# 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.

# Notation

When considering numerical features, we use

* (x1, x2, …, xn) for the features, where
  + Each feature is a number
  + A fixed order is assumed
* Y for the output value/class
* In particual, Jurafsky and Mratin use
  + ^y for the predicted value of the learner, ^y = f (x1, x2, …, xn)
  + Y for the true value

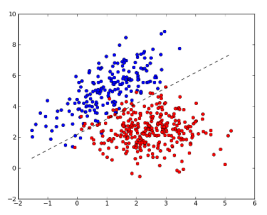
# Machine Learning

* In NLP, we often consider thousands of features (dimensions) and categorical data
* These are difficult to illustrate by figures
* To understand ML algorithms
  + it is 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

# Classifiers – two classes

* Many classification methods are made for two classes and then generalize to more classes
* The goal is to find a curve that separates the two classes: the **decision boundary** or a **(hyper-) surface** with more dimensions

# Linear Classifiers

* Try to find a **straight** **line** that separates the two classes (in 2 dimensions)
* 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 on unseen data
* Decision boundary
  + A line has the form ax + by + c = 0
  + ax + by < -c for the red points
  + ax + by >-c for the blue points

# 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

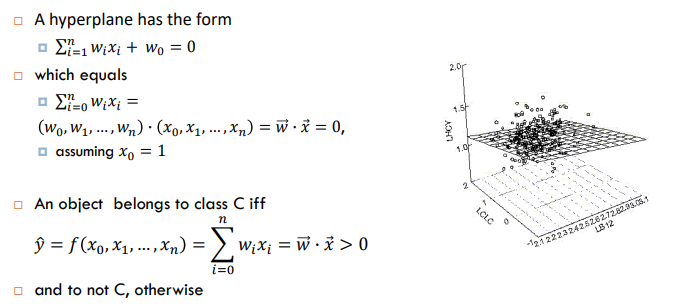
# More dimensions

* 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**

# Higher dimension

* With one variable, consider
  + ax + b
  + alternatively, write it w0 + w1x1
* With two variables, consider
  + w0 + w1x1 + w2x2
* Vector form:
  + w0 + w1x1 + w2x2 = (w0, w1, w2) . (1, x1, x2) where we add an extra feature x0 = 1 to each observation (dot product: multiply first element with first, second with second…)

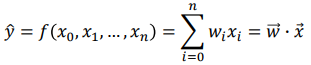
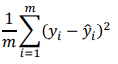
# Linear Classifiers: n dimensions



