Machine Learning

Lecture 4: Linear Classification

Prof. Dr. Stephan Günnemann

Data Mining and Analytics
Technical University of Munich

12.11.2018

Reading material

Reading material

- Bishop: chapters 4, 4.1.1, 4.1.2, 4.1.7, 4.2, 4.3.0 - 4.3.4

Acknowledgements

- Slides are based on an older version by G. Jensen and J. Bayer
- Some figures are from C. Bishop: "Pattern Recognition and Machine Learning"

Notation

Symbol	Meaning
s	scalar is lowercase and not bold
$\boldsymbol{s}$	vector is lowercase and bold
$\boldsymbol{S}$	matrix is uppercase and bold
$\hat{y}$	predicted class label
y	actual class label
$\mathbb{I}(a)$	Indicator function; $\mathbb{I}(a) = 1$ if a is true, else 0

Section 1

Introduction to linear classification

Classification vs regression

Regression

Output y is continuous (i.e. $y \in \mathbb{R}$).
For example, predict the price of a house given its area.

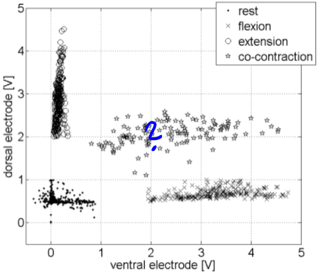
Classification

Output y belongs to one of C predetermined classes (i.e. $y \in \{1, ..., C\}$).
For example, determine whether the picture shows a cat or a dog.

Classification problem

Given

- observations [1]
  $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_i \in \mathbb{R}^D$
- set of possible classes.
  $\mathcal{C} = \{1, ..., C\}$
- labels
  $\boldsymbol{y} = \{y_1, y_2, \dots, y_N\}, \quad y_i \in \mathcal{C}$



Find

- function $f : \mathbb{R}^D \to \mathcal{C}$ that maps observations $\boldsymbol{x}_i$ to class labels y_i

$$y_i = f(\boldsymbol{x}_i) \qquad \text{for } i \in \{1, ..., N\}$$

[1] Like before, we represent samples as a data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$.

Zero-one loss

DOG : 5

CAT : 1

BIRD : 2

REGRESSION: $\sum_i (\underbrace{w^T x_i}_{\hat{y}_i} - y_i)^2$

How do we measure quality of a prediction $\hat{\boldsymbol{y}} := f(\boldsymbol{X})$? [2]

Zero-one loss denotes the number of misclassified samples.

$$\ell_{01}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{N} \mathbb{I}(\hat{y}_i \neq y_i).$$

How do we choose a good $f(\cdot)$?

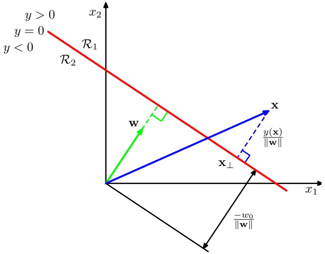
[2] For brevity, we denote the generated prediction $f(\boldsymbol{x}_i)$ as $\hat{y}_i$, i.e. $\hat{\boldsymbol{y}}$ is the vector of predictions for entire $\boldsymbol{X}$

Hyperplane as a decision boundary

For a 2 class problem ($\mathcal{C} = \{0, 1\}$) we can try to separate points from the two classes by a hyperplane.

Hyperplane as a decision boundary

For a 2 class problem ($\mathcal{C} = \{0, 1\}$) we can try to separate points from the two classes by a hyperplane.



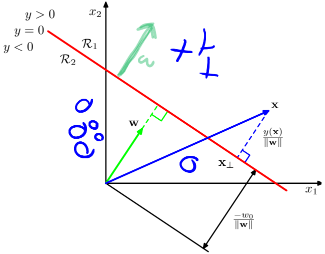
A hyperplane be defined by a normal vector $\boldsymbol{w}$ and an offset w_0.

