IN4080 – 2020 FALL

NATURAL LANGUAGE PROCESSING

Jan Tore Lønning

Logistic Regression

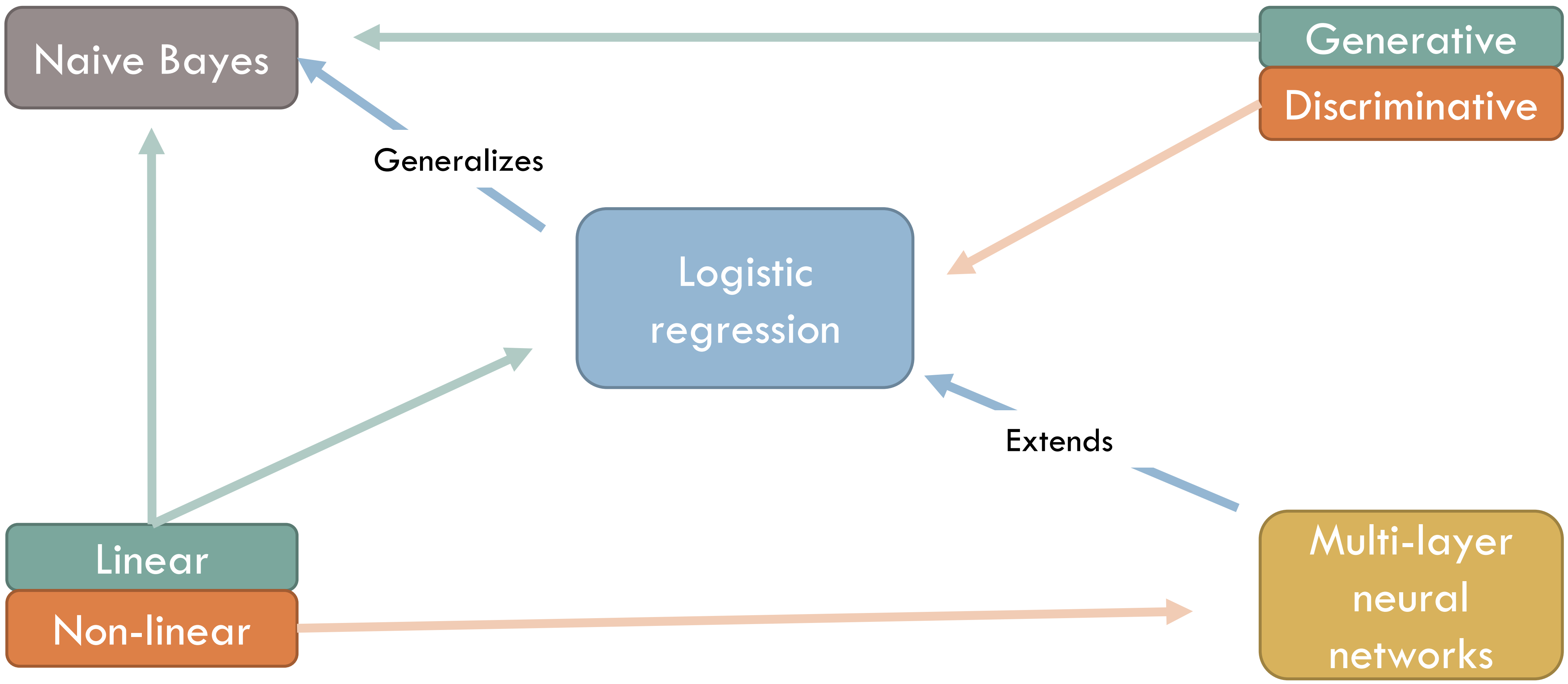
Lecture 4, 7 Sept

Logistic regression

In natural language processing, logistic regression is the baseline supervised machine learning algorithm for classification, and also has a very close relationship with neural networks.

(J&M, 3. ed., Ch. 5)

Relationships



Today

- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

Machine learning

- Last week: Naive Bayes
  - Probabilistic classifier
  - Categorical features
- Today
  - A geometrical view on classification
  - Numeric features
- Eventually see that both Naive Bayes and Logistic regression can fit both descriptions

Notation

When considering numerical features, it is usual to use

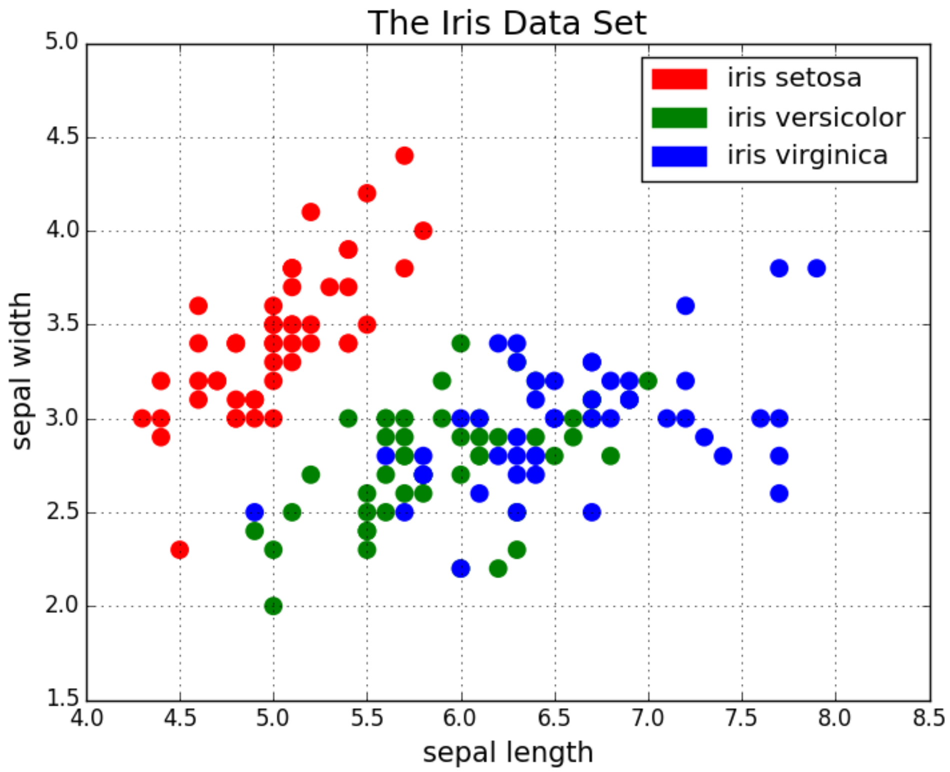
- $(x_1, x_2, \ldots, x_n)$ for the features, where
  - each feature is a number
  - a fixed order is assumed
- y for the output value/class
- In particular, J&M use
  - $\hat{y}$ for the predicted value of the learner, $\hat{y} = f(x_1, x_2, \ldots, x_n)$
  - y for the true value
  - (where Marsland, IN3050, uses y and t, resp.)

Machine learning

- In NLP, we often consider
  - thousands of features (dimension)
  - categorical data
- These are difficult to illustrate by figures
- To understand ML algorithms
  - it easier to use one or two features, 2-3 dimensions, to be able to draw figures
  - and then to use numerical data, to get non-trivial figures

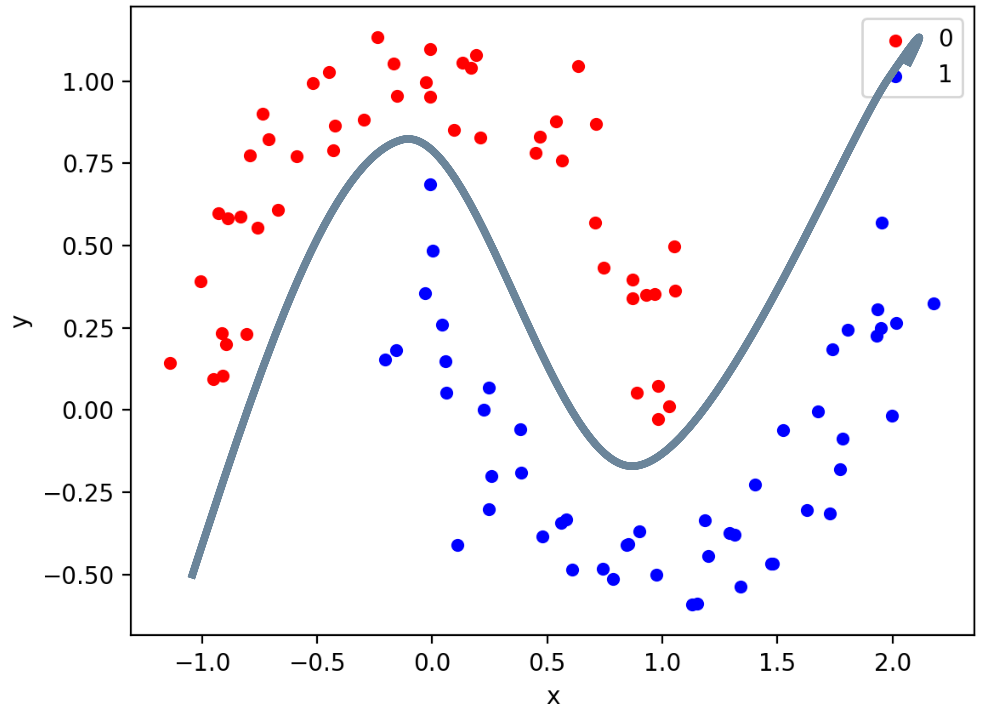
Scatter plot example

- Two numeric features
- Three classes
- We may indicate the classes by colors or symbols



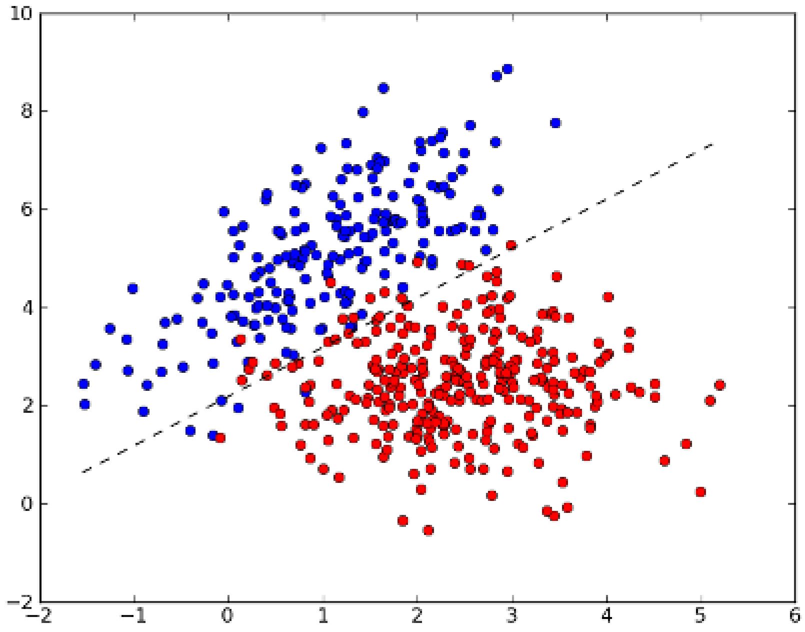
Classifiers – two classes

- Many classification methods are made for two classes
  - And then generalizes to more classes
- The goal is to find a curve that separates the two classes
- With more dimensions: to find a (hyper-)surface



