SDU Summer School

Deep Learning

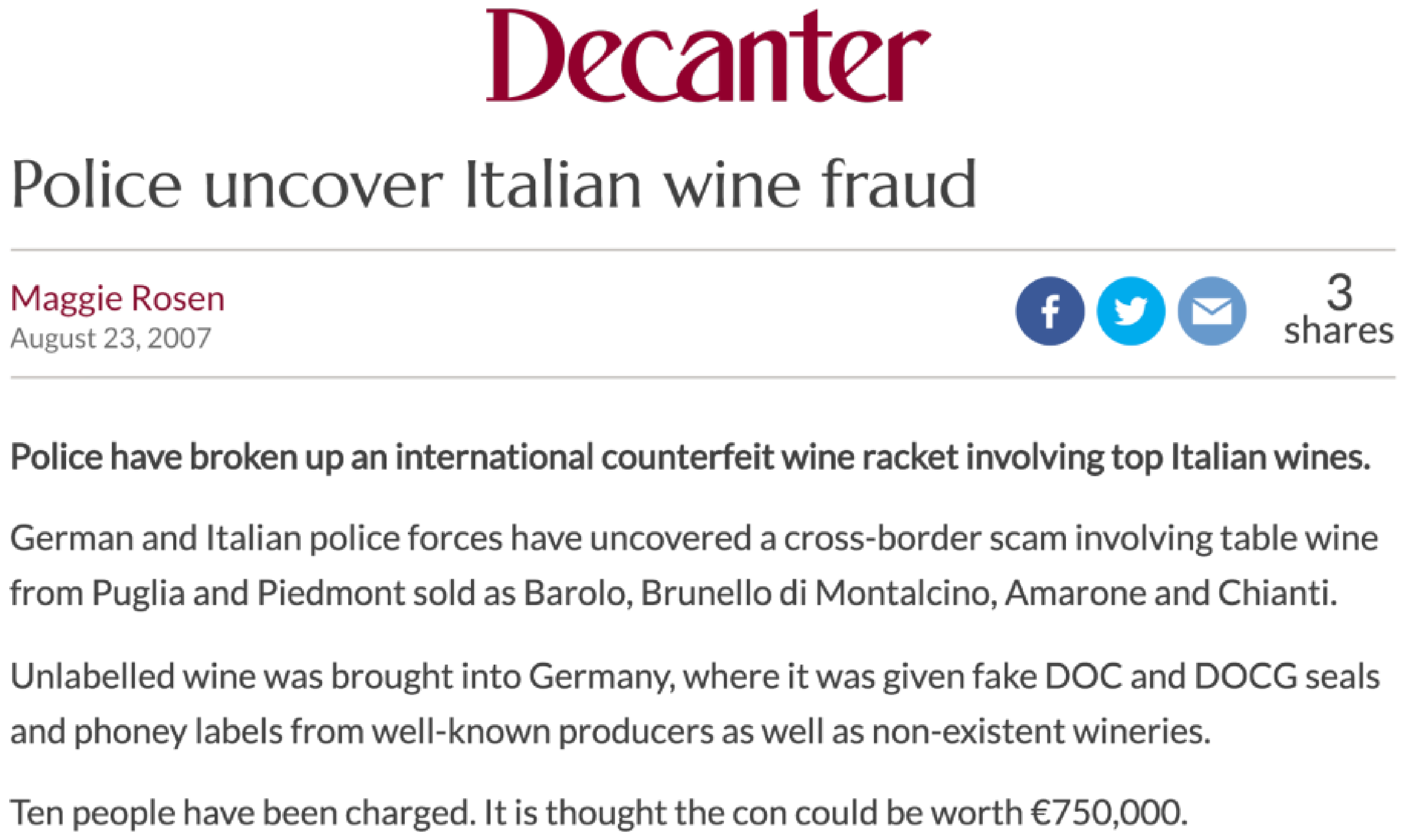
Summer 2022

Welcome to the Summer School

Logistic Regression & Output Functions

- Generative Models
- Discriminative Models & Logistic Regression
- Non-binary Classification

One Example


Decanter

Police uncover Italian wine fraud

Maggie Rosen
August 23, 2007

3 shares

Police have broken up an international counterfeit wine racket involving top Italian wines.

German and Italian police forces have uncovered a cross-border scam involving table wine from Puglia and Piedmont sold as Barolo, Brunello di Montalcino, Amarone and Chianti.

Unlabelled wine was brought into Germany, where it was given fake DOC and DOCG seals and phoney labels from well-known producers as well as non-existent wineries.