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 \begin{cases} = 0 & \text{if } \boldsymbol{x} \text{ on the plane} \\ > 0 & \text{if } \boldsymbol{x} \text{ on normal's side} \\ < 0 & \text{else} \end{cases}$$

Hyperplanes are computationally very convenient: easy to evaluate.

Hyperplane as a decision boundary

For a 2 class problem ($\mathcal{C} = \{0, 1\}$) we can try to separate points from the two classes by a hyperplane.



A hyperplane be defined by a normal vector $\boldsymbol{w}$ and an offset w_0.

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 \begin{cases} = 0 & \text{if } \boldsymbol{x} \text{ on the plane} \\ > 0 & \text{if } \boldsymbol{x} \text{ on normal's side} \\ < 0 & \text{else} \end{cases}$$

Hyperplanes are computationally very convenient: easy to evaluate.

A data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$ is linearly separable if there exists a hyperplane for which all $\boldsymbol{x}_i$ with $y_i = 0$ are on one and all $\boldsymbol{x}_i$ with $y_i = 1$ on the other side.

Perceptron

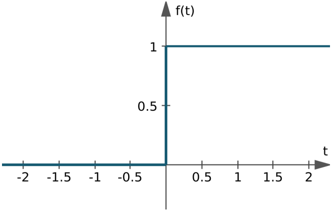
Perceptron algorithm is one of the oldest methods for binary classification.

Decision rule

$$\hat{y} = f(\boldsymbol{w}^T \boldsymbol{x} + w_0)$$

where f is the step function defined as:

$$f(t) = \begin{cases} 1 \text{ if } t > 0, \\ 0 \text{ else.} \end{cases}$$



Learning rule for the perceptron

Initialize parameters to any value, e.g., a zero vector: $\boldsymbol{w}, w_0 \leftarrow \boldsymbol{0}$.

[3] However, there is no way to determine the number of required iterations in advance.

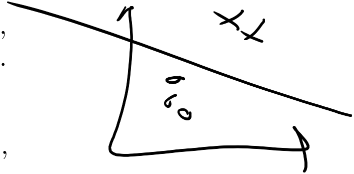
Learning rule for the perceptron

$\tilde{x}_i = [x_i, 1]$

Initialize parameters to any value, e.g., a zero vector: $\boldsymbol{w}, w_0 \leftarrow \mathbf{0}$.

For each misclassified sample $\boldsymbol{x}_i$ in the training set update

$$\boldsymbol{w} \leftarrow \begin{cases} \boldsymbol{w} + \boldsymbol{x}_i \text{ if } y_i = 1, \\ \boldsymbol{w} - \boldsymbol{x}_i \text{ if } y_i = 0. \end{cases}$$



$$w_0 \leftarrow \begin{cases} w_0 + 1 \text{ if } y_i = 1, \\ w_0 - 1 \text{ if } y_i = 0. \end{cases}$$

until all samples are classified correctly.

This method takes a finite number of steps to converge to a $(\boldsymbol{w}, w_0)$ discriminating between two classes if it exists. [3]

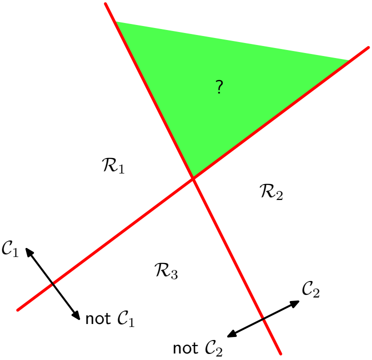
[3] However, there is no way to determine the number of required iterations in advance.

Does this scale up to multiple classes?