Linear classifiers

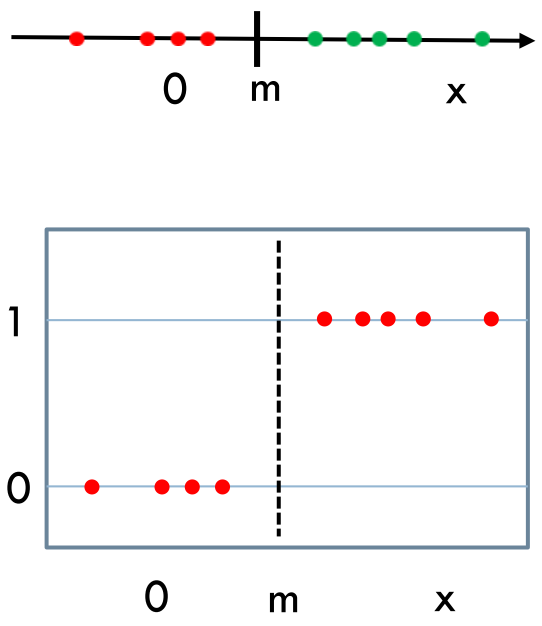
- Linear classifiers try to find a straight line that separates the two classes (in 2-dim)
- The two classes are linearly separable_if they can be separated by a straight line
- If the data isn't linearly separable, the classifier will make mistakes.
- Then: the goal is to make as few mistakes as possible



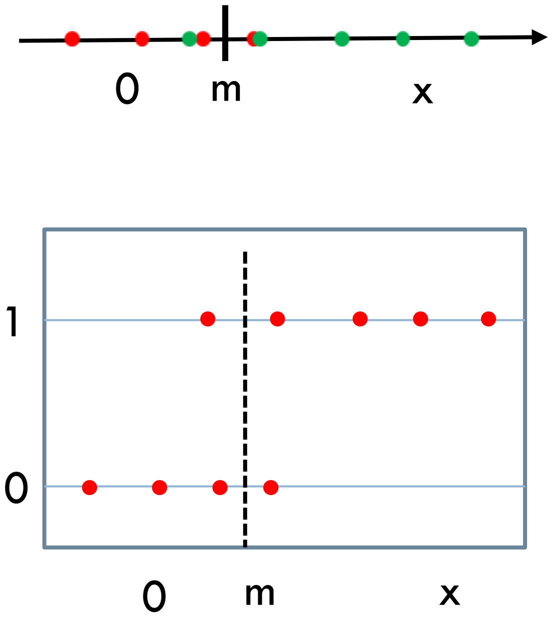
One-dimensional classification

- A linear separator is simply a point
- An observation is classified as
  - class 1 iff x>m
  - Class 0 iff x<m

Data set 1:
linerarly separable

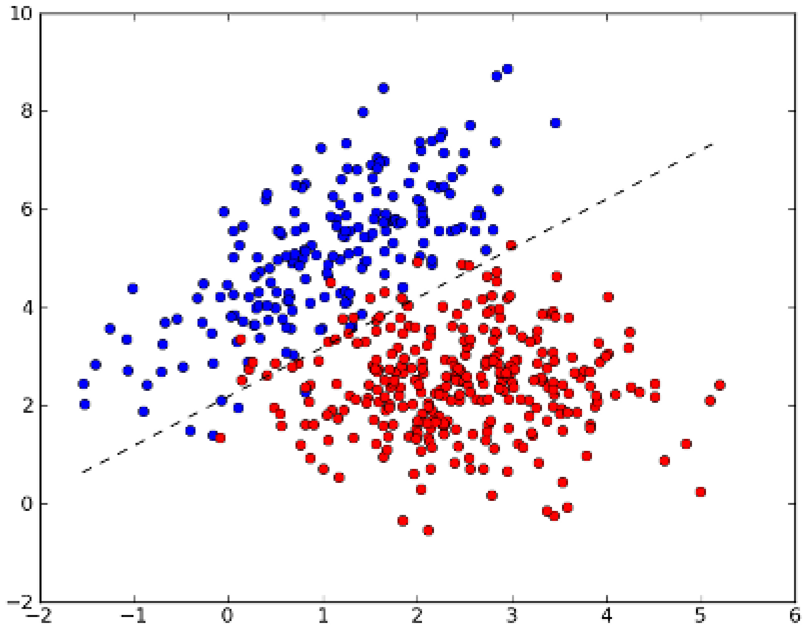


Data set 2:
not linerarly separable



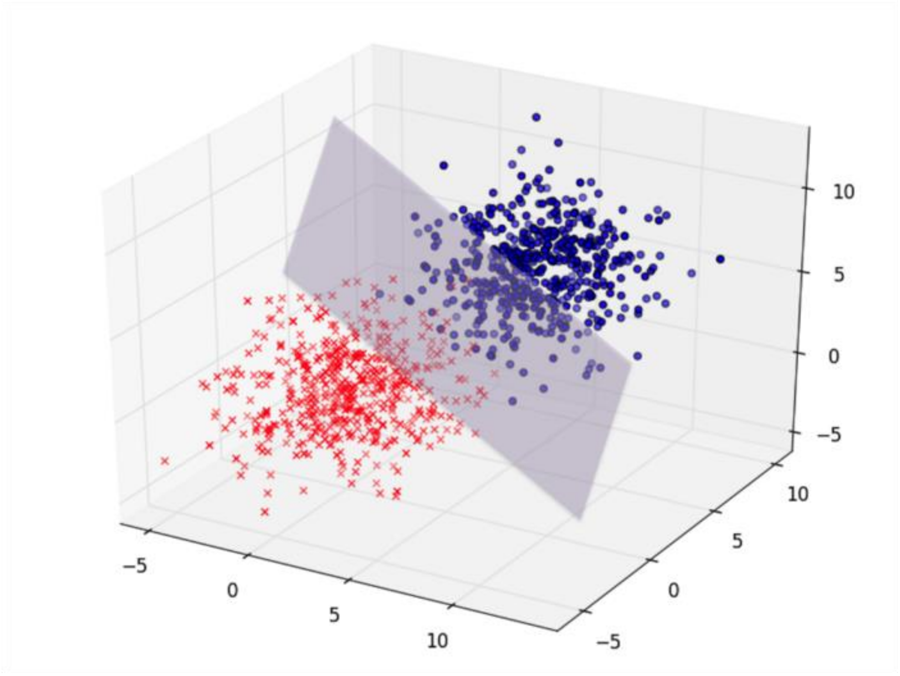
Linear classifiers: two dimensions

- a line has the form $ax+by+c=0$
- $ax + by < -c$ for red points
- $ax + by > -c$ for blue points



More dimensions

- In a 3 dimensional space (3 features) a linear classifier corresponds to a plane
- In a higher-dimensional space it is called a hyper-plane



Linear classifiers: *n* dimensions

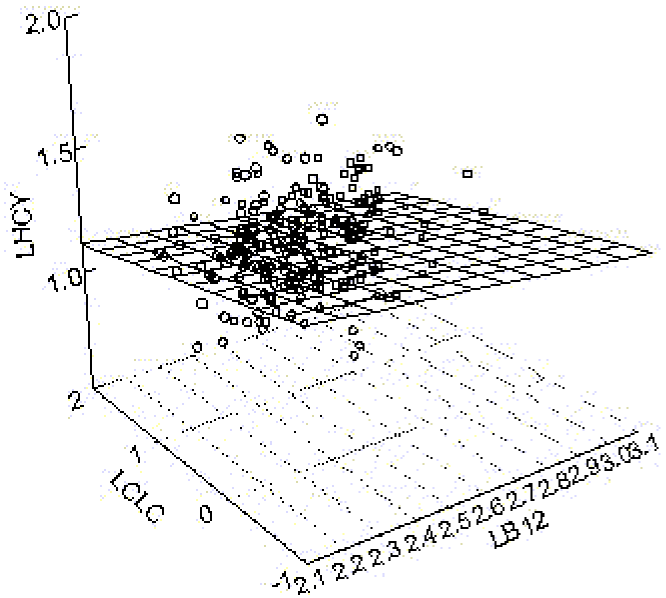
- A hyperplane has the form
  - $\sum_{i=1}^{n} w_i x_i + w_0 = 0$
- which equals
  - $\sum_{i=0}^{n} w_i x_i =$

  $(w_0, w_1, \dots, w_n) \cdot (x_0, x_1, \dots, x_n) = \vec{w} \cdot \vec{x} = 0,$
  - assuming $x_0 = 1$

- An object belongs to class C iff

$$\hat{y} = f(x_0, x_1, \dots, x_n) = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x} > 0$$

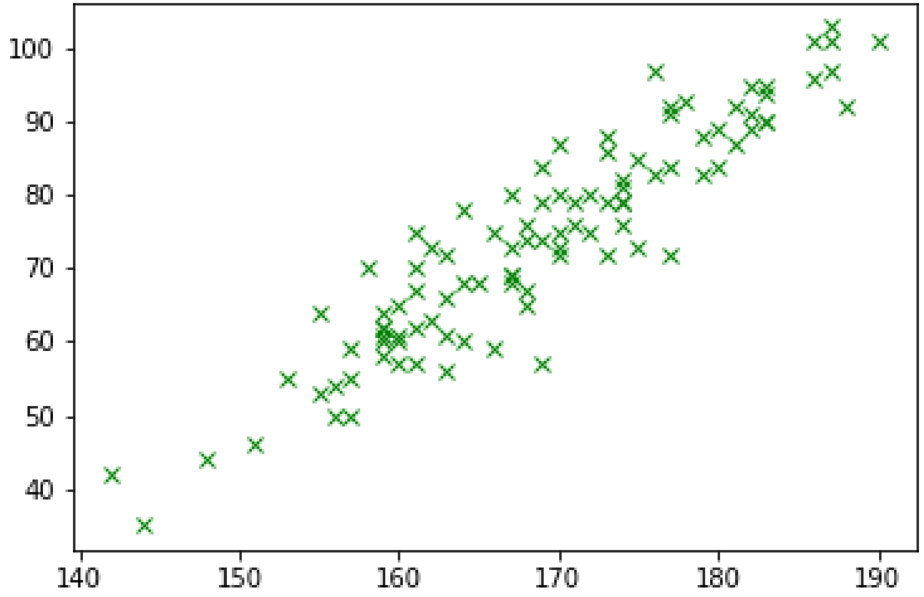
- and to not C, otherwise



Today

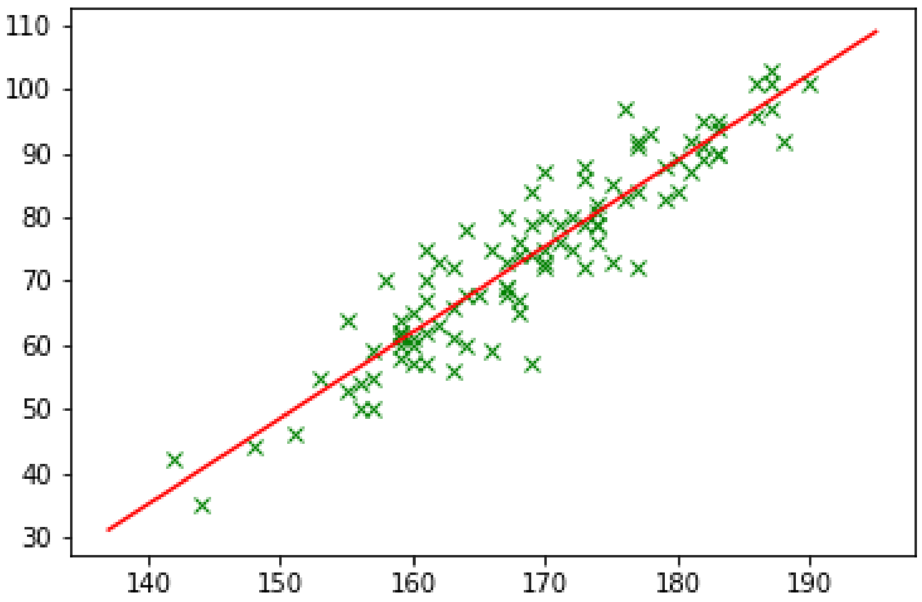
- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

