SVM

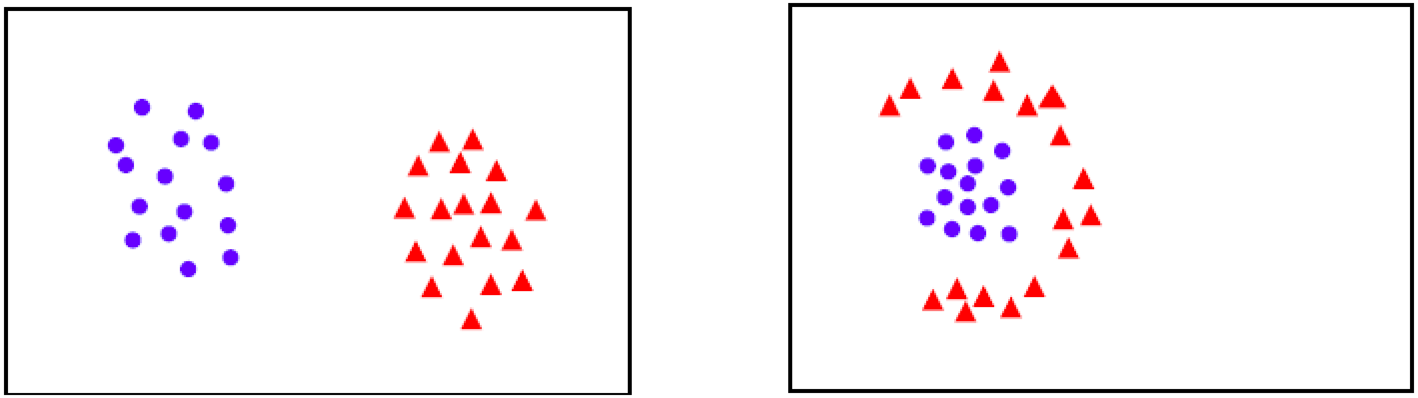
Gonzalo Vaca-Castano

Binary Classification

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

$$f(\mathbf{x}_i) \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases}$$

i.e. $y_i f(\mathbf{x}_i) > 0$ for a correct classification.