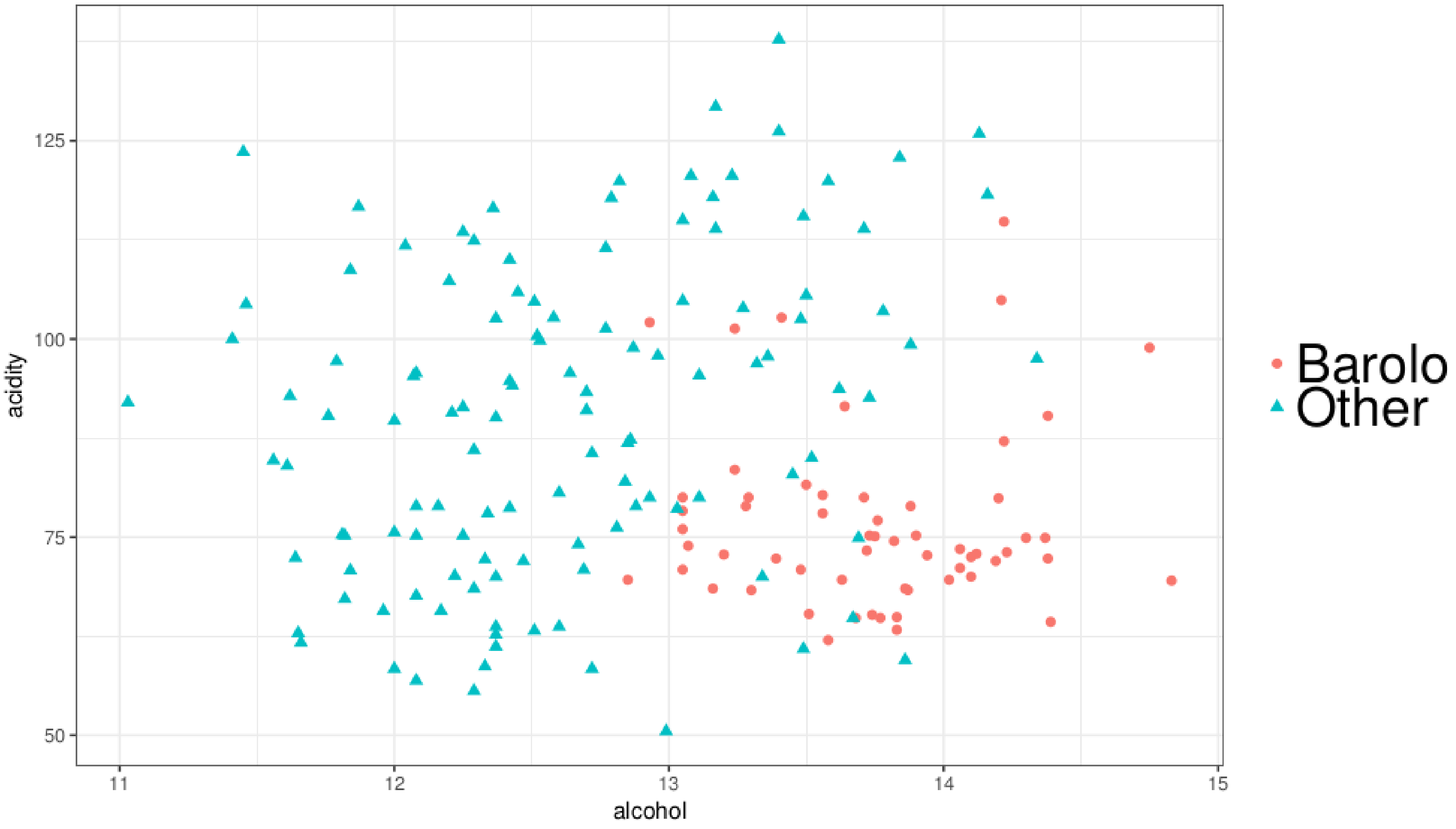
Ten people have been charged. It is thought the con could be worth €750,000.

- The following slides are adapted from Pierre-Alexandre Mattei and his "Machine Learning" course

One Example

- Would it be possible to create an AI device that learns to estimates the probability that a bottle of wine is fake or not?
- The device would need some information about the wine, for example some chemical properties:
  - alcohol content
  - acidity
- One of the wines the bad guys counterfeited was from the Barolo region.
  - Those wines have "pronounced tannins and acidity", and "moderate to high alcohol levels (Minimum 13%)". (Wikipedia)
  - So, can we use a computer to do so?

Our Dataset



Probabilistic Model for Classification

- There are two schools of modelling when it comes to building a probabilistic model of our data $\{x_n, t_n\}$.
- Both make the assumption that the data are sampled from a distribution $p(x,t)$, or equivalently $p(x, C_k)$.
- The **generative school** builds models for the joint distribution $p(x,t)$.
  - This is quite hard, because this means we'll have in particular to learn the distribution of the data $p(x)$, which can be very complicated.
- The **discriminative school** builds models for the conditional distribution $p(t|x)$.
  - This is much easier, because we don't care about $p(x)$, but this is also often less powerful (in particular, all the training set needs to be labelled).

A Generative Model

- So, we want to define a distribution $p(x, C_k)$ Using the product rule, we can write

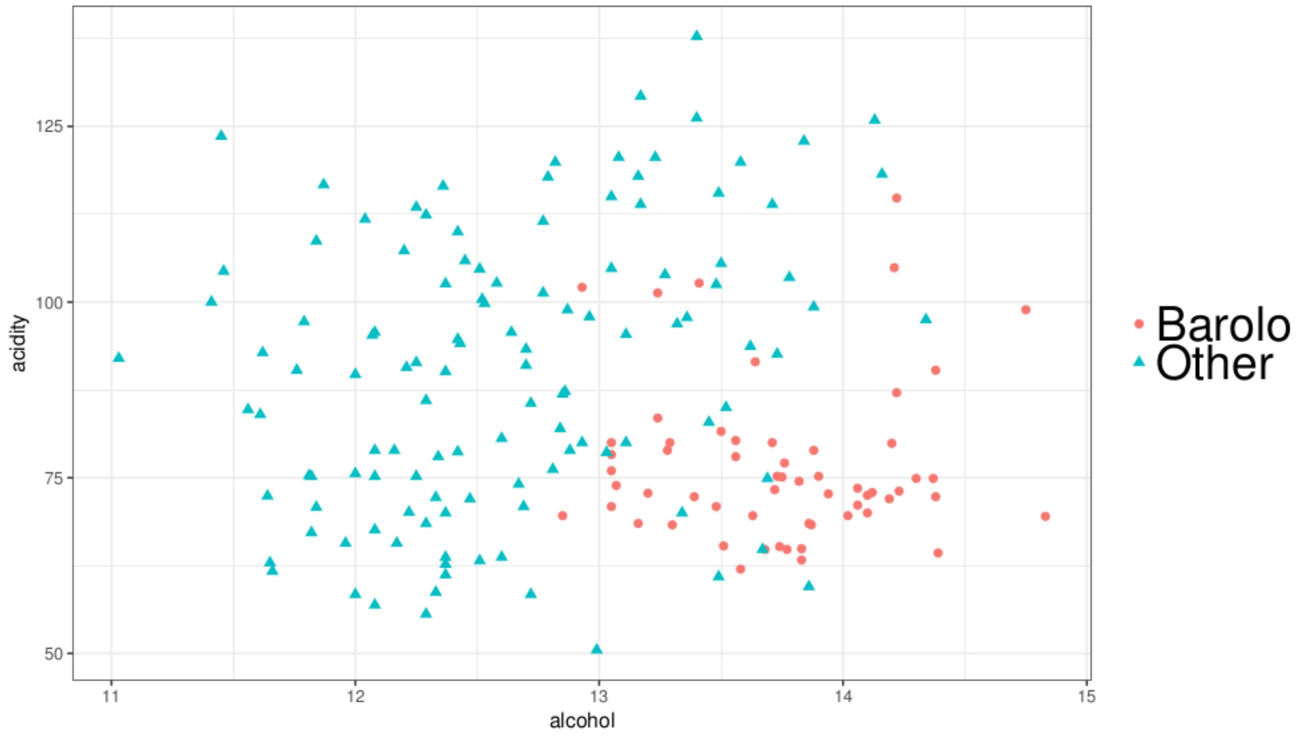
$$p(x, C_k) = p(x|C_k)p(C_k)$$

- This involve two terms:
  - The class conditional densities $p(x|C_k)$, which are the K different distributions $p(x|C_1), \ldots, (x|C_K)$ of the K classes.
  - The class prior probabilities $p(C_k)$, which are the proportions $p(C_1), \ldots, (C_K)$ of the classes in the population.