One-versus-rest classifier

Each hyperplane $\mathcal{H}_i$ makes a decision

$$\text{class } \mathcal{C}_i \leftrightarrow \text{not class } \mathcal{C}_i$$

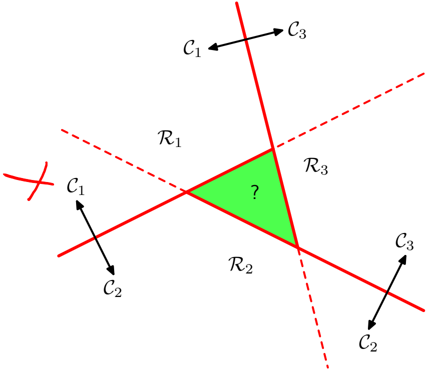


One-versus-one classifier

Hyperplane $\mathcal{H}_{ij}$ makes a decision for each pair of classes

$$\text{class } \mathcal{C}_i \leftrightarrow \text{class } \mathcal{C}_j$$

Use majority vote to classify.



$C_1 : 2$

$C_2 : 1$

$C_3 : 0$

Multiclass discriminant

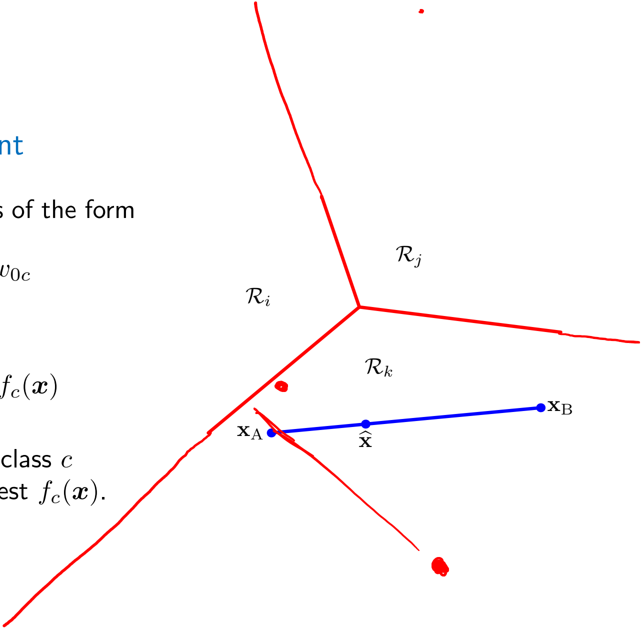
Define C linear functions of the form

$$f_c(\boldsymbol{x}) = \boldsymbol{w}_c^T \boldsymbol{x} + w_{0c}$$

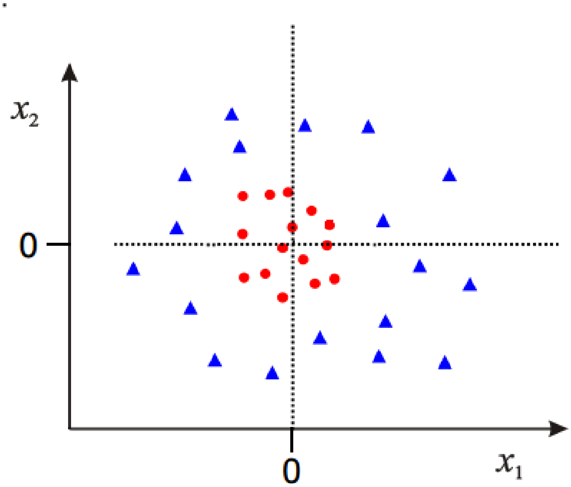
with the decision rule

$$\hat{y} = \arg\max_{c \in \mathcal{C}} f_c(\boldsymbol{x})$$

That is, assign $\boldsymbol{x}$ to the class c which produces the highest $f_c(\boldsymbol{x})$.