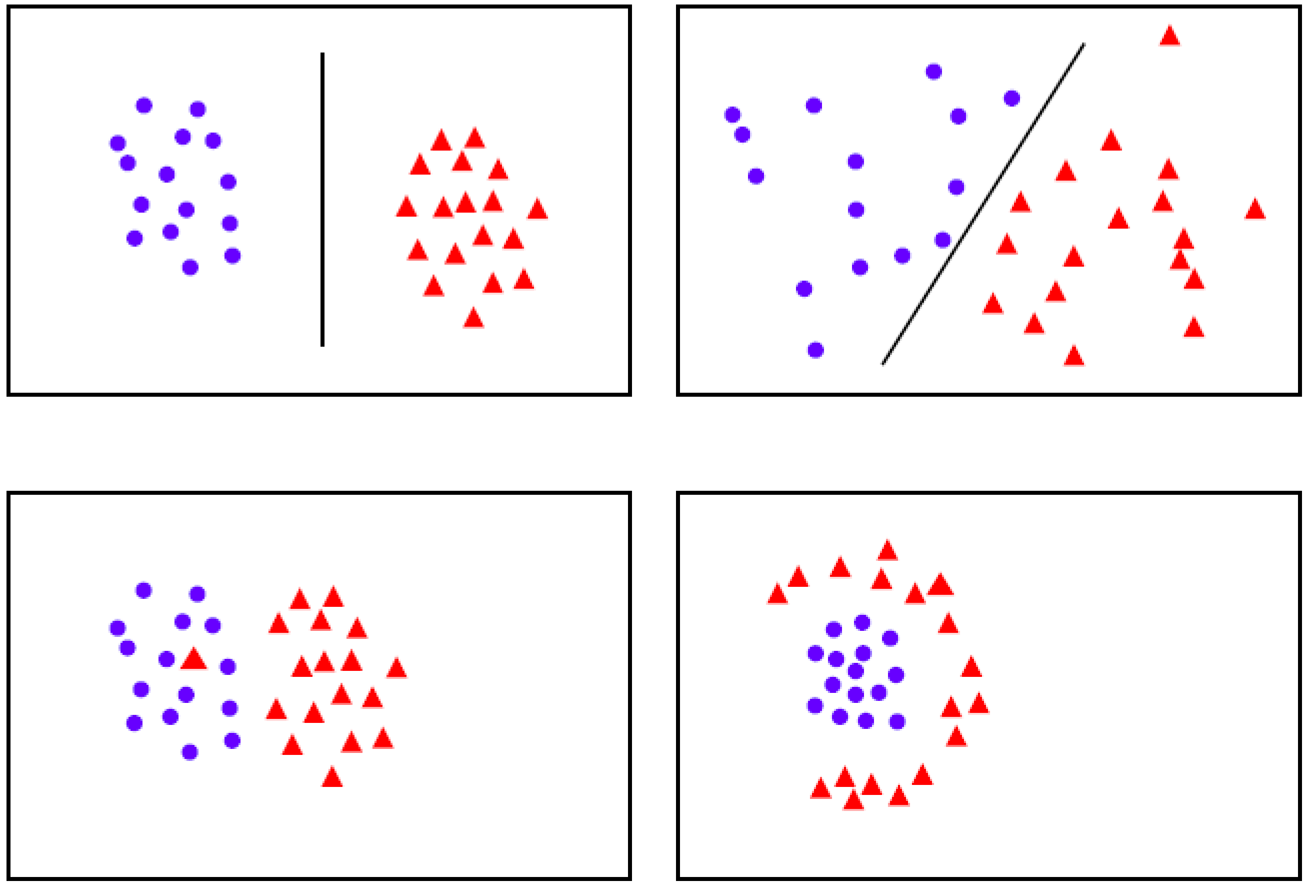


Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

Linear separability

linearly separable

not linearly separable

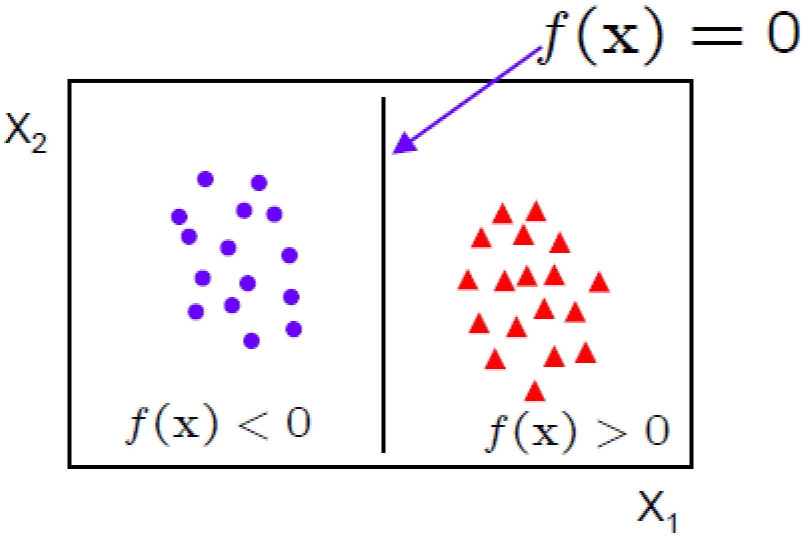


Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

Linear classifiers

