Machine Learning – Lecture 6

Linear Discriminants II

05.11.2018

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de
leibe@vision.rwth-aachen.de
Course Outline

• Fundamentals
  ➢ Bayes Decision Theory
  ➢ Probability Density Estimation

• Classification Approaches
  ➢ Linear Discriminants
  ➢ Support Vector Machines
  ➢ Ensemble Methods & Boosting
  ➢ Randomized Trees, Forests & Ferns

• Deep Learning
  ➢ Foundations
  ➢ Convolutional Neural Networks
  ➢ Recurrent Neural Networks
Recap: Linear Discriminant Functions

• Basic idea
  - Directly encode decision boundary
  - Minimize misclassification probability directly.

• Linear discriminant functions
  
  \[ y(x) = w^T x + w_0 \]

  - \( w, w_0 \) define a hyperplane in \( \mathbb{R}^D \).
  - If a data set can be perfectly classified by a linear discriminant, then we call it **linearly separable**.
Recap: Least-Squares Classification

• Simplest approach
  - Directly try to minimize the sum-of-squares error
    \[
    E(w) = \sum_{n=1}^{N} (y(x_n; w) - t_n)^2
    \]
    \[
    E_D(\tilde{W}) = \frac{1}{2} \text{Tr} \left\{ (\tilde{X}\tilde{W} - T)^T(\tilde{X}\tilde{W} - T) \right\}
    \]
  - Setting the derivative to zero yields
    \[
    \tilde{W} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T T = \tilde{X}^\dagger T = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T T
    \]
  - We then obtain the discriminant function as
    \[
    y(x) = \tilde{W}^T\tilde{x} = T^T(\tilde{X}^\dagger)^T\tilde{x}
    \]
    \Rightarrow Exact, closed-form solution for the discriminant function parameters.
Recap: Problems with Least Squares

- Least-squares is very sensitive to outliers!
  - The error function penalizes predictions that are “too correct”.

Image source: C.M. Bishop, 2006
Recap: Generalized Linear Models

• Generalized linear model

\[ y(x) = g(w^T x + w_0) \]

- \( g(\cdot) \) is called an activation function and may be nonlinear.
- The decision surfaces correspond to

\[ y(x) = \text{const.} \iff w^T x + w_0 = \text{const.} \]

- If \( g \) is monotonous (which is typically the case), the resulting decision boundaries are still linear functions of \( x \).

• Advantages of the non-linearity

- Can be used to bound the influence of outliers and “too correct” data points.
- When using a sigmoid for \( g(\cdot) \), we can interpret the \( y(x) \) as posterior probabilities.

\[ g(a) = \frac{1}{1 + \exp(-a)} \]
Linear Separability

• Up to now: restrictive assumption
  ➢ Only consider linear decision boundaries

• Classical counterexample: XOR

```
x_1

C_1

C_2

C_1

C_2

x_2
```
Generalized Linear Discriminants

• Generalization
  - Transform vector $\mathbf{x}$ with $M$ nonlinear basis functions $\phi_j(\mathbf{x})$:
    \[ y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj} \phi_j(\mathbf{x}) + w_{k0} \]
  - Purpose of $\phi_j(\mathbf{x})$: basis functions
  - Allow non-linear decision boundaries.
  - By choosing the right $\phi_j$, every continuous function can (in principle) be approximated with arbitrary accuracy.

• Notation
  \[ y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj} \phi_j(\mathbf{x}) \quad \text{with} \quad \phi_0(\mathbf{x}) = 1 \]

Slide credit: Bernt Schiele
Linear Basis Function Models

• Generalized Linear Discriminantant Model

\[ y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x) \]

- where \( \phi_j(x) \) are known as *basis functions*.
- Typically, \( \phi_0(x) = 1 \), so that \( w_0 \) acts as a bias.
- In the simplest case, we use linear basis functions: \( \phi_d(x) = x_d \).

• *Let’s take a look at some other possible basis functions...*
Linear Basis Function Models (2)

- **Polynomial** basis functions
  
  \[ \phi_j(x) = x^j. \]

- **Properties**
  - Global
    - A small change in \( x \) affects all basis functions.

- **Result**
  - If we use polynomial basis functions, the decision boundary will be a polynomial function of \( x \).
  - Nonlinear decision boundaries
  - However, we still solve a linear problem in \( \phi(x) \).
Linear Basis Function Models (3)

- **Gaussian** basis functions

\[ \phi_j(x) = \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\} \]

- **Properties**
  - Local
    - A small change in \( x \) affects only nearby basis functions.
  - \( \mu_j \) and \( s \) control location and scale (width).

