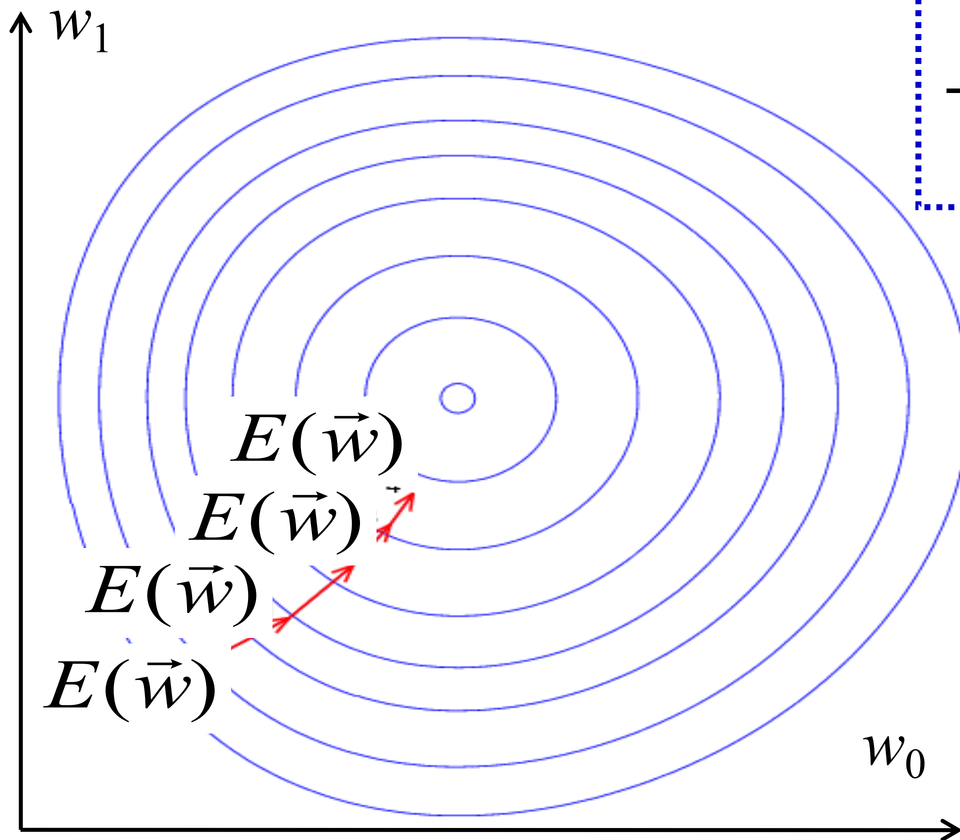
Image source: http://en.wikipedia.org/wiki/Gradient_descent

Gradient descent

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

Start with **random** weights $\vec{w}$. **Compute** $E(\vec{w})$ on the training data. Which way to modify $\vec{w}$?

The **gradient** $\nabla E(\vec{w})$ is a vector showing the **direction** we need to **modify** $\vec{w}$ to obtain the **steepest increase** of $E(\vec{w})$. Hence, $-\nabla E(\vec{w})$ is the direction that leads to the **steepest decrease** of $E(\vec{w})$.



At each iteration, modify $\vec{w}$ by taking **a step** to the direction $-\nabla E(\vec{w})$:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E(\vec{w})$$

In the simplest case, η is a small **positive constant**.

Weights update rule

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

$$\nabla E(\vec{w}) = \left\langle \frac{\partial E(\vec{w})}{\partial w_0}, \frac{\partial E(\vec{w})}{\partial w_1}, \ldots, \frac{\partial E(\vec{w})}{\partial w_l}, \ldots, \frac{\partial E(\vec{w})}{\partial w_n} \right\rangle$$

$$\frac{\partial E(\vec{w})}{\partial w_l} = \sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot \frac{\partial (f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)})}{\partial w_l}$$

$$= \sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot \frac{\partial (\sum_{j=0}^{n} w_j x_j^{(i)} - y^{(i)})}{\partial w_l}$$

Weights update rule – continued

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

$$\nabla E(\vec{w}) = \left\langle \frac{\partial E(\vec{w})}{\partial w_0}, \frac{\partial E(\vec{w})}{\partial w_1}, \ldots, \frac{\partial E(\vec{w})}{\partial w_l}, \ldots, \frac{\partial E(\vec{w})}{\partial w_n} \right\rangle$$

$$\frac{\partial E(\vec{w})}{\partial w_l} = \ldots = \sum_{i=1}^{m}[f_{\vec{w}}(x^{(i)}) - y^{(i)}] \cdot x_l^{(i)}$$

Hence:

$$\nabla E(\vec{w}) = \sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]\left\langle x_1^{(i)}, \ldots x_l^{(i)}, \ldots, x_n^{(i)} \right\rangle$$

$$= \sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot \vec{x}^{(i)}$$

Weights update rule – continued

Hence, the weights update rule:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E(\vec{w})$$

becomes:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \sum_{i=1}^{m} [f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot \vec{x}^{(i)}$$

and each individual weight is updated as follows:

$$w_l \leftarrow w_l - \eta \cdot \sum_{i=1}^{m} [f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot x_l^{(i)}$$

Batch gradient descent
(not used in practice)

1. Start with random weights $\vec{w}$.
2. While $E(\vec{w})$ has not converged (or $\nabla E(\vec{w}) > \varepsilon$):
3. Update the weights:

$$w_l \leftarrow w_l - \eta \cdot \sum_{i=1}^{m} [f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot x_l^{(i)}$$

4. Go to step 2.

In the simplest case, η is a small positive constant. In more elaborate versions, η is adjusted (e.g., decreased) at each iteration.

Stochastic gradient descent (SGD)

1. Start with random weights $\vec{w}$.
2. Shuffle the training instances. Set $i \leftarrow 1$ and $s \leftarrow 0$.
3. Compute $E_i(\vec{w}) = \frac{1}{2}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$ **only for the current** (i-th) **training example**.
4. $s \leftarrow s + E_i(\vec{w})$
5. Update the weights: $\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E_i(\vec{w})$
   i.e., $w_l \leftarrow w_l - \eta \cdot [f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}] \cdot x_l^{(i)}$
6. If a (i+1)-th training example exists, set $i \leftarrow i+1$ and go to step 3.
7. If s has not converged and max number of scans ("epochs") of training data not exceeded, go to step 2.

Obtained by computing the partial derivatives...

Stochastic gradient descent – continued

- **Much smaller computational cost**.
  - **Loss computed** on a **single training example** per step. In practice (esp. with GPUs) on a **mini-batch** of training examples.
- The **update steps** do **not always** go **towards** the **minimum** of the **total error** $E(\vec{w})$. Each step goes **towards** the minimum of the **local error** $E_i(\vec{w})$.
  - With **larger mini-batches** the gradient of $E_i(\vec{w})$ is **closer to that of $E(\vec{w})$**, which often allows using a **larger learning rate**.
- SGD **may not reach** the **exact minimum** of $E(\vec{w})$.
  - It may start **wandering around** the minimum, but in practice, it arrives **close enough**, much faster than batch GD.
  - Optionally see https://en.wikipedia.org/wiki/Stochastic_gradient_descent for improvements (e.g., momentum, AdaGrad, Adam).

Closed-form solution

- A **closed-form solution** to obtain **directly** the **weights** that **minimize** the **squared error loss** also exists.

$$E(\vec{w}) = \frac{1}{2}\sum_{i=1}^{m}[f_{\vec{w}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

- Set $\nabla E(\vec{w}) = \vec{0}$ and solve for $\vec{w}$.
- The **solution** is: $W^* = (X^T X)^{-1} X^T Y$

where:

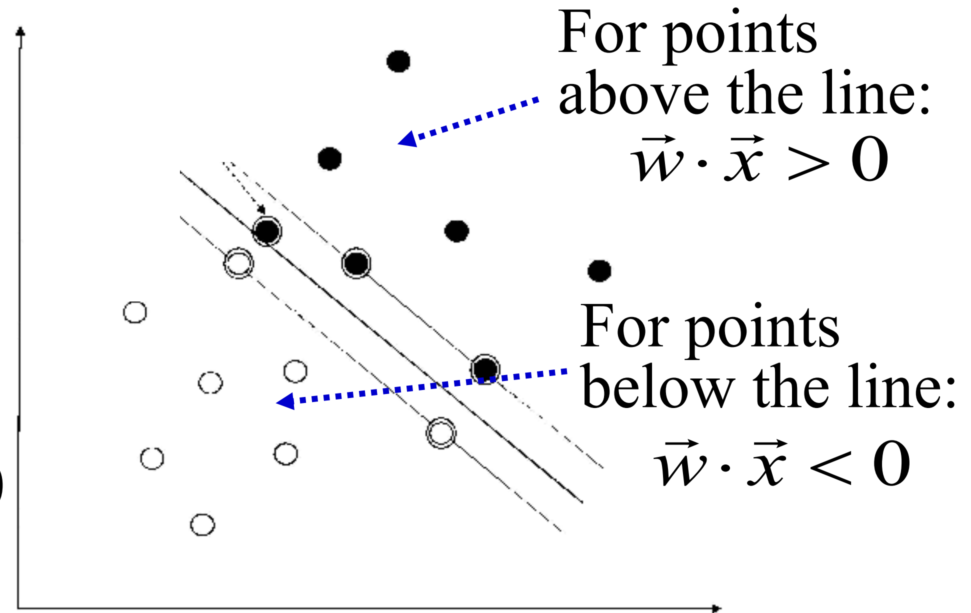
$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

Closed-form solution – continued

- The closed-form solution requires inverting $(X^T X)$.
  - May be **time-consuming** if we have a very large number of training examples and features.
- **Gradient descent** (or SGD) can also be used in **problems** where there is **no closed-form solution**.
  - We will encounter such problems shortly.

Linear separators

- For two features x_1, x_2, we search for a **straight line** that **separates** the two classes.



$$w_2 x_2 + w_1 x_1 + w_0 = 0$$

- For more features x_1, x_2, …, x_n, we search for a **hyper-plane** that separates the two classes.

$$w_n x_n + \ldots + w_1 x_1 + w_0 = \sum_{l=0}^{n} w_l x_l = \vec{w} \cdot \vec{x} = 0$$

Setting again $x_0 = 1$.