- Of course, both $p(x|C_k)$ and $p(C_k)$ will need to be learned, but let's assume that we just know them.

A Generative Model

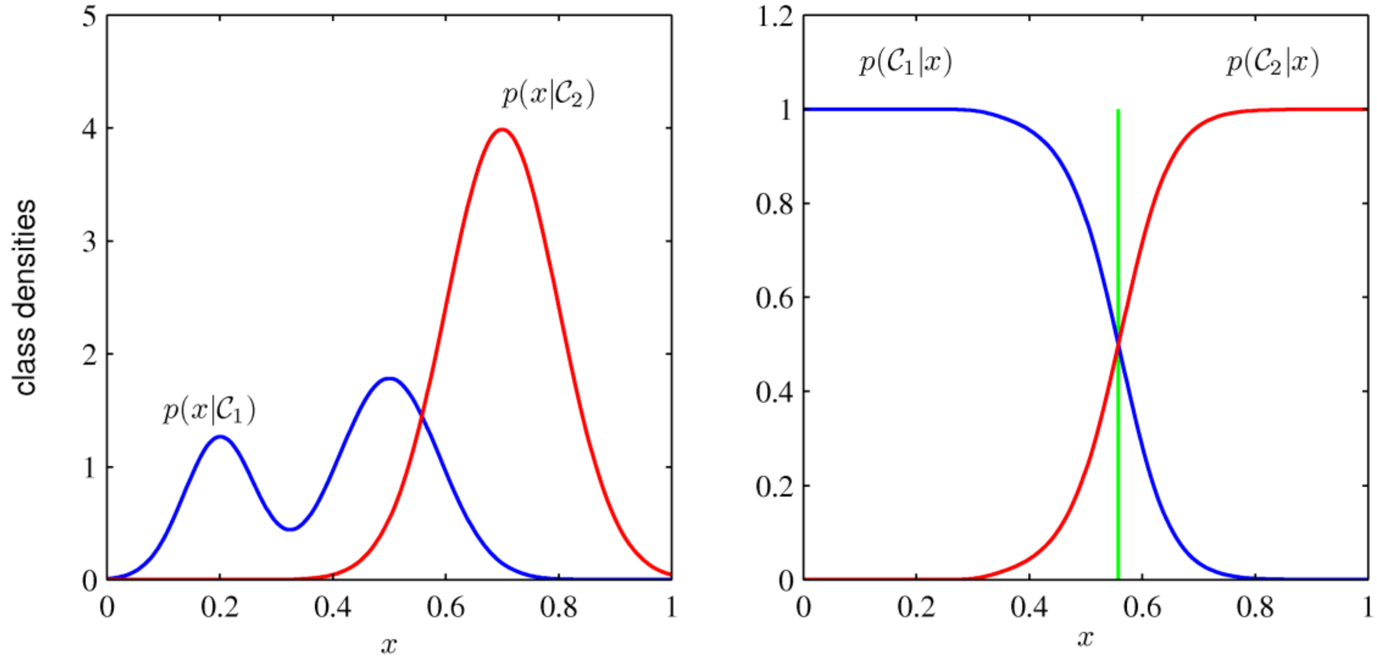
- Let's assume we have learned $p(x|C_1)$ (density of the Barolos), $p(x|C_2)$ (density of the other wines), $p(C_1)$ (proportion of Barolos), $p(C_2)$ (proportion of the other wines).



Estimate Classes

- Our main goal is to estimate the probability that the bottle is truly a Barolo. Using Bayes:

$$p(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_1)p(C_1) + p(\boldsymbol{x}|C_2)p(C_2)}$$



- Image: Simplified view on only one variable

Closer Look to the Formula

$$p(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_1)p(C_1) + p(\boldsymbol{x}|C_2)p(C_2)}$$

- It only involves the class-conditional densities and the class proportions. Rewrite:

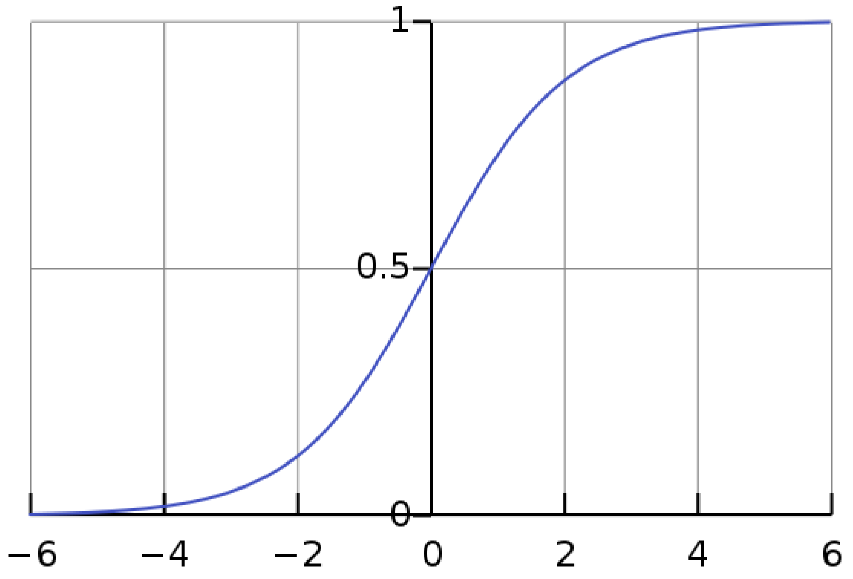
$$p(C_1|\boldsymbol{x}) = \sigma\big(a(\boldsymbol{x})\big), \text{with } a(\boldsymbol{x}) = \ln \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_2)p(C_2)} = \ln \frac{p(\boldsymbol{x}, C_1)}{p(\boldsymbol{x}, C_2)}$$

