Statistical Natural Language Processing

Classification

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2018

When/why do we do classification

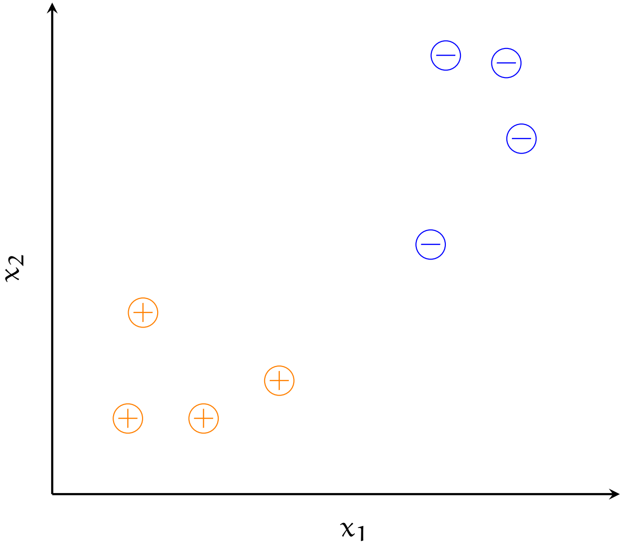
- Is a given email spam or not?
- What is the gender of the author of a document?
- Is a product review positive or negative?
- Who is the author of a document?
- What is the subject of an article?
- …

When/why do we do classification

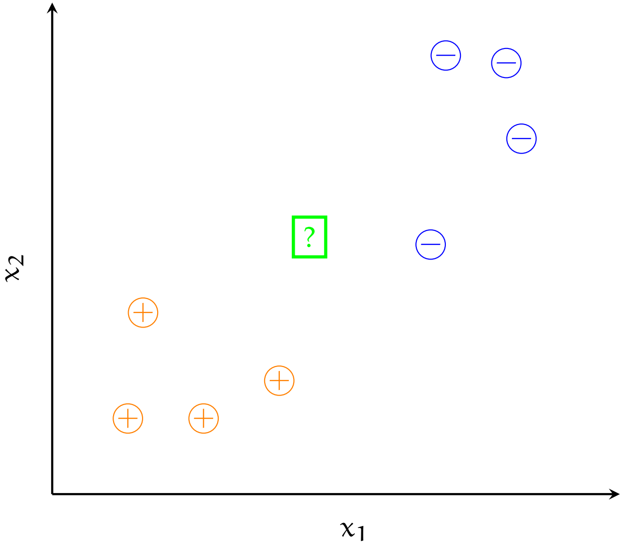
- Is a given email spam or not?
- What is the gender of the author of a document?
- Is a product review positive or negative?
- Who is the author of a document?
- What is the subject of an article?
- …

As opposed to regression the outcome is a ‘category’.