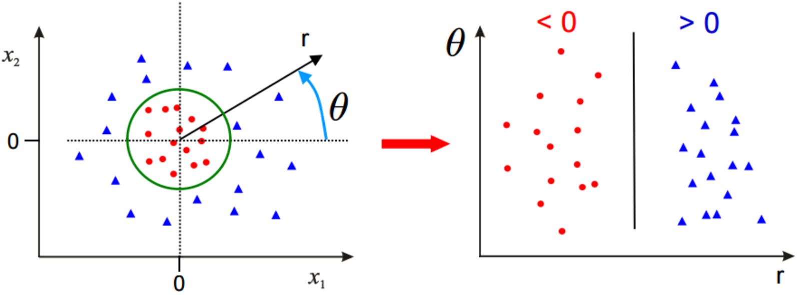


What if the classes are not linearly separable?



Basis functions

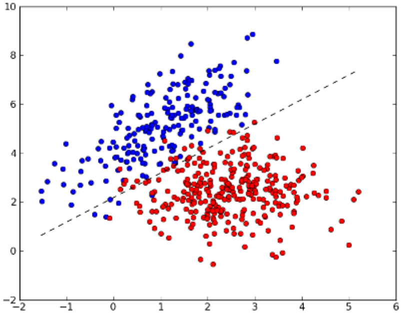
Like in the Linear Regression lecture last week, we can apply a nonlinear transformation $\phi : \mathbb{R}^D \to \mathbb{R}^M$. We need to choose a ϕ that maps samples to a space where they are linearly separable.



Here, $\boldsymbol{\phi}(\boldsymbol{x}) = (\theta, r) = (\text{angle}(\boldsymbol{x}), \sqrt{\boldsymbol{x}^T\boldsymbol{x}})$.

Limitations of hard-decision based classifiers

- No measure of uncertainty
- Can't handle noisy data
- Poor generalization
- Difficult to optimize



What are the alternatives?

Probabilistic models for classification

Solution: model the distribution of the class label y given the data $\boldsymbol{x}$.

$$p(y = c \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = c) \cdot p(y = c)}{p(\boldsymbol{x})}$$

Two types of models:

Generative

- Model the joint distribution $p(\boldsymbol{x}, y = c) = p(\boldsymbol{x} \mid y = c) \cdot p(y = c)$

Discriminative

↗ DOG
CAT

- Directly model the posterior $p(y = c \mid \boldsymbol{x})$

Given $p(y \mid \boldsymbol{x})$ we can make the prediction $\hat{y}$ based on our problem.
Popular choice is the mode: $\hat{y} = \arg\max_{c \in \mathcal{C}} p(y = c \mid \boldsymbol{x})$.

Section 2

Probabilistic generative models for linear classification

Generative model

The idea is to obtain the class posterior using the Bayes formula

$$p(y = c \mid \boldsymbol{x}) \propto \underbrace{p(\boldsymbol{x} \mid y = c)}_{\text{class conditional}} \cdot \underbrace{p(y = c)}_{\text{class prior}} \tag{1}$$

The model consists of

- class prior - a priori probability of a point belonging to a class c
- class conditional - probability of generating a point $\boldsymbol{x}$, given that it belongs to class c

Applying a generative model

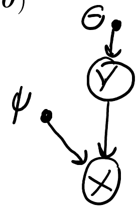
$y_i \sim N(f_w(x_i), \beta)$

Applying a generative model typically works as following

- Choose a parametric model for the class conditional $p(\boldsymbol{x} \mid y = c, \boldsymbol{\psi})$ and the class prior $p(y = c \mid \boldsymbol{\theta})$.
- Estimate the parameters of our model $\{\boldsymbol{\psi}, \boldsymbol{\theta}\}$ from the data $\mathcal{D}$ (e.g., using maximum likelihood - obtain estimates $\{\hat{\boldsymbol{\psi}}, \hat{\boldsymbol{\theta}}\}$). This step is called learning.

Once fitted, we can perform inference - classify a new $\boldsymbol{x}$ using Bayes rule

$$p(y = c \mid \boldsymbol{x}, \hat{\boldsymbol{\psi}}, \hat{\boldsymbol{\theta}}) \propto p(\boldsymbol{x} \mid y = c, \hat{\boldsymbol{\psi}}) p(y = c \mid \hat{\boldsymbol{\theta}}) \tag{2}$$



Additionally, we can generate new data - hence the name.