- Classification **decision**:

$$C = \mathrm{sign}(\vec{w} \cdot \vec{x})$$

Linear separators– continued

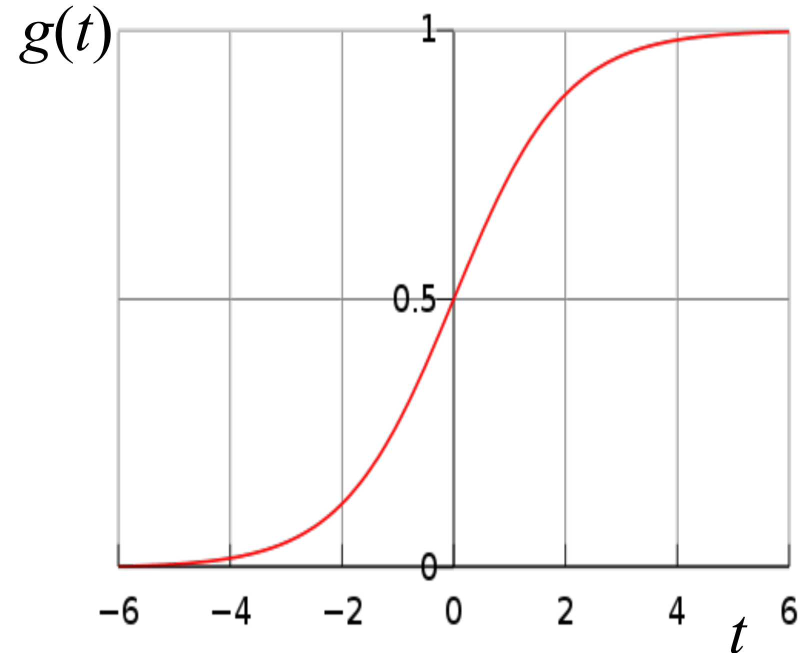
- We often want the classifier to also return a **probability-like certainty score**.
  - How **probable** is it for a **document** with **vector** $\vec{x}$ to belong in the **positive** or **negative** class?
- The **signed distance** $d_{\vec{w}}(\vec{x})$ from the separating **hyper-plane** is **<u>not</u>** a **good** certainty score.

$$d_{\vec{w}}(\vec{x}) = \frac{\vec{w} \cdot \vec{x}}{\|\vec{w}\|}$$

Without w_0.

  - **Not confined** to **[0, 1].**
  - For **large** (positive or negative) **distances**, we want the certainty to **approach 1**.
  - For **small distances**, we want the certainty to **approach 0**.

Sigmoid (logistic) function



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

Image source: http://en.wikipedia.org/wiki/Logistic_function

- In our case, t will be the **unsigned** (and usually unnormalized) **distance** from the separating **hyper-plane**:

$$t = \vec{w} \cdot \vec{x}$$

- **Probability** for $\vec{x}$ to belong in the **positive** class:

$$P(c_+ \mid \vec{x}) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$$

- **Probability** to belong in the **negative** class: $P(c_- \mid \vec{x}) = 1 - P(c_+ \mid \vec{x})$

Logistic regression classifiers

$$P(c_+ \mid \vec{x}) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}, \quad P(c_- \mid \vec{x}) = 1 - P(c_+ \mid \vec{x}) = \frac{e^{-\vec{w}\cdot\vec{x}}}{1+e^{-\vec{w}\cdot\vec{x}}}$$

- During **training**, they **select the $\vec{w}$** that makes the classifier more **confident** that the **training examples** belong in their **correct classes**.
  - They maximize the (conditional) **likelihood** of the examples.

$$L(\vec{w}) = P(y^{(1)}, \ldots, y^{(m)} \mid \vec{x}^{(1)}, \ldots, \vec{x}^{(m)}; \vec{w})$$

The **correct classes** of the training examples.

The **feature vectors** of the training examples.

Maximizing the likelihood

- Assuming that the **training examples** are **independent** and sampled from the **same population**:

$$L(\vec{w}) = P(y^{(1)}, \ldots, y^{(m)} \mid \vec{x}^{(1)}, \ldots, \vec{x}^{(m)}; \vec{w})$$

$$= \prod_{i=1}^{m} P(y^{(i)} \mid \vec{x}^{(i)}; \vec{w})$$

- Instead of maximizing $L(\vec{w})$, easier to maximize the (conditional) **log-likelihood**:

$$l(\vec{w}) = \log L(\vec{w}) = \sum_{i=1}^{m} \log P(y^{(i)} \mid \vec{x}^{(i)}; \vec{w})$$

Maximizing the likelihood – continued

- If we represent the classes as ***y* = 1** (**positive** class) and ***y* = 0** (**negative** class), then:

$$P(y \mid \vec{x}; \vec{w}) = P(c_+ \mid \vec{x}; \vec{w})^y \cdot P(c_- \mid \vec{x}; \vec{w})^{(1-y)}$$

  - For $y = 1$ (positive): $P(y = 1 | \vec{x}; \vec{w}) = P(c_+ | \vec{x}; \vec{w}) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
  - For $y = 0$ (negative): $P(y = 0 | \vec{x}; \vec{w}) = P(c_- | \vec{x}; \vec{w}) = \frac{e^{-\vec{w}\cdot\vec{x}}}{1+e^{-\vec{w}\cdot\vec{x}}}$

- Hence:

$$l(\vec{w}) = \sum_{i=1}^{m} \log P(c_+ \mid \vec{x}^{(i)}; \vec{w})^{y^{(i)}} + \log P(c_- \mid \vec{x}^{(i)}; \vec{w})^{(1-y^{(i)})}$$

$$= \sum_{i=1}^{m} y^{(i)} \log P(c_+ \mid \vec{x}^{(i)}; \vec{w}) + (1 - y^{(i)}) \log P(c_- \mid \vec{x}^{(i)}; \vec{w})$$

For each training example, we **minimize** the **cross-entropy**…

Maximizing the likelihood – continued

- With **batch gradient <u>ascent</u>**:

$$\vec{w} \leftarrow \vec{w} + \eta \cdot \nabla l(\vec{w})$$

We now **maximize** $\boldsymbol{l(\vec{w})}$, instead of minimizing $E(\vec{w})$.

we obtain the **weights update rule**:

$$w_l \leftarrow w_l + \eta \cdot \sum_{i=1}^{m} [y^{(i)} - P(c_+ \mid \vec{x}^{(i)})] \cdot x_l^{(i)}$$

- In practice, we use **stochastic gradient ascent**.
    - Or stochastic gradient descent (**SGD**, or variant), if we **minimize** the **cross-entropy** of each training example.
- **No closed-form** solution.

Regularization (very important)

- Instead of maximizing the log-likelihood alone:

$$l(\vec{w}) = \sum_{i=1}^{m} \log P(y^{(i)} \mid \vec{x}^{(i)}; \vec{w})$$

we usually add a **regularization term**:

$$l(\vec{w}) - \lambda \cdot \|\vec{w}\|^2 = l(\vec{w}) - \lambda \cdot \sum_{l=0}^{n} w_l^2$$

L2 regularization ("ridge regression")

to **reward $\vec{w}$** vectors with many **small weights**.

L1 regularization ("lasso regression") uses the L1 norm, adding $-\lambda \sum_{l=0}^{n} |w_l|$ instead. It leads to sparser $\vec{w}$.

- Lower risk of **over-fitting** the training data:
  - Intuitively, if **many weights are small** (or zero), we **do not rely** much on the **corresponding features**. With fewer features, less likely to over-fit the training data.
  - $\lambda > 0$. Value usually **tuned** on **held-out/development** data.

Important tricks

- For each attribute X_i, assuming normal distribution:

$$X_i \leftarrow \frac{X_i - \mu_i}{\sigma_i}$$

Mean and standard deviation of X_i in the training data.

- Also important: **start** with **random small weights**.
  - E.g., sample them from a **zero-centered Gaussian with small $\boldsymbol{\sigma}$**.
  - See the material of the **Deep Learning course** for alternative/better weight initialization schemes.

Multinomial Logistic Regression

- Extension for **multiple (non-overlapping) classes** $c_1, \ldots, c_K$.
  - Intuitively, we learn a **separate linear separator** for **each class** c_j, with its **own weights** vector $\vec{w}_j$.

probability that $\vec{x}$ belongs in c_j

$$P(c_j \mid \vec{x}) = \frac{e^{\vec{w}_j \cdot \vec{x}}}{\sum_{j'=1}^{K} e^{\vec{w}_{j'} \cdot \vec{x}}}$$

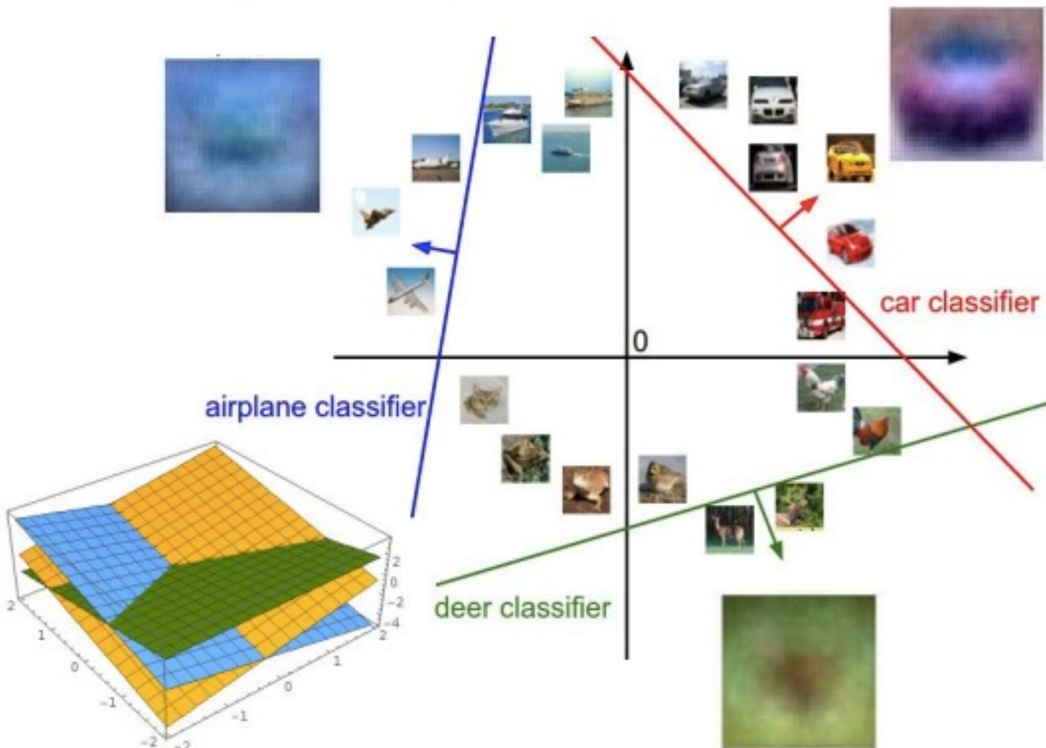
different weights vector per class

normalization factor

- Alternative view: we compute a **score** $\boldsymbol{z_j = \vec{w}_j \cdot \vec{x}}$ for **each class** $\boldsymbol{c_j}$, and we **apply** the **softmax** function $\frac{\exp(z_j)}{\sum_{j'} \exp(z_{j'})}$ to the **scores** to turn them into **probabilities** that sum up to 1.
- We **train** by **maximizing** the conditional **log-likelihood**.
  - Same as **minimizing** the **cross-entropy** of the training examples.