Closer Look to the Formula

$$p(C_1|\boldsymbol{x}) = \sigma\big(a(\boldsymbol{x})\big), \text{with } a(\boldsymbol{x}) = \ln\frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_2)p(C_2)} = \ln\frac{p(\boldsymbol{x},C_1)}{p(\boldsymbol{x},C_2)}$$

- With σ being the logistic sigmoid function

$$\sigma(a) = \frac{1}{1+\exp(-a)} = \frac{e^x}{e^x+1}$$



The Sigmoid Function

- Why on earth do we need this weird sigmoid function?

- Interpretation: we can interpret $a(x)$ as the amount of evidence about x being a true Barolo, because it's the logarithm of the ratio of probabilities of being true and being fake:
  - using Bayes's rule, we can see that $a(\boldsymbol{x}) = \ln \frac{p(C_1|\boldsymbol{x})}{p(C_2|\boldsymbol{x})}$

- Unification of the generative and discriminative schools: using the logistic function makes a bridge between the generative methods, and the discriminative ones and neural networks … you will see soon

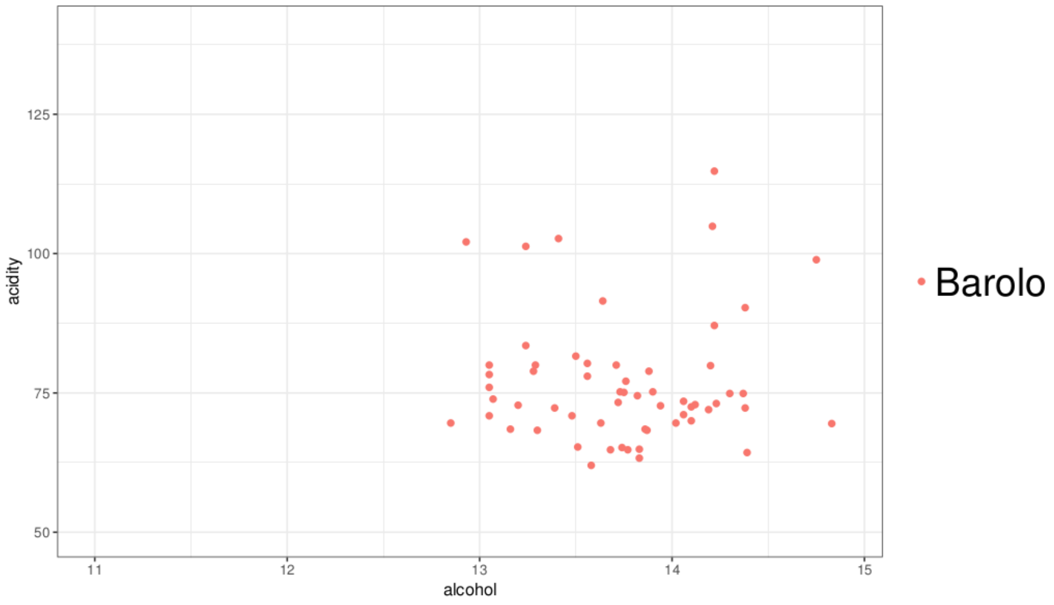
What we have done, what we still have to do...

- What we have done.
  - If we know the class-conditional densities $p(x|C_k)$ and the class proportions $p(C_k)$, we can compute the posterior probabilities $p(C_k|x)$, and use them to classify data.

- What we still have to do. How can we find the class-conditional densities and the proportions?
  - We can choose a parametrized model family
  - For that we could use for instance Gaussian Mixture Models
  - Then use MLE to find the best parameters for a Gaussian

Maximum Likelihood Estimators

- Let's look at the Barolos:



- We now need to fit a Gaussian that describes the data best:

$$N(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sqrt{\frac{1}{(2\pi)^2|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

Maximum Likelihood Estimation

- The Maximum Likelihood Estimation (MLE) is a method of estimating the parameters of a model. This estimation method is one of the most widely used.

- The method of maximum likelihood selects the set of values of the model parameters that maximizes the likelihood function. Intuitively, this maximizes the "agreement" of the selected model with the observed data.

- The Maximum-likelihood Estimation gives an unified approach to estimation.

What is a Maximum Likelihood Estimator?

- The likelihood that one datapoint was generated by any distribution function is given by

$$l(\boldsymbol{x}, \boldsymbol{\theta}) = p(\boldsymbol{x}; \boldsymbol{\theta})$$

- For the entire dataset, we have the likelihood

$$L(X, \boldsymbol{\theta}) = \prod_{i=1}^{n} p(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

What is a Maximum Likelihood Estimator?

- Goal of the MLE is to find $\boldsymbol{\theta}$ in such a way that $L(X, \boldsymbol{\theta})$ is maximized

$$\boldsymbol{\theta}_{\text{MLE}} = \text{argmax}_{\theta}\, L(X, \boldsymbol{\theta})$$

- Very often (for simplicity), the log likelihood is used

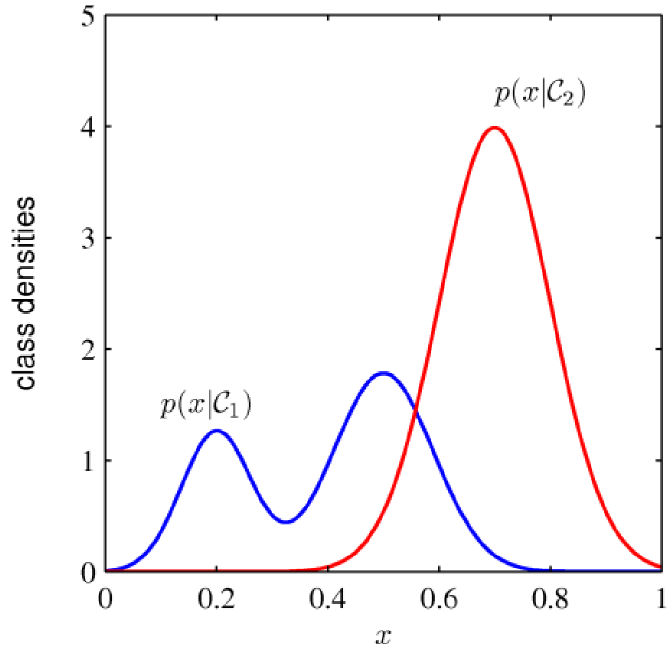
$$\boldsymbol{\theta}_{\text{MLE}} = \text{argmax}_{\theta} \log \prod_{i=1}^{n} p(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) =$$

$$= \text{argmax}_{\theta} \sum_{i=1}^{n} \log p(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