- sample a class label $y_{new} \sim p(y \mid \hat{\boldsymbol{\theta}})$
- sample a feature vector $\boldsymbol{x}_{new} \sim p(\boldsymbol{x} \mid y = y_{new}, \hat{\boldsymbol{\psi}})$

Choosing class prior

How do we choose the class prior $p(y = c)$?

The label y can take one of C discrete values.
$\implies$ Use categorical distribution!

$$y \sim \text{Categorical}(\boldsymbol{\theta})$$

The parameter $\boldsymbol{\theta} \in \mathbb{R}^C$ specifies the probability of each class

$$p(y = c) = \theta_c \quad \text{or equivalently} \quad p(y) = \prod_{c=1}^{C} \theta_c^{\mathbb{I}(y=c)}$$

and is subject to the constraints $0 \leq \theta_c \leq 1$ and $\sum_{c=1}^{C} \theta_c = 1$.

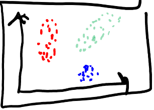
The maximum likelihood estimate for $\boldsymbol{\theta}$ given the data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}$ is

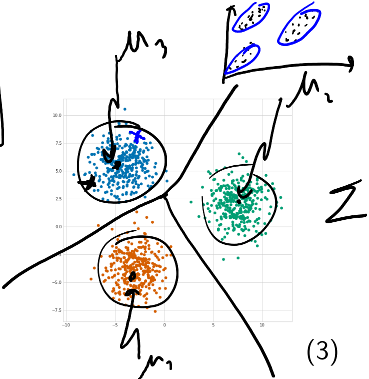
$$\theta_c^{MLE} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i = c)$$

Class conditionals

$\Sigma_c : C \cdot D^2 + C \cdot D$

How do we choose the class conditionals $p(\boldsymbol{x} \mid y = c)$?





The feature vector $\boldsymbol{x} \in \mathbb{R}^D$ is continuous.
$\implies$ Use a multivariate normal for each class!

$$p(\boldsymbol{x} \mid y = c) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}) \tag{3}$$

$X \sim \mathcal{N}(\mu_c, \Sigma)$

$$= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_c)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_c)\right\} \tag{4}$$

We use the same $\boldsymbol{\Sigma}$ for each class, as estimating all $\boldsymbol{\Sigma}_c$'s behaves badly numerically, unless we have **lots** of data.

The MLE estimates for $\{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_C, \boldsymbol{\Sigma}\}$ will be derived in the tutorial (or see Bishop 4.2.2).

HOW MANY PARAMETERS TO LEARN: $\underbrace{\binom{D}{2}+D}_{O(D^2)} + C \cdot D$

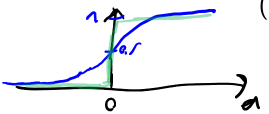
Posterior distribution

Now that we have chosen $p(\boldsymbol{x} \mid y)$ and $p(y)$, and have estimated their parameters from the training data [4], how do we perform classification?

Let's assume for simplicity that we have two classes $\mathcal{C} = \{0, 1\}$.

$$p(y = 1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = 1)\, p(y = 1)}{p(\boldsymbol{x} \mid y = 1) p(y = 1) + p(\boldsymbol{x} \mid y = 0)\, p(y = 0)} \tag{5}$$

$$= \frac{1}{1 + \exp(-a)} =: \sigma(a) \tag{6}$$



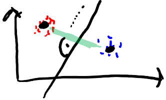
where we defined

$$a = \ln \frac{p(\boldsymbol{x} \mid y = 1) p(y = 1)}{p(\boldsymbol{x} \mid y = 0) p(y = 0)} \tag{7}$$

and σ is the sigmoid function.

