Machine Learning – Lecture 7

Linear Discriminants III

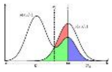
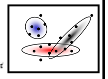
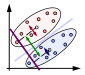
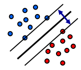
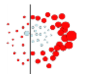
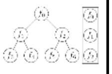
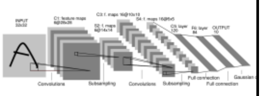
31.10.2019

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: Generalized Linear Models

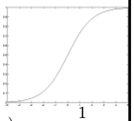
- Generalized linear model

$$y(\mathbf{x}) = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0)$$

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

$$y(\mathbf{x}) = const. \quad \Leftrightarrow \quad \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = const.$$

  - If g is monotonous (which is typically the case), the resulting decision boundaries are still linear functions of $\mathbf{x}$.
- 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 $y(\mathbf{x})$ as posterior probabilities.



$$g(a) \equiv \frac{1}{1 + \exp(-a)}$$

Recap: Extension to Nonlinear Basis Fcts.

- Generalization
  - Transform vector $\mathbf{x}$ with M nonlinear basis functions $\phi_j(\mathbf{x})$:

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$$

- Advantages
  - Transformation allows non-linear decision boundaries.
  - By choosing the right ϕ_j, every continuous function can (in principle) be approximated with arbitrary accuracy.
- Disadvantage
  - The error function can in general no longer be minimized in closed form.

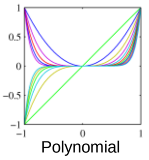
  $\Rightarrow$ Minimization with Gradient Descent

Recap: Basis Functions

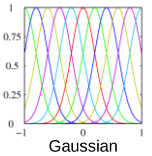
- Generally, we consider models of the following form

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x})$$

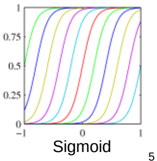
  - where $\phi_j(\mathbf{x})$ are known as *basis functions*.
  - In the simplest case, we use linear basis functions: $\phi_d(\mathbf{x}) = x_d$.
- Other popular basis functions



Polynomial



Gaussian



Sigmoid

Recap: Iterative Methods for Estimation

- Gradient Descent (1st order)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \left.\nabla E(\mathbf{w})\right|_{\mathbf{w}^{(\tau)}}$$

  - Simple and general
  - Relatively slow to converge, has problems with some functions
- Newton-Raphson (2nd order)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\, \mathbf{H}^{-1}\left.\nabla E(\mathbf{w})\right|_{\mathbf{w}^{(\tau)}}$$

  where $\mathbf{H} = \nabla\nabla E(\mathbf{w})$ is the Hessian matrix, i.e. the matrix of second derivatives.
  - Local quadratic approximation to the target function
  - Faster convergence

Recap: 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.
- Basic strategies
  - "Batch learning" $$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left.\frac{\partial E(\mathbf{w})}{\partial w_{kj}}\right|_{\mathbf{w}^{(\tau)}}$$
  - "Sequential updating" $$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left.\frac{\partial E_n(\mathbf{w})}{\partial w_{kj}}\right|_{\mathbf{w}^{(\tau)}}$$

    where $$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

Machine Learning Winter '19

Recap: Gradient Descent

- Example: Quadratic error function
$$E(\mathbf{w}) = \sum_{n=1}^{N} \left(y(\mathbf{x}_n;\mathbf{w}) - \mathbf{t}_n\right)^2$$
- Sequential updating leads to delta rule (=LMS rule)
$$\begin{aligned} w_{kj}^{(\tau+1)} &= w_{kj}^{(\tau)} - \eta\left(y_k(\mathbf{x}_n;\mathbf{w}) - t_{kn}\right)\phi_j(\mathbf{x}_n) \\ &= w_{kj}^{(\tau)} - \eta\delta_{kn}\phi_j(\mathbf{x}_n) \end{aligned}$$
  - where
$$\delta_{kn} = y_k(\mathbf{x}_n;\mathbf{w}) - t_{kn}$$

  $\Rightarrow$ Simply feed back the input data point, weighted by the classification error.