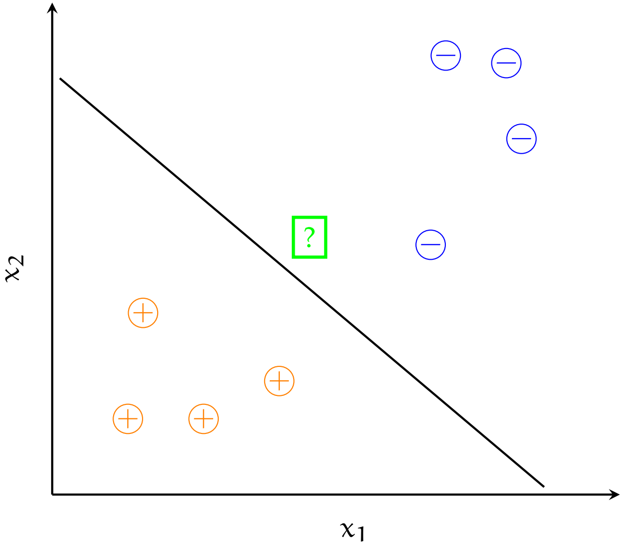
The task



The task

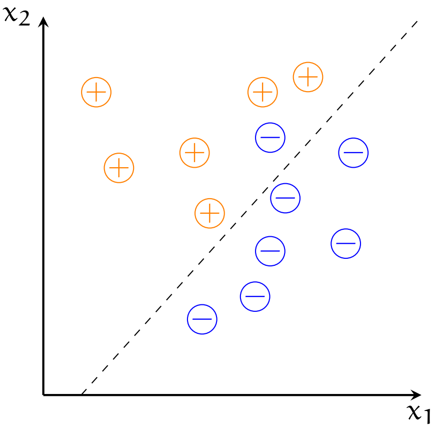


The task



A quick survey of some solutions

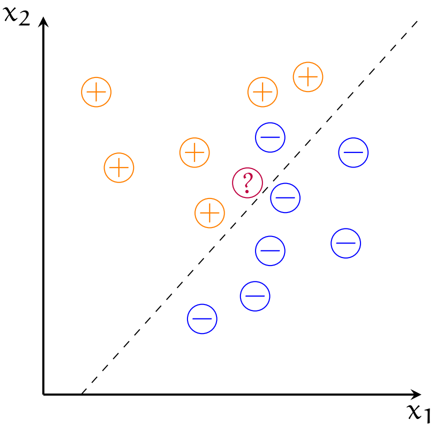
(Linear) discriminant functions



- Find a discriminant function (f) that separates the training instance best (for a definition of 'best')

A quick survey of some solutions

(Linear) discriminant functions

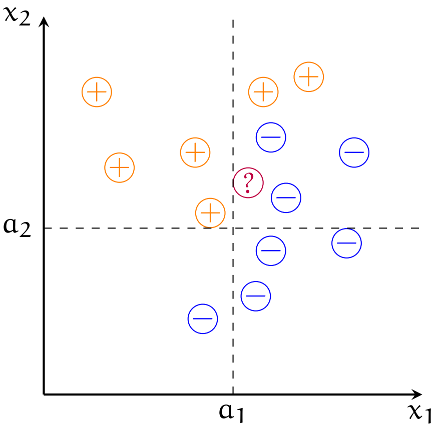


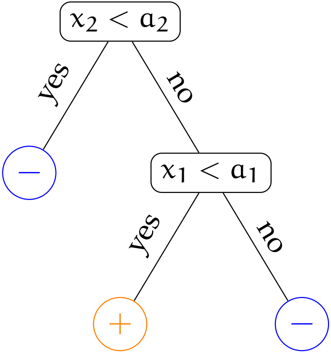
- Find a discriminant function (f) that separates the training instance best (for a definition of 'best')
- Use the discriminant to predict the label of unknown instances

$$\hat{y} = \begin{cases} \oplus & f(\mathbf{x}) > 0 \\ \ominus & f(\mathbf{x}) < 0 \end{cases}$$

A quick survey of some solutions

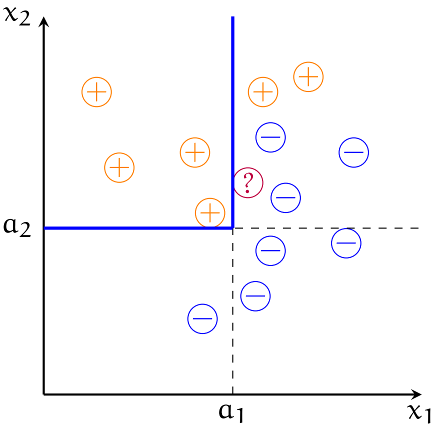
Decision trees

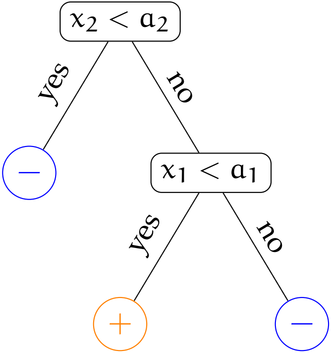




A quick survey of some solutions

Decision trees

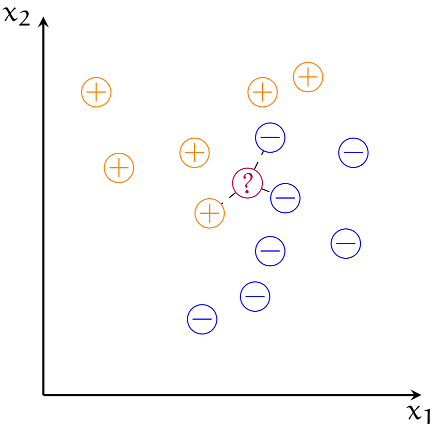




- Note that the decision boundary is non-linear

A quick survey of some solutions

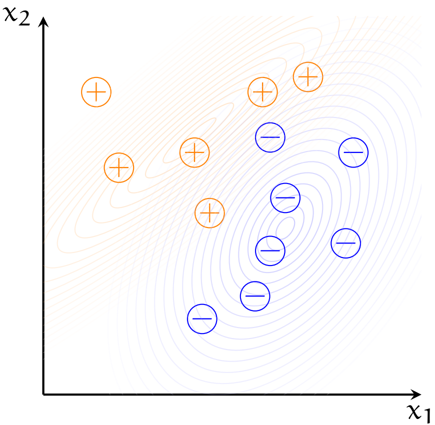
Instance/memory based methods



- No training: just memorize the instances
- During test time, decide based on the k nearest neighbors
- Like decision trees, kNN is non-linear
- It can also be used for regression

