Topics of This Lecture

- **Support Vector Machines (Recap)**
  - Lagrangian (primal) formulation
  - Dual formulation
  - Soft-margin classification
- **Nonlinear Support Vector Machines**
  - Nonlinear basis functions
  - The Kernel trick
  - Mercer’s condition
  - Popular kernels
- **Analysis**
  - VC dimensions
  - Error function
- **Applications**

Recap: SVM - Primal Formulation

- Lagrangian primal form
  \[ L_p = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \{ t_n(w^T x_n + b) - 1 \} \]
  \[ = \frac{1}{2} \|w\|^2 - \sum_{n=1}^{N} a_n \{ t_n y(x_n) - 1 \} \]
- The solution of \( L_p \) needs to fulfill the KKT conditions
  - Necessary and sufficient conditions
    \[ a_n \geq 0 \]
    \[ t_n y(x_n) - 1 \geq 0 \]
    \[ a_n \{ t_n y(x_n) - 1 \} = 0 \]
  - KKT:
    \[ \lambda \geq 0 \]
    \[ f(x) \geq 0 \]
    \[ \lambda f(x) = 0 \]

Recap: SVM - Solution

- Solution for the hyperplane
  - Computed as a linear combination of the training examples
    \[ w = \sum_{n=1}^{N} a_n t_n x_n \]
  - Sparse solution: \( a_i \neq 0 \) only for some points, the support vectors
  - Only the SVs actually influence the decision boundary!
  - Compute \( b \) by averaging over all support vectors:
    \[ b = \frac{1}{N_S} \sum_{n \in S} \left( t_n - \sum_{m \in S} a_m t_m x_m^T x_n \right) \]
Recap: 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.

\[ \Rightarrow \text{All other data points can be discarded!} \]

Recap: SVM - Dual Formulation

- Maximize

\[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m) \]

under the conditions

\[ a_n \geq 0 \quad \forall n \]
\[ \sum_{n=1}^{N} a_n t_n = 0 \]

- Comparison
  - \( L_d \) is equivalent to the primal form \( L_p \), but only depends on \( a_n \).
  - \( L_p \) scales with \( O(D^3) \).
  - \( L_d \) scales with \( O(N^3) \)--in practice between \( O(N) \) and \( O(N^2) \).

Recap: SVM for Non-Separable Data

- Slack variables
  - One slack variable \( \xi_n \geq 0 \) for each training data point.
- Interpretation
  - \( \xi_n = 0 \) for points that are on the correct side of the margin.
  - \( \xi_n = y_n - y(x_n) \) for all other points.

\[ \Rightarrow \text{We do not have to set the slack variables ourselves!} \]
\[ \Rightarrow \text{They are jointly optimized together with } w. \]

Recap: SVM - New Dual Formulation

- New SVM Dual: Maximize

\[ L_d(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m (x_n^T x_m) \]

under the conditions

\[ 0 \cdot a_n \cdot C \quad \text{This is all that changed!} \]
\[ \sum_{n=1}^{N} a_n t_n = 0 \]

- This is again a quadratic programming problem

\[ \Rightarrow \text{Solve as before...} \]

Interpretation of Support Vectors

- Those are the hard examples!
  - We can visualize them, e.g., for face detection

Topics of This Lecture

- Support Vector Machines (Recap)
  - Lagrangian (primal) formulation
  - Dual formulation
  - Soft-margin classification
- Nonlinear Support Vector Machines
  - Nonlinear basis functions
  - The Kernel trick
  - Mercer's condition
  - Popular kernels
- Analysis
  - VC dimensions
  - Error function
- Applications
So Far...

- Only looked at linearly separable case...
  - Current problem formulation has no solution if the data are not linearly separable!
  - Need to introduce some tolerance to outlier data points.
    ⇒ Slack variables.

- Only looked at linear decision boundaries...
  - This is not sufficient for many applications.
  - Want to generalize the ideas to non-linear boundaries.

Nonlinear SVM

- Linear SVMs
  - Datasets that are linearly separable with some noise work well:
    ⇒ But what are we going to do if the dataset is just too hard?
  - How about... mapping data to a higher-dimensional space:

Another Example

- Non-separable by a hyperplane in 2D

Another Example

- Separable by a surface in 3D

Nonlinear SVM - Feature Spaces

- General idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable:

Nonlinear SVM

- General idea
  - Nonlinear transformation \( \phi \) of the data points \( x_n \):
    \[ x \in \mathbb{R}^D \quad \phi : \mathbb{R}^D \to \mathcal{H} \]
  - Hyperplane in higher-dim. space \( \mathcal{H} \) (linear classifier in \( \mathcal{H} \))
    \[ w^T \phi(x) + b = 0 \]
  ⇒ Nonlinear classifier in \( \mathbb{R}^D \).
Mapping to polynomial space, $x, y \in \mathbb{R}^2$:

$$
\phi(x) = \begin{bmatrix}
    x^2_1 \\
    x^2_2
\end{bmatrix}
$$

Motivation: Easier to separate data in higher-dimensional space.