# Linear Regression

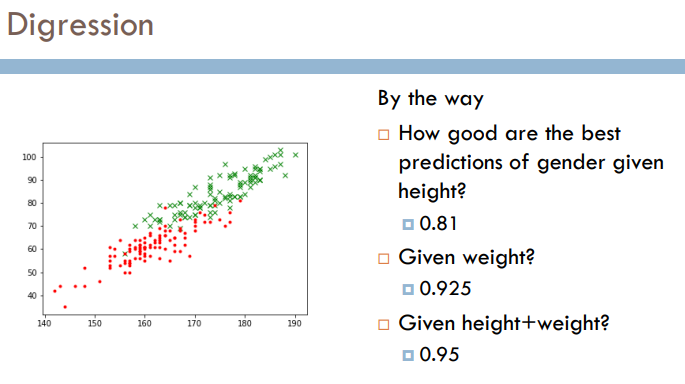
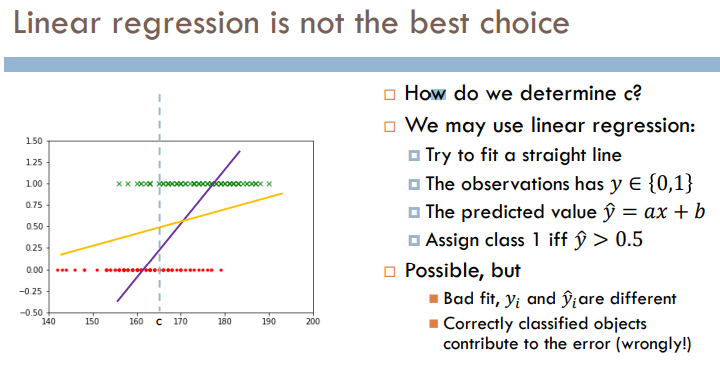
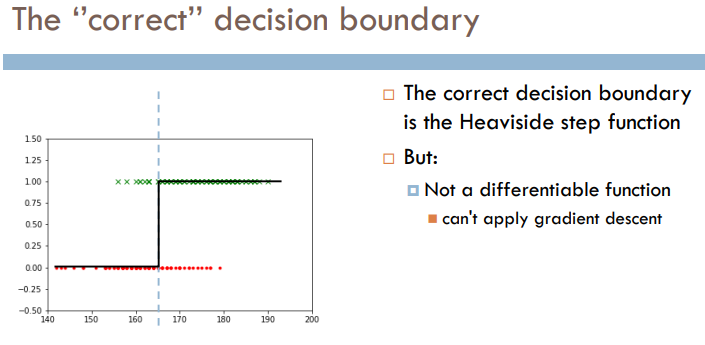
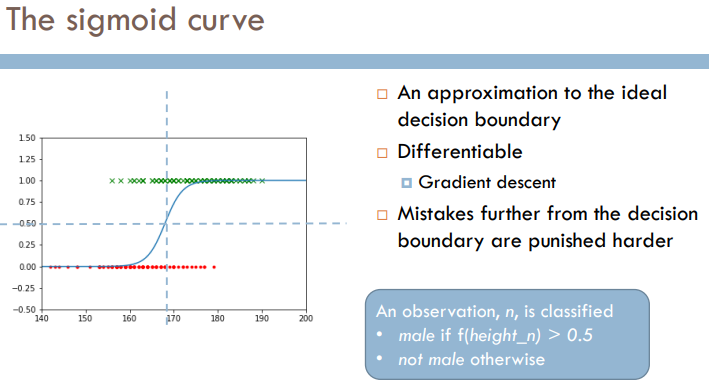
* Data: 100 males: height and weight
* Goal: guess the weight of other males when you only know the height

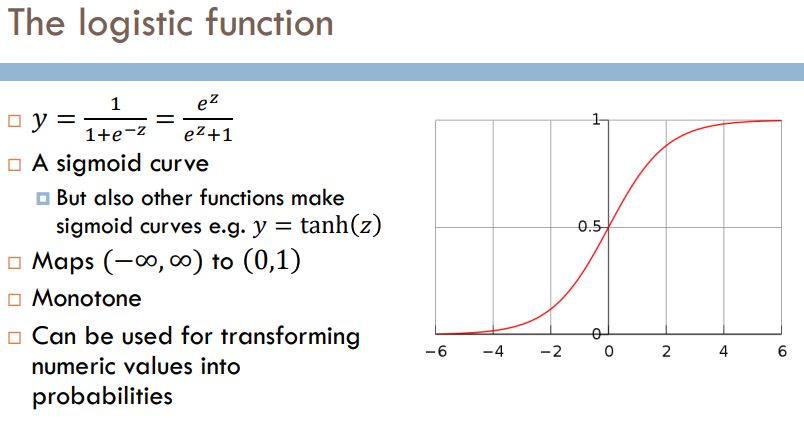
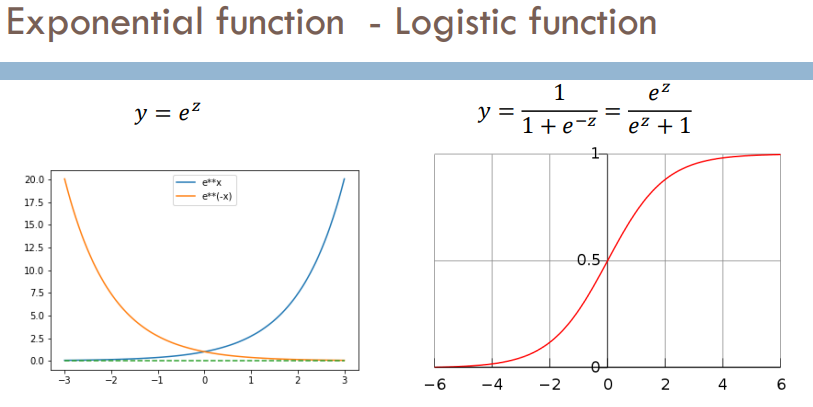
# Linear Regression