Slide adapted from C.M. Bishop, 2006

Image source: C.M. Bishop, 2006
Linear Basis Function Models (4)

- **Sigmoid** basis functions

  \[ \phi_j(x) = \sigma \left( \frac{x - \mu_j}{s} \right) \]

  - where

  \[ \sigma(a) = \frac{1}{1 + \exp(-a)}. \]

- **Properties**
  - Local
    \[ \Rightarrow \text{A small change in } x \text{ affects only nearby basis functions.} \]
  - \( \mu_j \) and \( s \) control location and scale (slope).
Topics of This Lecture

• Gradient Descent

• Logistic Regression
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares

• Softmax Regression
  - Multi-class generalization
  - Gradient descent solution

• Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
Gradient Descent

- Learning the weights $w$:
  - $N$ training data points: $X = \{x_1, \ldots, x_N\}$
  - $K$ outputs of decision functions: $y_k(x_n; w)$
  - Target vector for each data point: $T = \{t_1, \ldots, t_N\}$
  - Error function (least-squares error) of linear model

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} (y_k(x_n; w) - t_{kn})^2$$

$$= \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \left( \sum_{j=1}^{M} w_{kj} \phi_j(x_n) - t_{kn} \right)^2$$
Gradient Descent

• Problem
  ➢ The error function can in general no longer be minimized in closed form.

• Idea (Gradient Descent)
  ➢ Iterative minimization
  ➢ Start with an initial guess for the parameter values $w_{kj}^{(0)}$
  ➢ Move towards a (local) minimum by following the gradient.

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E(w)}{\partial w_{kj}} \bigg|_{w^{(\tau)}}$$

$\eta$: Learning rate

➢ This simple scheme corresponds to a 1\textsuperscript{st}-order Taylor expansion (There are more complex procedures available).
Gradient Descent – Basic Strategies

• “Batch learning”

\[ w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(w)}{\partial w_{kj}} \right|_{w^{(\tau)}} \]

\( \eta: \text{Learning rate} \)

⇒ Compute the gradient based on all training data:

\[ \frac{\partial E(w)}{\partial w_{kj}} \]
Gradient Descent – Basic Strategies

• “Sequential updating”

\[ E(w) = \sum_{n=1}^{N} E_n(w) \]

\[ w^{(\tau+1)}_{k,j} = w^{(\tau)}_{k,j} - \eta \left. \frac{\partial E_n(w)}{\partial w_{k,j}} \right|_{w^{(\tau)}} \]

\( \eta \): Learning rate

- Compute the gradient based on a single data point at a time:

\[ \frac{\partial E_n(w)}{\partial w_{k,j}} \]
Gradient Descent

- Error function

\[
E(w) = \sum_{n=1}^{N} E_n(w) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \left( \sum_{j=1}^{M} w_{kj} \phi_j(x_n) - t_{kn} \right)^2
\]

\[
E_n(w) = \frac{1}{2} \sum_{k=1}^{K} \left( \sum_{j=1}^{M} w_{kj} \phi_j(x_n) - t_{kn} \right)^2
\]

\[
\frac{\partial E_n(w)}{\partial w_{kj}} = \left( \sum_{\tilde{j}=1}^{M} w_{k\tilde{j}} \tilde{\phi}_{\tilde{j}}(x_n) - t_{kn} \right) \phi_j(x_n)
\]

\[
= (y_k(x_n; w) - t_{kn}) \phi_j(x_n)
\]

Slide credit: Bernt Schiele
Gradient Descent

- Delta rule (=LMS rule)

\[ w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta (y_k(x_n; w) - t_{kn}) \phi_j(x_n) \]

\[ = w_{kj}^{(\tau)} - \eta \delta_{kn} \phi_j(x_n) \]

- where

\[ \delta_{kn} = y_k(x_n; w) - t_{kn} \]

⇒ Simply feed back the input data point, weighted by the classification error.

\[ w(\tau+1)_{kj} = w(\tau)_{kj} - \eta \delta_{kn} \phi_j(x_n) \]

\[ \delta_{kn} = y_k(x_n; w) - t_{kn} \]
Gradient Descent

• Cases with differentiable, non-linear activation function

\[
y_k(x) = g(a_k) = g \left( \sum_{j=0}^{M} w_{ki} \phi_j(x_n) \right)
\]