[4] To avoid clutter, we implicitly condition the distributions on their respective parameters $(\boldsymbol{\theta}, \boldsymbol{\mu}_c, \boldsymbol{\Sigma})$

Linear discriminant analysis (LDA)



Let's look at how this function looks for Gaussian class-conditional with the same covariance $\mathbf{\Sigma}$

$$
\begin{aligned}
a =& \ln \frac{p(\boldsymbol{x} \mid y=1)p(y=1)}{p(\boldsymbol{x} \mid y=0)p(y=0)} && (8) \\
=& -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_1)^T \mathbf{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1) + \ln p(y=1) && (9) \\
& + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_0)^T \mathbf{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_0) - \ln p(y=0) && (10) \\
=& \boldsymbol{w}^T \boldsymbol{x} + w_0 && (11)
\end{aligned}
$$

where we define

$$
\begin{aligned}
\boldsymbol{w} =& \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) && (12) \\
w_0 =& -\frac{1}{2}\boldsymbol{\mu}_1 \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0 \mathbf{\Sigma}^{-1} \boldsymbol{\mu}_0 + \ln \frac{p(y=1)}{p(y=0)} && (13)
\end{aligned}
$$

LDA for $C = 2$ classes

$(x,y) \rightsquigarrow \mu_c, \Sigma \rightsquigarrow w$

This means, that the posterior distribution is a sigmoid of a linear function of $\boldsymbol{x}$

$$p(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + \exp(-(\boldsymbol{w}^T \boldsymbol{x} + w_0))} \tag{14}$$

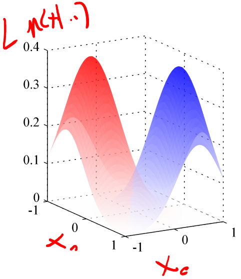
$$= \sigma(\boldsymbol{w}^T \boldsymbol{x} + w_0) \tag{15}$$

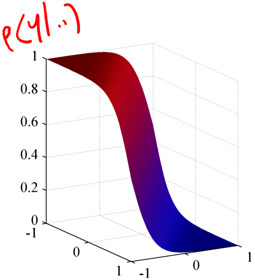
or equivalently

$$y \mid \boldsymbol{x} \sim \text{Bernoulli}(\sigma(\boldsymbol{w}^T \boldsymbol{x} + w_0)) \tag{16}$$

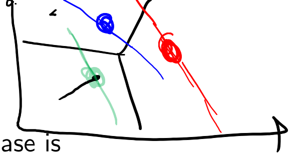
This is how this function looks for $D = 2$

- Right:
  $p(\boldsymbol{x} \mid y = 1)$ - red
  $p(\boldsymbol{x} \mid y = 0)$ - blue
- Left:
  $p(y = 1 \mid \boldsymbol{x})$





LDA for $C > 2$ classes



Using Bayes formula, the posterior for the $C > 2$ case is

$$p(y = c \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = c)p(y = c)}{\sum_{c'=1}^{C} p(\boldsymbol{x} \mid y = c')p(y = c')} \tag{17}$$

working out through some math we get

$$= \frac{\exp(\boldsymbol{w}_c^T \boldsymbol{x} + w_{c0})}{\sum_{c'=1}^{C} \exp(\boldsymbol{w}_{c'}^T \boldsymbol{x} + w_{c'0})} \tag{18}$$

where

$$\boldsymbol{w}_c = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c \tag{19}$$

$$\boxed{w_{c0}} = -\frac{1}{2} \boldsymbol{\mu}_c \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_c + \ln p(y = c) \tag{20}$$

Softmax function

In the slide above we made use of the softmax function.

Softmax σ is a generalization of sigmoid to multiple dimensions

$$\sigma : \mathbb{R}^K \to \triangle^{K-1} \tag{21}$$

where

$$\triangle^{K-1} = \left\{ \boldsymbol{x} \in \mathbb{R}^K \,|\, \sum_{k=1}^{K} x_k = 1 \text{ and } x_k \geq 0, k = 1, ..., K \right\} \tag{22}$$

is the standard probability simplex.