A linear classifier has the form

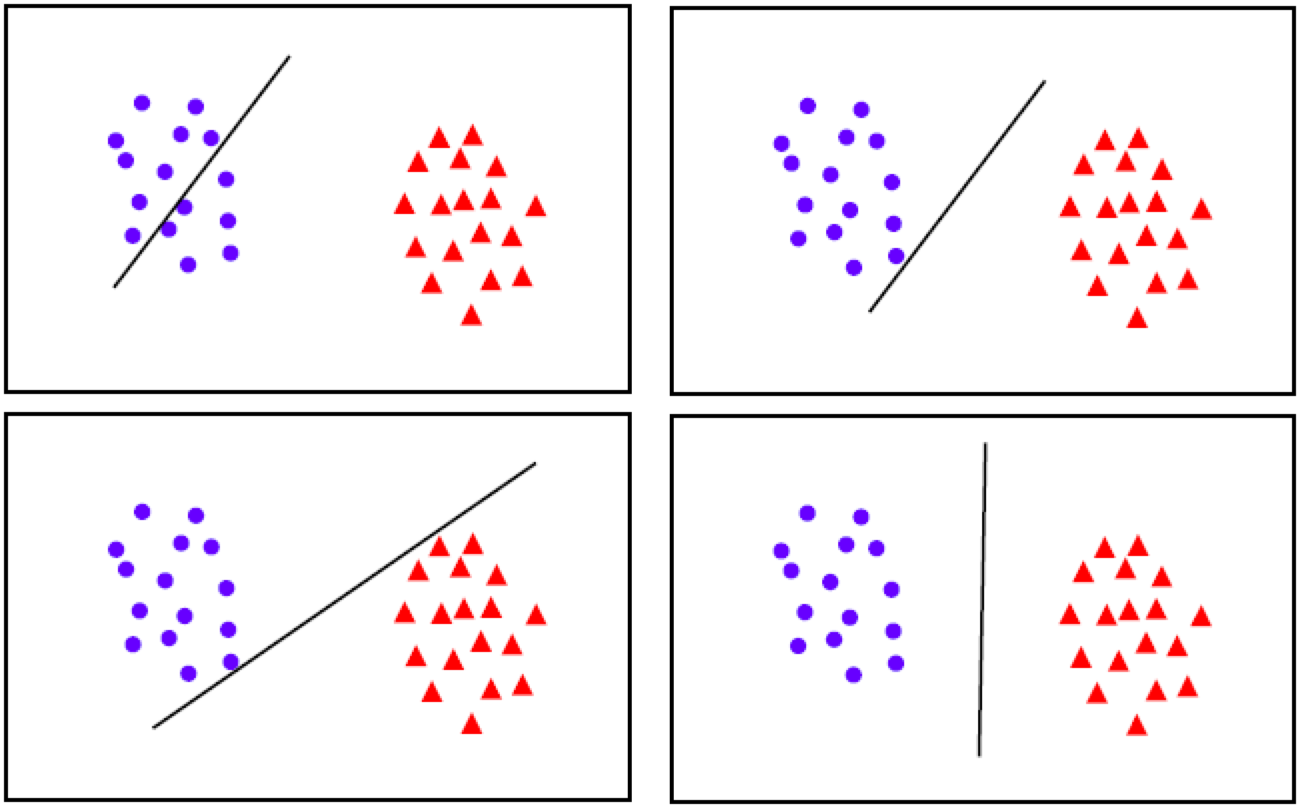
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 2D the discriminant is a line
- $\mathbf{w}$ is the normal to the line, and b the bias
- $\mathbf{w}$ is known as the weight vector