A quick survey of some solutions

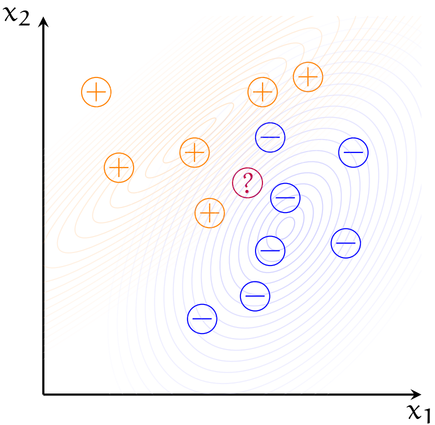
Probability-based solutions



- Estimate distributions of $p(\mathbf{x} \mid y = \oplus)$ and $p(\mathbf{x} \mid y = \ominus)$ from the training data
- Assign the new items to the class c with the highest $p(\mathbf{x} \mid y = c)$

A quick survey of some solutions

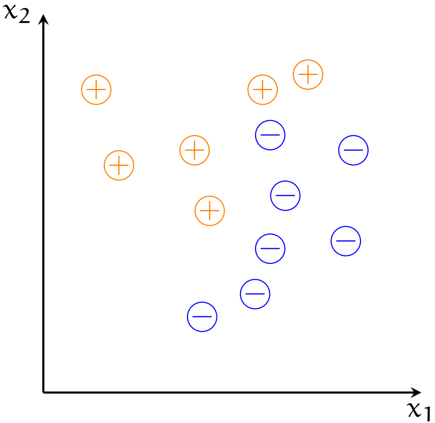
Probability-based solutions

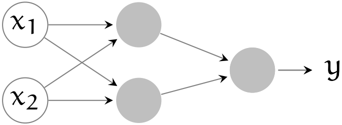


- Estimate distributions of $p(\mathbf{x} \mid y = \oplus)$ and $p(\mathbf{x} \mid y = \ominus)$ from the training data
- Assign the new items to the class c with the highest $p(\mathbf{x} \mid y = c)$

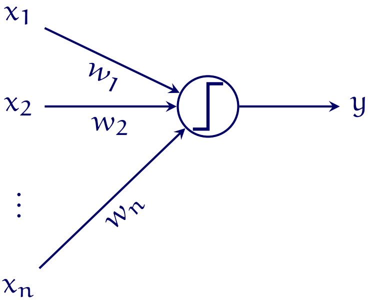
A quick survey of some solutions

Artificial neural networks





The perceptron

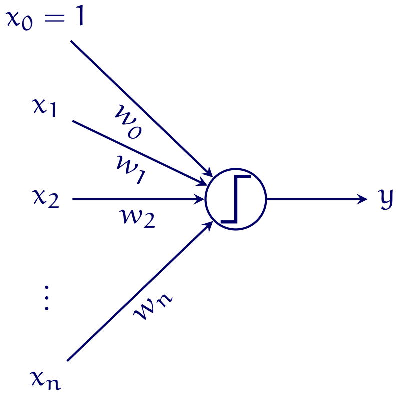


$$y = f\left(\sum_{i}^{n} w_i x_i\right)$$

where

$$f(x) = \begin{cases} +1 & \text{if} \quad \sum_i^n w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

The perceptron



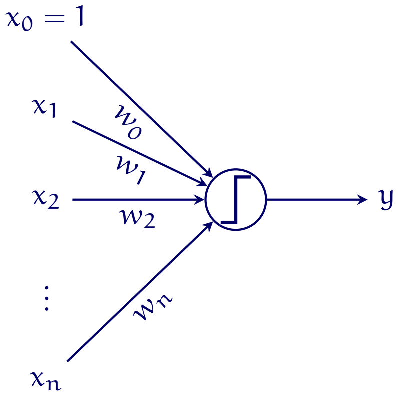
$$y = f\left(\sum_i^n w_i x_i\right)$$

where

$$f(x) = \begin{cases} +1 & \text{if} \quad \sum_i^n w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Similar to the *intercept* in linear models, an additional input x_0 which is always set to one is often used (called *bias* in ANN literature.)