But wait - isn't there a big problem?

How should we evaluate the decision function?

Solution: The Kernel Trick

$\phi(x)$ only appears in the form of dot products $\phi(x)^T \phi(y)$:

$$
y(x) = w^T \phi(x) + b
$$

Trick: Define a so-called kernel function $k(x, y) = \phi(x)^T \phi(y)$.

Now, in place of the dot product, use the kernel instead:

$$
y(x) = \sum_{n=1}^{N} a_n t_n \phi(x_n) + b
$$

The kernel function implicitly maps the data to the higher-dimensional space (without having to compute $\phi(x)$ explicitly)!

Problem with High-dim. Basis Functions

Problem

In order to apply the SVM, we need to evaluate the function

$$
y(x) = w^T \phi(x) + b
$$

Using the hyperplane, which is itself defined as

$$
w = \sum_{n=1}^{N} a_n t_n \phi(x_n)
$$

$\Rightarrow$ What happens if we try this for a million-dimensional feature space $\phi(x)$?

Oh-oh...

Solution: What Could This Look Like?

• Example:
  * Mapping to polynomial space, $x, y \in \mathbb{R}^2$:

$$
\phi(x) = \begin{bmatrix}
    x^2_1 \\
    x^2_2
\end{bmatrix}
$$

Back to Our Previous Example...

$2^\text{nd}$ degree polynomial kernel:

$$
\phi(x)^T \phi(y) = \begin{bmatrix}
    x^2_1 \\
    x^2_2
\end{bmatrix} \begin{bmatrix}
    y^2_1 \\
    y^2_2
\end{bmatrix} = x^2_1 y^2_1 + 2x_1 x_2 y_1 y_2 + x^2_2 y^2_2 = (x^T y)^2 =: k(x, y)
$$

Whenever we evaluate the kernel function $k(x,y) = (x^T y)^2$, we implicitly compute the dot product in the higher-dimensional feature space.

SVMs with Kernels

• Using kernels
  * Applying the kernel trick is easy. Just replace every dot product by a kernel function...

$$
x^T y \rightarrow k(x, y)
$$

...and we’re done.

Instead of the raw input space, we’re now working in a higher-dimensional (potentially infinite dimensional) space, where the data is more easily separable.

“Sounds like magic…”

• Wait - does this always work?
  * The kernel needs to define an implicit mapping to a higher-dimensional feature space $\phi(x)$.
  * When is this the case?

Which Functions are Valid Kernels?

• Mercer’s theorem (modernized version):
  * Every positive definite symmetric function is a kernel.

• Positive definite symmetric functions correspond to a positive definite symmetric Gram matrix:

$$
K = \begin{bmatrix}
    k(x_1, x_1) & \cdots & k(x_1, x_N) \\
    \vdots & \ddots & \vdots \\
    k(x_N, x_1) & \cdots & k(x_N, x_N)
\end{bmatrix}
$$

(positive definite = all eigenvalues are $> 0$)
Kernels Fulfilling Mercer’s Condition

- Polynomial kernel
  \[ k(x, y) = (x^T y + 1)^p \]
- Radial Basis Function kernel
  \[ k(x, y) = \exp \left\{ -\frac{(x - y)^2}{2\sigma^2} \right\} \quad \text{e.g., Gaussian} \]
- Hyperbolic tangent kernel
  \[ k(x, y) = \tanh(x^T y + b) \quad \text{e.g., Sigmoid} \]

(and many, many more…)

Example: Bag of Visual Words Representation

- General framework in visual recognition
  - Create a codebook (vocabulary) of prototypical image features
  - Represent images as histograms over codebook activations
  - Compare two images by any histogram kernel, e.g. \( \chi^2 \) kernel

Nonlinear SVM - Dual Formulation

- SVM Dual: Maximize
  \[ L_\alpha(a) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \]
  under the conditions
  \[ \sum_{n=1}^{N} a_n = C \]
  \[ \sum_{n=1}^{N} a_n t_n = 0 \]
- Classify new data points using
  \[ y(x) = \sum_{n=1}^{N} a_n t_n k(x_n, x) + b \]

