Binary Classification

Given training data \((x_i, y_i)\) for \(i = 1 \ldots N\), with \(x_i \in \mathbb{R}^d\) and \(y_i \in \{-1, 1\}\), learn a classifier \(f(x)\) such that

\[
\begin{align*}
    f(x_i) \geq 0 & \quad y_i = +1 \\
    f(x_i) < 0 & \quad y_i = -1
\end{align*}
\]

i.e. \(y_i f(x_i) > 0\) for a correct classification.

Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman
Linear separability

- linearly separable
  - Example 1
  - Example 2

- not linearly separable
  - Example 3
  - Example 4

Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman
Linear classifiers

A linear classifier has the form

$$f(x) = w^T x + b$$

- in 2D the discriminant is a line
- $w$ is the normal to the line, and $b$ the bias
- $w$ is known as the weight vector
What is the best $w$?

- maximum margin solution: most stable under perturbations of the inputs
Support Vector Machine

Linearly separable data

\[ f(x) = \sum_i \alpha_i y_i (x_i^T x) + b \]

where

- \( f(x) \) is the decision function.
- \( \alpha_i \) are the Lagrange multipliers.
- \( y_i \) are the class labels (1 or -1).
- \( x_i \) are the data points.
- \( x \) is the input vector.
- \( b \) is the bias term.
- \( (x_i^T x) \) is the inner product of \( x_i \) and \( x \).

The margin is defined by the distance of the support vectors to the decision boundary.

The SVM classifier maximizes the margin between the two classes.
SVM – sketch derivation

- Since $w^T x + b = 0$ and $c(w^T x + b) = 0$ define the same plane, we have the freedom to choose the normalization of $w$

- Choose normalization such that $w^T x_+ + b = +1$ and $w^T x_- + b = -1$ for the positive and negative support vectors respectively

- Then the margin is given by

$$\frac{w}{\|w\|} \cdot (x_+ - x_-) = \frac{w^T (x_+ - x_-)}{\|w\|} = \frac{2}{\|w\|}$$
Support Vector Machine

linearly separable data

Margin = \frac{2}{||w||}

\( w^T x + b = 1 \)
\( w^T x + b = 0 \)
\( w^T x + b = -1 \)

Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman
SVM – Optimization

- Learning the SVM can be formulated as an optimization:
  \[
  \max \frac{2}{||w||} \quad \text{subject to} \quad \begin{cases} 
  w^\top x_i + b \geq 1 & \text{if } y_i = +1 \\
  \leq -1 & \text{if } y_i = -1 
  \end{cases} \quad \text{for } i = 1 \ldots N
  \]

- Or equivalently
  \[
  \min ||w||^2 \quad \text{subject to } y_i \left( w^\top x_i + b \right) \geq 1 \quad \text{for } i = 1 \ldots N
  \]

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum.
Linear separability again: What is the best $w$?

- the points can be linearly separated but there is a very narrow margin

- but possibly the large margin solution is better, even though one constraint is violated

In general there is a trade off between the margin and the number of mistakes on the training data.
Introduce “slack” variables

\[ \xi_i \geq 0 \]

- for \( 0 < \xi \leq \frac{1}{||w||} \) point is between margin and correct side of hyperplane. This is a margin violation
- for \( \xi > \frac{1}{||w||} \) point is misclassified

\[ w^T x + b = 1 \]
\[ w^T x + b = 0 \]
\[ w^T x + b = -1 \]
“Soft” margin solution

The optimization problem becomes

\[
\min_{w \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} ||w||^2 + C \sum_{i=1}^{N} \xi_i
\]

subject to

\[
y_i \left( w^T x_i + b \right) \geq 1 - \xi_i \text{ for } i = 1 \ldots N
\]

- Every constraint can be satisfied if \( \xi_i \) is sufficiently large
- \( C \) is a **regularization** parameter:
  - small \( C \) allows constraints to be easily ignored \( \rightarrow \) large margin
  - large \( C \) makes constraints hard to ignore \( \rightarrow \) narrow margin
  - \( C = \infty \) enforces all constraints: hard margin
- This is still a quadratic optimization problem and there is a unique minimum. Note, there is only one parameter, \( C' \).
Loss function

\[
\min_{w \in \mathbb{R}^d} \|w\|^2 + C \sum_{i}^{N} \max(0, 1 - y_i f(x_i))
\]

Points are in three categories:

1. \(y_i f(x_i) > 1\)
   - Point is outside margin.
   - No contribution to loss.

2. \(y_i f(x_i) = 1\)
   - Point is on margin.
   - No contribution to loss.
   - As in hard margin case.

