Linear classification

\[ x \rightarrow t \]

- Outputs, \( t \), are discrete and can take on \( K \) possible values \( C_1, \ldots, C_K \).
- Task: Learn to predict \( t \) from \( x \).
- Like linear regression, this is a supervised learning problem.
- Like in linear regression, linearity makes the task simple.
- But “linear” obviously has to mean something different here than in the case of regression.

Discriminative vs. generative

\[ x \rightarrow t \]

- There are two fundamentally different ways to approach classification tasks.
- **Discriminative methods** try to directly learn the mapping.
- **Generative methods** first learn a (probabilistic) model for each class \( C_k \), or equivalently a conditional distribution \( p(x|t) \) (note \( K \) is finite!), and then “invert” the model to get the mapping.
Discriminant functions

▶ In the case of $K = 2$ classes, a linear classifier can be defined as follows:

**Linear classifier (two classes)**

▶ Define the linear function

$$y(x) = w^T x + w_0$$

with parameters $w, w_0$

▶ Assign $x$ to class $C_1$ if $y(x) \geq 0$, to class $C_2$ otherwise.

▶ The function $y(x)$ is known as discriminant function.

▶ Learning a classifier can be achieved by adapting the parameters $w$ as we will soon see.

**Two classes**

▶ $w_0 \neq 0$ will move this boundary away from the origin.

▶ We can use the bias trick that we used for linear regression to eliminate $w_0$ from our model:

$$
\begin{pmatrix}
  x_1 \\
  \vdots \\
  x_D \\
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  1 \\
  x_1 \\
  \vdots \\
  x_D \\
\end{pmatrix}
$$

▶ Parameter $w_0$, which multiplies the 1 in the input, can be thought of as a negative threshold.

**Extension to more than two classes**

▶ We can define a multi-class classifier for $K > 2$ classes as follows:

**Multi-class linear classifier**

▶ Define $K$ discriminant functions

$$y_k(x) = w_k^T x + w_{k0}$$

with parameters $w_k, w_{k0}$

▶ Assign $x$ to class $C_k$ if $y_k(x) > y_j(x)$ for all $j \neq k$

▶ This amounts to choosing the maximum over all $k$ discriminants.

▶ We can define a 2-class model in this way, too.

**Geometric interpretation**

▶ The classification rule leads to a linear boundary separating the classes. In $D > 2$ dimensions, this will be a $(D - 1)$-dimensional hyperplane.

▶ Here, $w_0 = 0$
Extension to more than two classes

- A linear multi-class classifier partitions the input-space into $K$ convex regions:
  - Any convex combination of two points in the same class will also be in that class (see Bishop, page 183)

Learning linear classifiers

- We defined how to assign points to classes, given a model.
- The real question is, how to learn the parameters based on training data $D = \{(x_n, t_n)\}$, such that we will make as few mis-classifications as possible on test data.
- It may, or may not, be possible to learn a linear classifier that makes no mistakes at all on a given training set.
- Data-sets for which this is possible are called linearly separable.

Example with linearly separable data (before learning).

Example with linearly separable data (after learning).
Learning linear classifiers

▶ Example with data that is not linearly separable.

\[ y = 0 \]

Roland Memisevic
Machine Learning
13

Remark: How the represent discrete values

▶ There is a particularly useful way to represent one out of a set of \( K \) values:

▶ As a \( K \)-vector with \((K-1)\) 0’s, and one 1 at position \( k \):

\[
t = \begin{pmatrix}
0 \\
\vdots \\
1 \\
\vdots \\
0
\end{pmatrix}
\]

▶ This is known as one-of-\( K \) encoding, one-hot-encoding, or as orthogonal encoding.

▶ Note that we can interpret \( t \) as a probability distribution, because \( t_k > 0 \ \forall k \) and \( \sum_k t_k = 1 \).

▶ We can now represent discrete class-labels row-wise in a matrix \( T \), like we did before for continuous vectors.

(LMulti-class) logistic regression

▶ Logistic regression is probably the oldest and one of the most heavily used classifiers.

▶ Logistic regression defines a probabilistic model over classes, given inputs:

\[
p(C_k|x) = \frac{\exp(w_k^T x_n)}{\sum_{j=1}^{K} \exp(w_j^T x_n)}
\]

▶ Why this way?
Because the \( \exp \)-function ensures positivity, and the normalization that the outputs sum to one.

▶ Note that this amounts to “squashing” the components of \( y = W^T x \) to lie in the range between 0 and 1.

In the following, we shall stack the parameter vectors \( w_k \) column-wise in a matrix \( W \).

▶ Note that, for a given input \( x \),

\[
y = W^T x
\]

yields a vector of linear “scores” \( w_k^T x \) for each class.