Machine Learning Winter '19

Slide adapted from Bernt Schiele

Recap: Gradient Descent

- Cases with differentiable, non-linear activation function
$$y_k(\mathbf{x}) = g(a_k) = g\left(\sum_{j=0}^{M} w_{ki}\phi_j(\mathbf{x}_n)\right)$$
- Gradient descent (again with quadratic error function)
$$\begin{aligned} \frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} &= \frac{\partial g(a_k)}{\partial w_{kj}}\left(y_k(\mathbf{x}_n;\mathbf{w}) - t_{kn}\right)\phi_j(\mathbf{x}_n) \\ w_{kj}^{(\tau+1)} &= w_{kj}^{(\tau)} - \eta\delta_{kn}\phi_j(\mathbf{x}_n) \\ \delta_{kn} &= \frac{\partial g(a_k)}{\partial w_{kj}}\left(y_k(\mathbf{x}_n;\mathbf{w}) - t_{kn}\right) \end{aligned}$$

Machine Learning Winter '19

Slide adapted from Bernt Schiele

Recap: Probabilistic Discriminative Models

- Consider models of the form
$$\begin{aligned} p(\mathcal{C}_1|\boldsymbol{\phi}) &= y(\boldsymbol{\phi}) = \sigma(\mathrm{w}^T\boldsymbol{\phi}) \\ \text{with} \quad p(\mathcal{C}_2|\boldsymbol{\phi}) &= 1 - p(\mathcal{C}_1|\boldsymbol{\phi}) \end{aligned}$$
- This model is called logistic regression.
- Properties
  - Probabilistic interpretation
  - But discriminative method: only focus on decision hyperplane
  - Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling $p(\phi|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$.

Machine Learning Winter '19

Recap: Logistic Regression

- Let's consider a data set $\{\phi_n, t_n\}$ with $n = 1,\ldots,N$, where $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$ and $t_n \in \{0,1\}$, $\mathbf{t} = (t_1,\ldots,t_N)^T$.
- With $y_n = p(\mathcal{C}_1|\phi_n)$, we can write the likelihood as
$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}\left\{1 - y_n\right\}^{1-t_n}$$
- Define the error function as the negative log-likelihood
$$\begin{aligned} E(\mathbf{w}) &= -\ln p(\mathbf{t}|\mathbf{w}) \\ &= -\sum_{n=1}^{N}\left\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\right\} \end{aligned}$$
  - This is the so-called cross-entropy error function.

Machine Learning Winter '19

Gradient of the Error Function

$$\boxed{\begin{aligned} y_n &= \sigma(\mathbf{w}^T\boldsymbol{\phi}_n) \\ \frac{dy_n}{d\mathbf{w}} &= y_n(1-y_n)\boldsymbol{\phi}_n \end{aligned}}$$

- Error function
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\left\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\right\}$$
- Gradient
$$\begin{aligned} \nabla E(\mathbf{w}) &= -\sum_{n=1}^{N}\left\{t_n \frac{\frac{d}{d\mathbf{w}}y_n}{y_n} + (1-t_n)\frac{\frac{d}{d\mathbf{w}}(1-y_n)}{(1-y_n)}\right\} \\ &= -\sum_{n=1}^{N}\left\{t_n \frac{\cancel{y_n}(1-y_n)}{\cancel{y_n}}\boldsymbol{\phi}_n - (1-t_n)\frac{y_n\cancel{(1-y_n)}}{\cancel{(1-y_n)}}\boldsymbol{\phi}_n\right\} \\ &= -\sum_{n=1}^{N}\left\{(t_n - \cancel{t_n y_n} - y_n + \cancel{t_n y_n})\boldsymbol{\phi}_n\right\} \\ &= \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n \end{aligned}$$

Machine Learning Winter '19

Gradient of the Error Function

- Gradient for logistic regression

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n$$

- Does this look familiar to you?
- This is the same result as for the Delta (=LMS) rule

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta(y_k(\mathbf{x}_n;\mathbf{w}) - t_{kn})\phi_j(\mathbf{x}_n)$$

- We can use this to derive a sequential estimation algorithm.
  - However, this will be quite slow…

Newton-Raphson for Least-Squares Estimation

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

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^T\boldsymbol{\phi}_n - t_n\right)^2$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}\left(\mathbf{w}^T\boldsymbol{\phi}_n - t_n\right)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w} - \boldsymbol{\Phi}^T\mathbf{t}$$

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N}\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

where $\boldsymbol{\Phi} = \begin{bmatrix}\boldsymbol{\phi}_1^T \\ \vdots \\ \boldsymbol{\phi}_N^T\end{bmatrix}$

- Resulting update scheme:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}(\boldsymbol{\Phi}^T\boldsymbol{\Phi}\mathbf{w}^{(\tau)} - \boldsymbol{\Phi}^T\mathbf{t})$$

$$= (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{t}$$

Closed-form solution!