* Method:
  + Try to find 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 yi (green points)
    - To the corresponding predicted value ^yi (on the red line)
  + We define a **loss function**
    - Which measures the discrepancy between the yi-s and the ^yi-s
  + The goal is to **minimize the loss**
* Loss for linear regression
  + For linear regression we usually use the **Mean Square Error**, where di = (yi - ^yi) and ^yi = (axi + b)
  + Mean Square Error:
  + Why squaring?
    - To not get 0 when we sum the differences
    - Large mistakes are punished more severely
* Learning = minimizing the loss
  + For linear regression there is a formula, but this is very slow with many features
  + Alternative
    - Start with one candidate line
    - Try to find better weights
    - 🡪 Search Problem, use **Gradient Descent**
* Higher dimensions
  + Linear regression of more than two variables works similarly
  + We try to fit the best (hyper-) plane 🡪
  + We can use the same mean square error

# From Regression to Classification

* Goal: predict gender from two features – height and weight





# Logistic Regression

* 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 | x ) > 0.5
* We could also choose other cutoffs, e.g. if the classes are not equally important
* With two features x1, x2
* Apply weights: w0 + w1x1 + w2x2
* Let y = w0 + w1x1 + w2x2
* Apply the logistic function σ and check whether
* Geometrically: Folding a plane along a sigmoid  
    
  The decision boundary is the intersection of this surface and the plane 0.5: a straight line
* How to find the best curve?
  + What are the best choices of *a* and *b* in ?
  + Geometrically, a and b determine the curve’s
    - Midpoint: x =
    - Steepness: larger *a* 🡪 steeper curve

# Learning in the Logistic Regression Model

* A training instance consists of
  + A feature vector 
  + A label (class) y, which is 1 or 0
* With a set of weights , the classifier will assign

to this training instance 

* + Where P (C = 0 | ) = 1 - ^y
* Goal: find  that maximizes P (C = 0 | ) of all training instances

# Loss Function

* In machine learning, we have to determine an objective for the training
* We can do this 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

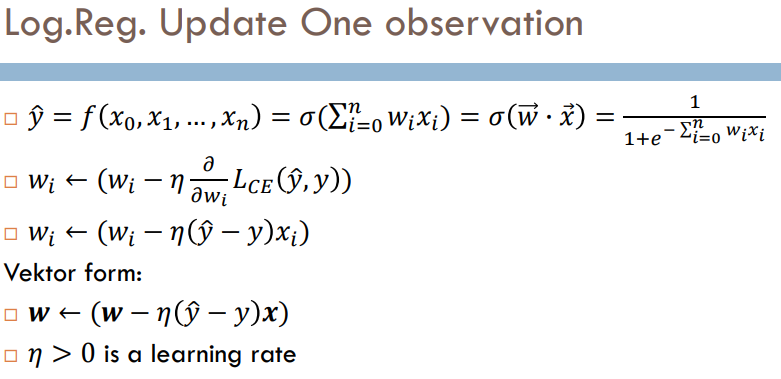
* The underlying idea is that we want to maximize the joint probability of all the predictions we make 
* This is the same as maximizing 
* This is the same as minimizing 

Which is an instance of what is called the cross-entropy loss

# Gradient Descent

* We use the derivate of the loss function to point in which direction to move
* We are approaching a unique global minimum
* To minimize the loss function, we can use gradient descrent
* The gradient (= partial derivates of the loss function) tells us in which direction we should move, i.e. where the steepest direction is
* Good news: the loss function is convex, you are not stuck in local minima