The perceptron: in plain words



- Sum all input x_i weighted with corresponding weight w_i
- Classify the input using a threshold function

positive	the sum is larger than 0
negative	otherwise

Learning with perceptron

- We do not update the parameters if classification is correct
- For misclassified examples, we try to minimize

$$E(w) = -\sum_{i} \boldsymbol{w}\boldsymbol{x}_i y_i$$

where i ranges over all misclassified examples

- Perceptron algorithm updates the weights such that

$$w \leftarrow w - \eta \nabla E(w)$$
$$w \leftarrow w + \eta \boldsymbol{x}_i y_i$$

for a misclassified example (η is the learning rate)

The perceptron algorithm

- The perceptron algorithm can be

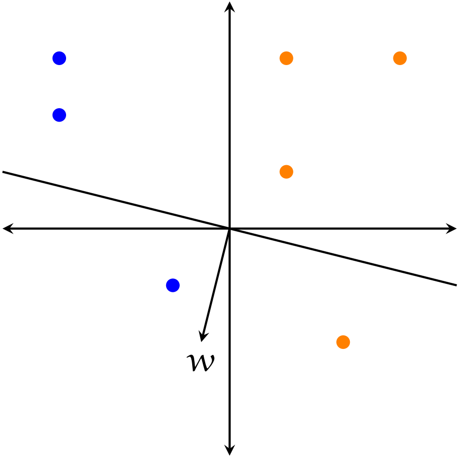
online update weights for a single misclassified example

batch updates weights for all misclassified examples at once

- The perceptron algorithm converges to the global minimum if the classes are *linearly separable*
- If the classes are not linearly separable, the perceptron algorithm will not stop
- We do not know whether the classes are linearly separable or not before the algorithm converges

Perceptron algorithm (online)

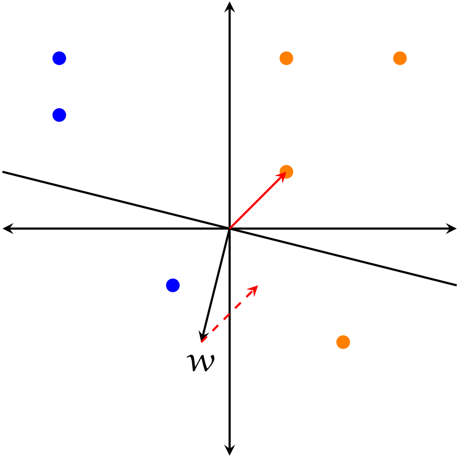
demonstration



1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence

Perceptron algorithm (online)

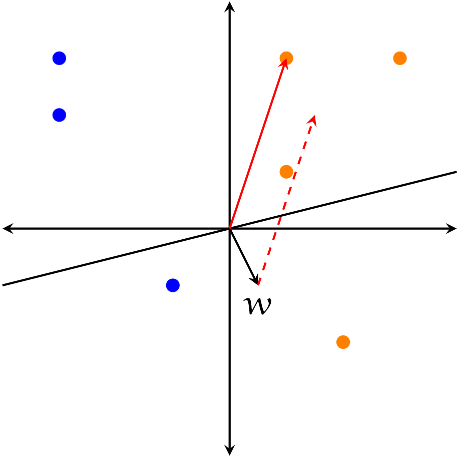
demonstration



1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence

Perceptron algorithm (online)

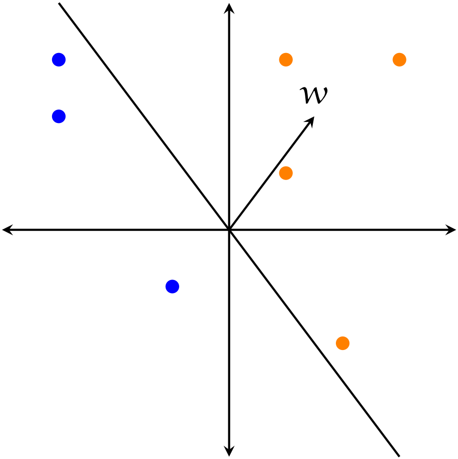
demonstration



1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i\boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i\boldsymbol{x}_i$, go to step 2 until convergence

Perceptron algorithm (online)

demonstration



1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add $y_i \boldsymbol{x}_i$ to $\boldsymbol{w}$
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$, go to step 2 until convergence