Multinomial Logistic Regression

Interpreting a Linear Classifier



$$f(x,W) = Wx + b$$

Array of **32x32x3** numbers
(3072 numbers total)

Plot created using Wolfram Cloud

Cat image by Nikita is licensed under CC-BY 2.0

From Stanford's course "Convolutional Neural Networks for Visual Recognition", F.F. Li, J. Johnson, S. Yeung, 2017. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture2.pdf

Evaluating classifiers

- **Accuracy** (correct decisions/total decisions) is **not always** a **good evaluation measure**!
  - If we have two classes and one is much more frequent (e.g., 80% of instances), a **majority classifier** that always classifies in the most frequent class will have an accuracy of 80%!
- **Precision of a class**:
  - **How many** of the instances **classified in the class** (true positives + false positives) **are true members** of the class (true positives).

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

- **Recall of a class**:
  - **How many** of the **true members** of a class (true positives + false negatives) are **classified in the class** (true positives).

Evaluating classifiers – continued

- **F-measure**: $$F_\beta = \frac{(\beta^2+1)\cdot \text{Precision}\cdot \text{Recall}}{\beta^2\cdot \text{Precision}+\text{Recall}}$$
  - **Combination** of **precision** and **recall** (weighted harmonic mean).
  - For $\boldsymbol{\beta}$ **= 1**, **equal importance** to **precision** and **recall**. (But the harmonic mean is closer to the min of the two values than the arithmetic mean.)
- **Averaging precision or recall** over ***n*** **classes**:
  - **Macro-averaging** (equal weight assigned to all classes):

$$\text{MacroPrecision} = \frac{1}{n}\sum_{i=1}^{n}\text{Precision}_i \qquad \text{MacroRecall} = \frac{1}{n}\sum_{i=1}^{n}\text{Recall}_i$$

  - **Micro-averaging** (frequent classes treated as more important):

$$\text{MicroPrecision} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + FP_i} \qquad \text{MicroRecall} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + FN_i}$$

Precision-recall diagrams

- In many algorithms, we can **opt for higher precision** at the expense of **lower recall**, or **vice versa** by **tuning** a **threshold**.
  - E.g., in Logistic Regression, classify as spam iff $P(C = 1|\vec{x}) > \boldsymbol{t}$
  - For **different values** of the **threshold** $\boldsymbol{t}$, we obtain **different pairs of precision-recall scores** (on test data).
  - The **larger** the **area under the curve** (**AUC of Precision-Recall** curve, a.k.a. **Average Precision**) the **better** the system. (AP can be slightly different in IR.)
  - For **multiple classes**, we can **average AP** over classes, obtaining **Mean Average Precision** (**MAP**).

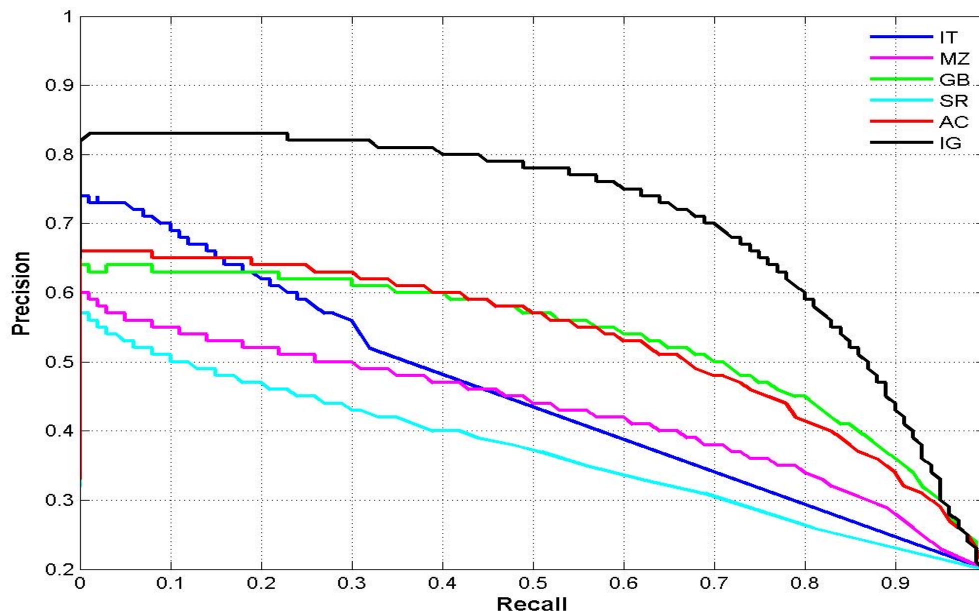


Image source: http://ivrgwww.epfl.ch/supplementary_material/RK_CVPR09/Images/prcurve_1000.jpg

ROC curves

- Instead of Precision-Recall curves, it is also common to plot **Receiver Operating Characteristic** (ROC) curves.
  - **True Positive Rate** = $\frac{TP}{TP+FN}$ = **Sensitivity** = Recall of positive class
  - **False Positive Rate** = $\frac{FP}{FP+TN} = 1 - \frac{TN}{TN+FP}$ = **1 – Specificity** = 1 – Recall of negative class
  - The **larger** the **AUC** (of ROC curve) the **better** the system.

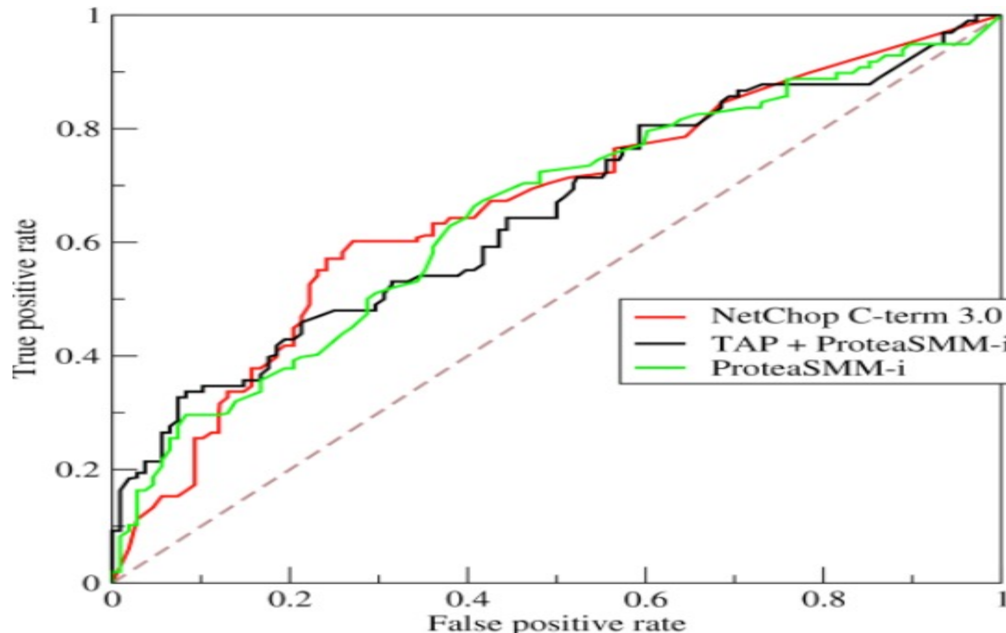
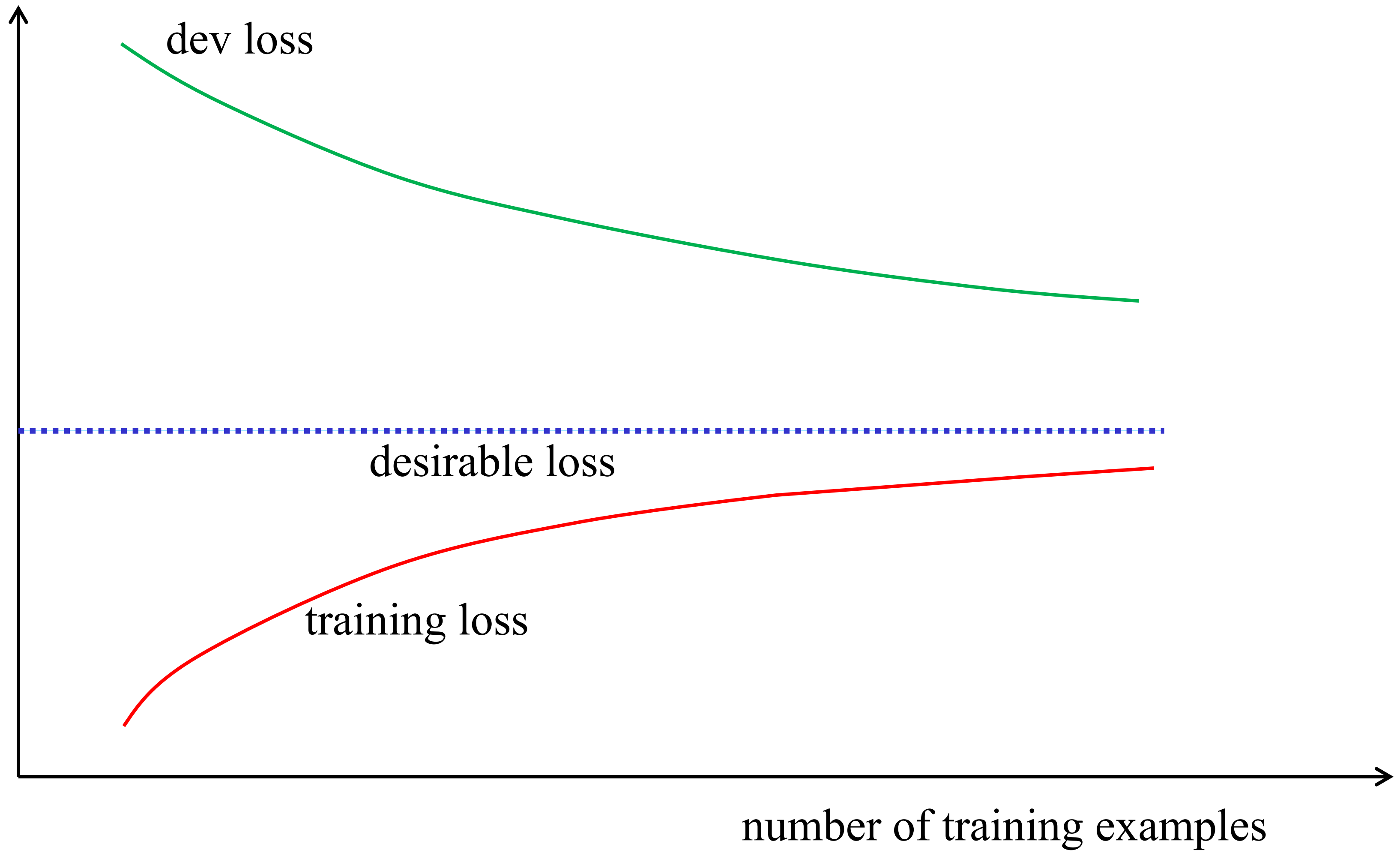