Linear Regression



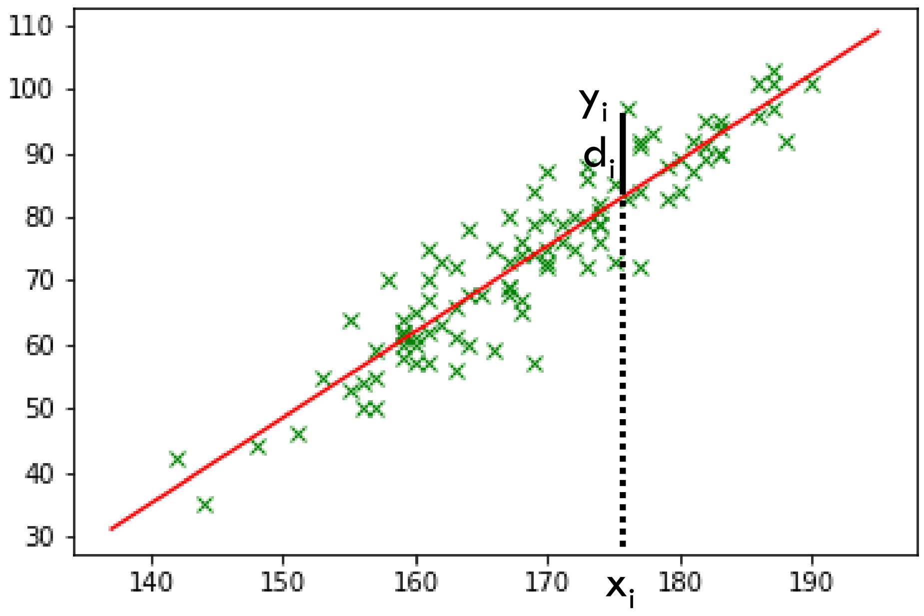
- Data:
  - 100 males: height and weight
- Goal:
  - Guess the weight of other males when you only know the height

Linear Regression



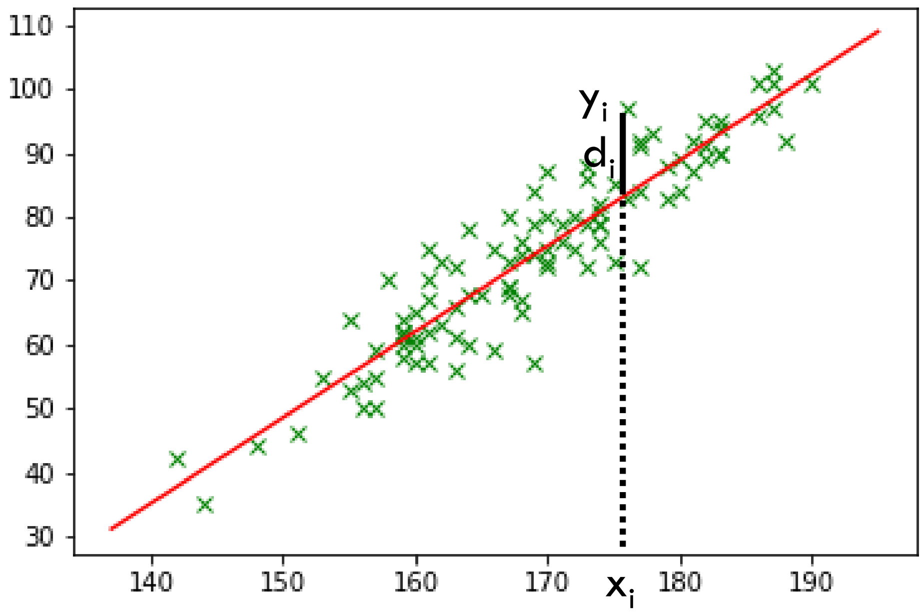
- Method:
  - Try to fit a straight line to the observed data
  - Predict that unseen data are placed on the line
- Questions:
  - What is the best line?
  - How do we find it?

Best fit



- To find the best fit, we compare each
  - true value y_i (green point)
  - to the corresponding predicted value $\hat{y}_i$ (on the red line)
- We define a loss function
  - which measures the discrepancy between the y_i-s and $\hat{y}_i$-s
  - (alternatively called *error function*)
- The goal is to minimize the loss

Loss for linear regression



For linear regression, usual to use:

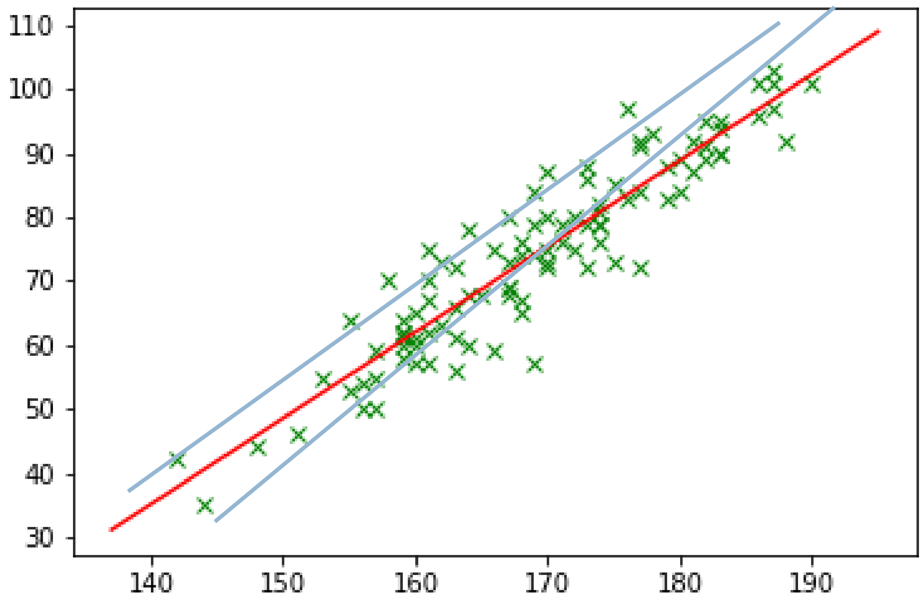
- Mean square error:

$$\frac{1}{m}\sum_{i=1}^{m} {d_i}^2$$

  - where
    - $d_i = (y_i - \hat{y}_i)$
    - $\hat{y}_i = \ (ax_i + b)$

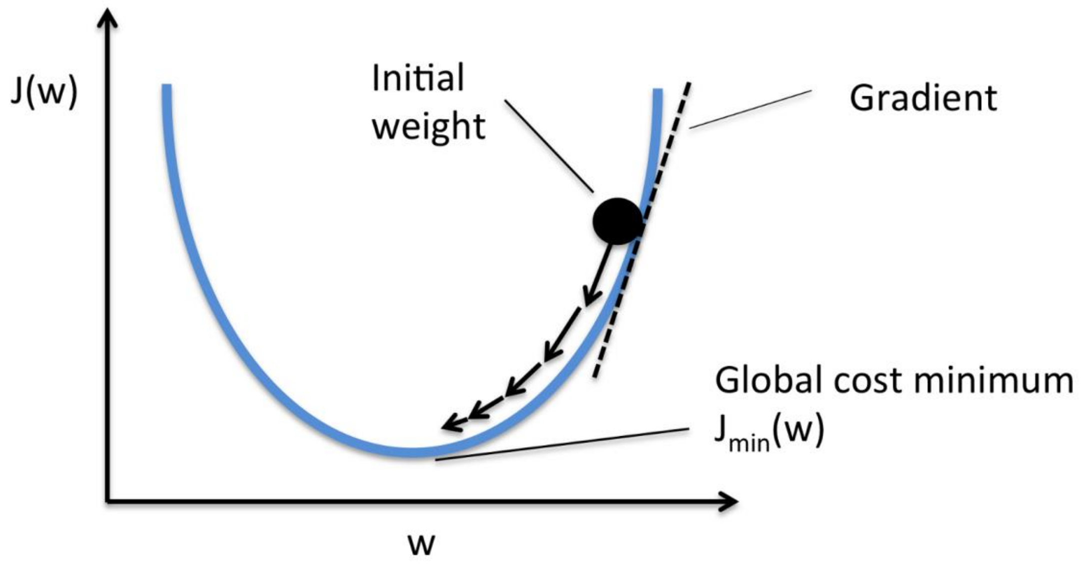
- Why squaring?
  - To not get 0 when we sum the diff.s.
  - Large mistakes are punished more severly

Learning = minimizing the loss



- For lin. regr. there is a formula
  - (this is called an analytic solution)
  - But slow with many (millions) of features
- Alternative:
  - Start with one candidate line
  - Try to find better weights
  - Use Gradient Descent
  - A kind of search problem

Gradient descent



- We use the derivative of the (mse) loss function to point in which direction to move
- We are approaching a unique global minimum
- For details:
  - IN3050/4050 (spring)