• Gradient descent

\[
\frac{\partial E_n(w)}{\partial w_{kj}} = \frac{\partial g(a_k)}{\partial w_{kj}} (y_k(x_n; w) - t_{kn}) \phi_j(x_n)
\]

\[
w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \delta_{kn} \phi_j(x_n)
\]

\[
\delta_{kn} = \frac{\partial g(a_k)}{\partial w_{kj}} (y_k(x_n; w) - t_{kn})
\]

Slide credit: Bernt Schiele
Summary: Generalized Linear Discriminants

• Properties
  - General class of decision functions.
  - Nonlinearity $g(\cdot)$ and basis functions $\phi_j$ allow us to address linearly non-separable problems.
  - Shown simple sequential learning approach for parameter estimation using gradient descent.
  - Better 2nd order gradient descent approaches are available (e.g. Newton-Raphson), but they are more expensive to compute.

• Limitations / Caveats
  - Flexibility of model is limited by curse of dimensionality
    - $g(\cdot)$ and $\phi_j$ often introduce additional parameters.
    - Models are either limited to lower-dimensional input space or need to share parameters.
  - Linearly separable case often leads to overfitting.
    - Several possible parameter choices minimize training error.
Topics of This Lecture

• Gradient Descent

• **Logistic Regression**
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares

• **Softmax Regression**
  - Multi-class generalization
  - Gradient descent solution

• **Note on Error Functions**
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
Probabilistic Discriminative Models

• We have seen that we can write

\[ P(C_1 | x) = \sigma(a) \]

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

• We can obtain the familiar probabilistic model by setting

\[ a = \ln \frac{p(x | C_1) p(C_1)}{p(x | C_2) p(C_2)} \]

• Or we can use generalized linear discriminant models

\[ a = w^T x \]

or

\[ a = w^T \phi(x) \]
Probabilistic Discriminative Models

• In the following, we will consider models of the form

\[ p(C_1 | \phi) = y(\phi) = \sigma(w^T \phi) \]

with

\[ p(C_2 | \phi) = 1 - p(C_1 | \phi) \]

• This model is called logistic regression.

• Why should we do this? What advantage does such a model have compared to modeling the probabilities?

\[ p(C_1 | \phi) = \frac{p(\phi | C_1) p(C_1)}{p(\phi | C_1) p(C_1) + p(\phi | C_2) p(C_2)} \]

• Any ideas?
Comparison

• Let’s look at the number of parameters…
  - Assume we have an $M$-dimensional feature space $\phi$.
  - And assume we represent $p(\phi|C_k)$ and $p(C_k)$ by Gaussians.
  - How many parameters do we need?
    - For the means: $2M$
    - For the covariances: $M(M+1)/2$
    - Together with the class priors, this gives $M(M+5)/2+1$ parameters!
  
  - How many parameters do we need for logistic regression?
    $$p(C_1|\phi) = y(\phi) = \sigma(w^T \phi)$$
    - Just the values of $w \Rightarrow M$ parameters.

$\Rightarrow$ For large $M$, logistic regression has clear advantages!
Logistic Sigmoid

• Properties
  ➢ Definition: \[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]
  ➢ Inverse: \[ a = \ln \left( \frac{\sigma}{1 - \sigma} \right) \] “logit” function
  ➢ Symmetry property: \[ \sigma(-a) = 1 - \sigma(a) \]
  ➢ Derivative: \[ \frac{d\sigma}{da} = \sigma(1 - \sigma) \]
Logistic Regression

• Let’s consider a data set \( \{ \phi_n, t_n \} \) with \( n = 1, \ldots, N \), where \( \phi_n = \phi(x_n) \) and \( t_n \in \{0, 1\} \), \( t = (t_1, \ldots, t_N)^T \).

• With \( y_n = p(C_1 | \phi_n) \), we can write the likelihood as

\[
p(t|w) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}
\]

• Define the error function as the negative log-likelihood

\[
E(w) = - \ln p(t|w)
= - \sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}
\]

\( \checkmark \) This is the so-called cross-entropy error function.
Gradient of the Error Function

• Error function

\[ E(w) = - \sum_{n=1}^{N} \{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \} \]

• Gradient

\[ \nabla E(w) = - \sum_{n=1}^{N} \left\{ t_n \frac{d}{dw} \frac{y_n}{y_n} + (1 - t_n) \frac{d}{dw} \frac{1 - y_n}{1 - y_n} \right\} \]

\[ = - \sum_{n=1}^{N} \left\{ t_n \frac{y_n(1 - y_n)}{y_n} \phi_n - (1 - t_n) \frac{y_n(1 - y_n)}{1 - y_n} \phi_n \right\} \]

\[ = - \sum_{n=1}^{N} \left\{ (t_n - t_n y_n - y_n + t_n y_n) \phi_n \right\} \]

\[ = \sum_{n=1}^{N} (y_n - t_n) \phi_n \]
Gradient of the Error Function

- Gradient for logistic regression
  \[ \nabla E(w) = \sum_{n=1}^{N} (y_n - t_n) \phi_n \]

- Does this look familiar to you?