  - This doesn’t change the maximum of the function
  - Eases many calculations

Example: MLE for a Gaussian

- For the sake of simplicity, we only approximate a one-dimensional gaussian for the Barolo, i.e., we just concentrate on one feature



$$\frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

Example: MLE for a Gaussian

- Target: Estimate μ and σ for a Gaussian. The likelihood is

$$L(X|\theta) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

- Taking the log results in an easier version:

$$LL(X|\theta) = -\frac{1}{2} n \log 2\pi - n \log \sigma - \sum_{i=1}^{n} \frac{(x-\mu)^2}{2\sigma^2}$$

Example: MLE for a Gaussian

- Derivate for μ:

$$\frac{\partial LL}{\partial \mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu)$$

- And for σ^2:

$$\frac{\partial LL}{\partial \sigma^2} = \frac{1}{2\sigma^2}\left[\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2 - n\right]$$

- These derivates are 0 if

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{and} \qquad \sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Result

- The Estimator works the same in the multi-dimensional case, except the math gets a bit more iffy (Matrices)

- Now, we have a model for the Barolos
  - We Could now fit a Gaussian over the other wines
  - We calculate our priors, i.e., class prior probabilities, which is simply the ratio of Barolos of all wines in our dataset

- We have everything to compute:

$$p(C_1|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_1)p(C_1)}{p(\boldsymbol{x}|C_1)p(C_1) + p(\boldsymbol{x}|C_2)p(C_2)}$$

Logistic Regression & Output Functions

- Generative Models
- Discriminative Models & Logistic Regression
- Non-binary Classification

From generative to discriminative models

- We defined a distribution $p(\boldsymbol{x}, C_k)$ over features and classes. We saw how to learn it using the decomposition

$$p(\boldsymbol{x}, C_k) = p(\boldsymbol{x}|C_k)p(C_k)$$

- Using this distribution, we were able to compute the posterior probabilities of the classes $p(C_k|\boldsymbol{x})$, and to use them for classification.
- But, the only thing that we actually need for classification are those posterior probabilities.

- **So, could we avoid having to compute $p(x, C_k)$ and directly target the posterior probabilities?**

From generative to discriminative models

- The answer to this question is the general rationale of discriminative models:

Learn a way to transform any input vector x into its posterior class distributions

$$p(C_1|x), \dots, p(C_K|x)$$

From generative to discriminative models

- So, we're looking for a way to transform any input vector x into its posterior class distributions $p(C_1|x), \dots, p(C_K|x)$.
  - For simplicity, let's start with the binary case ($K = 2$).

- We have already seen:

$$p(C_1|\boldsymbol{x}) = \sigma\big(a(\boldsymbol{x})\big), \text{with } a(\boldsymbol{x}) = \ln \frac{p(x, C_1)}{p(x, C_2)}$$

From generative to discriminative models

- So, with generative models, we have the formula $p(C_1|x) = \sigma\left(a(x)\right)$, where the function $a(x)$ comes from the class proportions and the model $p(x, C_k)$.

- The main idea of discriminative modelling will be to forget that $a(x)$ comes from a generative model, and to rather treat $a(x)$ as an unknown function to be learned from the data.

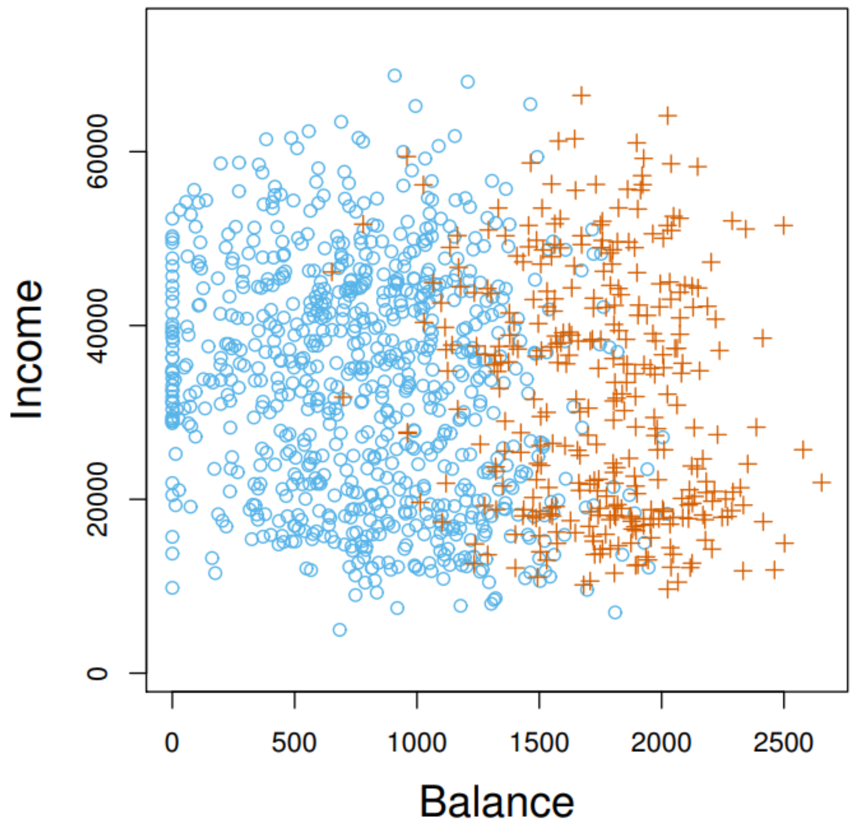
Say Hi to the Big Family of Discriminative Models