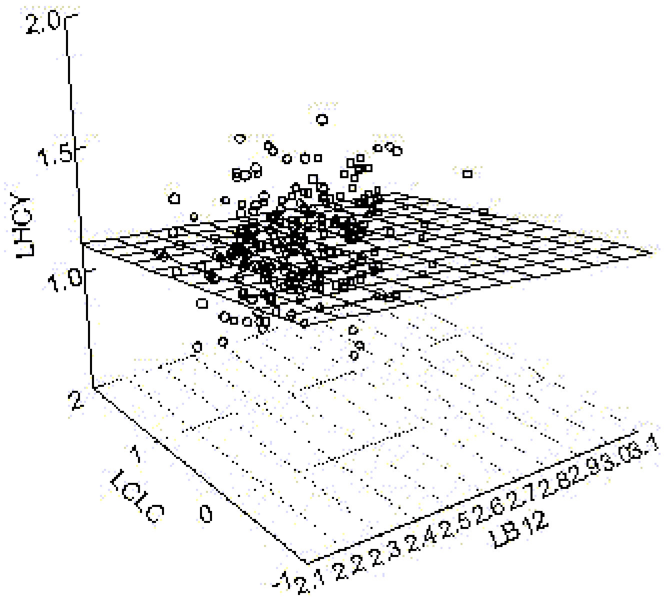
Linear regression: higher dimensions

- Linear regression of more than two variables works similarly
- We try to fit the best (hyper-)plane

$$\hat{y} = f(x_0, x_1, \dots, x_n) = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$

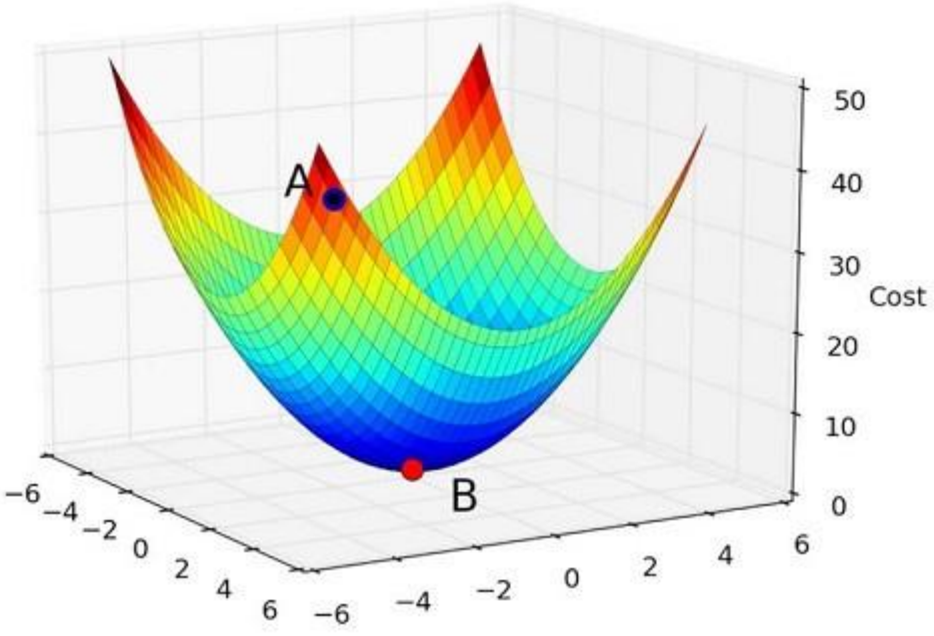
- We can use the same mean square error:

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



Gradient descent

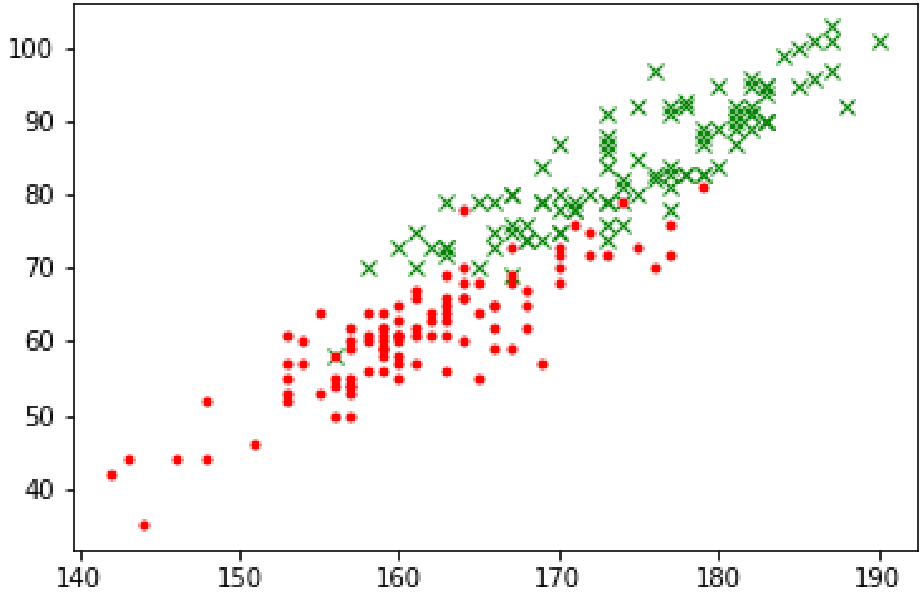
- The loss function is convex: you are not stuck in local minima
- The gradient
  - (= the partial derivatives of the loss function)
- tells us in which direction we should move
  - = how long steps in each direction



Today

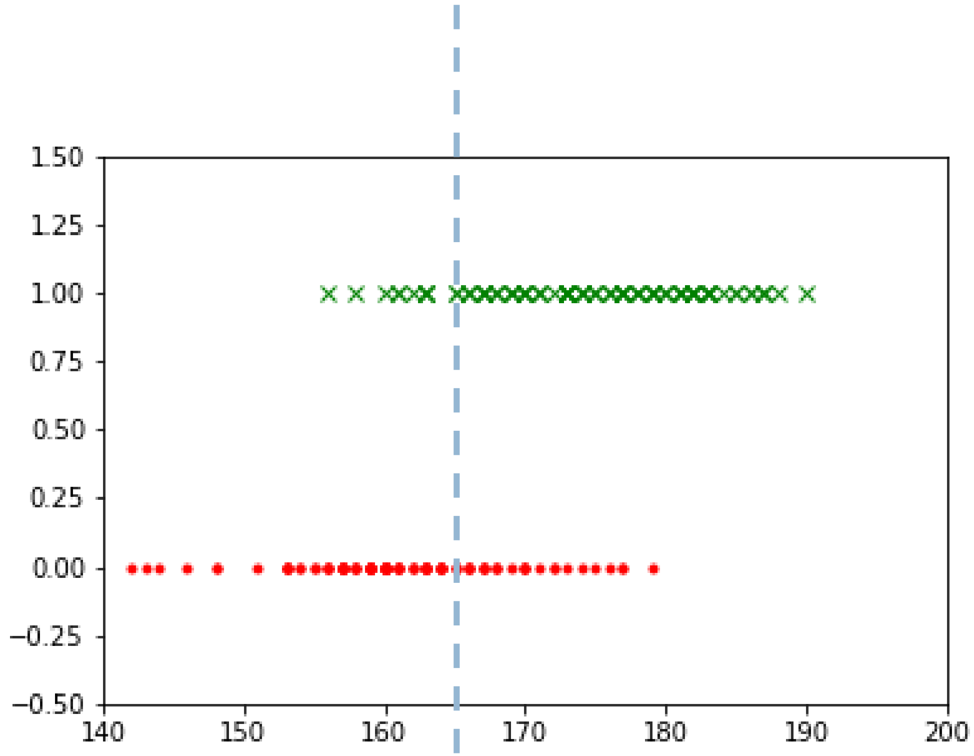
- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

From regression to classification



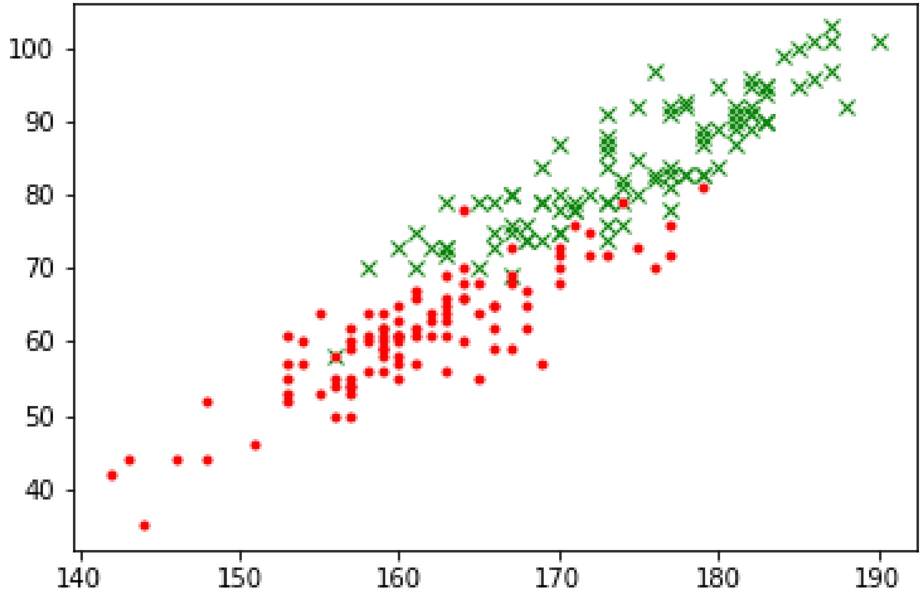
- Goal: predict gender from two features: height and weight

Predicting gender from height



- First:
  try to predict from height only
- The decision boundary should be a number: c
- An observation, *n*, is classified
  - 1(*male*) if *height_n* > c
  - 0 (*not male*) otherwise
- How do we determine c?