Image source: https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roccurves.png

Advice for using supervised learning
(based on lectures of Andrew Ng)

- With most **supervised learning** algorithms, the **loss on the training data** is **lower** than the loss on the development (or test) data.
  - Training loss: Performance on the **same data** we used for training.
  - Dev loss: Performance on **different data** than the data used for training (but from the same population).
- The training loss is roughly a **lower bound** of the dev loss.
- Comparing the training loss to the dev loss we can also see **how much we overfit** the training data.

Diagnostics: **overfitting**



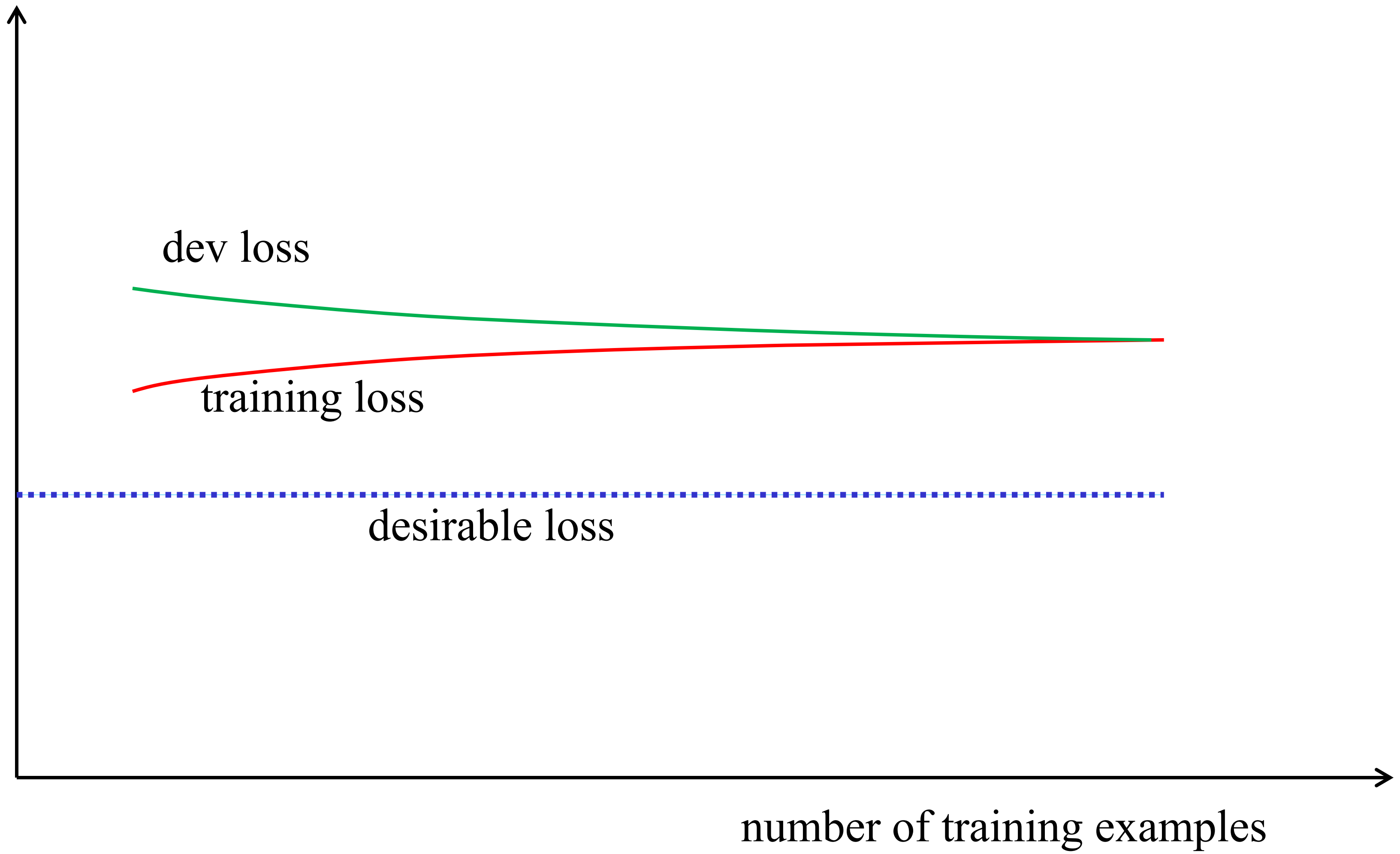
Diagnostics: **overfitting**

- If we **observe** that:
  - The training loss increases (**becomes worse**) **rapidly** as we add more training examples.
  - The dev loss decreases (**becomes better**) **rapidly** as we add more training examples.
  - **Most importantly**: there is a **big difference** between the two.
- The system is probably **overfitting** the training data:
  - It performs much better on the training data than on the dev data, because it learns **peculiarities** of the **training data**.
  - Easier to happen with **few training examples** (and large numbers of features).
  - The **more** the **training examples**, the **more difficult** it becomes for the system to **overfit** them. The system **generalizes better** and, hence, **performs better** on dev data.

Diagnostics: **overfitting**

- What **<u>may</u> help** reduce the overfitting:
  - 👍 **More training data**.
  - 👍 **Fewer** (and better) **features** (e.g., feature selection).
  - 👍 **Simpler models** (e.g., linear classifiers instead of *k*-NN, linear instead of non-liner SVM, simpler neural nets).
  - 👍 **Stronger regularization** (e.g., larger λ in logistic regression). May be better than using simpler models.
- What **<u>may</u> not** be **worth doing**, especially if the **training loss** is **below** the **desirable** performance level:
  - 👎 Spending time to think of (or collect data for) **more features**.
  - 👎 Repeating the experiments with **more complex models** .