Note that with every update the set of misclassified examples change

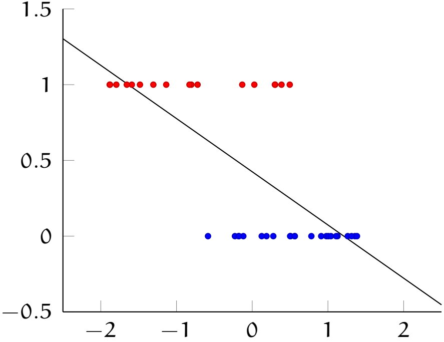
Perceptron: a bit of history

- The perceptron was developed in late 1950’s and early 1960’s (Rosenblatt 1958)
- It caused excitement in many fields including computer science, artificial intelligence, cognitive science
- The excitement (and funding) died away in early 1970’s (after the criticism by Minsky and Papert 1969)
- The main issue was the fact that the perceptron algorithm cannot handle problems that are not linearly separable

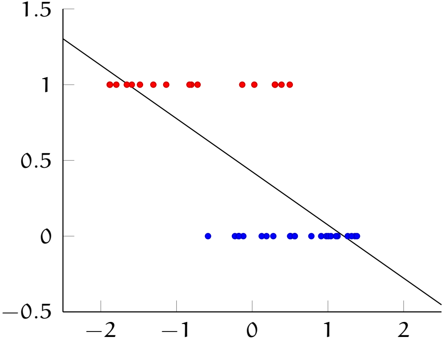
Logistic regression

- Logistic *regression* is a *classification* method
- In logistic regression, we fit a model that predicts $P(y \mid x)$
- Logistic regression is an extension of linear regression
  - it is a member of the family of models called generalized linear models
- Typically formulated for binary classification, but it has a natural extension to multiple classes
- The multi-class logistic regression is often called *maximum-entropy model* (or max-ent) in the NLP literature

Why not linear regression?



Why not linear regression?



- What is $P(y \mid x = 2)$?
- Is RMS error appropriate?

Fixing the outcome: transforming the output variable

Instead of predicting the probability p, we predict logit(p)

$$\hat{y} = \text{logit}(p) = \log \frac{p}{1-p} = w_0 + w_1 x$$

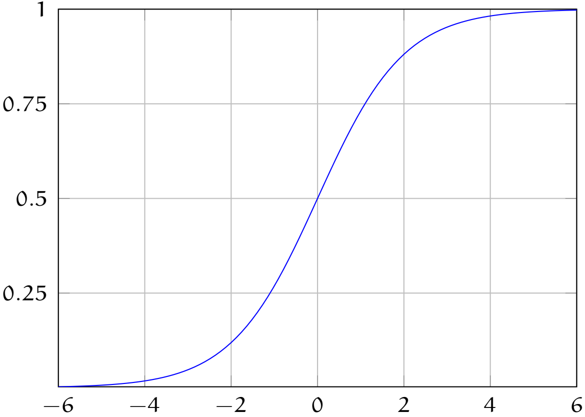
- $\frac{p}{1-p}$ (odds) is bounded between 0 and ∞
- $\log \frac{p}{1-p}$ (log odds) is bounded between $-\infty$ and ∞
- we can estimate $\mathtt{logit}(p)$ with regression, and convert it to a probability using the inverse of $\mathtt{logit}$

$$\hat{p} = \frac{e^{w_0 + w_1 x}}{1 + e^{w_0 + w_1 x}} = \frac{1}{1 + e^{-w_0 - w_1 x}}$$

which is called logistic function (or sometimes sigmoid function, with some ambiguity).

Logistic function

$$\text{logistic}(x) = \frac{1}{1 + e^{-x}}$$



How to fit a logistic regression model

with maximum-likelihood estimation

$$P(y = 1|\boldsymbol{x}) = p = \frac{1}{1 + e^{-\boldsymbol{wx}}} \qquad P(y = 0|\boldsymbol{x}) = 1 - p = \frac{e^{-\boldsymbol{wx}}}{1 + e^{-\boldsymbol{wx}}}$$

The likelihood of the training set is,

$$\mathcal{L}(\boldsymbol{w}) = \prod_i p^{y_i} (1 - p)^{1 - y_i}$$

In practice, we maximize $\log$ likelihood, or minimize $-\log$ likelihood:

$$-\log \mathcal{L}(\boldsymbol{w}) = -\sum_i y_i \log p + (1 - y_i) \log(1 - p)$$

How to fit a logistic regression model (2)