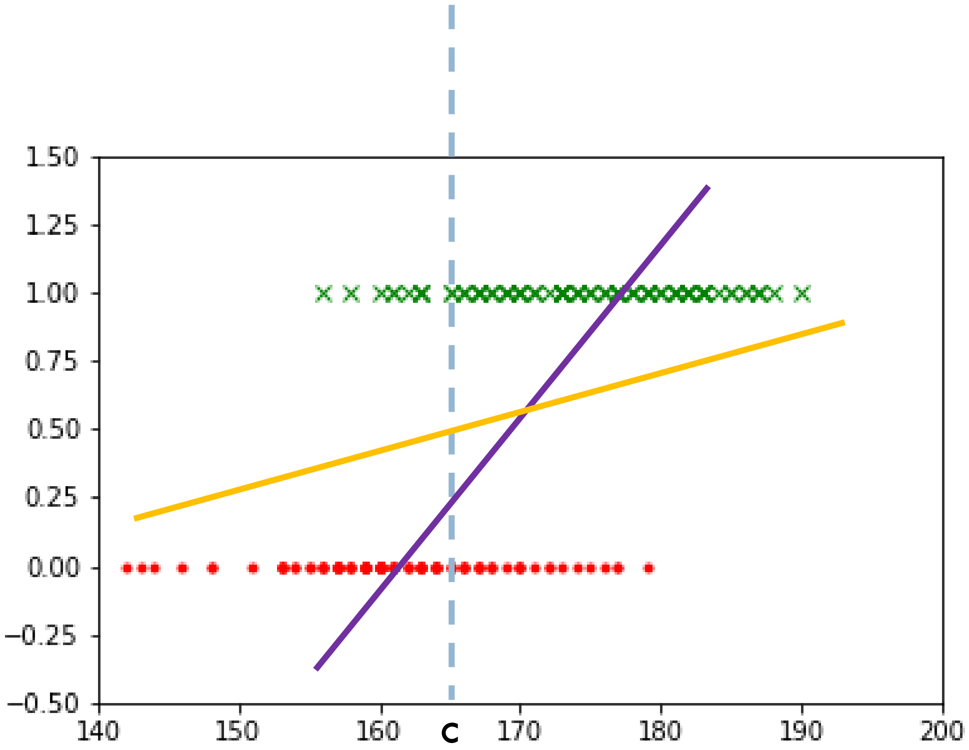
Digression



By the way

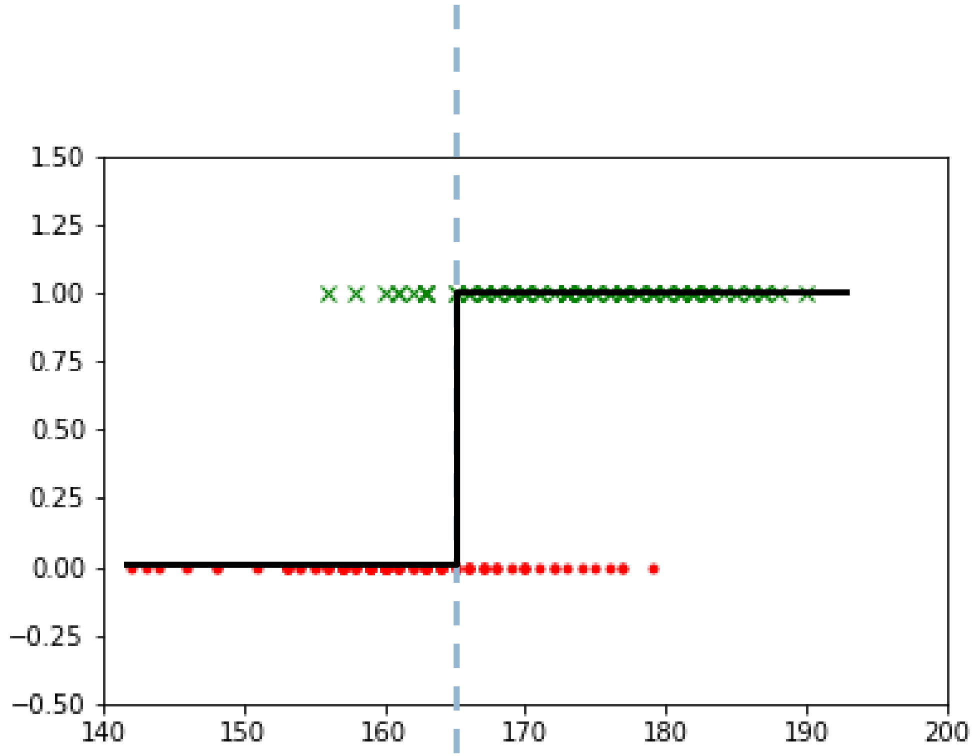
- How good are the best predictions og gender given height?
- Given weight?
- Given height+weight?

Linear regression is not the best choice



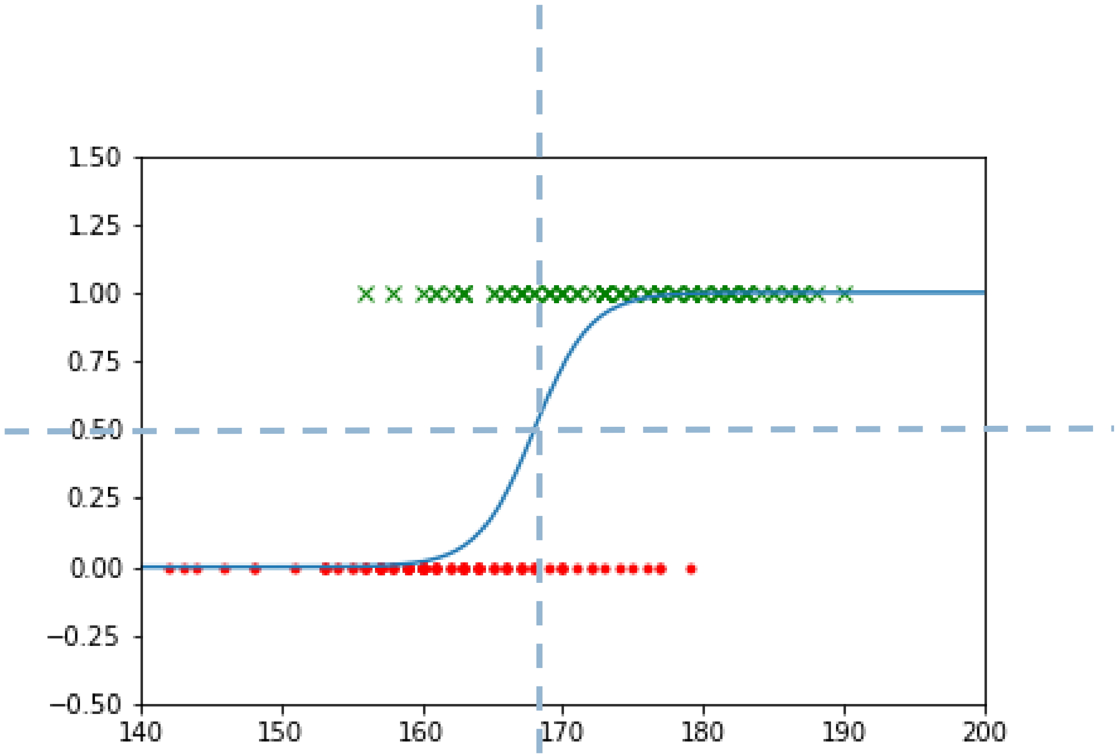
- How do we determine c?
- We may use linear regression:
  - Try to fit a straight line
  - The observations has $y \in \{0,1\}$
  - The predicted value $\hat{y} = ax + b$
- Possible, but
  - Bad fit, y_i and $\hat{y}_i$ are different
  - Correctly classified objects contribute to the error (wrongly!)

The "correct" decision boundary



- The correct decision boundary is the Heaviside step function
- But:
  - Not a differentiable function
    - can't apply gradient descent

The sigmoid curve

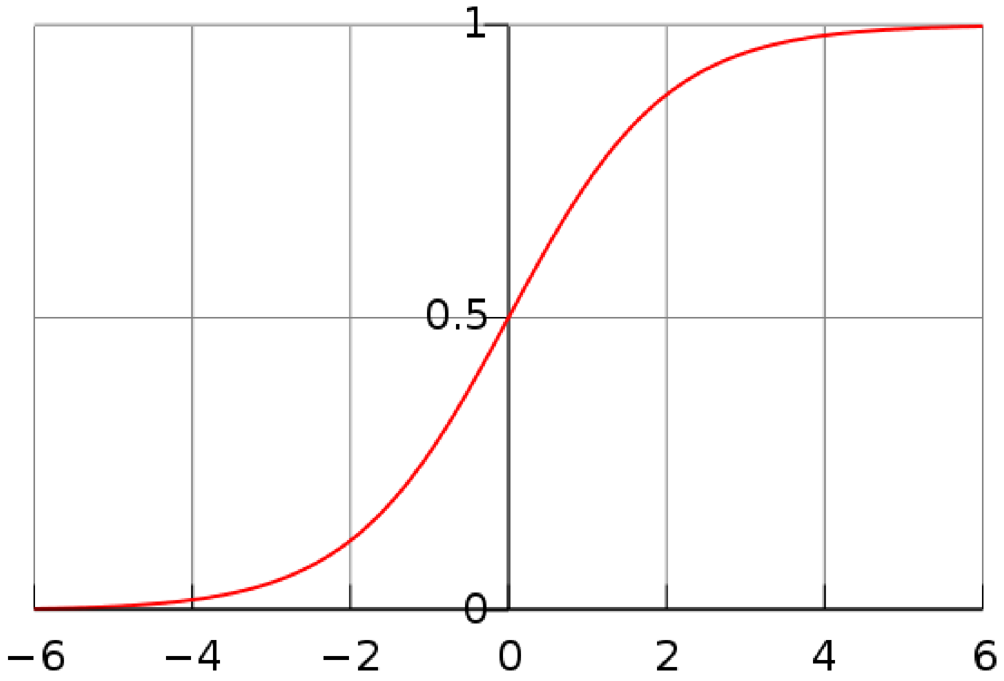


- An approximation to the ideal decision boundary
- Differentiable
  - Gradient descent
- Mistakes further from the decision boundary are punished harder

An observation, *n*, is classified
- *male* if f(*height_n*) > *0.5*
- *not male* otherwise

The logistic function

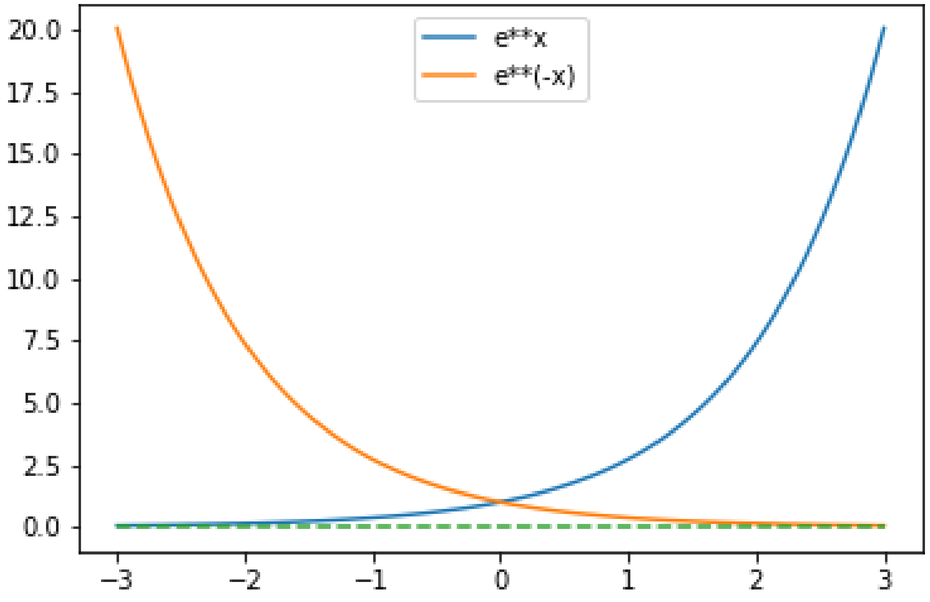
- $y = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$
- A sigmoid curve
  - But also other functions make sigmoid curves e.g. $y = \tanh(z)$
- Maps $(-\infty, \infty)$ to $(0,1)$
- Monotone
- Can be used for transforming numeric values into probabilities