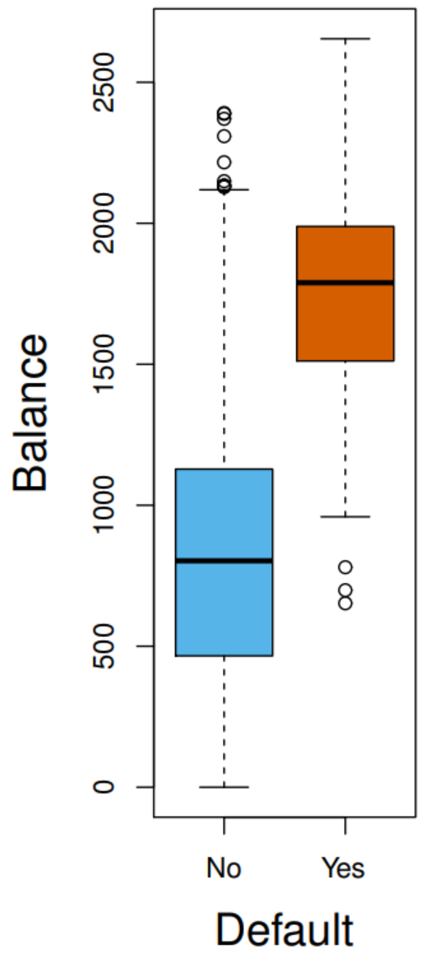
- The General model is

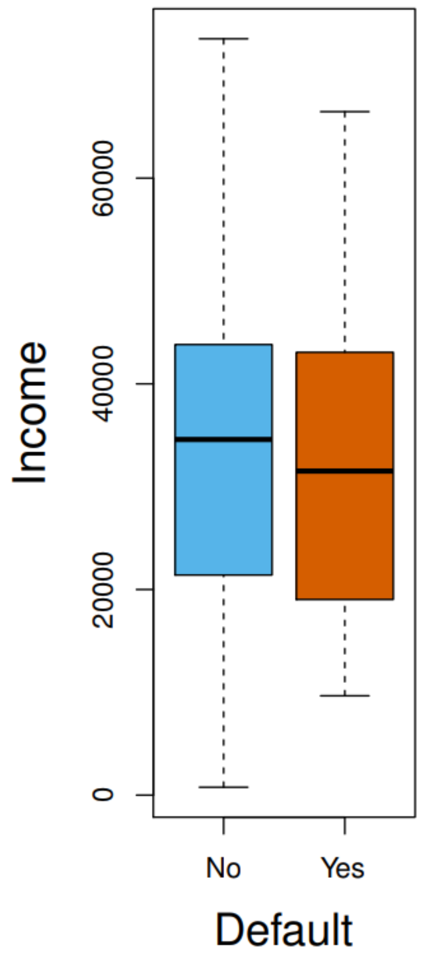
$$p(C_1|\boldsymbol{x}) = \sigma\big(a(\boldsymbol{x})\big)$$

- it would be impractically hard to learn any kind of function without making additional assumptions:
  - **Logistic regression** assumes that the function a is linear
  - **Deep learning** assumes that the function a is a composition of many linear and nonlinear functions
  - **Gaussian processes** and **nonparametric methods** more generally, try to make as few assumptions as possible regarding a

Example for Logistic Regression: Credit Card Default







Logistic Regression

- In Logistic regression we model $a(\boldsymbol{x})$ as a linear regression

$$\beta_0 + \beta_1 x$$

- Put everything together, with $\boldsymbol{\theta} = (\beta_0, \beta_1)^T$:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

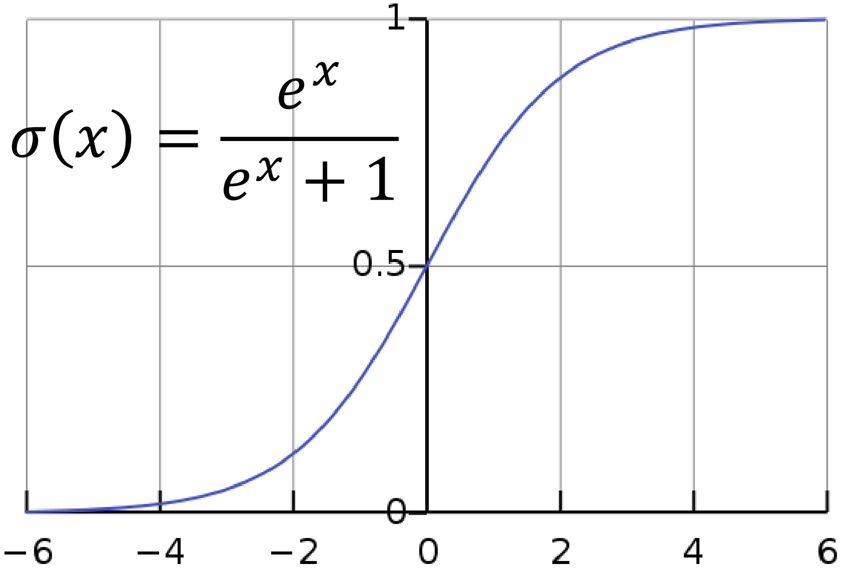
Learning Goal

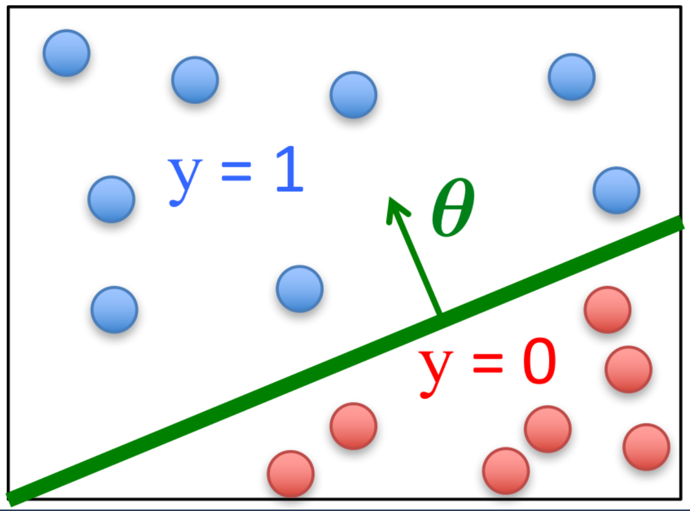
- Our Model:
  $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = p(y|\boldsymbol{x},\boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^T\boldsymbol{x})$