Newton-Raphson for Logistic Regression

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

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

$$\boxed{\frac{dy_n}{d\mathbf{w}} = y_n(1-y_n)\boldsymbol{\phi}_n}$$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y}-\mathbf{t})$$

$$\mathbf{H} = \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N}y_n(1-y_n)\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T = \boldsymbol{\Phi}^T\mathbf{R}\boldsymbol{\Phi}$$

where $\mathbf{R}$ 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 $\mathbf{w}$ through the weighting matrix $\mathbf{R}$.

Iteratively Reweighted Least Squares

- Update equations

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

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

$$= (\boldsymbol{\Phi}^T\mathbf{R}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{R}\mathbf{z}$$

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

- 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.
  - $\Rightarrow$ Iteratively Reweighted Least-Squares (IRLS)

Summary: Logistic Regression

- Properties
  - Directly represent posterior distribution $p(\phi|\mathcal{C}_k)$
  - 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

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- Linear Support Vector Machines
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

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.

$$\mathbf{y}(\mathbf{x};\mathbf{w}) = \begin{bmatrix} P(y=1|\mathbf{x};\mathbf{w}) \\ P(y=2|\mathbf{x};\mathbf{w}) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})} \begin{bmatrix} \exp(\mathbf{w}_1^\top \mathbf{x}) \\ \exp(\mathbf{w}_2^\top \mathbf{x}) \\ \vdots \\ \exp(\mathbf{w}_K^\top \mathbf{x}) \end{bmatrix}$$

  - 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 with indicator function $\mathbb{I}(\cdot)$

$$\begin{aligned} E(\mathbf{w}) &= -\sum_{n=1}^{N} \{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \\ &= -\sum_{n=1}^{N}\sum_{k=0}^{1} \{\mathbb{I}(t_n = k)\ln P(y_n = k|\mathbf{x}_n;\mathbf{w})\} \end{aligned}$$

- Softmax regression
  - Generalization to K classes using indicator functions.

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})} \right\}$$

Optimization

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

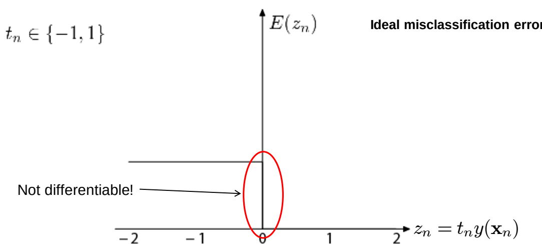
$$\nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\sum_{n=1}^{N} \left[\mathbb{I}(t_n = k)\ln P(y_n = k|\mathbf{x}_n;\mathbf{w})\right]$$

- 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

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- Linear Support Vector Machines
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

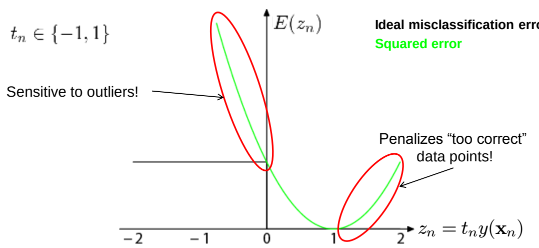
Note on Error Functions



- 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.

Image source: Bishop, 2006

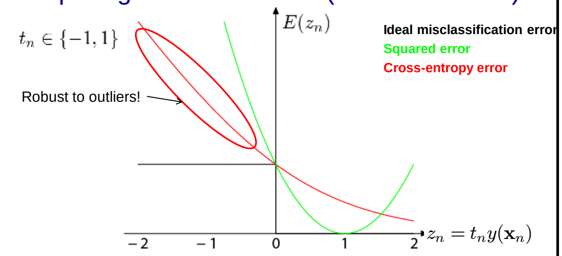
Note on Error Functions



- 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
  - $\Rightarrow$ Generally does not lead to good classifiers.

Image source: Bishop, 2006

Comparing Error Functions (Loss Functions)

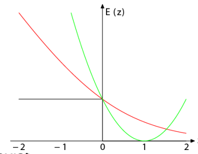


- Cross-Entropy Error
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - Robust to outliers, error increases only roughly linearly
  - But no closed-form solution, requires iterative estimation.

Image source: Bishop, 2006

Machine Learning Winter '19

Overview: Error Functions



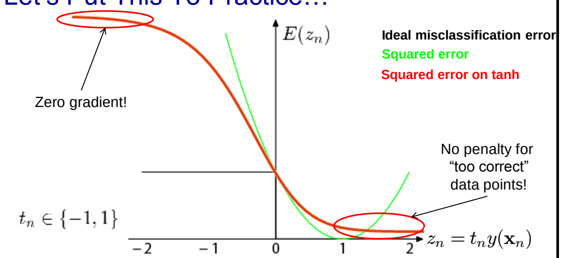
- Ideal Misclassification Error
  - This is what we would like to optimize.
  - But cannot compute gradients here.
- Quadratic Error
  - Easy to optimize, closed-form solutions exist.
  - But not robust to outliers.
- Cross-Entropy Error
  - Minimizer of this error is given by posterior class probabilities.
  - 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.*

B. Leibe

Machine Learning Winter '19

Let's Put This To Practice…



- Squared error on sigmoid/tanh output function
  - Avoids penalizing "too correct" data points.
  - But: zero gradient for confidently incorrect classifications!
  - ⇒ Do not use L_2 loss with sigmoid outputs (instead: cross-entropy)!

Image source: Bishop, 2006

Machine Learning Winter '19

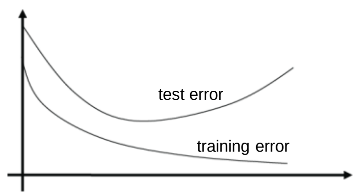
Topics of This Lecture

- Softmax Regression
  - Multi-class generalization
  - Gradient descent solution
- Note on Error Functions
  - Ideal error function
  - Quadratic error
  - Cross-entropy error
- Linear Support Vector Machines
  - Lagrangian (primal) formulation
  - Dual formulation
  - Discussion

B. Leibe

Machine Learning Winter '19

Generalization and Overfitting



- Goal: predict class labels of new observations
  - Train classification model on limited training set.
  - The further we optimize the model parameters, the more the training error will decrease.
  - However, at some point the test error will go up again.
  - ⇒ *Overfitting to the training set!*

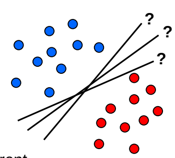
B. Leibe

Image source: B. Schiele

Machine Learning Winter '19

Example: Linearly Separable Data

- Overfitting is often a problem with linearly separable data
  - Which of the many possible decision boundaries is correct?
  - All of them have zero error on the training set…
  - However, they will most likely result in different predictions on novel test data.
    ⇒ Different generalization performance



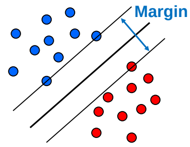
- How to select the classifier with the best generalization performance?

B. Leibe

Machine Learning Winter '19

Revisiting Our Previous Example…

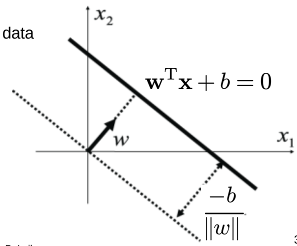
- How to select the classifier with the best generalization performance?
  - Intuitively, we would like to select the classifier which leaves maximal "safety room" for future data points.
  - This can be obtained by maximizing the margin between positive and negative data points.
  - It can be shown that the larger the margin, the lower the corresponding classifier's VC dimension (capacity for overfitting).



- The SVM takes up this idea
  - It searches for the classifier with maximum margin.
  - Formulation as a convex optimization problem
    $\Rightarrow$ Possible to find the globally optimal solution!

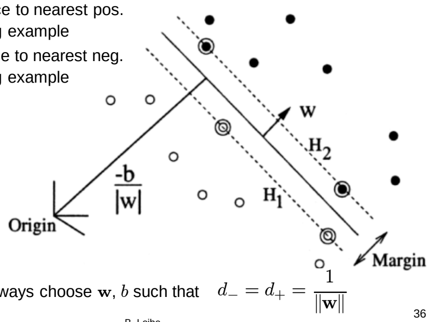
Support Vector Machine (SVM)

- Let's first consider linearly separable data
  - N training data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ $\quad \mathbf{x}_i \in \mathbb{R}^d$
  - Target values $\quad t_i \in \{-1, 1\}$
  - Hyperplane separating the data



Support Vector Machine (SVM)

- Margin of the hyperplane: $\quad d_- + d_+$
  - d_+: distance to nearest pos. training example
  - d_-: distance to nearest neg. training example



  - We can always choose $\mathbf{w}$, b such that $\quad d_- = d_+ = \frac{1}{\|\mathbf{w}\|}$

Support Vector Machine (SVM)

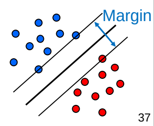
- Since the data is linearly separable, there exists a hyperplane with

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b \geq +1 \quad \text{for} \quad t_n = +1$$
$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b \cdot \ -1 \quad \text{for} \quad t_n = -1$$

- Combined in one equation, this can be written as

$$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

$\Rightarrow$ Canonical representation of the decision hyperplane.
  - The equation will hold exactly for the points on the margin
$$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) = 1$$
  - By definition, there will always be at least one such point.



Support Vector Machine (SVM)

- We can choose w such that
$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b = +1 \quad \text{for one} \quad t_n = +1$$
$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b = -1 \quad \text{for one} \quad t_n = -1$$

- The distance between those two hyperplanes is then the margin
$$d_- = d_+ = \frac{1}{\|\mathbf{w}\|}$$
$$d_- + d_+ = \frac{2}{\|\mathbf{w}\|}$$

$\Rightarrow$ We can find the hyperplane with maximal margin by minimizing $\|\mathbf{w}\|^2$

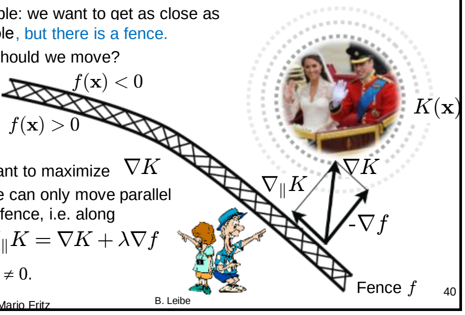
Support Vector Machine (SVM)

- Optimization problem
  - Find the hyperplane satisfying
$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$
    under the constraints
$$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 \quad \forall n$$
  - Quadratic programming problem with linear constraints.
  - Can be formulated using Lagrange multipliers.

- *Who is already familiar with Lagrange multipliers?*
  - Let's look at a real-life example…

Recap: Lagrange Multipliers

- Problem
  - We want to maximize $K(\mathbf{x})$ subject to constraints $f(\mathbf{x}) = 0$.
  - Example: we want to get as close as possible, but there is a fence.
  - How should we move?

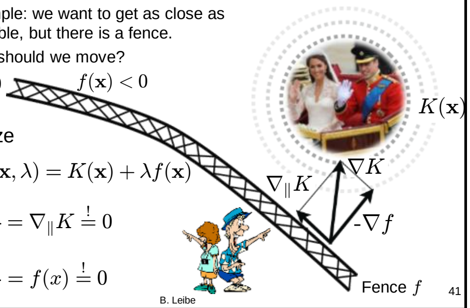


  - We want to maximize ∇K
  - But we can only move parallel to the fence, i.e. along
    $$\nabla_{\|} K = \nabla K + \lambda \nabla f$$
    with $\lambda \neq 0$.

Slide adapted from Mario Fritz

Recap: Lagrange Multipliers

- Problem
  - We want to maximize $K(\mathbf{x})$ subject to constraints $f(\mathbf{x}) = 0$.
  - Example: we want to get as close as possible, but there is a fence.
  - How should we move?



$\Rightarrow$ Optimize

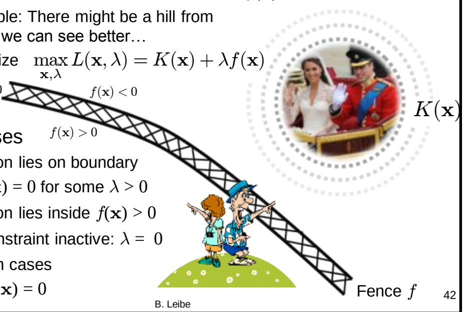
$$\max_{\mathbf{x},\lambda} L(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$$

$$\frac{\partial L}{\partial \mathbf{x}} = \nabla_{\|} K \overset{!}{=} 0$$

$$\frac{\partial L}{\partial \lambda} = f(x) \overset{!}{=} 0$$

Recap: Lagrange Multipliers

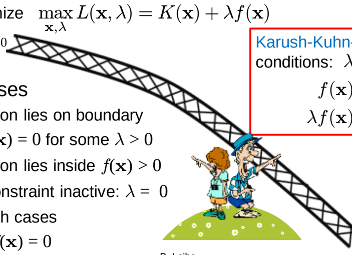
- Problem
  - Now let's look at constraints of the form $f(\mathbf{x}) \geq 0$.
  - Example: There might be a hill from which we can see better…
  - Optimize $\max_{\mathbf{x},\lambda} L(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$



- Two cases
  - Solution lies on boundary
    $\Rightarrow f(\mathbf{x}) = 0$ for some $\lambda > 0$
  - Solution lies inside $f(\mathbf{x}) > 0$
    $\Rightarrow$ Constraint inactive: $\lambda = 0$
  - In both cases
    $\Rightarrow \lambda f(\mathbf{x}) = 0$

Recap: Lagrange Multipliers

- Problem
  - Now let's look at constraints of the form $f(\mathbf{x}) \geq 0$.
  - Example: There might be a hill from which we can see better…
  - Optimize $\max_{\mathbf{x},\lambda} L(\mathbf{x}, \lambda) = K(\mathbf{x}) + \lambda f(\mathbf{x})$



Karush-Kuhn-Tucker (KKT) conditions:
$$\begin{aligned} \lambda &\geq 0 \\ f(\mathbf{x}) &\geq 0 \\ \lambda f(\mathbf{x}) &= 0 \end{aligned}$$

- Two cases
  - Solution lies on boundary
    $\Rightarrow f(\mathbf{x}) = 0$ for some $\lambda > 0$
  - Solution lies inside $f(\mathbf{x}) > 0$
    $\Rightarrow$ Constraint inactive: $\lambda = 0$
  - In both cases
    $\Rightarrow \lambda f(\mathbf{x}) = 0$

SVM – Lagrangian Formulation

- Find hyperplane minimizing $\|\mathbf{w}\|^2$ under the constraints
  $$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1 \geq 0 \quad \forall n$$
- Lagrangian formulation
  - Introduce positive Lagrange multipliers: $a_n \geq 0 \quad \forall n$
  - Minimize Lagrangian ("primal form")
    $$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1 \right\}$$
  - I.e., find $\mathbf{w}$, b, and $\mathbf{a}$ such that
    $$\frac{\partial L}{\partial b} = 0 \;\Rightarrow\; \boxed{\sum_{n=1}^{N} a_n t_n = 0} \qquad \frac{\partial L}{\partial \mathbf{w}} = 0 \;\Rightarrow\; \boxed{\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n}$$

SVM – Lagrangian Formulation

- Lagrangian primal form
  $$\begin{aligned} L_p &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1 \right\} \\ &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n y(\mathbf{x}_n) - 1 \right\} \end{aligned}$$
- The solution of L_p needs to fulfill the KKT conditions
  - Necessary and sufficient conditions
    $$\begin{aligned} a_n &\geq 0 \\ t_n y(\mathbf{x}_n) - 1 &\geq 0 \\ a_n \left\{ t_n y(\mathbf{x}_n) - 1 \right\} &= 0 \end{aligned}$$

KKT:
$$\begin{aligned} \lambda &\geq 0 \\ f(\mathbf{x}) &\geq 0 \\ \lambda f(\mathbf{x}) &= 0 \end{aligned}$$

SVM – Solution (Part 1)

- Solution for the hyperplane
  - Computed as a linear combination of the training examples

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$$

  - Because of the KKT conditions, the following must also hold

$$a_n \left(t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1\right) = 0$$

$\boxed{\text{KKT: } \lambda f(\mathbf{x}) = 0}$

  - This implies that $a_n > 0$ only for training data points for which

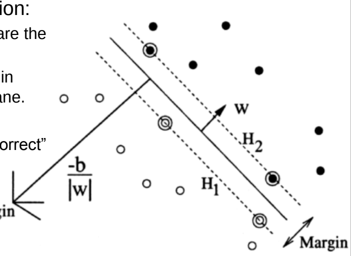
$$\left(t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1\right) = 0$$

$\Rightarrow$ *Only some of the data points actually influence the decision boundary!*

SVM – Support Vectors

- The training points for which $a_n > 0$ are called "support vectors".
- Graphical interpretation:
  - The support vectors are the points on the margin.
  - They *define* the margin and thus the hyperplane.

$\Rightarrow$ Robustness to "too correct" points!



SVM – Solution (Part 2)

- Solution for the hyperplane
  - To define the decision boundary, we still need to know b.
  - Observation: any support vector $\mathbf{x}_n$ satisfies

$$t_n y(\mathbf{x}_n) = t_n \left(\sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^{\mathrm{T}} \mathbf{x}_n + b \right) = 1$$

$\boxed{\text{KKT: } f(\mathbf{x}) \geq 0}$

  - Using $t_n^2 = 1$ we can derive: $\quad b = t_n - \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^{\mathrm{T}} \mathbf{x}_n$
  - In practice, it is more robust to average over all support vectors:

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m \mathbf{x}_m^{\mathrm{T}} \mathbf{x}_n \right)$$

SVM – Discussion (Part 1)

- Linear SVM
  - Linear classifier
  - SVMs have a "guaranteed" generalization capability.
  - Formulation as convex optimization problem.

  $\Rightarrow$ Globally optimal solution!
- Primal form formulation
  - Solution to quadratic prog. problem in M variables is in $\mathcal{O}(M^3)$.
  - Here: D variables $\Rightarrow \mathcal{O}(D^3)$
  - Problem: scaling with high-dim. data ("curse of dimensionality")

SVM – Dual Formulation

- Improving the scaling behavior: rewrite L_p in a dual form

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1 \right\}$$

$$= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n t_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n - b \overset{=0}{\cancel{\sum_{n=1}^{N} a_n t_n}} + \sum_{n=1}^{N} a_n$$

  - Using the constraint $\sum_{n=1}^{N} a_n t_n = 0$ we obtain

$\boxed{\dfrac{\partial L_p}{\partial b} = 0}$

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n t_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n + \sum_{n=1}^{N} a_n$$

SVM – Dual Formulation

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n t_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n + \sum_{n=1}^{N} a_n$$

- Using the constraint $\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$ we obtain

$\boxed{\dfrac{\partial L_p}{\partial \mathbf{w}} = 0}$

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n t_n \sum_{m=1}^{N} a_m t_m \mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n + \sum_{n=1}^{N} a_n$$

$$= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n) + \sum_{n=1}^{N} a_n$$

SVM – Dual Formulation

$$L = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n) + \sum_{n=1}^{N} a_n$$

- Applying $\frac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$ and again using $\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$

$$\frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} = \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n)$$

- Inserting this, we get the Wolfe dual

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n)$$

Machine Learning Winter '19

Slide adapted from Bernt Schiele

B. Leibe

53

SVM – Dual Formulation

- Maximize

$$L_d(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m (\mathbf{x}_m^{\mathrm{T}}\mathbf{x}_n)$$

under the conditions

$$a_n \geq 0 \quad \forall n$$

$$\sum_{n=1}^{N} a_n t_n = 0$$

- The hyperplane is given by the $N_{\mathcal{S}}$ support vectors:

$$\mathbf{w} = \sum_{n=1}^{N_{\mathcal{S}}} a_n t_n \mathbf{x}_n$$

Machine Learning Winter '19

Slide adapted from Bernt Schiele

B. Leibe

54

SVM – Discussion (Part 2)

- Dual form formulation
  - In going to the dual, we now have a problem in N variables (a_n).
  - Isn't this worse??? We penalize large training sets!
- However…
  1. SVMs have sparse solutions: $a_n \neq 0$ only for support vectors!

     ⇒ This makes it possible to construct efficient algorithms
     - e.g. Sequential Minimal Optimization (SMO)
     - Effective runtime between $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$.
  2. We have avoided the dependency on the dimensionality.

     ⇒ This makes it possible to work with infinite-dimensional feature spaces by using suitable basis functions $\phi(\mathbf{x})$.

     ⇒ We'll see that in the next lecture…

Machine Learning Winter '19

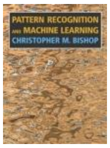
B. Leibe

55

References and Further Reading

- More information on SVMs can be found in Chapter 7.1 of Bishop's book.

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



- Additional information about Statistical Learning Theory and a more in-depth introduction to SVMs are available in the following tutorial:
  - C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, Vol. 2(2), pp. 121-167 1998.

Machine Learning Winter '19

B. Leibe

56