# Logistic Regression: Update One Observation

* use the sigmoid function on the sum of the weighted features

# Variations of Gradient Descent

|  |  |
| --- | --- |
| **Batch Training** | **Stochastic Gradient Descent** |
| * Calculate the loss for the whole training set * Make one move in the correct direction * Repeat an epoch * Can be slow | * 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 * Overall effect may be good * Can be faster |
| **Mini-batch Training** | **Comparision** |
| * pick up a subset of the training set of a certain size * calculate the loss for this subset * make on emove in the direction of this gradient * repeat an epoch * a good compromise between the two extremes above |  |

# Solvers / Optimizers

* There are various different solvers and optimizers for gradient descent
* Observe that you may specify between solvers in scikit-learn

## **Regularization**

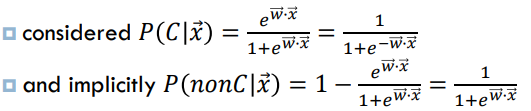
* LogReg is prone to overfitting to the training data
* Hence apply regularization 
* Regularization **punishes large weights**
* Most common is L2-regularization R(W) =
* Alternative: L1-regularization R(W) =

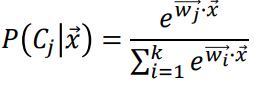
# Scikit-Learn: LogisticRegression

* LogisticRegression(penalty=’12’, …, 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

# Multinomial Logistic Regression

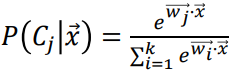
* Also called **Maximum Entropy (maxent) Classifier**, or softmax regression
* With one class we



* We now consider a linear expression  for each class *Ci, i = 1, …, k*
* The probability for each class is then given by the **softmax function 🡪**

# Example: Softmax

# Features in Multinomial Logistic Regression

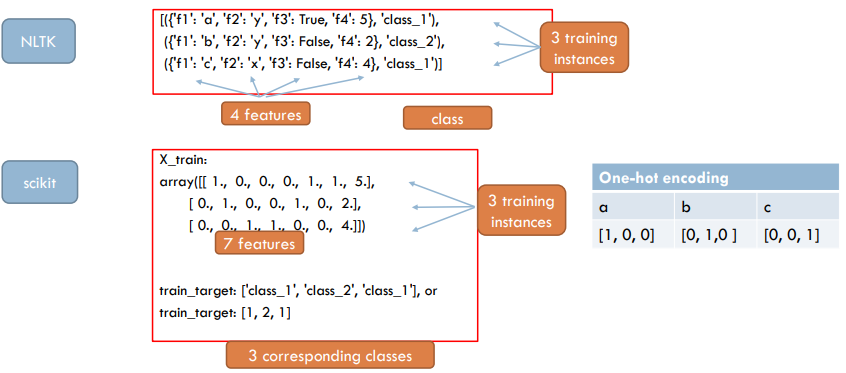
* Multinomial LR constructs  for each class
* This corresponds to one linear expression  , for each *Ci , i = 1, …, k*
* Alternatively think of this
  + Different features for each class: notation *fI (C, x)* feature *j* for the class C and observation x
  + And one set of weights for the features and classes
* In Scikit-learn, we write features as before and LogisticRegression constructs the match with labels during training

# Categories as Numbers

* In the Naïve Bayes model, we could handle categorical values directly, e.g. with characters: “What is the probability that c\_n = ‘z’?”
* But many classifiers 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!

# One-Hot Encoding

* Represent categorical variables as vectors / arrays of numerical variables
* Representation in scikit: “one hot” encoding:



# Converting a dictionary

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

# Multinomial NB in Scikit

* We can construct the data to scikit directly
* Scikit has methods for converting text to bag of words arrays
* Position 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 0s and a single 1
* Bernoulli NB text. Classifier with 2000 most frequent words: each word represented by a vector with 1999 0s and a single 1
* Scikit-learn uses internally a dictionary-like representation for these vectors, called “sparse vectors”