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

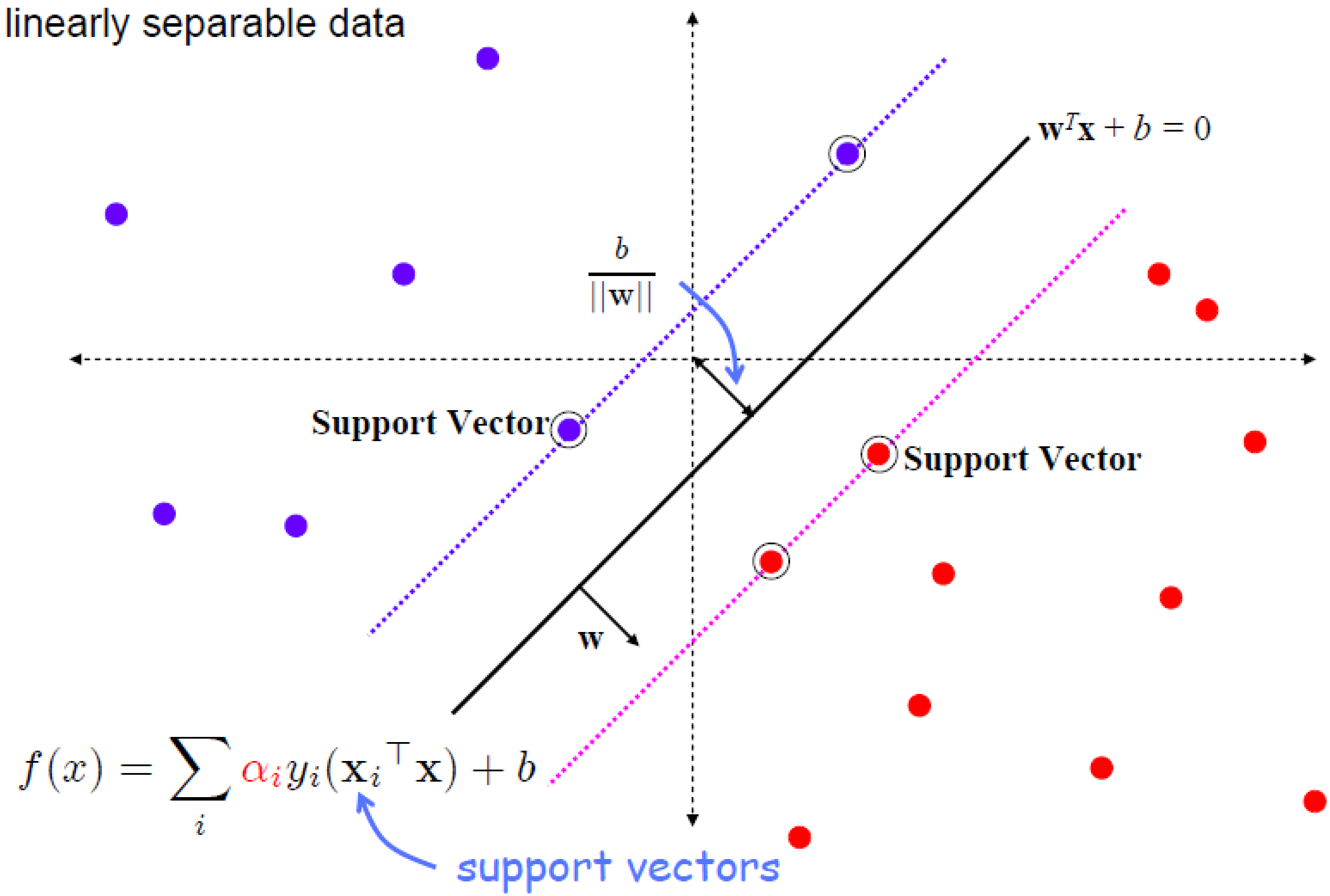
What is the best w?



• maximum margin solution: most stable under perturbations of the inputs

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

Support Vector Machine



$$f(x) = \sum_i \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

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

SVM – sketch derivation

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

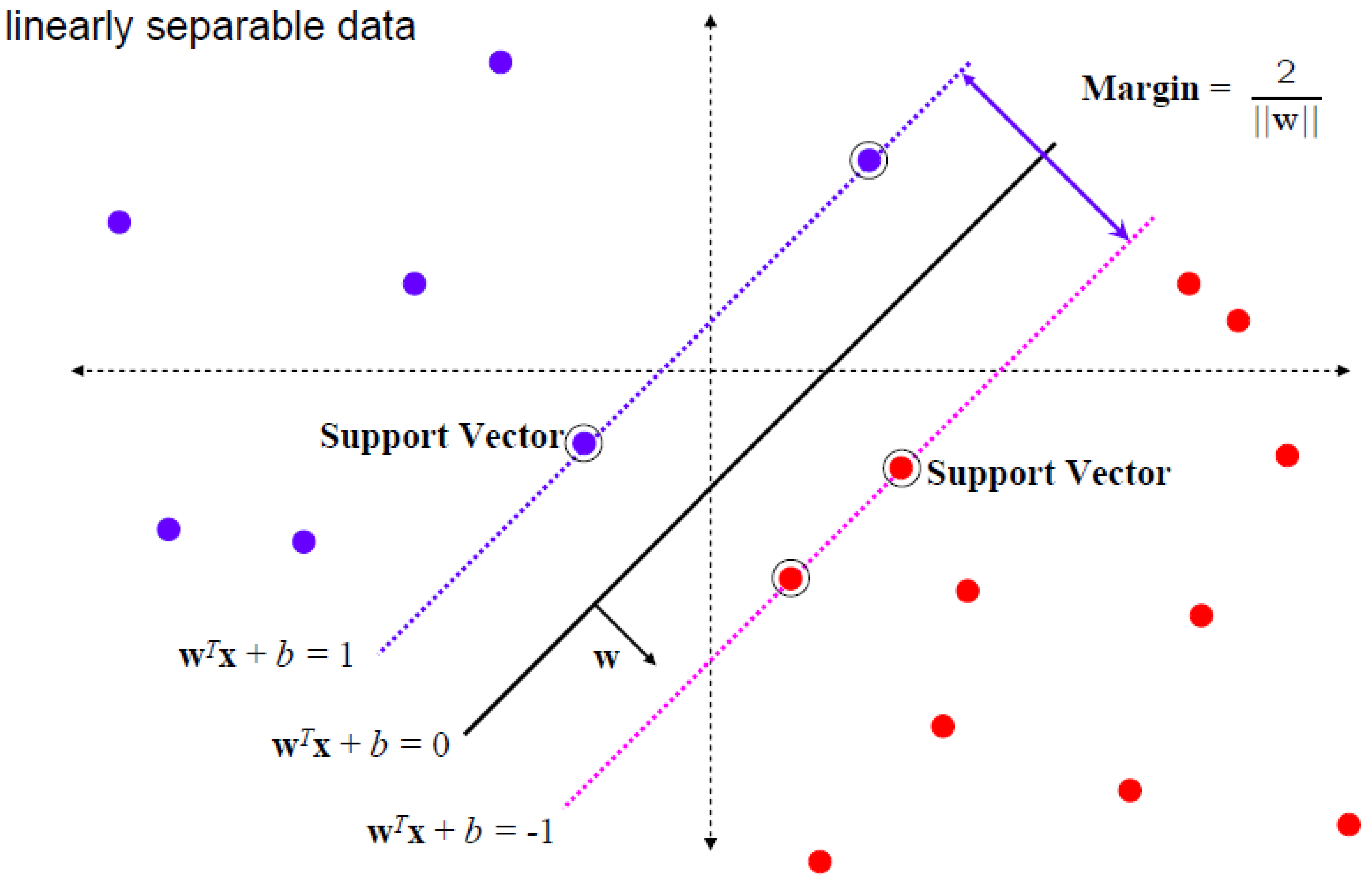
- Choose normalization such that $\mathbf{w}^\top \mathbf{x}_+ + b = +1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively

- Then the margin is given by

$$\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \left(\mathbf{x}_+ - \mathbf{x}_-\right) = \frac{\mathbf{w}^\top \left(\mathbf{x}_+ - \mathbf{x}_-\right)}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

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

Support Vector Machine



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:

$$\max_{\mathbf{w}} \frac{2}{||\mathbf{w}||} \text{ subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{array}{ll} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{array} \quad \text{for } i = 1 \ldots N$$

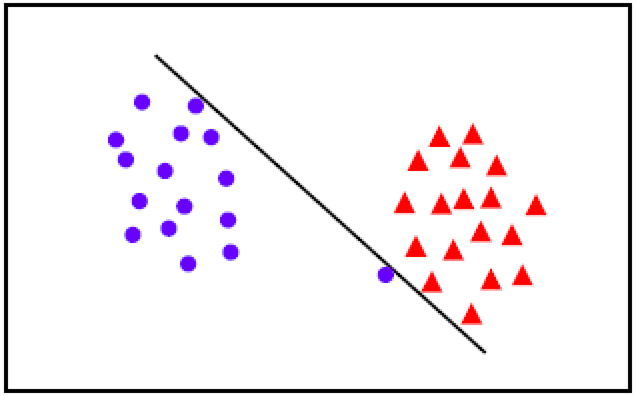
- Or equivalently

$$\min_{\mathbf{w}} ||\mathbf{w}||^2 \text{ subject to } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 \text{ for } i = 1 \ldots N$$

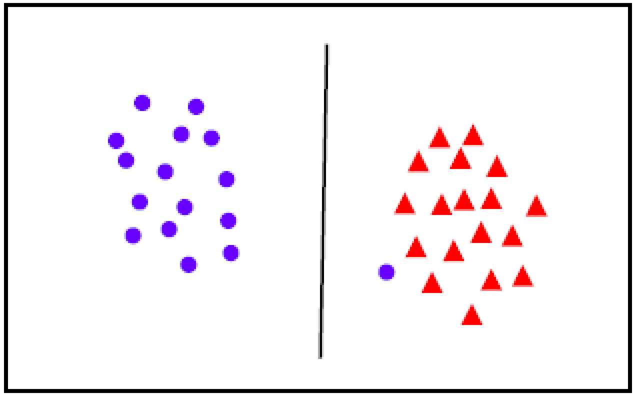
- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