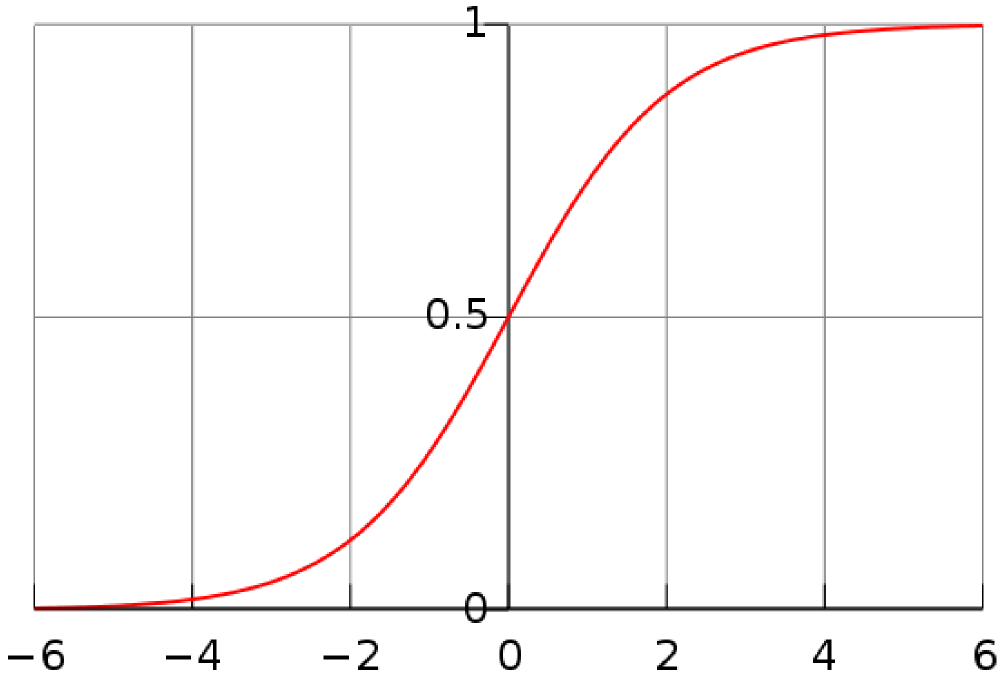


Exponential function - Logistic function

$$y = e^z$$

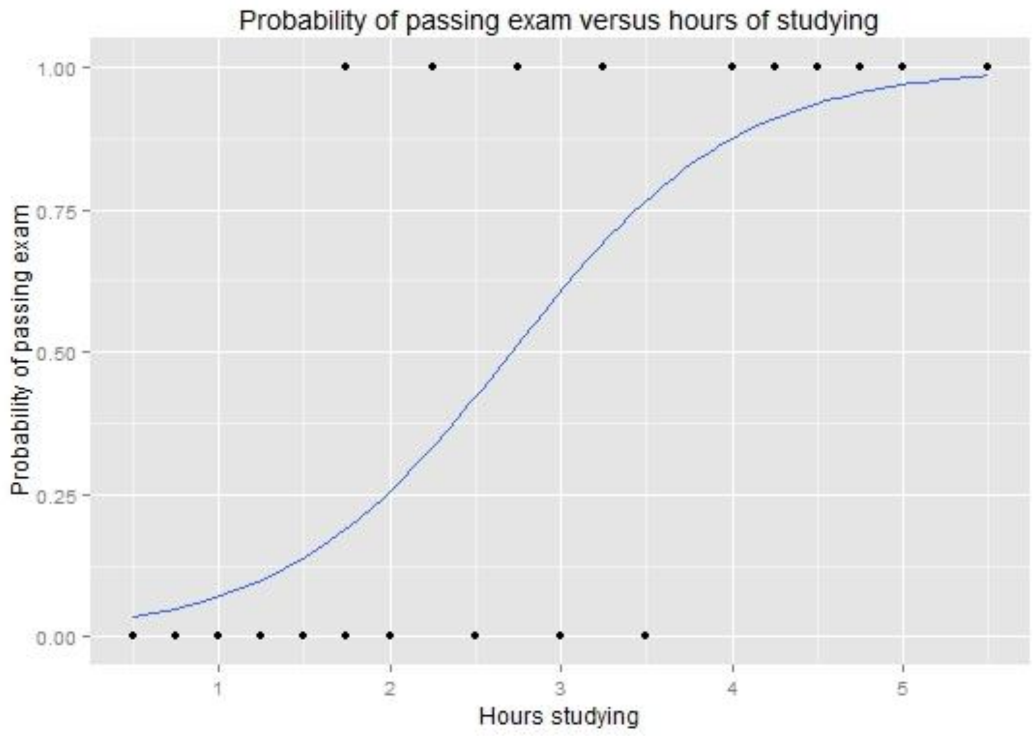
$$y = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$





The effect

- Instead of a linear classifier which will classify some instances incorrectly
- The logistic regression will ascribe a probability to all instances for the class C (and for notC)
- We can turn it into a classifier by ascribing class C if $P(C|\vec{x}) > 0.5$
- We could also choose other cut-offs, e.g. if the classes are not equally important



source: Wikipedia

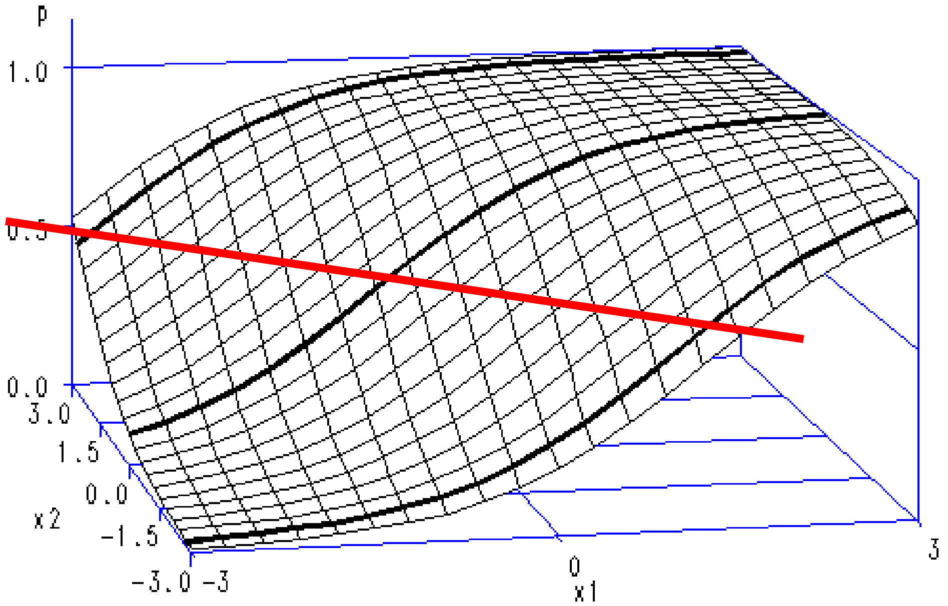
Logistic regression

- Logistic regression is probability-based
- Given to classes C, not-C,
  start with $P(C|\vec{x})$ and $P(notC|\vec{x})$ given a feature vector $\vec{x}$
- Consider the <u>odds</u> $\frac{P(C|\vec{x})}{P(notC|\vec{x})} = \frac{P(C|\vec{x})}{1-P(C|\vec{x})}$
  - If this is >1, $\vec{x}$ most probably belongs to C
  - It varies between 0 and infinity
- Take the logarithm of this $\log \frac{P(C|\vec{x})}{1-P(C|\vec{x})}$
  - If this is >0, $\vec{x}$ most probably belongs to C
  - It varies between minus infinity and pluss infinity

Logistic regression

- $\log \frac{P(C|\vec{x})}{1-P(C|\vec{x})} > 0$?
- Try to find a linear expression for this $\log \frac{P(C|\vec{x})}{1-P(C|\vec{x})} = \vec{w} \cdot \vec{x} > 0$
- Given such a linear expression
  - $\frac{P(C|\vec{x})}{1-P(C|\vec{x})} = e^{\vec{w}\cdot\vec{x}}$
  - $P(C|\vec{x}) = \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$

With two features



From IDRE, UCLA

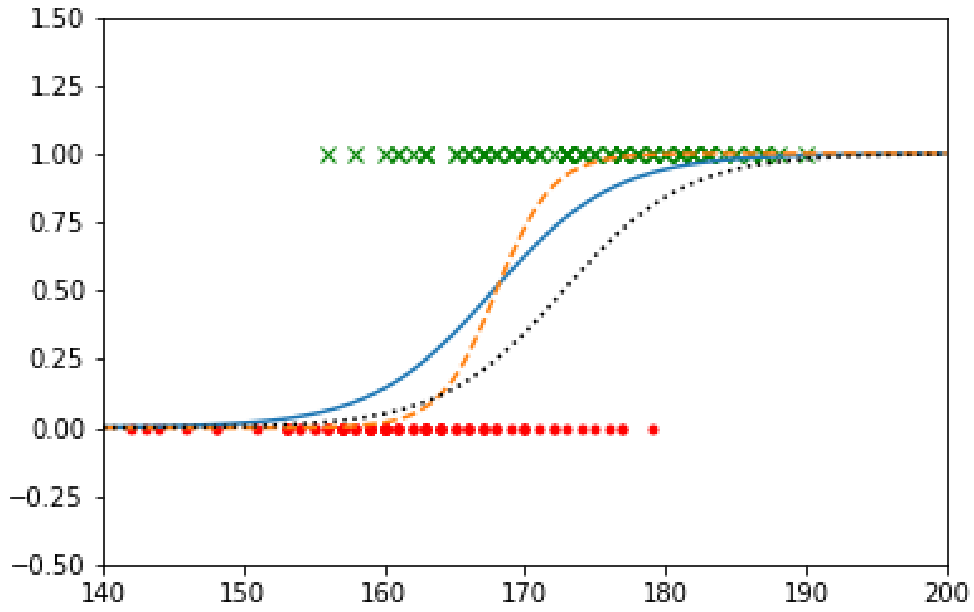
- Two features: x_1, x_2
- Apply weights: w_0, w_1, w_2
- Let $y = w_0 + w_1 x_1 + w_2 x_2$
- Apply the logistic function, σ, and check whether
  - $\sigma(y) = \frac{1}{1+e^{-y}} > 0.5$

Geometrically:
Folding a plane along a sigmoid
The decision boundary is the intersection of this surface and the plane 0.5: a straight line

Today

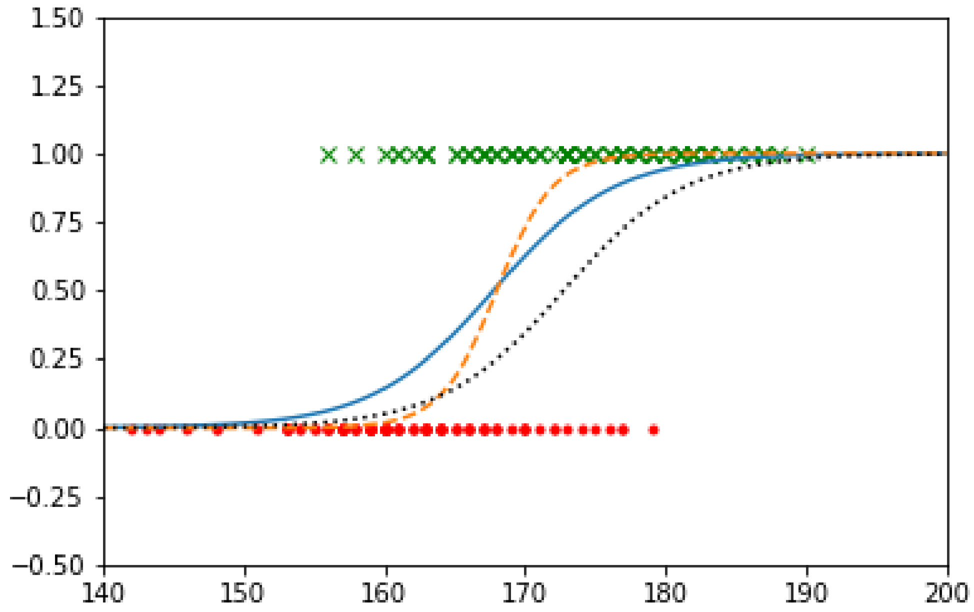
- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

How to find the best curve?