▶ A good learning criterion will adjust \( W \), such that on training cases with \( t_n = C_k \), the model will give large values \( w_k^T x_n \) and small values \( w_j^T x_n \ \forall j \neq k \).

▶ Note that least squares regression is not a good approach to training, because it would penalize large values even for the right class!
Multi-class logistic regression

- Since it is a probabilistic model, logistic regression allows us to use maximum likelihood for training.
- Like in the case of linear regression before, consider the negative log-likelihood cost:

\[
E(W; D) = -\log \prod_n p(t_n|\mathbf{x}_n) = -\log \prod_n \prod_k p(C_k|\mathbf{x}_n)^{t_{nk}} = -\sum_n \sum_k t_{nk} \log p(C_k|\mathbf{x}_n) = -\sum_{nk} t_{nk} (w_k^T \mathbf{x}_n - \log \sum_j \exp(w_j^T \mathbf{x}_n))
\]

The “logsumexp”-trick

- Expressions like

\[
\frac{\exp(w_k^T \mathbf{x}_n)}{\sum_{j=1}^K \exp(w_j^T \mathbf{x}_n)}
\]

are highly unstable in practice, because the “exp” in the denominator can cause an under- or overflow.
- Never compute sums \(\sum_i \exp(a_i)\) naively.
- Add a constant \(A\) to each argument in all exp’s, so that even the largest argument is small; then undo the operation after computing the sum!
- Many software packages supply a convenience function “logsumexp” for this purpose:

\[
\text{logsumexp}(a_1, \ldots, a_K) = \log \left( \sum_i \exp(a_i + A) \right) - A
\]

with \(A = -(\max_i a_i)\)

Multi-class logistic regression

- In contrast to linear regression, there is no closed-form solution for \(W\).
- But we can use gradient-based optimization to minimize \(E(W; D)\) iteratively.
- The gradient with respect to each parameter-vector \(w_k\) is

\[
\frac{\partial E(W; D)}{\partial w_k} = -\sum_n t_{nk} \mathbf{x}_n - \frac{\exp(w_k^T \mathbf{x}_n)}{\sum_j \exp(w_j^T \mathbf{x}_n)} \mathbf{x}_n = \sum_n (p(C_k|\mathbf{x}_n) - t_{nk}) \mathbf{x}_n
\]

- It can be shown that \(E(W; D)\) is convex, so there are no local minima!

Two-class Logistic regression

- The term “logistic regression” has been used traditionally to refer to the case of exactly two classes.
- Traditionally, the two-class case has been treated slightly differently, using one, rather than two, parameter vectors.
- Note that we can write

\[
p(C_1|\mathbf{x}) = \frac{\exp(w_1^T \mathbf{x})}{\exp(w_1^T \mathbf{x}) + \exp(w_2^T \mathbf{x})} = \frac{1}{1 + \exp(- (w_1 - w_2)^T \mathbf{x})}
\]

- And likewise, for class 2, we can write

\[
p(C_2|\mathbf{x}) = \frac{1}{1 + \exp((w_1 - w_2)^T \mathbf{x})}
\]
Two-class Logistic regression

- Thus, it is sufficient to use just one parameter vector $w := (w_1 - w_2)$:

Two-class logistic regression

$$p(C_1|x) = 1 - p(C_2|x) = \frac{1}{1 + \exp(-w^T x)}$$

- This amounts to removing the “over-parameterization” inherent in the original softmax-regression definition, and it could in principle be done for the multi-class model, not just the 2-class model. But this is not common in practice.

The sigmoid function

- The function

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

is an “S”-shaped 1-dimensional function, which known as “sigmoid” or as “logistic function”. (Hence the term “logistic regression”.)

- Its derivative can be written in a slightly unusual way, using the function itself, as:

$$\frac{d\sigma(a)}{da} = \sigma(1 - \sigma)$$

- Its inverse is given by $a = \log(\frac{\sigma}{1 - \sigma})$, which is known as the “logit-function” or as “log odds”.

Basis expansion and regularization

Basis expansions

- Like for linear regression, we can use a non-linear \textbf{basis-expansion} $x \rightarrow \Phi(x)$ and fit a non-linear model using logistic regression.

- This will, in general, amount to fitting non-linear class boundaries.

Regularization

- Like for linear regression, one should \textbf{regularize} the model by adding a weight penalty such as

$$E_W(W, \lambda) = \frac{\lambda}{2} \|W\|^2$$

to the objective function to avoid overfitting.
Generative methods

- An entirely different approach to classification is given by **generative methods**, which are based on learning, and then inverting, a model for each class-conditional distribution over inputs. This can be achieved by using Bayes’ rule:

\[
p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)}
\]

- Under this rule, the winning class is class $C_k$ for which $p(x|C_k)p(C_k) > p(x|C_j)p(C_j)$ $\forall j$
- Since they involve Bayes’ rule, generative classifiers are sometimes called “Bayes classifier” or “Bayesian classifier”. (This is as slight misnomer, as they are not really Bayesian models which integrate over parameters.)