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

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

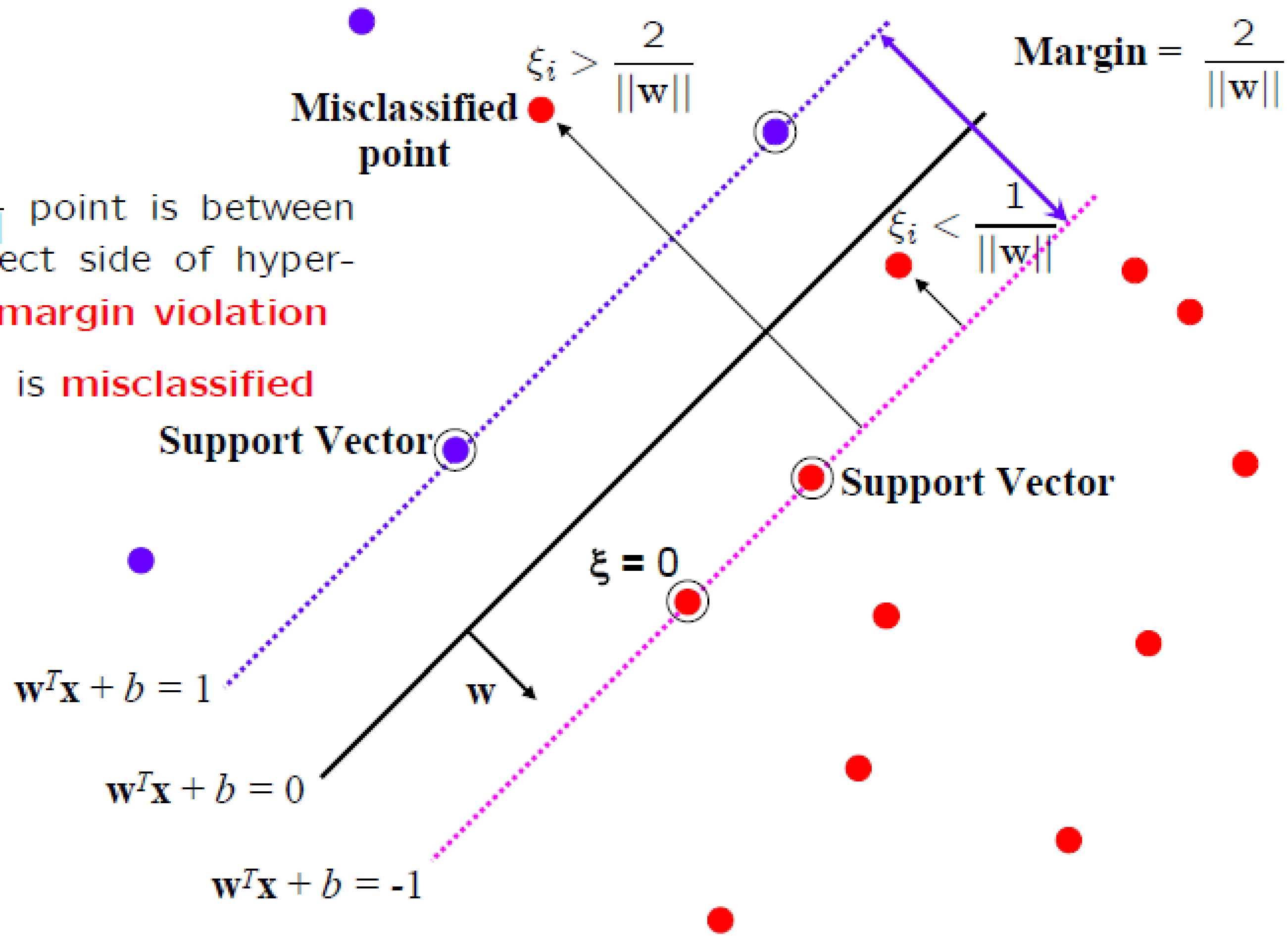
In general there is a trade off between the margin and the number of mistakes on the training data

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

Introduce "slack" variables

$\xi_i \geq 0$

- for $0 < \xi \leq \frac{1}{||\mathbf{w}||}$ point is between margin and correct side of hyper-plane. This is a margin violation
- for $\xi > \frac{1}{||\mathbf{w}||}$ point is misclassified



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

"Soft" margin solution

The optimization problem becomes

$$\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i\in\mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i$$

subject to

$$y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

- Every constraint can be satisfied if ξ_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.