- What are the best choices of a and b in $\frac{1}{1+e^{-(ax+b)}}$?
- Geometrically a and b determine the
  - Midpoint
  - Steepness
- of the curve

Learning in the logistic regression model



- A training instance consists of
  - a feature vector $\vec{x}$
  - a label (class), y, which is 1 or 0.
- With a set of weights, $\overrightarrow{w}$, the classifier will assign
  - $\hat{y} = P(C = 1|\vec{x}) = \frac{1}{1+e^{-\overrightarrow{w}\cdot\vec{x}}}$ to this training instance $\vec{x}$
  - where $P(C = 0|\vec{x}) = 1 - \hat{y}$
- Goal: find $\overrightarrow{w}$ that maximize $P(C = y|\vec{x})$ of all training inst.s

Loss function

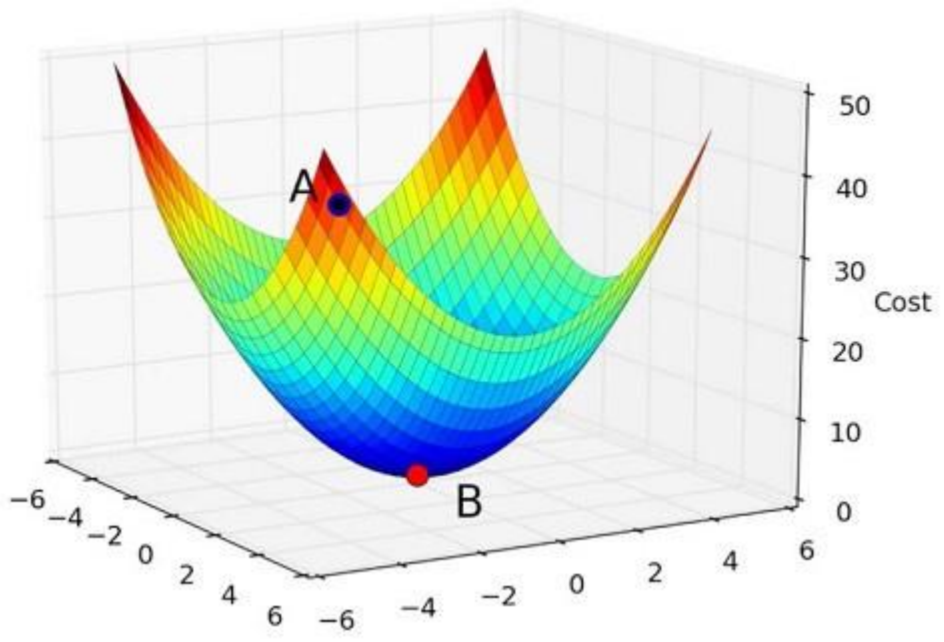
- In machine learning we have to determine an objective for the training.
- We can do that in terms of a loss function.
- The goal of the training is to minimize the loss function.
- Example: linear regression
  - Loss: Mean Square Error
- We can choose between various loss functions.
- The choice is partly determined by the learner.
- For logistic regression we choose (simplified) cross-entropy loss

Cross-entropy loss

- The underlying idea is that we want to maximize the joint probability of all the predictions we make
  - $\prod_{i=1}^{m} P\left(y^{(i)} \mid \vec{x}^{(i)}\right)$, over all the training data i = 1, 2, …m
- This is the same as maximizing
  - $\log \prod_{i=1}^{m} P\left(y^{(i)} \middle| \vec{x}^{(i)}\right) = \sum_{i=1}^{m} \log P\left(y^{(i)} \middle| \vec{x}^{(i)}\right)$
- This is the same as minimizing
  - $L_{CE}(\vec{w}) = -\log \prod_{i=1}^{m} P(y^{(i)} | \vec{x}^{(i)}) = \sum_{i=1}^{m} -\log P(y^{(i)} | \vec{x}^{(i)})$
  - Which is an instance of what is called the cross-entropy loss

Gradient descent

- To minimize the loss function we can use gradient descent.
- Good news:
  - The loss function is convex: you are not stuck in local minima
  - We know which way to go
- We skip the details of sec. 5.4



Variations of gradient descent

Batch training:

- Calculate the loss for the whole training set
- Make one move in the correct direction
- Repeat (an epoch)

- Can be slow

Stochastic gradient descent:

- Pick one item
- Calculate the loss for this item
- Move in the direction of the gradient for this item

- Each move does not have to be in the direction of the gradient for the whole set.
- But the overall effect may be good
- Can be faster

Variations of gradient descent

Mini-batch training:

- Pick a subset of the training set of a certain size
- Calculate the loss for this subset
- Make one move in the direction of this gradient
- Repeat (an epoch)

- A good compromise between the two extremes
- (The other two are subcases of this)

Solvers/optimizers

- There are various different solvers and optimizers for gradient descent (which you may meet later).
- Observe that you may specify between solvers in scikit-learn.

Regularization

- LogReg is prone to overfitting to the training data
- Hence apply regularization

$$\hat{w} = \underset{w}{\arg\max} \sum_{i=1}^{m} \log P(c^i \mid \vec{f}^{\,i}) - \alpha R(w)$$

- The regularization punishes large weights
- Most common is L2-regularization $R(W) = \sum_0^n w_i^2$
- Alternative: L1-regularization $R(W) = \sum_0^n |w_i|$

scikit-learn – LogisticRegression

- `LogisticRegression(penalty='l2', …, C=1.0, …)`
- By adjusting C, you may get better results
- The optimal C varies from task to task
- Uses L2-regularization as default
- Whether L1 or L2 may depend on the learner

Example: Features for sent. classification in LR

Var	Definition	Value in Fig. 5.2
x_1	$\text{count}(\text{positive lexicon}) \in \text{doc})$	3
x_2	$\text{count}(\text{negative lexicon}) \in \text{doc})$	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	$\text{count}(\text{1st and 2nd pronouns} \in \text{doc})$	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\log(\text{word count of doc})$	$\ln(64) = 4.15$

Today

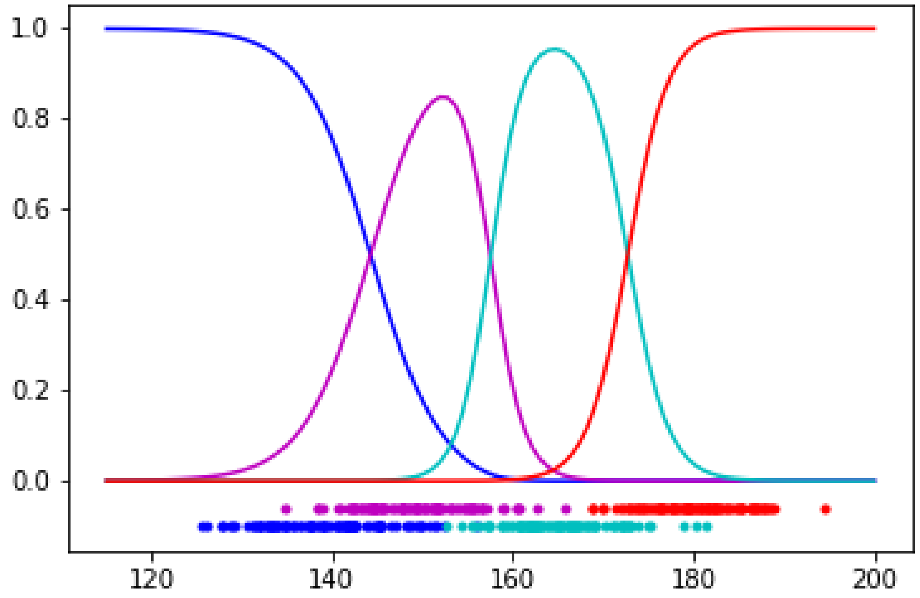
- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