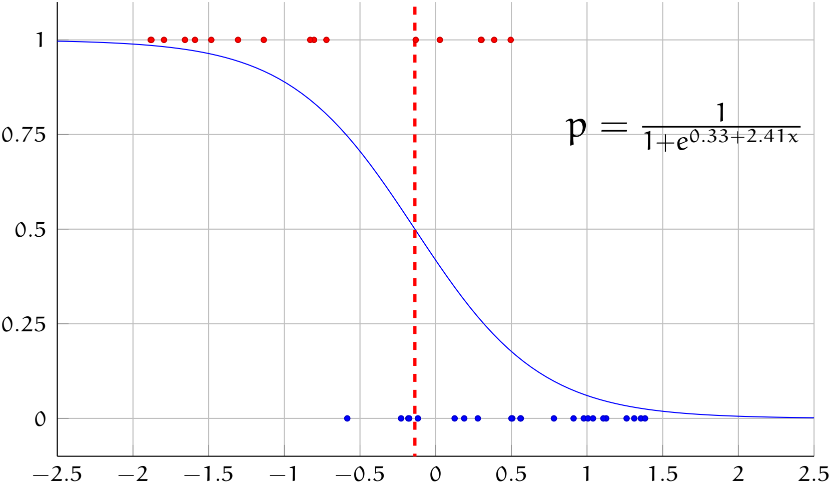
- Bad news: there is no analytic solution
- Good news: the (negative) log likelihood is a convex function
- We can use iterative methods such as *gradient descent* to find parameters that maximize the (log) likelihood
- Using gradient descent, we repeat

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla J(\boldsymbol{w})$$

until convergence, α is called the *learning rate*

Example logistic-regression

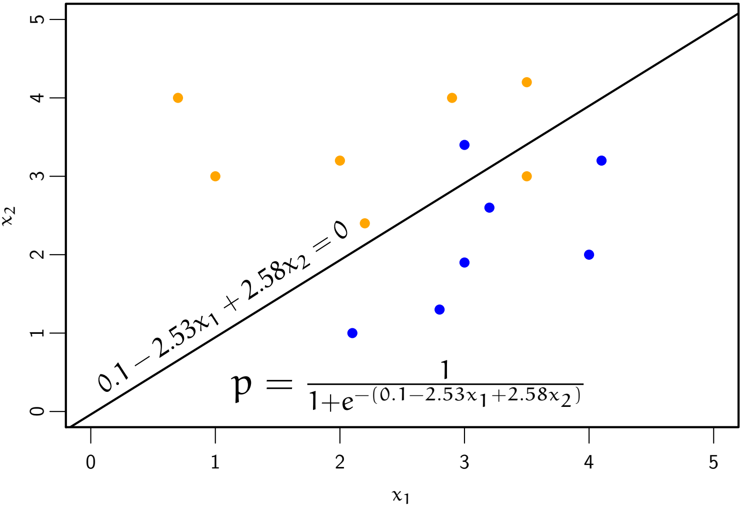
with single predictor



$$p = \frac{1}{1+e^{0.33+2.41x}}$$

Another example

two predictors



Logistic regression as a generalized linear model

Short divergence to statistics

Logistic regression is a special case of *generalized linear models* (GLM). GLMs are expressed with,

$$g(\mathbf{y}) = \mathbf{X}\boldsymbol{w} + \boldsymbol{\epsilon}$$

- The function $g()$ is called the *link function*
- $\boldsymbol{\epsilon}$ is distributed according to a distribution from *exponential family*
- For logistic regression, $g()$ is the logit function, $\boldsymbol{\epsilon}$ is distributed binomially

Naive Bayes classifier

- Naive Bayes (NB) classifier is a well-known simple classifier
- It was found to be effective on a number tasks, primarily in *document classification*
- Popularized by practical spam detection applications
- *Naive* part comes from the strong independent assumption
- *Bayes* part comes from use of Bayes' formula for inverting conditional probabilities

Naive Bayes classifier

- Naive Bayes (NB) classifier is a well-known simple classifier
- It was found to be effective on a number tasks, primarily in *document classification*
- Popularized by practical spam detection applications
- *Naive* part comes from the strong independent assumption
- *Bayes* part comes from use of Bayes’ formula for inverting conditional probabilities
- However, learning is (typically) ‘not really’ Bayesian

Naive Bayes: estimation

- Given a set of features $\boldsymbol{x}$, we want to know whether the class c of the object we want to classify
- During prediction time we pick the class, $\hat{c}$

$$\hat{c} = \arg\max_{c} P(c \mid \boldsymbol{x})$$

- Instead of directly estimating the conditional probability, we invert it using the Bayes' formula

$$\hat{c} = \arg\max_{c} \frac{p(\boldsymbol{x} \mid c)p(c)}{p(\boldsymbol{x})} = \arg\max_{c} p(\boldsymbol{x} \mid c)p(c)$$

- Now the task becomes estimating $p(\boldsymbol{x} \mid c)$ and $p(c)$

Naive Bayes: estimation (cont.)