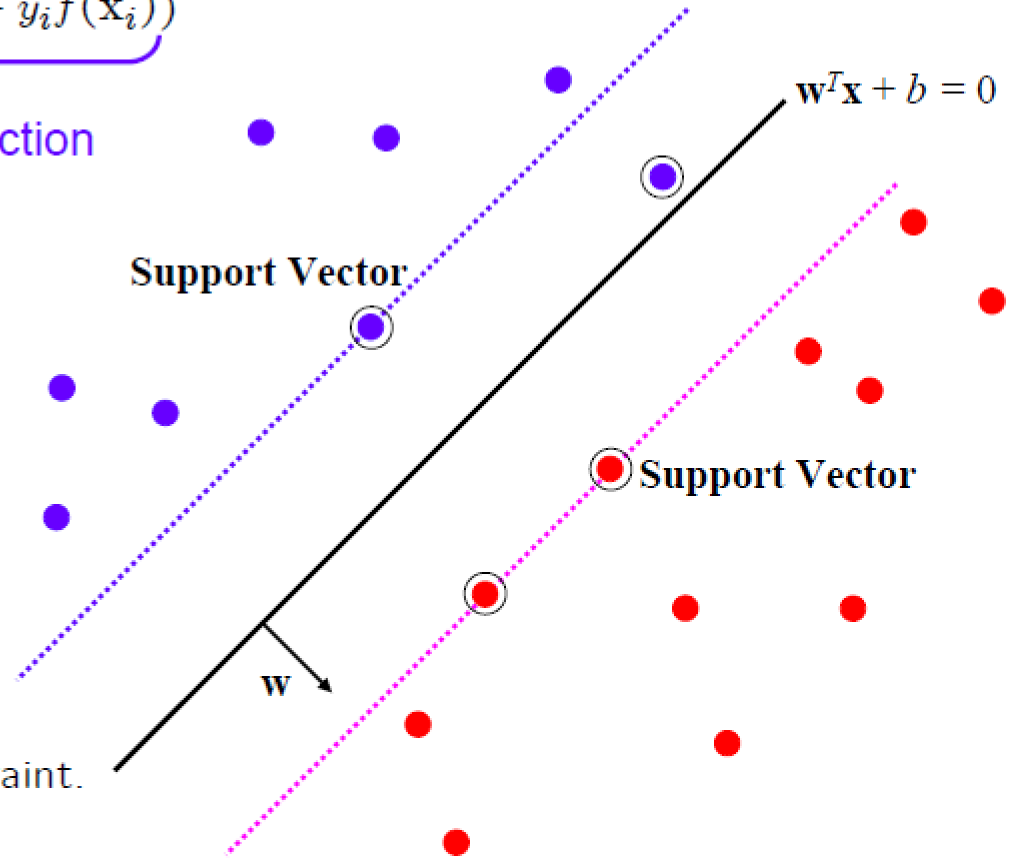
Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

Loss function

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_i^N \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

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



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

SVM – review

- We have seen that for an SVM learning a linear classifier

$$f(x) = \mathbf{w}^\top \mathbf{x} + b$$

is formulated as solving an optimization problem over $\mathbf{w}$:

$$\min_{\mathbf{w} \in \mathbb{R}^d} ||\mathbf{w}||^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- This quadratic optimization problem is known as the primal problem.

- Instead, the SVM can be formulated to learn a linear classifier

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

by solving an optimization problem over α_i.

- This is know as the dual problem, and we will look at the advantages of this formulation.

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

Primal and dual formulations

View or add

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

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

$$\min_{\mathbf{w} \in \mathbb{R}^d} ||\mathbf{w}||^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