- Class boundaries between classes $C_1$ and $C_2$ are defined by

\[
p(x|C_1)p(C_1) = p(x|C_2)p(C_2)
\]
Generative models

- A generative classifier requires two specifications:
  1. The prior probabilities \( p(C_k) \) over classes. In practice, these are usually set to \( \frac{\sum t_{nk}}{N} \).
  2. The class-conditional distributions over inputs \( p(x|C_k) \). There are a lot of possible choices. We shall discuss the most common ones in the following.
- Note that both, logistic regression, and generative models, define a classifier by modeling the conditional probability \( p(C_k|x) \). They differ only in how they define this probability.

Generative models with continuous inputs

- For continuous \((D\text{-dimensional})\) inputs \( x \), the most common and simplest class-conditional model is the multivariate Gaussian:
  \[
  p(x|C_k) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) \right)
  \]
- In the most general case, each class \( C_k \) gets its own mean \( \mu_k \) and own covariance matrix \( \Sigma_k \).
- Fitting a generative classifier then amounts to simply fitting \( K \) class-conditional Gaussians.

Class boundaries

- We now look at the class boundaries between two classes \( C_1, C_2 \) in the case of a shared covariance matrix.
- For this end, set:
  \[
  \log p(x|C_1) + \log p(C_1) = \log p(x|C_2) + \log p(C_2)
  \]
  \[
  \Leftrightarrow \ (x - \mu_1)^T \Sigma^{-1} (x - \mu_1) - \log p(C_1)
  \]
  \[
  = (x - \mu_2)^T \Sigma^{-1} (x - \mu_2) - \log p(C_2)
  \]
  \[
  \Leftrightarrow \ x^T \Sigma^{-1} x + \mu_1^T \Sigma^{-1} \mu_1 - 2 \mu_1^T \Sigma^{-1} x - \log p(C_1)
  \]
  \[
  = x^T \Sigma^{-1} x + \mu_2^T \Sigma^{-1} \mu_2 - 2 \mu_2^T \Sigma^{-1} x - \log p(C_2)
  \]
  \[
  \Leftrightarrow \ (\mu_1 - \mu_2)^T \Sigma^{-1} x = \text{const}
  \]
- This is a linear condition on the inputs \( x \). Thus when using class-conditional Gaussians with shared covariance matrix, we obtain linear class boundaries!
- Otherwise they will be quadratic.
Class boundaries

Example with three classes, where two share a covariance matrix:

Class-conditional Gaussians and logistic regression

Consider the conditional class-probabilities \( p(C_1|x), p(C_2|x) \) under the two-class model:

We have:

\[
p(C_1|x) = \frac{p(C_1|x)p(C_1)}{p(C_1|x)p(C_1) + p(C_2|x)p(C_2)} = \frac{1}{1 + \exp(-a)}
\]

with \( a = \log \frac{p(x|C_2)p(C_1)}{p(x|C_1)p(C_2)} = (\mu_1 - \mu_2)^T\Sigma^{-1}x := w^T x \)

Thus, a Gaussian generative classifier with shared covariance matrix has the same functional form as logistic regression!

Discrete inputs and Naive Bayes

Discrete features are easy to deal with, if we assume inputs to be independent given classes.
A generative classifier that makes this assumption is known as a “Naive Bayes classifier”.
Class-conditional independence seems like a severe restriction in practice, but it often works surprisingly well.

Bernoulli Naive Bayes classifier

If we assume independent Bernoulli variables in each input dimension, we can write:

\[
p(x|C_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i}(1 - \mu_{ki})^{1-x_i}
\]

where \( \mu_{ki} \) is the conditional probability for input \( i \) to be equal to 1, given class \( k \).
The maximum likelihood estimates for \( \mu_{ki} \) are simply counts:

\[
\mu_{ki} = \frac{\sum_n t_{nk}x_{ni}}{\sum_n t_{nk}}
\]
If we assume independent discrete ("multinomial") variables with $M$ outcomes in each input dimension, and represent each single input case by a $D \times M$-matrix of "stacked" one-hot encodings, we can write:

$$p(x|C_k) = \prod_{i=1}^{D} \prod_{j=1}^{M} \mu_{kij}^{x_{ij}}$$

with parameters $\mu_{kij}$ that represent the probability that for class $k$, input dimension $i$ takes on the value $j$.

Stack training data in a 3d-"tensor" of dimensions $(N \times D \times M)$.

The maximum likelihood estimates for the parameters $\mu_{kij}$ are again simply counts:

$$\mu_{kij} = \frac{\sum_n t_{nk} x_{nij}}{\sum_n t_{nk}}$$

How do we know? Maximize the log-likelihood, enforce constraints using Lagrange multipliers.