Diagnostics: **underfitting**



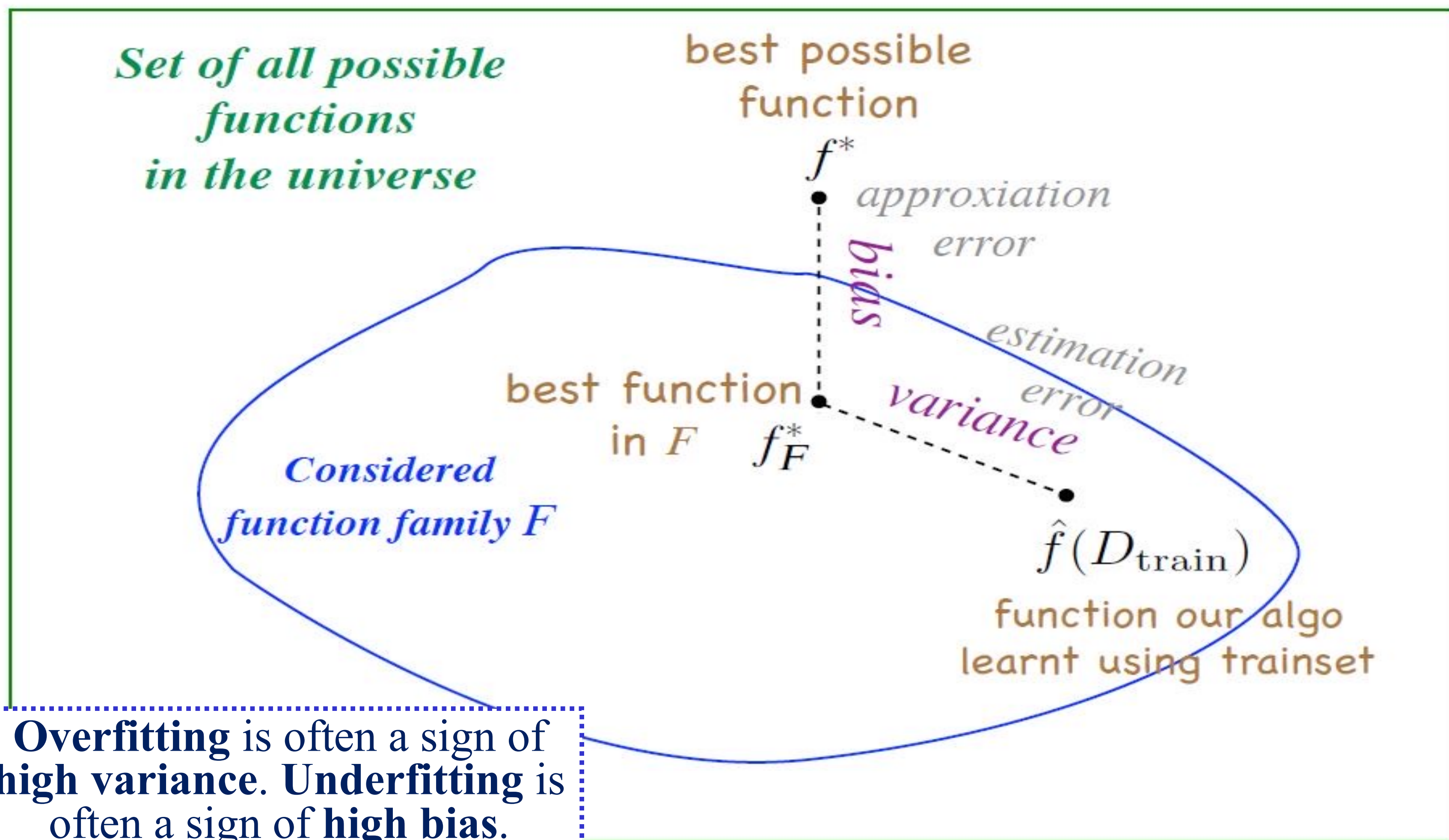
Diagnostics: **underfitting**

- If we **observe** that:
  - The training loss **increases (becomes worse) very slowly** as we add new training examples.
  - The dev loss **decreases (becomes better) very slowly** as we add more training examples.
  - **Most importantly**: **very small difference** between the two, but we are still **above the desired level** of performance.
- It may be that the **search space** is **too narrow**:
  - The **function** we **need to learn** is **not in the family** of functions we are **searching** in.
  - The **search space** does **not** contain **functions** that can **generalize** well enough.

Diagnostics: **underfitting**

- What **may help**:
  - **More (and better) features** (additional information, feature combinations, like ANDs of features in logistic regression).
  - **More complex models** (e.g., non-linear rather than linear SVM, more complex neural net) or **ensembles** of classifiers (e.g., trained with different learning algorithms, or using different kinds of features, or trained on different training subsets).
  - **Weaker regularization** (e.g., smaller λ in logistic regression).
- What will **probably not help**:
  - **More training data** (on its own).
  - **Fewer features** (e.g., feature selection).

Decomposing the generalization error



Overfitting is often a sign of **high variance**. **Underfitting** is often a sign of **high bias**.

From the presentation “Introduction to Machine Learning” of P. Vincent at the Deep Learning Summer School 2015 (http://videolectures.net/deeplearning2015_montreal/).

Additional optional reading slides.

Naive Bayes

- Example: incoming message represented as:

$$\vec{X} = \langle X_1, X_2, \ldots, X_m \rangle = \langle 0, 1, \ldots, 1 \rangle$$

- Classification function:

$$h(\vec{X}) = 1 \text{ , iff } P(C = 1 | \vec{X}) > P(C = 0 | \vec{X})$$

- Using Bayes' rule (for $c = 0$ or $c = 1$):

$$P(C = c \mid \vec{X}) = \frac{P(C = c) \cdot P(\vec{X} \mid C = c)}{P(\vec{X})}$$

We need to estimate the probabilities of all the combinations x_1, x_2, ..., $x_m \mid c$. Too many and many will not occur in the training data.

Conditional independence assumption

- Naive Bayes classifiers assume that $\boldsymbol{X_1, \ldots, X_m}$ are **conditionally independent given the value of C**.
  - Usually not true, but NB text classifiers often **work well**!

$$P(\overrightarrow{X} = \langle x_1, x_2, \ldots, x_m \rangle \mid C = c) =$$

$$P(X_1 = x_1 \wedge X_2 = x_2 \wedge \ldots \wedge X_m = x_m \mid C = c) \approx$$

$$P(X_1 = x_1 \mid C = c) \cdot \ldots \cdot P(X_m = x_m \mid C = c) =$$

$$\prod_{i=1}^{m} P(X_i = x_i \mid C = c)$$

  - This is the **multivariate Naive Bayes**. For **text classification**, the **multinomial Naive Bayes** works **better** (see references).

Naive Bayes classifiers – continued

- Then:

$$P(C=1 \mid \vec{X}) = \frac{P(C=1) \cdot \prod_{i=1}^{m} P(X_i = x_i \mid C=1)}{P(\vec{X})}$$

$$P(C=0 \mid \vec{X}) = \frac{P(C=0) \cdot \prod_{i=1}^{m} P(X_i = x_i \mid C=0)}{P(\vec{X})}$$

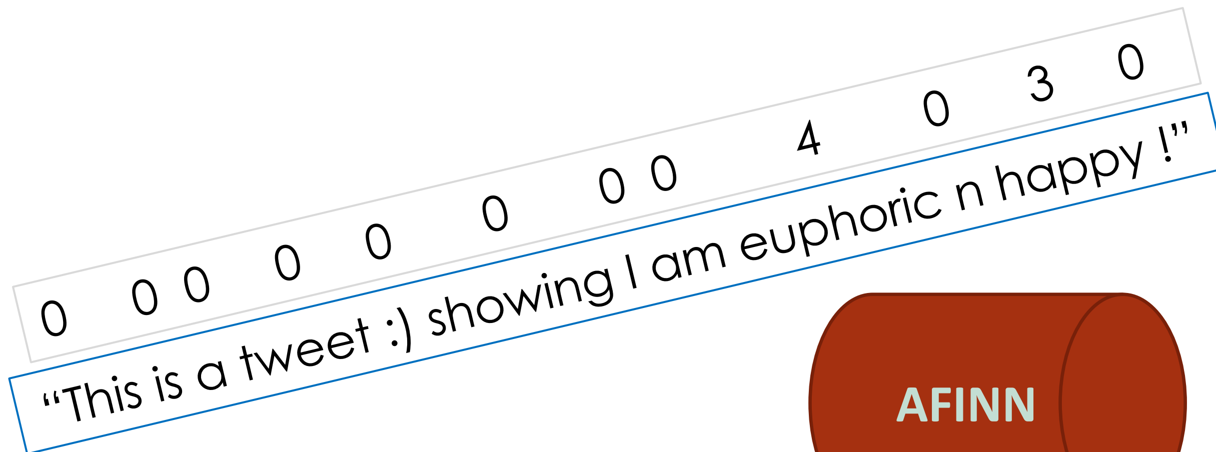
- We discard the **denominators**, because they are the **same**.
- Now all the **probabilities** are **easy to estimate** from the training messages (e.g., using Laplace smoothing).

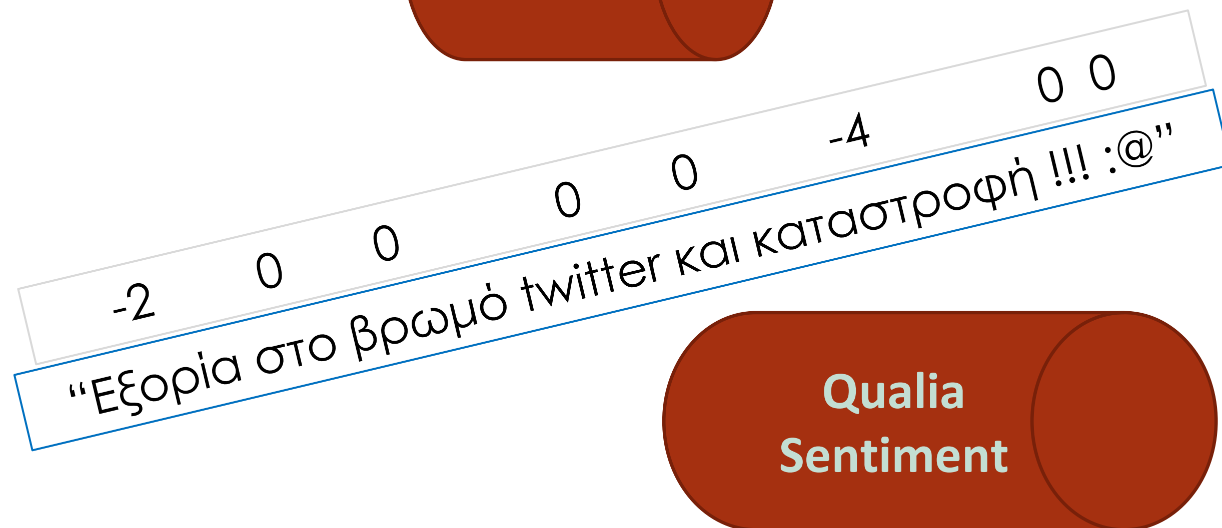
Pros and cons of Naive Bayes

- **Computationally cheap**:
  - For N **training examples**, m **features**,
  - $O(mN)$ steps during **training** to estimate the $P(X_i|C)$ probabilities,
  - $O(m)$ steps at **classification** time to compute the product of $P(X_i|C)$ probabilities.
- **Low memory requirements**:
  - $O(m)$ **counters** needed for the $P(X_i|C)$ estimates.
- Also very **easy to implement**.
- **Often works well**, but **may not** be a **top-performer**.
  - Often used in **spam filters**. Very good results in **sentiment** detection with **bigram** features.

Lexicon features for sentiment analysis

Optional reading





Images and feature examples (next slide) from the MSc thesis and presentation of M. Karampatsis "Social media sentiment analysis", AUEB, 2014 (http://nlp.cs.aueb.gr/theses.html).