Softmax is defined as

$$\sigma(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{k=1}^{K} \exp(x_k)} \tag{23}$$

Section 3

Probabilistic discriminative models for linear classification

Probabilistic discriminative model

An alternative approach to generative modeling is to model the posterior distribution $p(y \mid \boldsymbol{x})$ directly. Such models are called discriminative.

We saw in the previous section that a generative approach with Gaussian class-conditional distributions with same covariance matrix $\boldsymbol{\Sigma}$ produces the following decision rule

$$p(y = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{x}^T \boldsymbol{w} + w_0), \tag{24}$$

$$p(y = 0 \mid \boldsymbol{x}) = 1 - \sigma(\boldsymbol{x}^T \boldsymbol{w} + w_0) \tag{25}$$

where $\boldsymbol{w}, w_0$ depend on the parameters of class-conditionals $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}$.

Why not just let $\boldsymbol{w}$ and w_0 be free parameters and choose them directly?

Logistic regression

(x_i, y_i)

$(\approx, 0) = (x_1, 0)$

$(\approx, 1)$

$(\approx, 0)$

We model the posterior distribution as

$$y \mid \boldsymbol{x} \sim \text{Bernoulli}(\underline{\sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0)}) \qquad (26)$$

where

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \qquad (27)$$

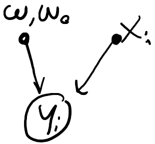
and $\boldsymbol{w}, w_0$ are the free model parameters.

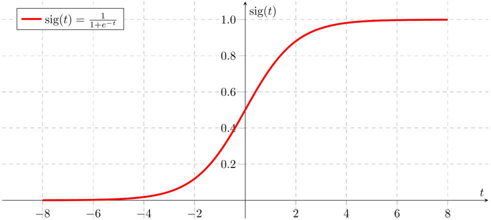
Likelihood

$$p(y \mid x, w, w_0)$$

$$= \prod_{i=1}^{N} p(y_i \mid x_i, w, w_0)$$

$$=$$





This model is called logistic regression.

Absorbing the bias term

Like in the previous lecture, we again absorb the bias term by overloading the notation and defining

$$\boldsymbol{w}^T \boldsymbol{x} := \boxed{w_0} + w_1 x_1 + \ldots + w_D x_D \tag{28}$$

Which is equivalent to defining $x_0 = 1$.

Likelihood of logistic regression

Learning logistic regression comes down to finding a “good” setting of parameters $\boldsymbol{w}$ that ”explain” the training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$.

Assuming that all samples $(\boldsymbol{x}_i, y_i)$ are drawn i.i.d., we can write the likelihood as

$$
\begin{aligned}
p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}) &= \prod_{i=1}^{N} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}) && (29)\\
&= \prod_{i=1}^{N} \underbrace{p(y = 1 \mid \boldsymbol{x}_i, \boldsymbol{w})^{y_i}}_{=1 \text{ if } y_i = 0} \underbrace{\big(1 - p(y = 1 \mid \boldsymbol{x}_i, \boldsymbol{w})\big)^{1-y_i}}_{=1 \text{ if } y_i = 1} && (30)\\
&= \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_i} (1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i))^{1-y_i} && (31)
\end{aligned}
$$

Negative log-likelihood

$$\sum \left[y_i \cdot \ln p_i + (1-y_i) \cdot \ln(1-p_i) \right]$$

Similarly to the linear regression case, we can define an error function as negative log-likelihood

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}) \tag{32}$$

$$= -\sum_{i=1}^{N} \left(\underline{y_i} \ln \overbrace{\sigma(\boldsymbol{w}^T \boldsymbol{x}_i)}^{\text{PREDICTIONS } p_i} + \underline{(1-y_i)} \ln(1-\sigma(\boldsymbol{w}^T \boldsymbol{x}_i)) \right) \tag{33}$$

GIVEN/TRUE TARGETS

This loss function is called binary cross entropy.