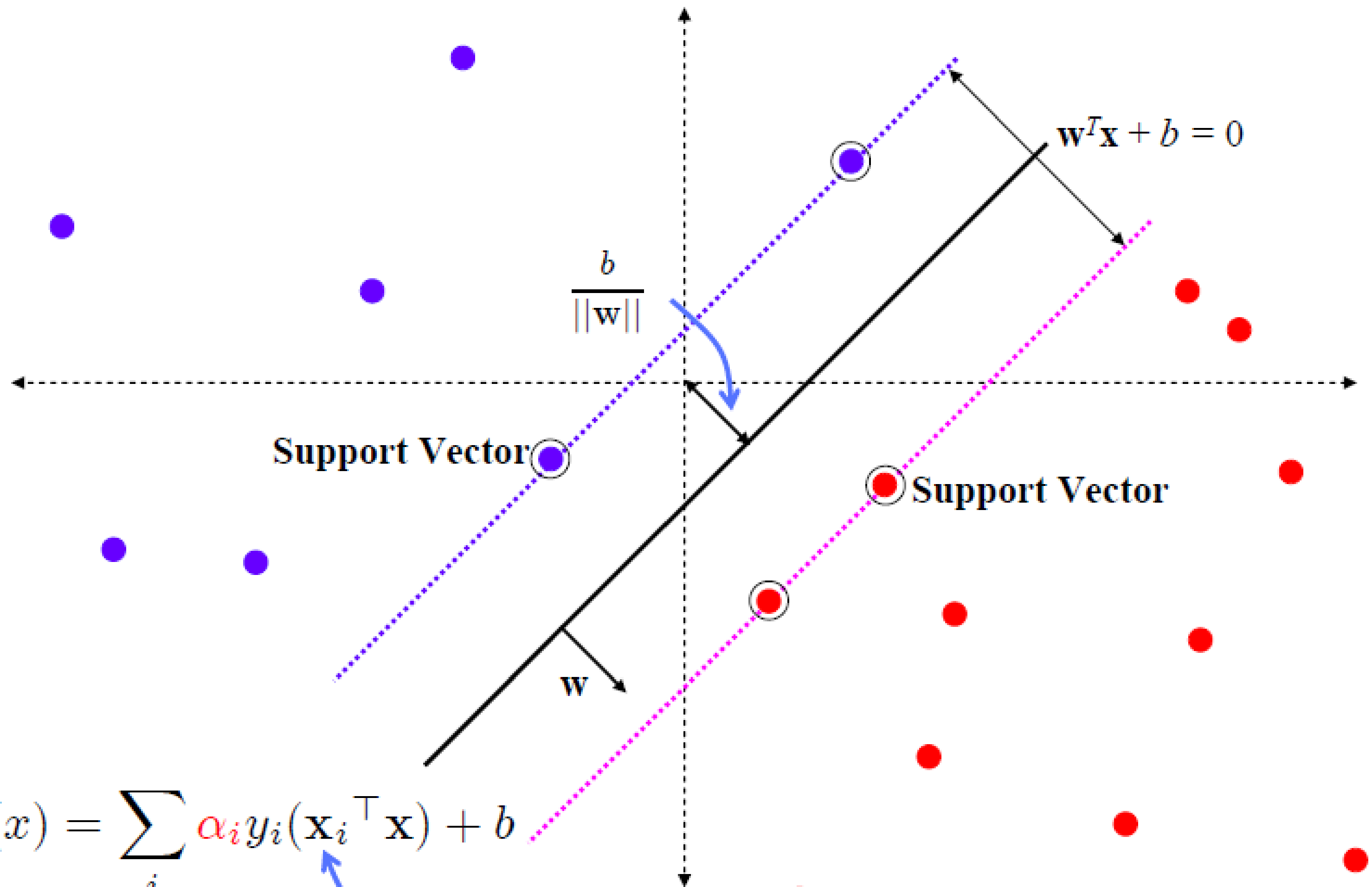
Dual problem: for $\boldsymbol{\alpha} \in \mathbb{R}^N$ (stated without proof):

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

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

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

Support Vector Machine



$$f(x) = \sum_i \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$$

support vectors

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

Dual Classifier in transformed feature space

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$$

$$\rightarrow f(\mathbf{x}) = \sum_i^N \alpha_i y_i \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$

$$\rightarrow \max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector $\boldsymbol{\alpha}$ needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a Kernel

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \, k(\mathbf{x}_j, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

Special transformations

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{aligned} \Phi(\mathbf{x})^\top \Phi(\mathbf{z}) &= \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \\ &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2 \end{aligned}$$

Kernel Trick

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

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

Example kernels

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

Slide from: Lecture 3:Dual problems and Kernels
C19 Machine Learning Hilary 2013 A. Zisserman

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, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, OCaml, LabVIEW, and PHP interfaces. C# .NET code and CUDA extension is available.

LibSVM installation

- Downlad 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)

```
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)

```
Usage: svm-train [options] training_set_file [model_file]
options:
-s 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)
-t 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)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-q : 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);
```