Lexicon features for sentiment analysis

- A **simplistic** approach to use them for **entire messages**:
  – **Max**, **min**, **avg score** of the lexicon's **words** in the message.
  – **Sum of positive (negative) scores** of words in the message.
- How to **construct** our own **sentiment lexicons**?
  – Collect **frequent words** from a corpus (e.g., tweets, reviews).
  – Treat **each word** as a **mini-classifier**.
  – **Scores** per **word**: its **precision**, **recall**, **F1 per class** (e.g., positive, negative), computed on **manually classified reviews**.
  – Also works with ***n*-grams** (bigrams are good for sentiment).
  – See J&M for other approaches (e.g., label propagation).
- Another trick for sentiment analysis: **prepend NOT_** to **words after a negation** (up to next punctuation).
  - "didn't like it, but…" → "didn't NOT_like NOT_it, but…"

Searching for feature subsets

Optional reading

- So far **we evaluated each feature separately**.
  - With *IG* or similar measures (e.g., χ^2).
- But **two features** may convey almost the **same information**.
  - It may be the case that "money" and "rich" always **co-occur**.
  - They may both have **high *IG* scores**, but one is **redundant**.
- Or two (or more) features may be **bad on their own** (low *IG* scores), but their **combination** may be a **good predictor**.
- We can treat the problem as a **state space search**:
  - E.g., **initial state**: use all (or none of) the candidate features.
  - **Transition operator**: remove (or add) a candidate feature.
  - **Loss function**: error rate (on held out/development data) when the feature subset of the particular state is used.
- We can use **heuristic search algorithms**:
  - E.g., hill climbing, beam search, simulated annealing, GAs, …

Semi-supervised classification

Optional reading

- We often have **training examples** that are **labeled** with the **correct answers**, and **others** that are **unlabeled**.
  - E.g., manually separated spam/ham messages or positive/negative/neutral tweets and many more unlabeled.
- We can **train** a classifier **on the labeled** examples (L).
  - E.g., train a Naive Bayes classifier.
- We can then (try to) **improve the classifier** using the **unlabeled examples** (U) with **labels generated** by the **previously trained** form of the **classifier**.
  - This is a form of **Expectation Maximization (EM)**.
  - It **does not always work** well in practice. We may end up learning **different classes** than the ones we intended to.

Semi-supervised learning with NB

Optional reading

- **Initial training** of a **Naive Bayes** classifier, with **Boolean features** and **|*C*| classes**, on the **labeled examples (*L*)**.

$$P(C=i) \approx \frac{1+\sum_{j=1}^{|L|} p_{i,j}}{|C|+|L|}$$

1 if the *j*-th labeled example belongs in class *i*; otherwise 0.

Number of labeled examples.

Laplace smoothing.

How many labeled examples of class *i* have $X_l = x_l$?

$$P(X_l = x_l \mid C=i) \approx \frac{1+\sum_{j=1}^{|L|} p_{i,j} \cdot 1\{X_{j,l} = x_l\}}{2+\sum_{j=1}^{|L|} p_{i,j}}$$

How many labeled examples of class *i* do we have?

Semi-supervised learning with NB

Optional reading

- **E-step:** Predict the **classes of the unlabeled examples**, using the **previously trained classifier** (its previously learned parameters $P(C = i)$ and $P(X_l = x_l \mid C = i)$).
  - **Each unlabeled** example is assigned a **probabilistic label** (a probability distribution over the classes).

Predicted probability that the unlabeled example belongs in class *i*. We get a prediction for every class.

We have previously learned these parameters from the labeled training examples.

$$p_{i,j} = P(C = i \mid \vec{X} = \langle x_1, \ldots, x_m \rangle) = \frac{P(C = i) \cdot \prod_{l=1}^{m} P(X_l = x_l \mid C = i)}{P(\vec{X} = \langle x_1, \ldots, x_m \rangle)}$$

$$\sum_{i'=1}^{k} P(C = i') \prod_{l=1}^{m} P(X_l = x_l | C = i')$$

The $p_{i,j}$ values of the labeled examples do not change. They remain 0 or 1.

Semi-supervised learning with NB

Optional reading

- **M-step: Re-train** the classifier (re-learn its parameters), now using **both** the **labeled** and the **unlabeled** examples.
  - Re-learn the $P(C = i)$ and $P(X_l = x_l \mid C = i)$ parameters.

$$P(C=i) \approx \frac{1 + \sum_{j=1}^{|L|+|U|} p_{i,j}}{|C| + |L| + |U|}$$

Count the j-th example to the extent that we believe it belongs in class i.

Count both the labeled and unlabeled examples.

$$P(X_l = x_l \mid C = i) \approx \frac{1 + \sum_{j=1}^{|L|+|U|} p_{i,j} \cdot 1\{X_{j,l} = x_l\}}{2 + \sum_{j=1}^{|L|+|U|} p_{i,j}}$$

How many examples of class i have $X_l = x_l$? Count each example to the extent we believe it belongs in class i.

How many examples of class i are there? Count each example to the extent we believe it belongs in class i.

- Repeat E, M until convergence.
  - It's a form of hill climbing to maximize the likelihood of the data…

The Newton-Raphson method

Optional reading

- Finds the **roots** (zeroes) of **differentiable functions**.
- Find x such that:

$$g(x) = 0$$

- At each iteration:

$$g'(x_n) = \frac{\Delta y}{\Delta x} = \frac{g(x_n) - 0}{x_n - x_{n+1}}$$

- Hence, the **update rule** is: $x_{n+1} \leftarrow x_n - \frac{g(x_n)}{g'(x_n)}$
- Works well and fast, if we start near a root…

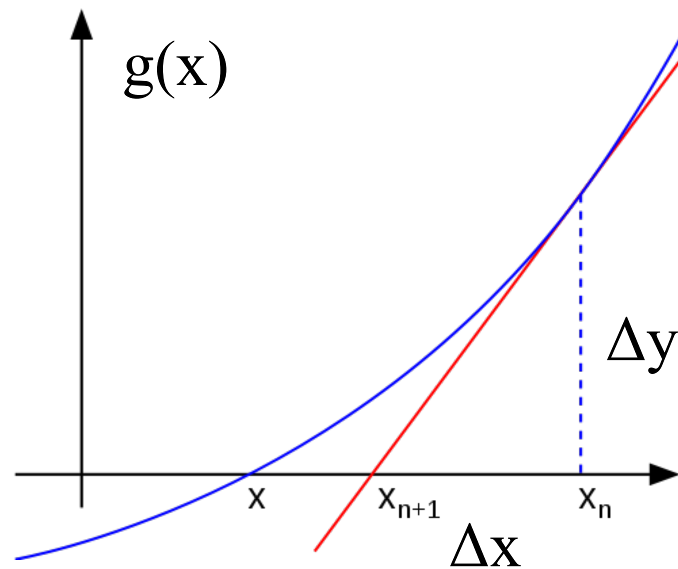


Image source: http://en.wikipedia.org/wiki/Newton's_method

The Newton-Raphson method – continued

Optional reading

- In our case, we want:

$$\vec{g}(\vec{w}) = \nabla E(\vec{w}) = \vec{0}$$

- The **update rule** becomes:

$$\vec{w}_{n+1} \leftarrow \vec{w}_n - H_E^{-1}(\vec{w}_n) \nabla E(\vec{w})$$

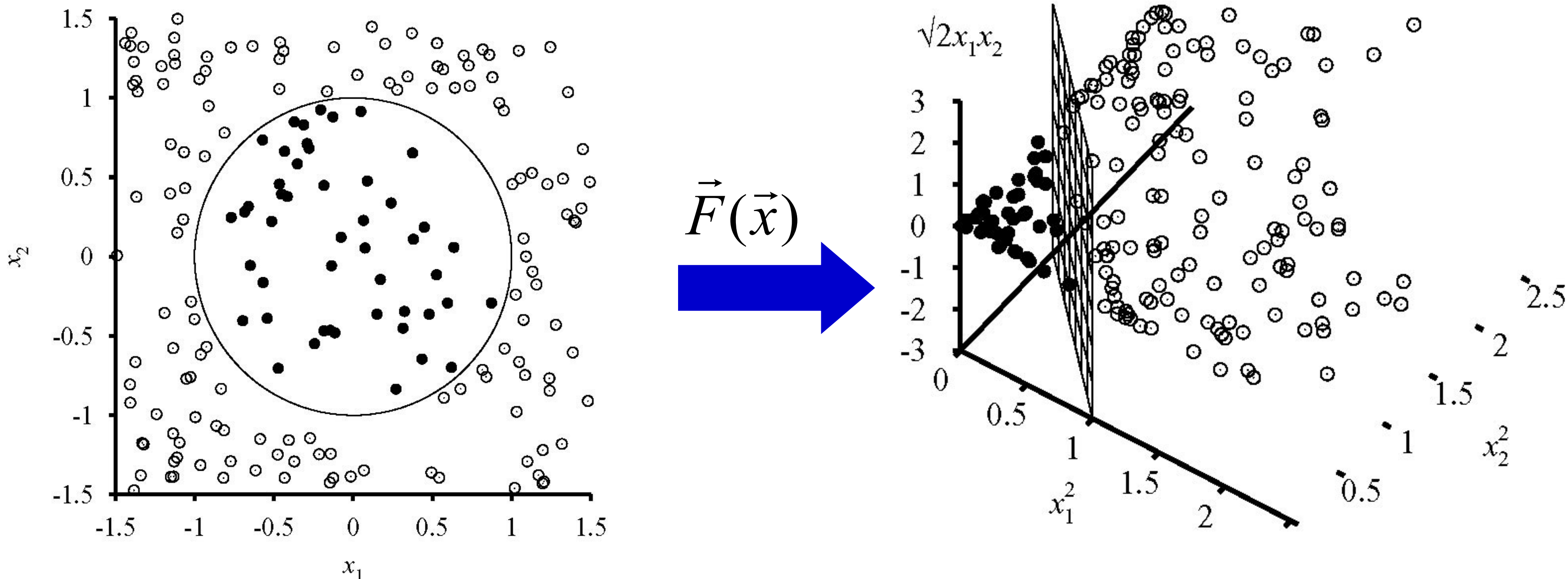
  where the matrix H_E (**Hessian matrix**) contains as elements $h_{i,j}$ all the partial **second derivatives**:

$$\frac{\partial^2 E(\vec{w})}{\partial w_i \partial w_j}$$

- **Works well**, if we **start near** the **minimum**.
- **Impractical for neural nets**, where the weights are billions (billions x billions 2nd derivatives)

Support Vector Machines (SVMs)