3. \(y_i f(x_i) < 1\)
   - Point violates margin constraint.
   - Contributes to loss.
SVM – review

- We have seen that for an SVM learning a linear classifier
  \[ f(x) = \mathbf{w}^\top x + b \]
  is formulated as solving an optimization problem over \( \mathbf{w} \):
  \[
  \min_{\mathbf{w} \in \mathbb{R}^d} ||\mathbf{w}||^2 + C \sum_{i} \max (0, 1 - y_i f(x_i))
  \]
- This quadratic optimization problem is known as the **primal** problem.

- Instead, the SVM can be formulated to learn a linear classifier
  \[ f(x) = \sum_{i}^{N} \alpha_i y_i (x_i^\top x) + b \]
  by solving an optimization problem over \( \alpha_i \).
- This is know as the **dual** problem, and we will look at the advantages of this formulation.
Primal and dual formulations

$N$ is number of training points, and $d$ is dimension of feature vector $\mathbf{x}$.

**Primal problem:** for $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_{i}^{N} \max(0, 1 - y_if(x_i))$$

**Dual problem:** for $\alpha \in \mathbb{R}^N$ (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_{i} \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k (x_j^\top x_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn $d$ parameters for primal, and $N$ for dual
- If $N << d$ then more efficient to solve for $\alpha$ than $\mathbf{w}$
- Dual form only involves $(x_j^\top x_k)$. We will return to why this is an advantage when we look at kernels.
Support Vector Machine

\[ f(x) = \sum_i \alpha_i y_i (x_i^\top x) + b \]

\[ w^\top x + b = 0 \]
Dual Classifier in transformed feature space

Classifier:

\[ f(x) = \sum_{i}^{N} \alpha_i y_i x_i^\top x + b \]

\[ \rightarrow f(x) = \sum_{i}^{N} \alpha_i y_i \Phi(x_i)^\top \Phi(x) + b \]

Learning:

\[ \max_{\alpha_i \geq 0} \sum_{i} \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k x_j^\top x_k \]

\[ \rightarrow \max_{\alpha_i \geq 0} \sum_{i} \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \Phi(x_j)^\top \Phi(x_k) \]

subject to

\[ 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_{i} |\alpha_i y_i| = 0 \]
Dual Classifier in transformed feature space

- Note, that $\Phi(x)$ only occurs in pairs $\Phi(x_j)^T \Phi(x_i)$

- Once the scalar products are computed, only the $N$ dimensional vector $\alpha$ needs to be learnt; it is not necessary to learn in the $D$ dimensional space, as it is for the primal

- Write $k(x_j, x_i) = \Phi(x_j)^T \Phi(x_i)$. This is known as a Kernel

Classifier:

$$f(x) = \sum_{i}^{N} \alpha_i y_i k(x_i, x) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_{i} \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k k(x_j, x_k)$$

subject to

$$0 \leq \alpha_i \leq C$$ for $\forall i$, and $\sum_{i} \alpha_i y_i = 0$
Special transformations

\[ \Phi: \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[ \Phi(x)^\top \Phi(z) = (x_1, x_2, \sqrt{2}x_1x_2) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \]

\[ = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \]

\[ = (x_1z_1 + x_2z_2)^2 \]

\[ = (x^\top z)^2 \]

Kernel Trick

- Classifier can be learnt and applied without explicitly computing \( \Phi(x) \)
- All that is required is the kernel \( k(x, z) = (x^\top z)^2 \)
- Complexity of learning depends on \( N \) (typically it is \( O(N^3) \)) not on \( D \)
Example kernels

- **Linear kernels** $k(x, x') = x^\top x'$
- **Polynomial kernels** $k(x, x') = (1 + x^\top x')^d$ for any $d > 0$
  - Contains all polynomials terms up to degree $d$
- **Gaussian kernels** $k(x, x') = \exp \left(-\|x - x'\|^2 / 2\sigma^2 \right)$ for $\sigma > 0$
  - Infinite dimensional feature space
LIBSVM FOR MATLAB
LibSVM

• **LIBSVM** is an integrated software for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, nu-SVR) and distribution estimation (one-class SVM). It supports multi-class classification.