- Class distribution, $p(c)$, is estimated using the MLE on the training set
- With many features, $\mathbf{x} = (x_1, x_2, \ldots x_n)$, $p(\mathbf{x} \mid c)$ is difficult to estimate
- Naive Bayes estimator makes a conditional independence assumption: given the class, we assume that the features are independent of each other

$$p(\mathbf{x} \mid c) = p(x_1, x_2, \ldots x_n \mid c) = \prod_{i=1}^{n} p(x_i \mid c)$$

Naive Bayes: estimation (cont.)

- The probability distributions $p(x_i \mid c)$ and $p(c)$ are typically estimated using MLE (count and divide)
- A *smoothing* technique may be used for unknown features (e.g., words)
- Note that $p(x_i \mid c)$ can be

binomial e.g, whether a word occurs in the document or not

cagegorical e.g, estimated using relative frequency of words

continious assuming the data is distributed according to a known distribution

a simple example: spam detection

Training set:

document	label
w_1 w_2 w_3 w_4	not spam
w_1 w_2 w_5	spam
w_1 w_4 w_1	not spam
w_3 w_4 w_5	not spam
w_2 w_4 w_5 w_6	spam

Test on:

w_1 w_4 w_5	not spam

a simple example: spam detection

Training set:

document	label
w_1 w_2 w_3 w_4	not spam
w_1 w_2 w_5	spam
w_1 w_4 w_1	not spam
w_3 w_4 w_5	not spam
w_2 w_4 w_5 w_6	spam

Test on:

w_1 w_4 w_5	not spam

How about:

w_1 w_5 w_7	not spam

Classifying classification methods

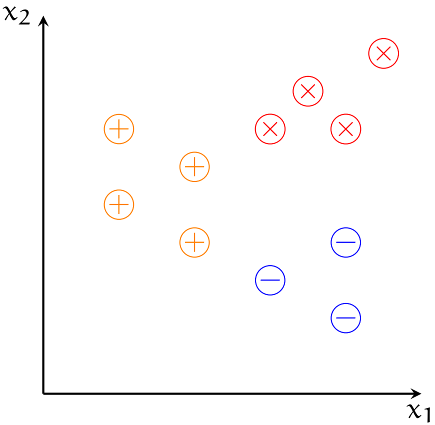
another short digression

- Some classification algorithms are non-probabilistic, discriminative: they return a label for a given input. Examples: perceptron, SVMs, decision trees
- Some classification algorithms are discriminative, probabilistic: they estimate the conditional probability distribution $p(\mathbf{c} \mid \mathbf{x})$ directly. Examples: logistic regression, (most) neural networks
- Some classification algorithms are generative: they estimate the joint distribution $p(\mathbf{c}, \mathbf{x})$. Examples: naive Bayes, Hidden Markov Models, (some) neural models

More than two classes

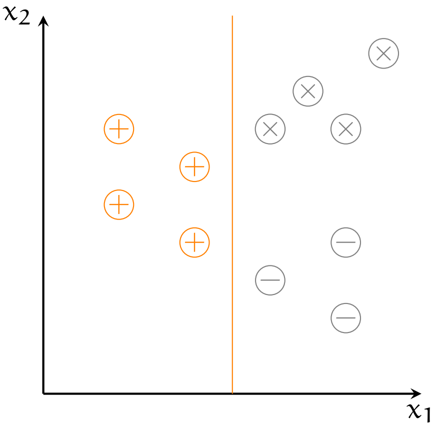
- Some algorithms can naturally be extended to multiple labels
- Others tend to work well in binary classification
- Any binary classifier can be turned into a k-way classifier by
  - training k one-vs.-rest (OvR) or one-vs.-all (OvA) classifiers.
  - Decisions are made based on the class with the highest confidence score.
  - This approach is feasible for classifiers that assign a weight or probability to the individual classes
  - training $\frac{k(k-1)}{2}$ one-vs.-one (OvO) classifiers. Decisions are made based on majority voting