Optional reading

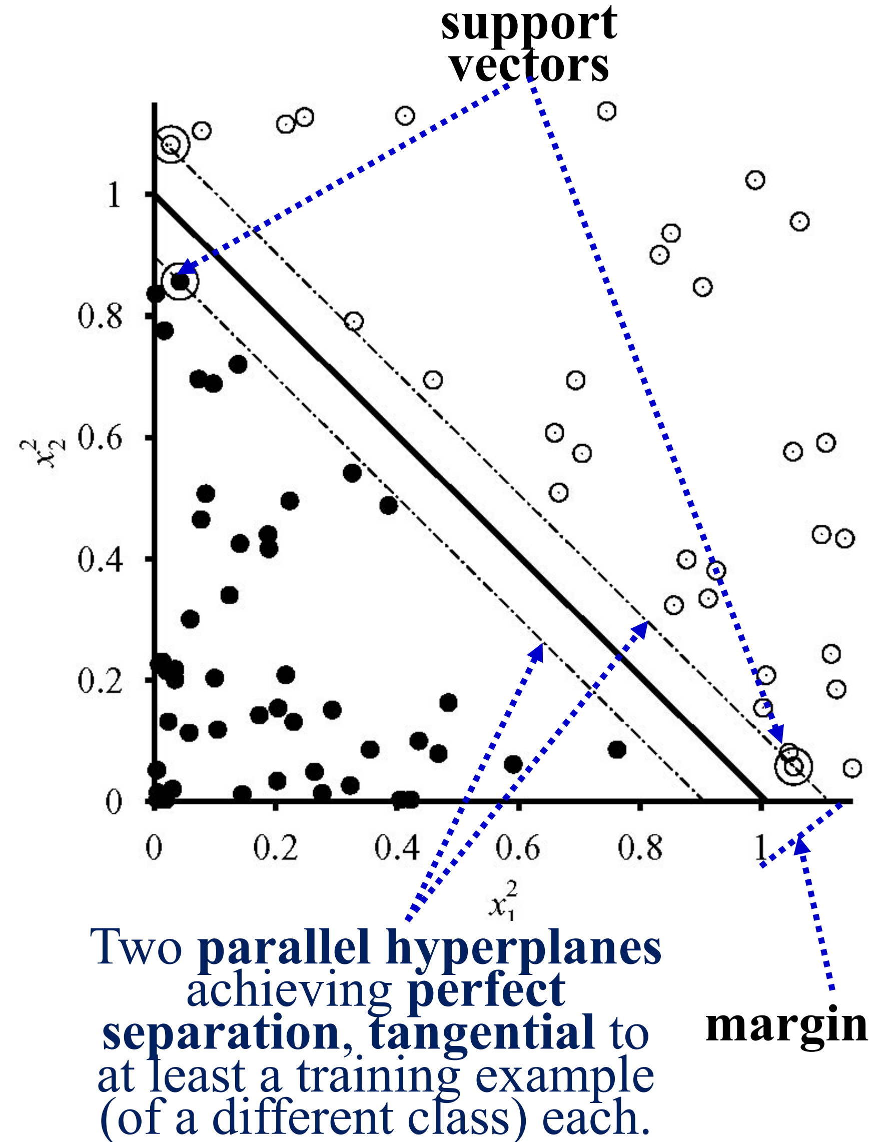


- With an appropriate **transformation**, an originally non-linearly separable dataset may become **linearly separable**.
  - In general, this is always possible, provided that we move to a vector space with sufficiently **large dimensionality**.
  - In the example above: $\vec{F}(\vec{x}) = \left\langle x_1^2, x_2^2, \sqrt{2}x_1x_2 \right\rangle$

Support Vector Machines

Optional reading

- SVMs search in the **new vector space** for a **hyperplane** that separates the examples, with the **maximum margin**.
  - The **separating hyperplane** (bold line) is in the **middle** of the **margin** (space between the two **tangential hyperplanes**).
  - **Maximizing** the **margin** leads to **better generalization** over the entire population.
  - **Support vectors** (definition to be revised): the vectors of the examples lying on the two **tangential hyperplanes**.



Support Vector Machines

Optional reading

- Equation of the separating hyperplane:

$$\vec{w} \cdot \vec{F}(\vec{x}) + b = 0$$

- For simplicity, we require the tangential hyperplanes to have the following equations:

$$\vec{w} \cdot \vec{F}(\vec{x}) + b = \pm 1$$

(Easy to rescale $\vec{w}, b$ to obtain ± 1.)

- Then the margin is: $2/\|\vec{w}\|$
- Minimization problem:

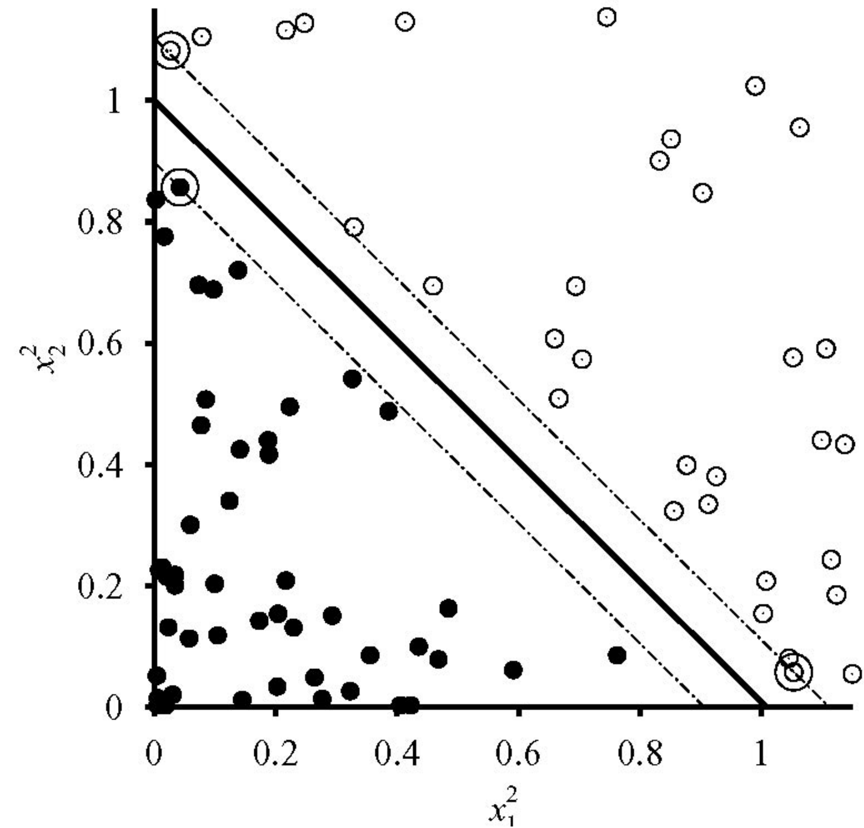
$$\min_{\vec{w},b} \frac{1}{2}\|\vec{w}\|^2$$

such that:

$$(\vec{w} \cdot \vec{F}(\vec{x}_j) + b) \cdot y_j \geq 1$$

a training example

its correct class (here +/-1)



We require all the training examples to be on the correct sides and outside the margin.

Optional reading

Relaxed problem

- We allow an error (**slack**) ξ_j at **each** training example $\vec{x}_j$.
- **Relaxed** problem:

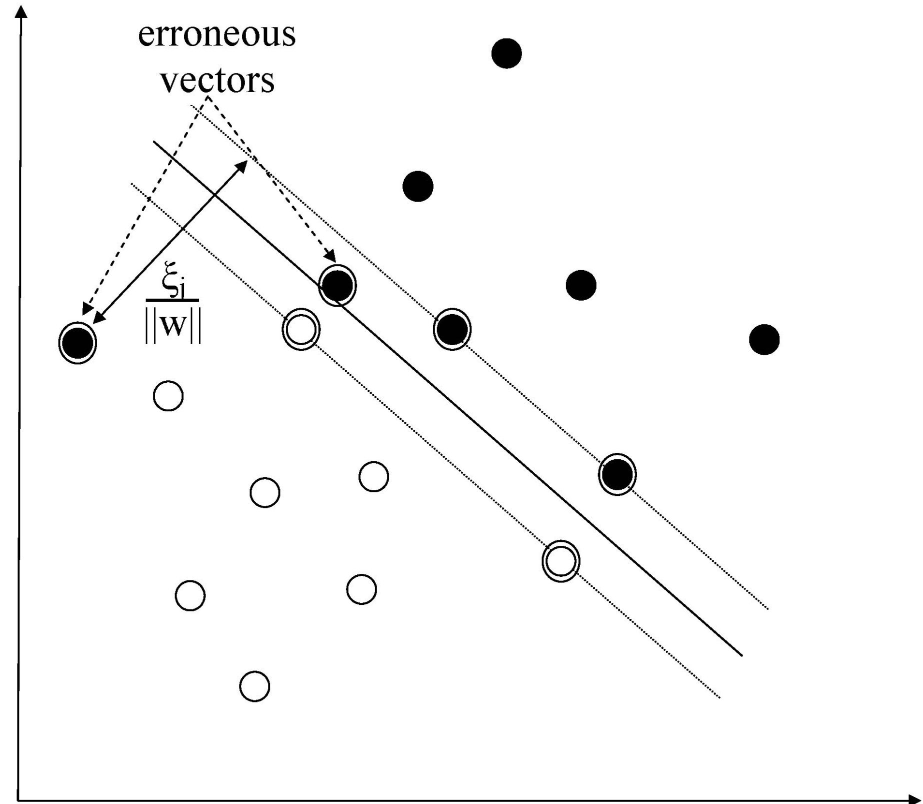
$$\min_{\vec{w},b,\vec{\xi}} \frac{1}{2}\|\vec{w}\|^2 + C \cdot \sum_j \xi_j$$

s.t:

$$(\vec{w} \cdot \vec{F}(\vec{x}_j) + b) \cdot y_j \geq 1 - \xi_j$$

$$\xi_j \geq 0$$

- We **optimize** the **sum** of the **margin** and the **total slack**.
  - The constant C is **tuned** on held-out data. It controls the **trade-off** between large **margin** and low **total slack**.



By allowing an error (slack) per training example, we may obtain a **larger margin**, and we may also find a separating hyperplane even when the training data are **not linearly separable** (e.g., when using a linear SVM).