- $\boldsymbol{\theta}^T\boldsymbol{x}$ should be large negative for negative instances

- $\boldsymbol{\theta}^T\boldsymbol{x}$ should be large positive for positive instances

- Select a threshold t and
  - Predict $y = 1$ if $p(y|\boldsymbol{x},\boldsymbol{\theta}) \geq t$
  - Predict $y = 0$ if $p(y|\boldsymbol{x},\boldsymbol{\theta}) < t$





Logistic Regression Objective Function

- We cannot just use the least squared approach as with linear regression

$$J(\theta) = \frac{1}{n}\sum_{i}^{n}\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2$$

  - Since the logistic regression model with the sigmoid function results in a non-convex optimization problem.
  - And due to other problems would not lead to the optimal parameter set

Remember MLE

- The Maximum Likelihood Estimation (MLE) is a method of estimating the parameters of a model. This estimation method is one of the most widely used.

- The method of maximum likelihood selects the set of values of the model parameters that maximizes the likelihood function. Intuitively, this maximizes the "agreement" of the selected model with the observed data.

- And it worked like a charm with Gaussian ☺

Computing the Likelihood

- Let's calculate the log-likelihood:

$$\ell(a) = \sum_{i=1}^{n} \log p\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) =$$

$$= \sum_{i=1}^{n} \log \left[h_\theta\left(\boldsymbol{x}^{(i)}\right)^{y^{(i)}} \cdot \left(1 - h_\theta\left(\boldsymbol{x}^{(i)}\right)\right)^{1-y^{(i)}} \right] =$$

$$\sum_{i=1}^{n} \left[y^{(i)} \log h_\theta\left(\boldsymbol{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(\boldsymbol{x}^{(i)}\right)\right) \right]$$

Computing the Likelihood

- We have

$$\sum_{i=1}^{n}\left[y^{(i)}\log h_\theta\left(\boldsymbol{x}^{(i)}\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

- We want to maximize this function, which is equivalent to minimizing its opposite, which is called the cross-entropy loss:

$$J(\theta)=-\sum_{i=1}^{n}\left[y^{(i)}\log h_\theta\left(\boldsymbol{x}^{(i)}\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

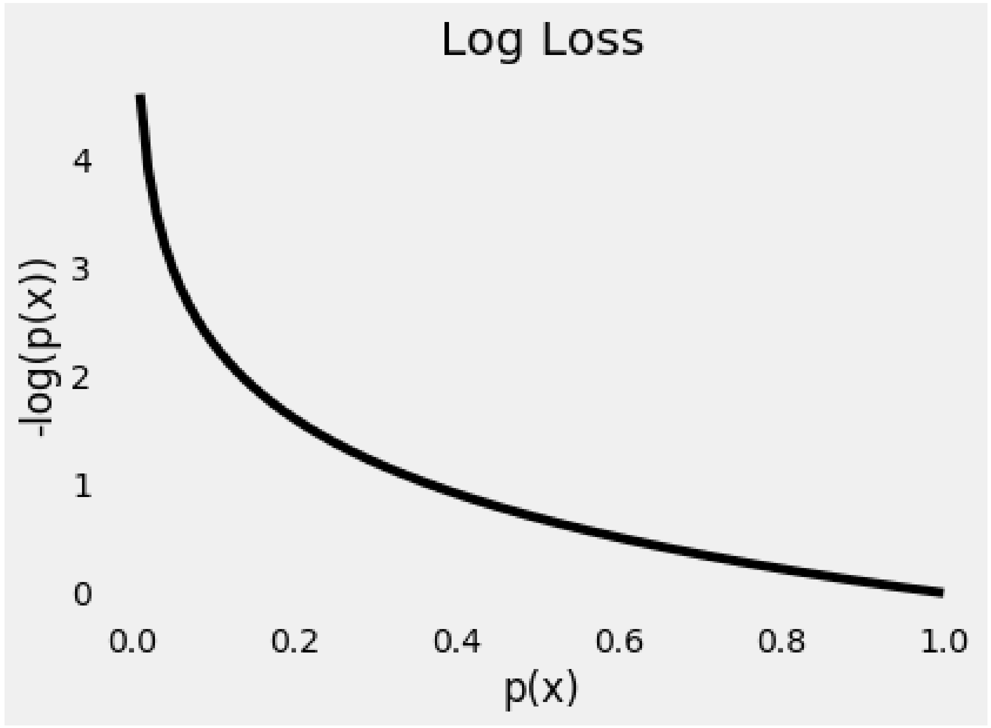
  - The cross-entropy loss is the most commonly used loss for neural networks, and is a way of doing maximum likelihood without necessarily saying it.

Intuition Behind the Objective

$$J(\theta) = -\sum_{i=1}^{n}\left[y^{(i)}\log h_\theta\left(\boldsymbol{x}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(\boldsymbol{x}^{(i)}\right)\right)\right]$$

- Depending on y, the loss function assumes these values:

$$J(\boldsymbol{\theta}) = \begin{cases} -\log\left(h_{\boldsymbol{\theta}}(\boldsymbol{x})\right) & \text{for} \quad y = 1 \\ -\log\left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})\right) & \text{for} \quad y = 0 \end{cases}$$