Summary: SVMs

- Properties
  - Empirically, SVMs work very, very well.
  - SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
  - SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
  - SVM techniques have been applied to a variety of other tasks
    - e.g. SV Regression, One-class SVMs, ...
  - The kernel trick has been used for a wide variety of applications. It can be applied wherever dot products are in use
    - e.g. Kernel PCA, kernel FLD, ...
    - Good overview, software, and tutorials available on http://www.kernel-machines.org/

- Limitations
  - How to select the right kernel?
    - Best practice guidelines are available for many applications
  - How to select the kernel parameters?
    - (Massive) cross-validation.
    - Usually, several parameters are optimized together in a grid search.
  - Solving the quadratic programming problem
    - Standard QP solvers do not perform too well on SVM task.
    - Dedicated methods have been developed for this, e.g. SMO.
  - Speed of evaluation
    - Evaluating \( y(x) \) scales linearly in the number of SVs.
    - Too expensive if we have a large number of support vectors.
    - \( \Rightarrow \) There are techniques to reduce the effective SV set.
  - Training for very large datasets (millions of data points)
    - Stochastic gradient descent and other approximations can be used
Topics of This Lecture

- Support Vector Machines (Recap)
  - Lagrangian (primal) formulation
  - Dual formulation
  - Soft-margin classification
- Nonlinear Support Vector Machines
  - Nonlinear basis functions
  - The Kernel trick
  - Mercer's condition
  - Popular kernels
- Analysis
  - VC dimensions
  - Error function
- Applications

Recap: Kernels Fulfilling Mercer’s Condition

- Polynomial kernel
  \[ k(x, y) = (x^T y + 1)^p \]
- Radial Basis Function kernel
  \[ k(x, y) = \exp \left\{-\frac{(x - y)^2}{2\sigma^2}\right\} \quad \text{e.g. Gaussian} \]
- Hyperbolic tangent kernel
  \[ k(x, y) = \tanh(ax^T y + \beta) \quad \text{e.g. Sigmoid} \]

(And many, many more...)

VC Dimension for Polynomial Kernel

- Polynomial kernel of degree \( p \):
  \[ k(x, y) = (x^T y)^p \]
- Dimensionality of \( \mathcal{H} \):
  \[ D + p - 1 \]
- Example:
  \[ D = 16 \times 16 = 256 \]
  \[ p = 4 \]
  \[ \dim(\mathcal{H}) = 183.181.376 \]
- The hyperplane in \( \mathcal{H} \) then has VC-dimension
  \[ \dim(\mathcal{H}) + 1 \]

VC Dimension for Gaussian RBF Kernel

- Intuitively
  - If we make the radius of the RBF kernel sufficiently small, then each data point can be associated with its own kernel.
  - However, this also means that we can get finite VC-dimension if we set a lower limit to the RBF radius.

Example: RBF Kernels

- Decision boundary on toy problem

Image source: B. Schoelkopf, A. Smola, 2002
But... but... but...

- Don’t we risk overfitting with those enormously high-dimensional feature spaces?
  - No matter what the basis functions are, there are really only up to $N$ parameters: $a_1, a_2, ...$, and most of them are usually set to zero by the maximum margin criterion.
  - The data effectively lives in a low-dimensional subspace of $H$.

- What about the VC dimension? I thought low VC-dim was good (in the sense of the risk bound)?
  - Yes, but the maximum margin classifier “magically” solves this.
  - Reason (Vapnik): by maximizing the margin, we can reduce the VC-dimension.
  - Empirically, SVMs have very good generalization performance.

Theoretical Justification for Maximum Margins

- For the general case, Vapnik has proven the following:
  - The class of optimal linear separators has VC dimension $h$ bounded from above as
    $$ h \leq \min \left( \frac{D^2}{\rho^2}, m_b \right) + 1 $$
    where $\rho$ is the margin, $D$ is the diameter of the smallest sphere that can enclose all of the training examples, and $m_b$ is the dimensionality.

  - Intuitively, this implies that regardless of dimensionality $m_b$, we can minimize the VC dimension by maximizing the margin $\rho$.

  - Thus, complexity of the classifier is kept small regardless of dimensionality.