Optional reading

Support Vectors

- Solving the optimization problem leads to:

$$\vec{w} = \sum_j a_j y_j \vec{F}(\vec{x}_j)$$

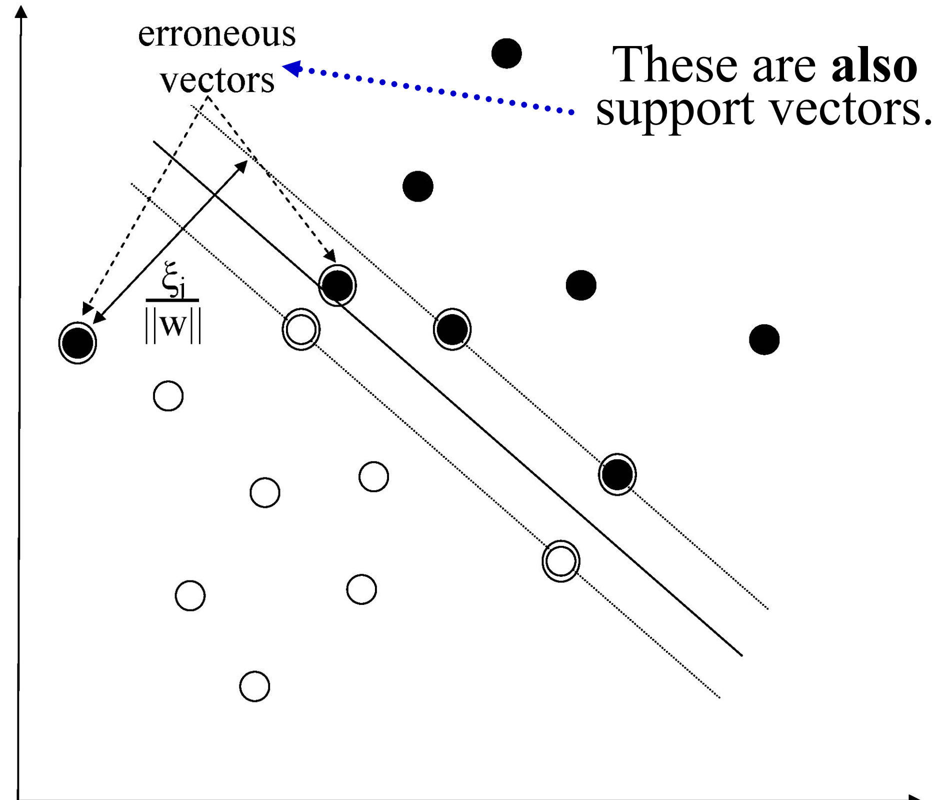
where $\boldsymbol{\alpha_j \neq 0}$ **only if** (⇒) $\boldsymbol{\vec{x}_j}$ **is a support vector**.

- Hence, the separating hyperplane is:

$$\vec{w} \cdot \vec{F}(\vec{x}) + b = 0$$

or:

$$\sum_j a_j y_j \vec{F}(\vec{x}_j) \cdot \vec{F}(\vec{x}) + b = 0$$



Training instances that are **not support vectors** are **ignored** at **classification time** (unlike *k*-NN) because they have $\boldsymbol{\alpha_j = 0}$.

Optional reading

Using kernels in SVMs

- It turns out that the **transformation function $\vec{F}$** is used **only** in **inner products** in the **new vector space.**

$$\vec{F}(\vec{x}_i) \cdot \vec{F}(\vec{x}_j)$$

- **For some functions $\vec{F}$** we can **compute** the **inner products** **without** first **computing** the **transformed vectors** $\vec{F}(\vec{x}_i), \vec{F}(\vec{x}_j)$.
  - In the initial example: $\vec{F}(\vec{x}) = \left\langle x_1^2, x_2^2, \sqrt{2}x_1 x_2 \right\rangle$

    $$\vec{F}(\vec{x}_i) \cdot \vec{F}(\vec{x}_j) = \cdots = (\vec{x}_i \cdot \vec{x}_j)^2$$
  - **No need** to compute the **transformed feature vectors** ($x_{i,1}^2$, $x_{i,2}^2$, $\sqrt{2}x_{i,1}x_{i,2}$, $x_{j,1}^2$, $x_{j,2}^2$, $\sqrt{2}x_{j,1}x_{j,2}$). We only **need** the values of the **original vectors**.
  - The **new vector space** usually has **many more** (possibly infinite) **features** (dimensions) than the original space.

Other kernel example

Optional reading

- Let:

If we have Boolean initial features (0 or 1), in effect the transformation constructs all the ANDs of the initial features.

$$\vec{F}(\vec{x}_i) = \vec{F}(\langle x_{i,1}, \ldots, x_{i,n} \rangle) =$$
$$< x_{i,1}x_{i,1}, x_{i,1}x_{i,2}, \ldots, x_{i,1}x_{i,n},$$
$$x_{i,2}x_{i,1}, x_{i,2}x_{i,2}, \ldots x_{i,2}x_{i,n}, \ldots x_{i,n}x_{i,n} >$$

- Then:

We need O(n^2) time to construct each transformed feature vector.

$$\vec{F}(\vec{x}_i) \cdot \vec{F}(\vec{x}_j) =$$
$$< x_{i,1}x_{i,1}, x_{i,1}x_{i,2}, \ldots, x_{i,1}x_{i,n},$$
$$x_{i,2}x_{i,1}, x_{i,2}x_{i,2}, \ldots x_{i,2}x_{i,n}, \ldots x_{i,n}x_{i,n} > \cdot$$
$$< x_{j,1}x_{j,1}, x_{j,1}x_{j,2}, \ldots, x_{j,1}x_{j,n},$$
$$x_{j,2}x_{j,1}, x_{j,2}x_{j,2}, \ldots x_{j,2}x_{j,n}, \ldots x_{j,n}x_{j,n} >$$
$$= \ldots = (\vec{x}_i \cdot \vec{x}_j)^2$$

We only need O(n) time to compute this!

Another kernel example

Optional reading

Now we also use both the initial features and their ANDs.

- If:

$$\vec{F}(\vec{x}_i) = \vec{F}(\langle x_{i,1}, \ldots, x_{i,n} \rangle) =$$

$$< x_{i,1}x_{i,1}, x_{i,1}x_{i,2}, \ldots, x_{i,1}x_{i,n},$$

$$x_{i,2}x_{i,1}, x_{i,2}x_{i,2}, \ldots x_{i,2}x_{i,n}, \ldots x_{i,n}x_{i,n},$$

$$\sqrt{2c} \cdot x_{i,1}, \sqrt{2c} \cdot x_{i,2}, \ldots, \sqrt{2c} \cdot x_{i,n}, c >$$

- Then:

$$\vec{F}(\vec{x}_i) \cdot \vec{F}(\vec{x}_j) = \ldots = (c + \vec{x}_i \cdot \vec{x}_j)^2$$

We need only O(n) time to compute this!

Using kernels in SVMs

Optional reading

- **Kernel**:
  - A function $K(\vec{x}_i, \vec{x}_j)$ that **computes** the **inner product** $\vec{F}(\vec{x}_i) \cdot \vec{F}(\vec{x}_j)$ in some **new vector space**, where a transformation F takes us. **No need to actually know *F*!**
  - E.g., $K(\vec{x}_i, \vec{x}_j) = (c + \vec{x}_i \cdot \vec{x}_j)^d$ (generalized polynomial kernel)
  - ***Mercer's theorem*** *specifies when a function (in effect, a similarity measure)* $K(\vec{x}_i, \vec{x}_j)$ *is indeed a kernel (see references).*
  - There are also **kernels** that compute the similarity between two texts by considering their **parse trees**, instead of BOW vectors.
- The **equation** of the **separating hyperplane** becomes:

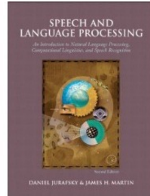
$$\sum_j a_j y_j K(\vec{x}_j, \vec{x}) + b = 0$$

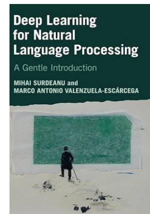
$$h(\vec{x}) = sign(\sum_j a_j y_j K(\vec{x}_j, \vec{x}) + b)$$

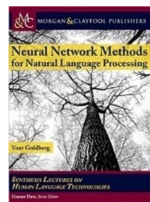
Training examples that are ***not*** support vectors are ignored, because their $\alpha_j = 0$. **Saving memory** and **time during classification**, unlike k-NN. But SVMs are much **slower** to **train**.

Recommended reading

- J&M (2nd ed.): Sections 6.6, 6.7, 20.2.2 (NB only).
  - MaxEnt classifiers are a variant of multinomial logistic regression with Boolean class-sensitive features.
  - See also the free draft of the 3rd edition: http://web.stanford.edu/~jurafsky/slp3/ (chapters 4, 5, 6, 25 – we'll discuss word2vec later in the course)
- S&V-A: Chapters 3–4 (LR only, Perceptron covered later).
- Goldberg: Chapter 2.
- Consult also the notes "Linear regression, classification and logistic regression, generalized linear models" of Andrew Ng at Stanford (pp. 1–7 and 16–21).
  - http://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf

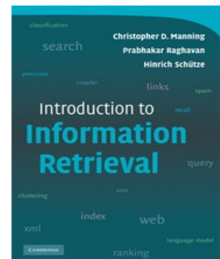






Recommended reading – continued

- There are many versions of Naive Bayes classifiers.
  - The version we considered (optional slides) uses Boolean features and is known as **multivariate Bernoulli Naive Bayes**.
  - The **multinomial Naive Bayes version** can also consider the **term frequencies** of the words in each text, and often **performs better in text classification**.
  - See: http://www.aueb.gr/users/ion/docs/ceas2006_paper.pdf
- For more information about Information Gain (also called Mutual Information, not to be confused with PMI), Naive Bayes, and SVMs consult "An introduction to Information Retrieval" by C.D. Manning, P. Raghavan and H. Schütze, Cambridge University Press, 2008.
  - Chapters 13 and 15.
  - Book freely available from: http://nlp.stanford.edu/IR-book/information-retrieval-book.html



Recommended reading – continued

- For an introduction to Machine Learning, see also "A Course in Machine Learning" by Hal Daumé III.
  - Freely available at http://ciml.info/.
- For more information about PCA (and SVD) see section 12.2 of K.P. Murphy's book "Machine Learning – A Probabilistic Perspective", MIT Press, 2012.
  - Available at AUEB's library.
  - Free draft of 2021 edition: https://probml.github.io/pml-book/book1.html