Multinomial Logistic Regression

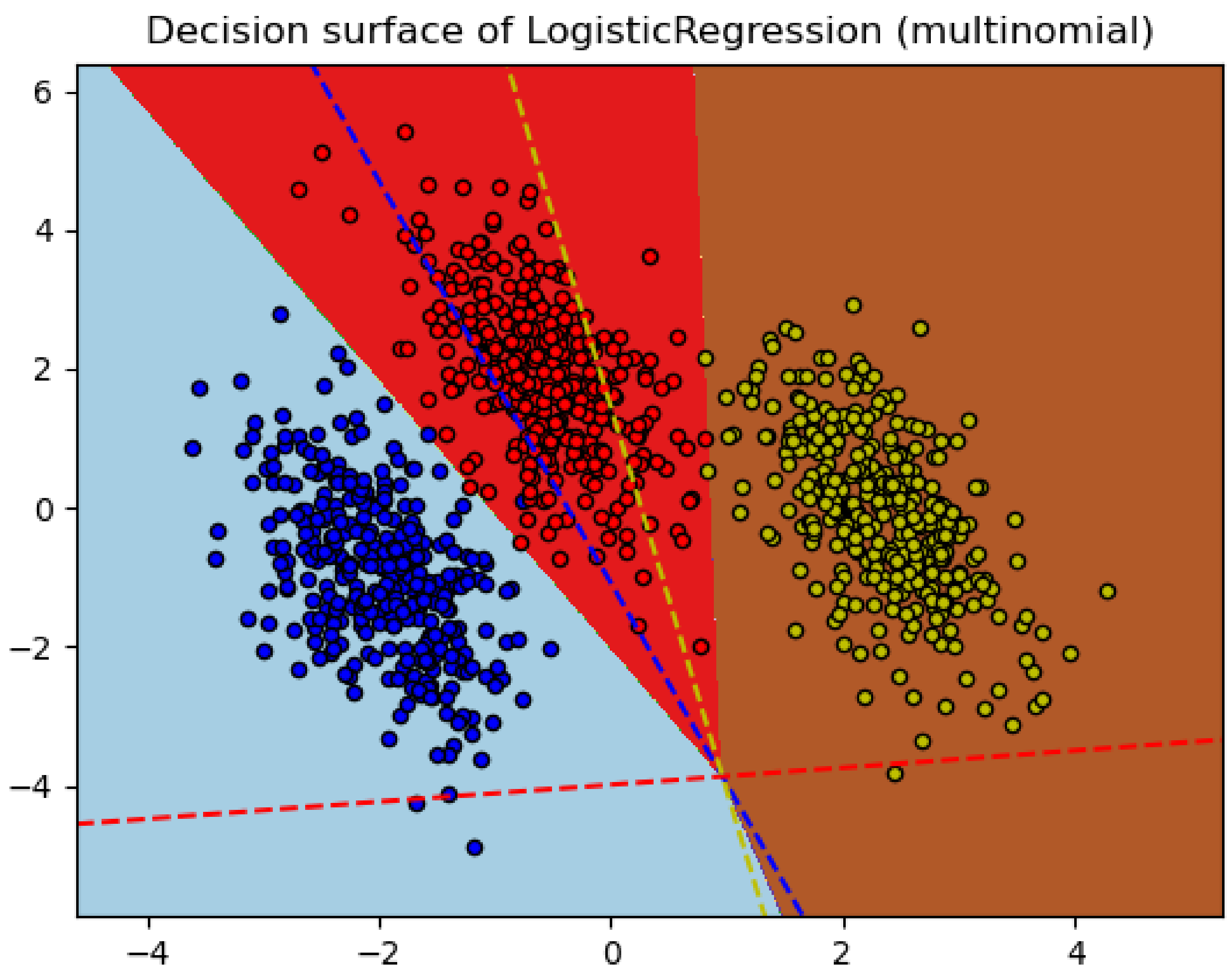
- Also called maximum entropy (maxent) classifier, or softmax regression
- With one class we
  - considered $P(C|\vec{x}) = \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
  - and implicitly $P(nonC|\vec{x}) = 1 - \frac{e^{\vec{w}\cdot\vec{x}}}{1+e^{\vec{w}\cdot\vec{x}}} = \frac{1}{1+e^{\vec{w}\cdot\vec{x}}}$
- We now consider a linear expression $\vec{w}_i$, for each class $C_i, i = 1, \dots, k$
- The probability for each class is then given by the softmax function

$$P(C_j|\vec{x}) = \frac{e^{\vec{w_j}\cdot\vec{x}}}{\sum_{i=1}^{k} e^{\vec{w_i}\cdot\vec{x}}}$$

Example: softmax



- 4 different classes corresponding to the dots below the 0-line
- For each of them a corresponding softmax curve
- This expresses the probability of the observation belonging to this class
- For classification of a new observation: Choose the class with the largest probability.
- In 3D
  - A surface for each class
  - They cut each other along straight lines
    - = decision boundaries



https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html

Training Multinomial Logistic Regression

- This is done similarly to the binary task
- We skip the details

Features in Multinomial LR

- Multinomial LR constructs $P\left(C_j \middle| \vec{x}\right) = \frac{e^{\vec{w_j} \cdot \vec{x}}}{\sum_{i=1}^{k} e^{\vec{w_i} \cdot \vec{x}}}$ for each class.
- This corresponds to one linear expression $\vec{w}_i$, for each $C_i, i = 1, \dots, k$
- Alternatively, think of this
  - different features for each class:
    - notation $f_j(C, x)$ feature *j* for the class C and observation x
  - and one set of weights for the features and classes:
- In scikit-learn we write features as before and LogisticRegression constructs the match with labels during training

Today

- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

Categories as numbers

- In the naive Bayes model we could handle categorical values directly, e.g., characters:
  - What is the probability that c_n = ‘z’
- But many classifier can only handle numerical data
- How can we represent categorical data by numerical data?
- (In general, it is not a good idea to just assign a single number to each category: ~~‘a’→1, ‘b’→2, ‘c’→ 3, …)~~

Data representation

Assume the following example

	4 different featues				Classes
feature	f1	f2	f3	f4	
type	cat	cat	Bool (num)	num	
Value set	a, b, c	x, y	True, False	0, 1, 2, 3, …	Class1, class2

Representation in NLTK

```
[({'f1': 'a', 'f2': 'z', 'f3': True, 'f4': 5}, 'class_1'),
 ({'f1': 'b', 'f2': 'z', 'f3': False, 'f4': 2}, 'class_2'),
 ({'f1': 'c', 'f2': 'x', 'f3': False, 'f4': 4}, 'class_1')]
```

3 training instances

4 features

class

One-hot encoding

feature 1				feature 2	
a	b	c		x	y
(1,0,0)	(0,1,0)	(0,0,1)		(1,0)	(0,1)

- Represent categorical variables as vectors/arrays of numerical variables

Representation in scikit: "one hot" encoding

NLTK

```
[({'f1': 'a', 'f2': 'z', 'f3': True, 'f4': 5}, 'class_1'),
 ({'f1': 'b', 'f2': 'z', 'f3': False, 'f4': 2}, 'class_2'),
 ({'f1': 'c', 'f2': 'x', 'f3': False, 'f4': 4}, 'class_1')]
```

3 training instances

4 features

class

scikit

```
X_train:
array([[ 1.,  0.,  0.,  0.,  1.,  1.,  5.],
       [ 0.,  1.,  0.,  0.,  1.,  0.,  2.],
       [ 0.,  0.,  1.,  1.,  0.,  0.,  4.]])

train_target: ['class_1', 'class_2', 'class_1'], or
train_target: [1, 2, 1]
```

7 features

3 training instances

3 corresponding classes

One-hot encoding		
a	b	c
[1, 0, 0]	[0, 1,]	[0, 0, 1

Converting a dictionary

- We can construct the data to scikit directly
- Scikit has methods for converting Python-dictionaries/NLTK-format to arrays

```
» train_data = [inst[0] for inst in train]
» train_target = [inst[1] for inst in train]
» v = DictVectorizer()
» X_train=v.fit_transform(train_data)

» X_test=v.transform(test_data)
```

1. Constructs (=fit) repr. format
2. Transform

Transform
Use same v as for train

Multinomial NB in scikit

- We can construct the data to scikit directly
- Scikit has methods for converting text to bag of words arrays

```
» train_data=["en rose er en rose",
              "anta en rose er en fiol"]
» v = CountVectorizer()
» X_train=v.fit_transform(train_data)

» print(X_train.toarray())
  [[0 2 1 0 2]
  [1 2 1 1 1]]
```

- Positions corresponds to [anta, en, er, fiol, rose]

Sparse vectors

- One hot encoding uses space
- 26 English characters:
  - Each is represented as a vector with 25 ‘0’-s and a singel ‘1’
- Bernoulli NB text. classifier with 2000 most frequent words
  - Each word represented by a vector with 1999 ‘0’-s and a singel ‘1’.
- scikit-learn uses internally a dictionary-like representation for these vectors, called ”sparse vectors”

Today

- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

Generative classifiers

$$P\left(s_m \mid \langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \rangle\right) = \frac{P\left(\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \rangle \mid s_m\right) P(s_m)}{P\left(\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \rangle\right)}$$

- Naive Bayes is an example of a generative classifier
- On its way to deciding which class is most probable:
  - It estimates the probability of the observation given the class
  - It "generates" the observation with a certain probability
- For an observation:
  - which model ascribes the highest probability
  - x the probability of the model

- Example: is this picture of a dog or cat?
- To decide:
  - Generate a picture of a dog
  - i.e. make a probability distribution over all picture: how probable is it you will draw a dog like this?
- Do the same for a cat

Generating positive movie reviews

- First choose the length of the review, say *n=1000* words
- Then choose the first word
  - according to the probability distribution P(w | 'pos') e.g.
    - $\hat{P}(w = the|pos) = 0.1$
    - $\hat{P}(w = pitt|pos) = \frac{31}{798\,742}$
- Then choose word 2, etc. up to word 1000

- Observation:
  - Whether we compare to negative film reviews of positive book reviews, we will use the same features

- Footnote:
  - The multinomial text model tacitly suppress "choose length of document", and assumes it is independent of class

Discriminative classifiers

- A discriminative classifier considers the probability of the class given the observation directly.
- E.g. a discriminative text classifier may focus on the features:
  - *terrible* and *terrific* for pos. vs. neg film review
  - *director* and *author* for pos. film vs. pos. book review
- The discriminative classifier
  - may be more efficient
  - but gives less explanation
  - and may eventually focus on wrong features

Today

- Linear classifiers
- Linear regression
- Logistic regression
- Training the logistic regression classifier
- Multinomial Logistic Regression
- Representing categorical features
- Generative and discriminative classifiers
- Logistic regression vs Naïve Bayes

Logistic regression and Naive Bayes

- Both are probability-based
- In the two-class case they consider whether $P(C|\vec{x}) > P(notC|\vec{x})$
- equivalently whether $\log\frac{P(C|\vec{x})}{P(notC|\vec{x})} > 0$

Comparing NB and LogReg

- NB is a generative classifier:
  - It has a model of how the data are generated
  - $P(C)P\left(\vec{f}\middle|C\right) = P(\vec{f}, C)$
- LogReg is a discriminative classifier
  - It only considers the conditional probability $P\left(C|\vec{f}\right)$

Logistic reg. and Naive Bayes are log-linear

- whether $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})}$>0
- For NB: $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} =$

$$\log P(c_1 \mid \vec{f}) - \log P(c_2 \mid \vec{f}) = \log P(c_1) + \sum_{j=1}^{n} \log P(f_j \mid c_1) - \left(\log P(c_2) + \sum_{j=1}^{n} \log P(f_j \mid c2)\right) > 0$$

  - one particular linear expression,
- For LR: $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
  - the linear expression that fits the training data best

Naive Bayes is an instance of log-linear

- LR: $\log \frac{P(C|\vec{x})}{P(notC|\vec{x})} = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$
- NB: $\log P(c_1 \mid \vec{f}) - \log P(c_2 \mid \vec{f}) = \log P(c_1) + \sum_{j=1}^{n} \log P(f_j \mid c_1) - \left(\log P(c_2) + \sum_{j=1}^{n} \log P(f_j \mid c2) \right) > 0$
- Where:
  - $w_0 = P(c_1) - P(c_2)$
  - $w_i = P(f_j \mid c_1) - P(f_j \mid c_2)$

Comparing NB and LogReg

- NB is an instance of LogReg,
  - i.e. one possible choice of weights
- LogReg will do at least as well as NB on the training data
  - (without any smoothing)
- When the independence assumptions of NB holds, NB will do as well as LogReg
- When the independence assumptions does not hold, NB may put too much weight on some features
- LogReg will not do this: If we add features that depend on other features, LogReg will put less weight on them

Ablation studies

- One way to see which features are important for LogReg
- Start with a classifier which uses many features
- Remove one feature f1, retrain and see whether it has an effect
- Remove another feature f2, instead of f1 or in addition to f1, and study the effect
- Beware of the possibility:
  - Removing f1 only has little effect
  - Removing f2 only has little effect
  - Removing both f1 and f2 might have a large effect
  - Why is this so?