• [Python](https://www.python.org), [R](https://www.r-project.org), [MATLAB](https://www.mathworks.com), [Perl](https://www.perl.org), [Ruby](https://www.ruby-lang.org), [Weka](https://www.cs.waikato.ac.nz/ml/weka), [Common LISP](https://www.lispworks.com), [CLISP](https://www.clisp.org), [Haskell](https://hackage.haskell.org), [OCaml](https://ocaml.org), [LabVIEW](https://www.labview.com), and [PHP](https://www.php.net) interfaces. [C# .NET](https://www.microsoft.com/en-us/net) code and [CUDA](https://www.nvidia.com/en-gb/cuda/) extension is available.
LibSVM installation

• Download from: http://www.csie.ntu.edu.tw/~cjlin/libsvm/
• Un-compress the folder
• Go to MATLAB subfolder
• Compile using make command (Apply for Linux and Mac users; Windows binaries are already built in windows folder)
• Copy binaries in the work directory
The magic commands (svmtrain, svmpredict)

```matlab
model = svmtrain(training_label_vector, training_instance_matrix [, 'libsvm_options']);

- training_label_vector:
  An m by 1 vector of training labels (type must be double).
- training_instance_matrix:
  An m by n matrix of m training instances with n features.
  It can be dense or sparse (type must be double).
- libsvm_options:
  A string of training options in the same format as that of LIBSVM.

[predicted_label, accuracy, decision_values/prob_estimates] = svmpredict(testing_label_vector, testing_instance_matrix, model [, 'libsvm_options']);

- testing_label_vector:
  An m by 1 vector of prediction labels. If labels of test
  data are unknown, simply use any random values. (type must be double)
- testing_instance_matrix:
  An m by n matrix of m testing instances with n features.
  It can be dense or sparse. (type must be double)
- model:
  The output of svmtrain.
- libsvm_options:
  A string of testing options in the same format as that of LIBSVM.
```
LibSVM options (svm-train)


options:
- `svm_type`: set type of SVM (default 0)
  - 0 -- C-SVC (multi-class classification)
  - 1 -- nu-SVC (multi-class classification)
  - 2 -- one-class SVM
  - 3 -- epsilon-SVR (regression)
  - 4 -- nu-SVR (regression)
- `kernel_type`: set type of kernel function (default 2)
  - 0 -- linear: u'*v
  - 1 -- polynomial: (gamma*u'*v + coef0)^degree
  - 2 -- radial basis function: exp(-gamma*|u-v|^2)
  - 3 -- sigmoid: tanh(gamma*u'*v + coef0)
  - 4 -- precomputed kernel (kernel values in training_set_file)
- `degree`: set degree in kernel function (default 3)
- `gamma`: set gamma in kernel function (default 1/num_features)
- `coef0`: set coef0 in kernel function (default 0)
- `cost`: set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
- `nu`: set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
- `epsilon`: set the epsilon in loss function of epsilon-SVR (default 0.1)
- `cache_size`: set cache memory size in MB (default 100)
- `epsilon`: set tolerance of termination criterion (default 0.001)
- `shrinking`: whether to use the shrinking heuristics (default 1)
- `probability_estimates`: whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
- `weight`: set the parameter C of class i to weight*C, for C-SVC (default 1)
- `n`: n-fold cross validation mode
- `quiet`: quiet mode (no outputs)

The k in the -g option means the number of attributes in the input data.

option -v randomly splits the data into n parts and calculates cross validation accuracy/mean squared error on them.
LibSVM options (svm-predict)

'svm-predict' Usage

Usage: svm-predict [options] test_file model_file output_file
options:
-b probability_estimates: whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM only 0 is supported

model_file is the model file generated by svm-train.
test_file is the test data you want to predict.
svm-predict will produce output in the output_file.
Example 1. Linear SVM

% read the data set
[heart_scale_label, heart_scale_inst] = libsvmread(fullfile(dirData,'heart_scale'));
[N D] = size(heart_scale_inst);

% Determine the train and test index
trainIndex = zeros(N,1); trainIndex(1:200) = 1;
testIndex = zeros(N,1); testIndex(201:N) = 1;
trainData = heart_scale_inst(trainIndex==1,:);
trainLabel = heart_scale_label(trainIndex==1,:);
testData = heart_scale_inst(testIndex==1,:);
testLabel = heart_scale_label(testIndex==1,:);

% Train the SVM
model = svmtrain(trainLabel, trainData, '-c 1 -g 0.07 -b 1');
% Use the SVM model to classify the data
[predict_label, accuracy, prob_values] = svmpredict(testLabel, testData, model, '-b 1');
Example 2. Multi-Class SVM

% Train the SVM in one-vs-rest (OVR) mode
model = svmtrain(trainLabel, trainData, '-s 0 -t 2 -c 1.5 -h 1 -b 1');
% Classify samples using OVR model
[predict_label, accuracy, prob_values] = svmpredict(testLabel, testData, model, '-b 1')
fprintf('Accuracy = %g%%\n', accuracy * 100);