- This is the same result as for the Delta (=LMS) rule
  \[ \omega_{kj}^{(\tau+1)} = \omega_{kj}^{(\tau)} - \eta (y_k(x_n; w) - t_{kn}) \phi_j(x_n) \]

- We can use this to derive a sequential estimation algorithm.
  - However, this will be quite slow…
A More Efficient Iterative Method…

• Second-order Newton-Raphson gradient descent scheme

\[ w^{(\tau+1)} = w^{(\tau)} - H^{-1}\nabla E(w) \]

where \( H = \nabla \nabla E(w) \) is the Hessian matrix, i.e. the matrix of second derivatives.

• Properties
  - Local quadratic approximation to the log-likelihood.
  - Faster convergence.
Newton-Raphson for Least-Squares Estimation

- Let’s first apply Newton-Raphson to the least-squares error function:

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (w^T \phi_n - t_n)^2
\]

\[
\nabla E(w) = \sum_{n=1}^{N} (w^T \phi_n - t_n) \phi_n = \Phi^T \Phi w - \Phi^T t
\]

\[
H = \nabla \nabla E(w) = \sum_{n=1}^{N} \phi_n \phi_n^T = \Phi^T \Phi
\]

where \( \Phi = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_N^T \end{bmatrix} \)

- Resulting update scheme:

\[
w^{(\tau+1)} = w^{(\tau)} - (\Phi^T \Phi)^{-1}(\Phi^T \Phi w^{(\tau)} - \Phi^T t)
\]

\[
= (\Phi^T \Phi)^{-1} \Phi^T t
\]

Closed-form solution!
Newton-Raphson for Logistic Regression

• Now, let’s try Newton-Raphson on the cross-entropy error function:

\[
E(w) = - \sum_{n=1}^{N} \{ t_n \ln y_n + (1 - t_n) \ln (1 - y_n) \}
\]

\[
\nabla E(w) = \sum_{n=1}^{N} (y_n - t_n) \phi_n = \Phi^T (y - t)
\]

\[
H = \nabla \nabla E(w) = \sum_{n=1}^{N} y_n (1 - y_n) \phi_n \phi_n^T = \Phi^T R \Phi
\]

where \( R \) is an \( N \times N \) diagonal matrix with \( R_{nn} = y_n (1 - y_n) \).

\[\Rightarrow\] The Hessian is no longer constant, but depends on \( w \) through the weighting matrix \( R \).
Iteratively Reweighted Least Squares

• Update equations

\[ \mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - (\Phi^T \mathbf{R} \Phi)^{-1} \Phi^T (\mathbf{y} - \mathbf{t}) \]

\[ = (\Phi^T \mathbf{R} \Phi)^{-1} \left\{ \Phi^T \mathbf{R} \Phi \mathbf{w}^{(\tau)} - \Phi^T (\mathbf{y} - \mathbf{t}) \right\} \]

\[ = (\Phi^T \mathbf{R} \Phi)^{-1} \Phi^T \mathbf{R} \mathbf{z} \]

with \[ \mathbf{z} = \Phi \mathbf{w}^{(\tau)} - \mathbf{R}^{-1} (\mathbf{y} - \mathbf{t}) \]

• Again very similar form (normal equations)
  - But now with non-constant weighing matrix \( \mathbf{R} \) (depends on \( \mathbf{w} \)).
  - Need to apply normal equations iteratively.
  \[ \Rightarrow \text{Iteratively Reweighted Least-Squares (IRLS) } \]
Summary: Logistic Regression

• Properties
  - Directly represent posterior distribution $p(\phi|C_k)$
  - Requires fewer parameters than modeling the likelihood + prior.
  - Very often used in statistics.
  - It can be shown that the cross-entropy error function is concave
    - Optimization leads to unique minimum
    - But no closed-form solution exists
    - Iterative optimization (IRLS)
  - Both online and batch optimizations exist