Finding the maximum likelihood estimate for $\boldsymbol{w}$ is equivalent to solving

$$\boldsymbol{w}^* = \arg\min E(\boldsymbol{w}) \tag{34}$$

Solving the minimization problem

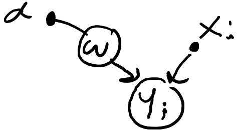
There doesn't exist a closed form solution for logistic regression. This means, we cannot compute the optimal $\boldsymbol{w}^*$ using standard mathematical operations, such as multiplication, matrix inversion, etc.

However, there is still hope! We can use optimization to numerically solve our problem. We will cover this in the next lecture (19.11.2018).

For now, just assume that we can find $\boldsymbol{w}^*$.

Logistic regression + weights regularization

$w \sim \mathcal{N}(0, \alpha^{-1} \cdot I)$



As we already well know, maximum likelihood estimation may often lead to overfitting. Just like in case of linear regression, we can control this by penalizing large weights.

$$E(\boldsymbol{w}) = -\ln p\big(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X}\big) + \lambda ||\boldsymbol{w}||_q^2 \tag{35}$$

Like before, for $q = 2$ this corresponds to MAP estimation with a Gaussian prior on $\boldsymbol{w}$.

Again, there is no closed form solution available.

Multiclass logistic regression

$$\text{softmax}\left(\begin{bmatrix} 3 \\ -5 \\ 15 \end{bmatrix}\right) = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix}$$

$$\sum \pi_i = 1 \quad \pi_i \geq 0$$

For the binary classification we used the sigmoid function to ”squeeze” the unnormalized probability $\boldsymbol{w}^T\boldsymbol{x}$ into the range $(0, 1)$.

The same can be done for multiple classes using the softmax function.

$$p(y = c \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_c^T \boldsymbol{x})}{\sum_{c'} \exp(\boldsymbol{w}_{c'}^T \boldsymbol{x})}$$

How does this relate to multiclass LDA?

Loss for multiclass logistic regression

The negative log-likelihood for multiclass LR can be written as

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{Y} \mid \boldsymbol{w}, \boldsymbol{X}) \tag{36}$$

$$= -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \ln p(y_i = c \mid \boldsymbol{x}_i, \boldsymbol{w}) \tag{37}$$

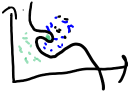
$$= -\sum_{i=1}^{N}\sum_{c=1}^{C} \underbrace{y_{ic}}_{\text{TRUE TARGETS}} \ln \underbrace{\frac{\exp(\boldsymbol{w}_c^T \boldsymbol{x})}{\sum_{c'} \exp(\boldsymbol{w}_{c'}^T \boldsymbol{x})}}_{\text{PREDICTION } p_i} \tag{38}$$

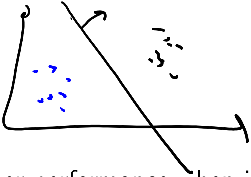
and is called cross entropy.

Here we use one-hot encoding: vector of categorical variables $\boldsymbol{y} \in \mathcal{C}^N$ is encoded as a binary matrix $\boldsymbol{Y} \in \{0, 1\}^{N \times C}$, where

$$y_{ic} = \begin{cases} 1 & \text{if sample } i \text{ belongs to class } c \\ 0 & \text{else} \end{cases} \tag{39}$$

Generative vs. discriminative models





- In general, discriminative models achieve better performance when it comes to pure classification tasks.

- While generative models work reasonably well when their assumptions hold, they are quite fragile when these assumptions are violated.

- Generative modeling for high-dimensional / strongly correlated data like images or graphs is still an open research challenge.

- Nevertheless, generative models provide the added benefits of better handling missing data, detecting outliers, generating new data and being more appropriate in the semi-supervised setting.