One vs. Rest



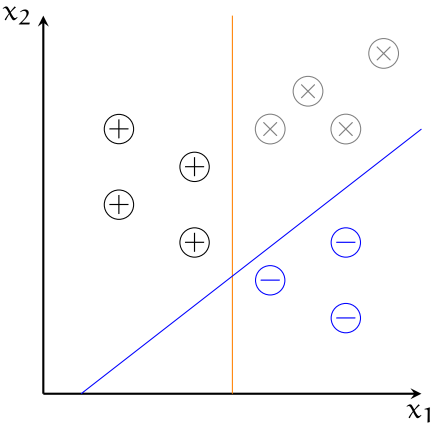
- For 3 classes we fit 3 classifiers separating one class from the rest

One vs. Rest



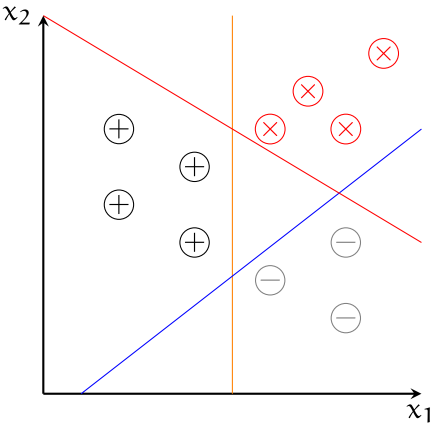
- For 3 classes we fit 3 classifiers separating one class from the rest

One vs. Rest



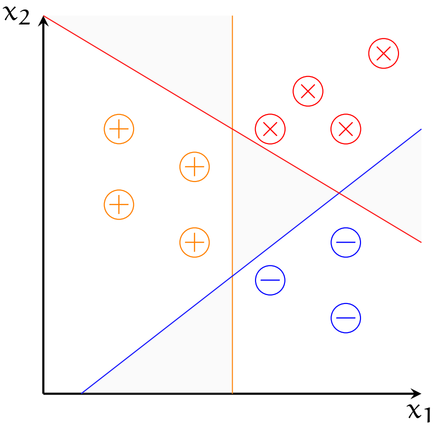
- For 3 classes we fit 3 classifiers separating one class from the rest

One vs. Rest



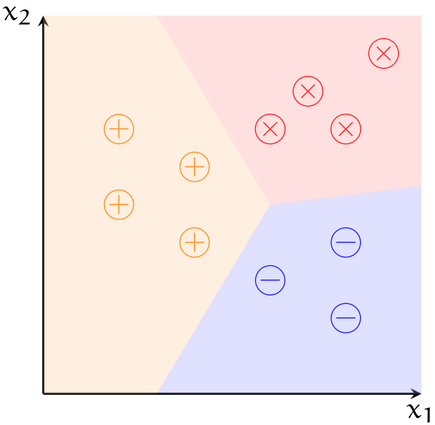
- For 3 classes we fit 3 classifiers separating one class from the rest

One vs. Rest



- For 3 classes we fit 3 classifiers separating one class from the rest
- Some regions of the feature space will be ambiguous

One vs. Rest



- For 3 classes we fit 3 classifiers separating one class from the rest
- Some regions of the feature space will be ambiguous
- We can assign labels based on probability or weight value, if classifier returns one
- One-vs.-one and majority voting is another option

Multi-class logistic regression

- Generalizing logistic regression to more than two classes is straightforward
- We estimate,

$$P(C_k \mid x) = \frac{e^{\boldsymbol{w}_k \boldsymbol{x}}}{\sum_j e^{\boldsymbol{w}_j \boldsymbol{x}}}$$

  Where C_k is the k^{th} class. j iterates over all classes.
- The function is also known as the *softmax* function, used frequently in neural network models as well
- This model is also known as a *log-linear model*, *Maximum entropy* model, *Boltzmann machine*

Summary

- We discussed two basic classification techniques: perceptron, logistic regression, naive Bayes
- We left out many others: SVMs, decision trees, …
- We will discuss some (non-linear) classification methods later

Next

Wed	practical session: solutions/discussion of assignment 1
Fri	ML evaluation, quick summary so far
Mon	Break

Additional reading, references, credits

- Hastie, Tibshirani, and Friedman (2009) covers logistic regression in section 4.4 and perceptron in section 4.5
- Jurafsky and Martin (2009) explains it in section 6.6, and it is moved to its own chapter (7) in the draft third edition

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer series in statistics. Springer-Verlag New York. ISBN: 9780387848587. URL: `http://web.stanford.edu/~hastie/ElemStatLearn/`.

Jurafsky, Daniel and James H. Martin (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. second. Pearson Prentice Hall. ISBN: 978-0-13-504196-3.

Minsky, Marvin and Seymour Papert (1969). *Perceptrons: An introduction to computational geometry*. MIT Press.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, pp. 386–408.