• Caveat
  - Logistic regression tends to systematically overestimate odds ratios when the sample size is less than ~500.
Topics of This Lecture

• Gradient Descent

• Logistic Regression
  ➢ Probabilistic discriminative models
  ➢ Logistic sigmoid (logit function)
  ➢ Cross-entropy error
  ➢ Iteratively Reweighted Least Squares

• Softmax Regression
  ➢ Multi-class generalization
  ➢ Gradient descent solution

• Note on Error Functions
  ➢ Ideal error function
  ➢ Quadratic error
  ➢ Cross-entropy error
Softmax Regression

- Multi-class generalization of logistic regression
  - In logistic regression, we assumed binary labels \( t_n \in \{0, 1\} \).
  - Softmax generalizes this to \( K \) values in 1-of-\( K \) notation.

\[
y(x; w) = \begin{bmatrix} P(y = 1|x; w) \\ P(y = 2|x; w) \\ \vdots \\ P(y = K|x; w) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(w_j^\top x)} \begin{bmatrix} \exp(w_1^\top x) \\ \exp(w_2^\top x) \\ \vdots \\ \exp(w_K^\top x) \end{bmatrix}
\]

- This uses the softmax function

\[
\frac{\exp(a_k)}{\sum_j \exp(a_j)}
\]

- Note: the resulting distribution is normalized.
Softmax Regression Cost Function

• Logistic regression
  
  Alternative way of writing the cost function

  \[
  E(w) = - \sum_{n=1}^{N} \left\{ t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right\}
  \]

  \[
  = - \sum_{n=1}^{N} \sum_{k=0}^{1} \left\{ \mathbb{I}(t_n = k) \ln P(y_n = k|\mathbf{x}_n; w) \right\}
  \]

• Softmax regression
  
  Generalization to K classes using indicator functions.

  \[
  E(w) = - \sum_{n=1}^{N} \sum_{k=1}^{K} \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(w_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(w_j^\top \mathbf{x})} \right\}
  \]
Optimization

• Again, no closed-form solution is available
  - Resort again to Gradient Descent
  - Gradient

\[ \nabla_{\mathbf{w}_k} E(\mathbf{w}) = - \sum_{n=1}^{N} \mathbb{I}(t_n = k) \ln P(y_n = k|\mathbf{x}_n; \mathbf{w}) \]

• Note
  - \( \nabla_{\mathbf{w}_k} E(\mathbf{w}) \) is itself a vector of partial derivatives for the different components of \( \mathbf{w}_k \).
  - We can now plug this into a standard optimization package.
Topics of This Lecture

• Gradient Descent

• Logistic Regression
  - Probabilistic discriminative models
  - Logistic sigmoid (logit function)
  - Cross-entropy error
  - Iteratively Reweighted Least Squares

• Softmax Regression
  - Multi-class generalization
  - Gradient descent solution

• Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
Note on Error Functions

\( t_n \in \{-1, 1\} \)

- Ideal misclassification error function (black)
  - This is what we want to approximate (error = #misclassifications)
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  \( \Rightarrow \) We cannot minimize it by gradient descent.

\[ z_n = t_n y(x_n) \]

Image source: Bishop, 2006
Note on Error Functions

\[ t_n \in \{-1, 1\} \]

Sensitive to outliers!

- **Squared error used in Least-Squares Classification**
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes “too correct” data points
  - Generally does not lead to good classifiers.

Ideal misclassification error
Squared error

\[ z_n = t_n y(x_n) \]
Comparing Error Functions (Loss Functions)

- Cross-Entropy Error
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - Robust to outliers, error increases only roughly linearly
  - But no closed-form solution, requires iterative estimation.

\[ t_n \in \{ -1, 1 \} \]

Robust to outliers!

\[ z_n = t_n y(x_n) \]

Image source: Bishop, 2006
Overview: Error Functions

• Ideal Misclassification Error
  - This is what we would like to optimize.
  - But cannot compute gradients here.

• Quadratic Error
  - Easy to optimize, closed-form solutions exist.
  - But not robust to outliers.

• Cross-Entropy Error
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - But no closed-form solution, requires iterative estimation.

⇒ Looking at the error function this way gives us an analysis tool to compare the properties of classification approaches.
References and Further Reading

• More information on Linear Discriminant Functions can be found in Chapter 4 of Bishop’s book (in particular Chapter 4.1 - 4.3).

Christopher M. Bishop
Pattern Recognition and Machine Learning
Springer, 2006