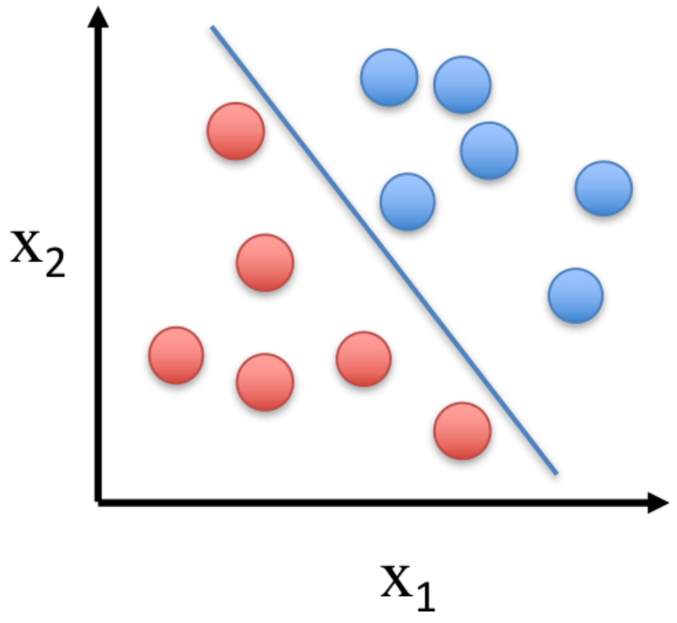
- Effect: Extremely negative with wrong classifications
- Trained with gradient descent methods

Logistic Regression & Output Functions

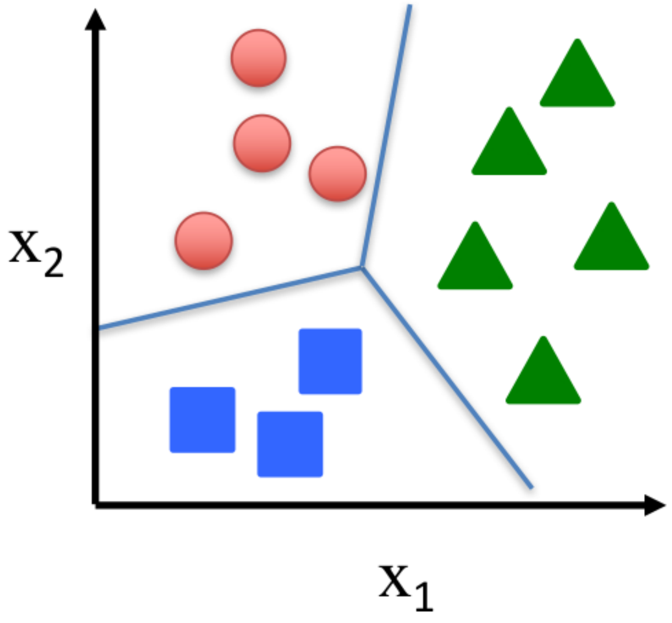
- Generative Models
- Discriminative Models & Logistic Regression
- Non-binary Classification

Multi-Class Classification

Binary classification:



Multi-class classification:



Disease diagnosis:	healthy / cold / flu / pneumonia
Object classification:	desk / chair / monitor / bookcase

Multi-Class Problems

- Can we find similar equations for more than two classes?
  - We assume that we know $p(x|C_1), p(x|C_2),\ p(x|C_3)$ …
- Bayes again:

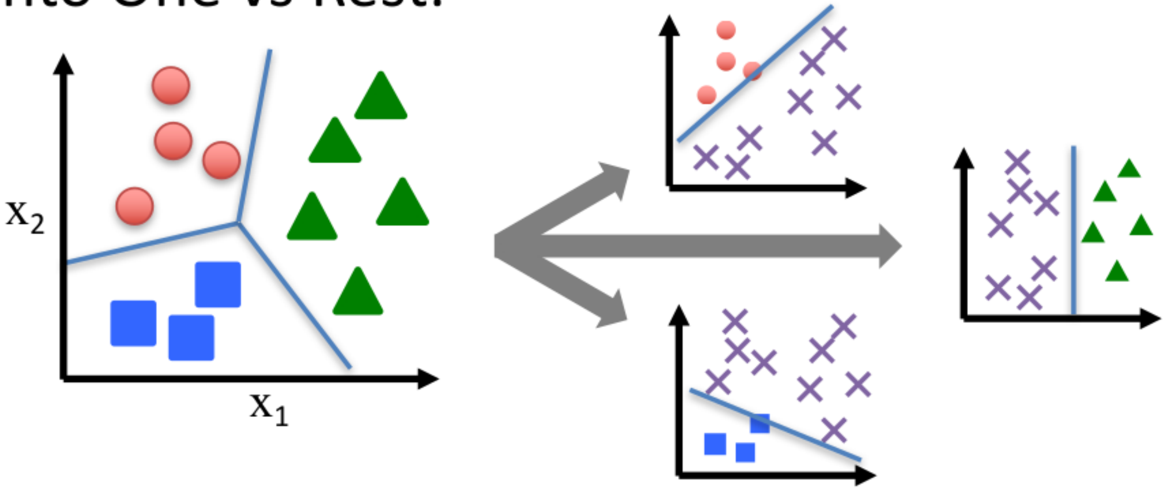
$$p(C_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|C_k)p(C_k)}{\sum_j^K p(x|C_j)p(C_j)}$$

- In this case we don't use the sigmoid function, but we can compute all posterior probabilities using the **softmax function** (aka normalized exponential), that's defined as:

$$\text{softmax}(a_1, \dots, a_K) = \left(\frac{\exp a_1}{\sum_j^K \exp a_j}, \frac{\exp a_2}{\sum_j^K \exp a_j}, \dots, \frac{\exp a_K}{\sum_j^K \exp a_j}\right)$$

Multi-Class Classification

Split into One vs Rest:



- Train a logistic regression classifier for each class i to predict the probability that $y = i$ with

$$h_c(\boldsymbol{x}) = \frac{\exp(\boldsymbol{\theta}_c^\top \boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{\theta}_c^\top \boldsymbol{x})}$$

Some Notes to Softmax

- For all possible $a_1, \ldots, a_K$, it's easy to check that the coefficients of $\mathrm{softmax}(a_1, \ldots, a_K)$ are ≥ 0 and sum to one: in other words, it always outputs a proper probability distribution functions over the classes K.

- Its name comes from the fact that it can be seen a smoothed version of the maximum function. Why?
  - Because, if we have some $a_1, \ldots, a_K$ and that one of them is much bigger than the others – say a_{j_0} – then, we'll have $[\mathrm{softmax}(a_1, \ldots, a_K)]_{j_0} \approx 0$ and for all other ≈ 0.