SVM - Analysis

- Traditional soft-margin formulation
  $$ \min_{w \in \mathbb{R}^D, z \in \mathbb{R}^+} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^N \xi_n $$
  subject to the constraints
  $$ t_n y(x_n) \geq 1 - \xi_n $$
  “Maximize the margin”
  “Most points should be on the correct side of the margin”

- Different way of looking at it
  - We can reformulate the constraints into the objective function.
    $$ \min_{w \in \mathbb{R}^D} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^N \left[ 1 - t_n y(x_n) \right]^+ $$
    $L_2$ regularizer  “Hinge loss”

- Gap Tolerant Classifier
  - Classifier is defined by a ball in $\mathbb{R}^D$ with diameter $D$ enclosing all points and two parallel hyperplanes with distance $\frac{D}{2}$ (the margin).
  - Points in the ball are assigned class $-1$; outside are assigned class $1$.

  - VC dimension of this classifier depends on the margin
    - $M \leq \frac{3}{4} D$  $\Rightarrow$  3 points can be shattered
    - $\frac{3}{4} D < M < D$  $\Rightarrow$  2 points can be shattered
    - $M \geq D$  $\Rightarrow$  1 point can be shattered

  - By maximizing the margin, we can minimize the VC dimension.

Topics of This Lecture

- Support Vector Machines (Recap)
  - Lagrangian (primal) formulation
  - Dual formulation
  - Soft-margin classification
- Nonlinear Support Vector Machines
  - Nonlinear basis functions
  - The Kernel trick
  - Mercer’s condition
  - Popular kernels
- Analysis
  - VC dimensions
  - Error function
- Applications

Recap: Error Functions

- $t_n \subset \{ -1, 1 \}$
- Ideal misclassification error function
  - $E(\xi_n)$

- Not differentiable!
  - Ideal misclassification error function (black)
    - This is what we want to approximate,
      - Unfortunately, it is not differentiable.
      - The gradient is zero for misclassified points.
    - We cannot minimize it by gradient descent.
Recap: 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
  - Generally does not lead to good classifiers.

Image source: Bishop, 2006

Ideal misclassification error
Squared error
Penalizes “too correct” data points!
Sensitive to outliers!

Error Functions (Loss Functions)

- “Hinge error” used in SVMs
  - Zero error for points outside the margin (zn > 1) ⇒ sparsity
  - Linear penalty for misclassified points (zn < 1) ⇒ robustness
  - Not differentiable around zn = 1 ⇒ Cannot be optimized directly.

Hinge loss enforces sparsity
- Only a subset of training data points actually influences the decision boundary.
- This is different from sparsity obtained through the regularizer!
  - There, only a subset of input dimensions are used.
- Unconstrained optimization, but non-differentiable function.
  - Solve, e.g. by subgradient descent
  - Currently most efficient: stochastic gradient descent

Topics of This Lecture

- Support Vector Machines (Recap)
  - Lagrangian (primal) formulation
  - Dual formulation
  - Soft-margin classification
- Nonlinear Support Vector Machines
  - Nonlinear basis functions
  - The Kernel trick
  - Mercer’s condition
  - Popular kernels
- Analysis
  - VC dimensions
  - Error function
- Applications

Example Application: Text Classification

- Problem:
  - Classify a document in a number of categories

- Representation:
  - “Bag-of-words” approach
  - Histogram of word counts (on learned dictionary)
  - Very high-dimensional feature space (~10,000 dimensions)
  - Few irrelevant features
- This was one of the first applications of SVMs
  - T. Joachims (1997)

<table>
<thead>
<tr>
<th>Rayon/Rochio/Cl5</th>
<th>SVM (pol)</th>
<th>SVM (rM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>degree d</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>corn</td>
<td>91.0</td>
<td>96.1</td>
</tr>
<tr>
<td>acy</td>
<td>91.0</td>
<td>92.1</td>
</tr>
<tr>
<td>money-fx</td>
<td>62.9</td>
<td>67.5</td>
</tr>
<tr>
<td>grain</td>
<td>72.9</td>
<td>79.5</td>
</tr>
<tr>
<td>crude</td>
<td>81.0</td>
<td>81.5</td>
</tr>
<tr>
<td>trade</td>
<td>50.0</td>
<td>77.4</td>
</tr>
<tr>
<td>internet</td>
<td>50.0</td>
<td>72.5</td>
</tr>
<tr>
<td>ship</td>
<td>78.7</td>
<td>85.1</td>
</tr>
<tr>
<td>wheat</td>
<td>86.0</td>
<td>79.4</td>
</tr>
<tr>
<td>corn</td>
<td>47.3</td>
<td>62.2</td>
</tr>
<tr>
<td>microg</td>
<td>72.0</td>
<td>79.0</td>
</tr>
</tbody>
</table>

Example Application: Text Classification

- Results:
Example Application: Text Classification

- This is also how you could implement a simple spam filter...

![Diagram](dictionary-svm-mailbox-trash)

Example Application: OCR

- Handwritten digit recognition
  - US Postal Service Database
  - Standard benchmark task for many learning algorithms

Historical Importance

- USPS benchmark
  - 2.5% error: human performance

- Different learning algorithms
  - 16.2% error: Decision tree (C4.5)
  - 5.9% error: (best) 2-layer Neural Network
  - 5.1% error: LeNet 1 - (massively hand-tuned) 5-layer network

- Different SVMs
  - 4.0% error: Polynomial kernel (p=3, 274 support vectors)
  - 4.1% error: Gaussian kernel (σ=0.3, 291 support vectors)

Example Application: OCR

- Results
  - Almost no overfitting with higher-degree kernels.

<table>
<thead>
<tr>
<th>degree of polynomial</th>
<th>dimensionality of feature space</th>
<th>support vectors</th>
<th>raw error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>256</td>
<td>282</td>
<td>8.9</td>
</tr>
<tr>
<td>2</td>
<td>≈ 33000</td>
<td>227</td>
<td>4.7</td>
</tr>
<tr>
<td>3</td>
<td>≈ 1 × 10^8</td>
<td>274</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>≈ 1 × 10^9</td>
<td>321</td>
<td>4.2</td>
</tr>
<tr>
<td>5</td>
<td>≈ 1 × 10^{12}</td>
<td>374</td>
<td>4.3</td>
</tr>
<tr>
<td>6</td>
<td>≈ 1 × 10^{14}</td>
<td>377</td>
<td>4.5</td>
</tr>
<tr>
<td>7</td>
<td>≈ 1 × 10^{16}</td>
<td>422</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Example Application: Object Detection

- Sliding-window approach

![Diagram](sliding-window-hog)

- E.g. histogram representation (HOG)
  - Map each grid cell in the input window to a histogram of gradient orientations.
  - Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

Example Application: Pedestrian Detection

- N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005
Many Other Applications

- Lots of other applications in all fields of technology
  - OCR
  - Text classification
  - Computer vision
  - ...
  - High-energy physics
  - Monitoring of household appliances
  - Protein secondary structure prediction
  - Design on decision feedback equalizers (DFE) in telephony

(Detailed references in Schoelkopf & Smola, 2002, pp. 221)

Topics of This Lecture

- Support Vector Machines (Recap)
  - Lagrangian (primal) formulation
  - Dual formulation
  - Soft-margin classification
  - Nonlinear Support Vector Machines
- Analysis
  - VC dimensions
  - Error function
- Applications
  - Extensions
    - One-class SVMs

Summary: SVMs

- Properties
  - Empirically, SVMs work very, very well.
  - SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
  - SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
  - SVM techniques have been applied to a variety of other tasks
    - e.g. SV Regression, One-class SVMs, ...
  - The kernel trick has been used for a wide variety of applications. It can be applied wherever dot products are in use
    - e.g. Kernel PCA, kernel FLD, ...
  - Good overview, software, and tutorials available on http://www.kernel-machines.org/

- Limitations
  - How to select the right kernel?
    - Requires domain knowledge and experiments...
  - How to select the kernel parameters?
    - (Massive) cross-validation.
    - Usually, several parameters are optimized together in a grid search.
  - Solving the quadratic programming problem
    - Standard QP solvers do not perform too well on SVM task.
    - Dedicated methods have been developed for this, e.g. SMO.
  - Speed of evaluation
    - Evaluating \( y(x) \) scales linearly in the number of SVs.
    - Too expensive if we have a large number of support vectors.
    => There are techniques to reduce the effective SV set.
  - Training for very large datasets (millions of data points)
    - Stochastic gradient descent and other approximations can be used

You Can Try It At Home...

- Lots of SVM software available, e.g.
  - svmlight (http://svmlight.joachims.org/)
    - Command-line based interface
    - Source code available (in C)
    - Interfaces to Python, MATLAB, Perl, Java, DLL, ...
  - libsvm (http://www.csie.ntu.edu.tw/~cjlin/libsvm/)
    - Library for inclusion with own code
    - C++ and Java sources
    - Interfaces to Python, R, MATLAB, Perl, Ruby, Weka, C+ .NET, ...
  - Both include fast training and evaluation algorithms, support for multi-class SVMs, automated training and cross-validation, ...
  => Easy to apply to your own problems!

References and Further Reading

- More information on SVMs can be found in Chapter 7.1 of Bishop’s book. You can also look at Schölkopf & Smola (some chapters available online).
- A more in-depth introduction to SVMs is available in the following tutorial: