Kernel Methods

COMP9417 Machine Learning and Data Mining

Term 2, 2023

Acknowledgements

Material derived from slides for the book
"Elements of Statistical Learning (2nd Ed.)" by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009)
http://statweb.stanford.edu/~tibs/ElemStatLearn/

Material derived from slides for the book
"Machine Learning: A Probabilistic Perspective" by P. Murphy
MIT Press (2012)
http://www.cs.ubc.ca/~murphyk/MLbook

Material derived from slides for the book
"Machine Learning" by P. Flach
Cambridge University Press (2012)
http://cs.bris.ac.uk/~flach/mlbook

Material derived from slides for the book
"Bayesian Reasoning and Machine Learning" by D. Barber
Cambridge University Press (2012)
http://www.cs.ucl.ac.uk/staff/d.barber/brml

Material derived from slides for the book
"Machine Learning" by T. Mitchell
McGraw-Hill (1997)
http://www-2.cs.cmu.edu/~tom/mlbook.html

Material derived from slides for the course
"Machine Learning" by A. Srinivasan
BITS Pilani Goa Campus, India (2016)

Contents

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximising the margin in linear classification

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximising the margin in linear classification
- outline the typical loss function for maximising the margin

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximising the margin in linear classification
- outline the typical loss function for maximising the margin
- describe the method of support vector machines (SVMs)

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximising the margin in linear classification
- outline the typical loss function for maximising the margin
- describe the method of support vector machines (SVMs)
- describe the concept of kernel functions

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximising the margin in linear classification
- outline the typical loss function for maximising the margin
- describe the method of support vector machines (SVMs)
- describe the concept of kernel functions
- outline the idea of using a kernel in a learning algorithm

Aims

This lecture will develop your understanding of kernel methods in machine learning. Following it you should be able to:

- describe learning with the dual perceptron
- outline the idea of learning in a dual space
- describe the concept of maximising the margin in linear classification
- outline the typical loss function for maximising the margin
- describe the method of support vector machines (SVMs)
- describe the concept of kernel functions
- outline the idea of using a kernel in a learning algorithm
- outline non-linear classification with kernel methods

Contents

Predictive machine learning scenarios

Task	*Label space*	*Output space*	*Learning problem*
Regression	$\mathbb{R}$	$\mathcal{Y} = \mathbb{R}$	learn an approximation $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}$ to the true labelling function f
Classification	$\mathcal{C}$	$\mathcal{Y} = \mathcal{C}$	learn an approximation $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$ to the true labelling function c
Scoring and ranking	$\mathcal{C}$	$\mathcal{Y} = \mathbb{R}^{\lvert\mathcal{C}\rvert}$	learn a model that outputs a score vector over classes
Probability estimation	$\mathcal{C}$	$\mathcal{Y} = [0, 1]^{\lvert\mathcal{C}\rvert}$	learn a model that outputs a probability vector over classes

Classification

Classification

A **classifier** is a mapping $\hat{c} : \mathcal{X} \to \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of **class labels**. We will sometimes also use C_i to indicate the set of examples of that class.

Classification

A **classifier** is a mapping $\hat{c} : \mathcal{X} \to \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of **class labels**. We will sometimes also use C_i to indicate the set of examples of that class.

We use the 'hat' to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathcal{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

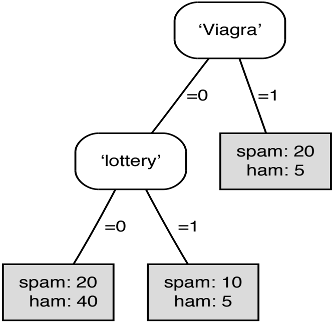
Classification

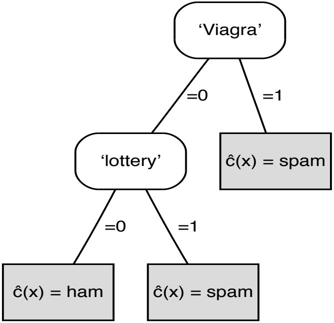
A **classifier** is a mapping $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$, where $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ is a finite and usually small set of **class labels**. We will sometimes also use C_i to indicate the set of examples of that class.

We use the 'hat' to indicate that $\hat{c}(x)$ is an estimate of the true but unknown function $c(x)$. Examples for a classifier take the form $(x, c(x))$, where $x \in \mathcal{X}$ is an instance and $c(x)$ is the true class of the instance (sometimes contaminated by noise).

Learning a classifier involves constructing the function $\hat{c}$ such that it matches c as closely as possible (and not just on the training set, but ideally on the entire instance space $\mathcal{X}$).

A decision tree





(left) A tree with the training set class distribution in the leaves.

(right) A tree with the majority class prediction rule in the leaves.

Scoring classifier

Scoring classifier

A **scoring classifier** is a mapping $\hat{\mathrm{s}} : \mathcal{X} \to \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers.

Scoring classifier

A **scoring classifier** is a mapping $\hat{\mathbf{s}} : \mathcal{X} \rightarrow \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers.

The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x.

Scoring classifier

A **scoring classifier** is a mapping $\hat{\mathbf{s}} : \mathcal{X} \to \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers.

The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x.

This score indicates how likely it is that class label C_i applies.

Scoring classifier

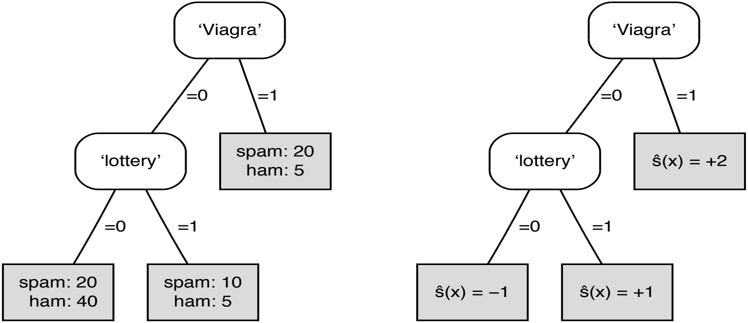
A **scoring classifier** is a mapping $\hat{\mathbf{s}} : \mathcal{X} \rightarrow \mathbb{R}^k$, i.e., a mapping from the instance space to a k-vector of real numbers.

The boldface notation indicates that a scoring classifier outputs a vector $\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ rather than a single number; $\hat{s}_i(x)$ is the score assigned to class C_i for instance x.

This score indicates how likely it is that class label C_i applies.

If we only have two classes, it usually suffices to consider the score for only one of the classes; in that case, we use $\hat{s}(x)$ to denote the score of the positive class for instance x.

A scoring tree



(left) A tree with the training set class distribution in the leaves.

(right) A tree using the logarithm of the class ratio as scores; spam is taken as the positive class.

Margins and loss functions

Margins and loss functions

If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity $z(x) = c(x)\hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions: this quantity is called the **margin** assigned by the scoring classifier to the example.

Margins and loss functions

If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity $z(x) = c(x)\hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions: this quantity is called the **margin** assigned by the scoring classifier to the example.

We would like to reward large positive margins, and penalise large negative values. This is achieved by means of a so-called **loss function** $L : \mathbb{R} \mapsto [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

Margins and loss functions

If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity $z(x) = c(x)\hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions: this quantity is called the **margin** assigned by the scoring classifier to the example.

We would like to reward large positive margins, and penalise large negative values. This is achieved by means of a so-called **loss function** $L : \mathbb{R} \mapsto [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

We will assume that $L(0) = 1$, which is the loss incurred by having an example on the decision boundary. We furthermore have $L(z) \geq 1$ for $z < 0$, and usually also $0 \leq L(z) < 1$ for $z > 0$.

Margins and loss functions

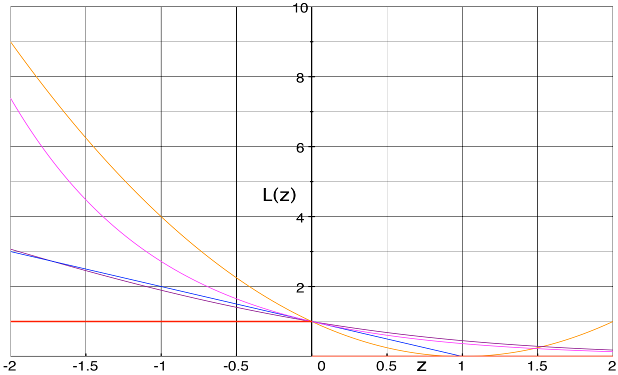
If we take the true class $c(x)$ as $+1$ for positive examples and -1 for negative examples, then the quantity $z(x) = c(x)\hat{s}(x)$ is positive for correct predictions and negative for incorrect predictions: this quantity is called the **margin** assigned by the scoring classifier to the example.

We would like to reward large positive margins, and penalise large negative values. This is achieved by means of a so-called **loss function** $L : \mathbb{R} \mapsto [0, \infty)$ which maps each example's margin $z(x)$ to an associated loss $L(z(x))$.

We will assume that $L(0) = 1$, which is the loss incurred by having an example on the decision boundary. We furthermore have $L(z) \geq 1$ for $z < 0$, and usually also $0 \leq L(z) < 1$ for $z > 0$.

The average loss over a test set Te is $\frac{1}{|Te|} \sum_{x \in Te} L(z(x))$.

Loss functions



From bottom-left: (i) 0–1 loss $L_{01}(z) = 1$ if $z \leq 0$, and $L_{01}(z) = 0$ if $z > 0$; (ii) hinge loss $L_{\mathrm{h}}(z) = (1 - z)$ if $z \leq 1$, and $L_{\mathrm{h}}(z) = 0$ if $z > 1$; (iii) logistic loss $L_{\log}(z) = \log_2(1 + \exp(-z))$; (iv) exponential loss $L_{\exp}(z) = \exp(-z)$; (v) squared loss $L_{\mathrm{sq}}(z) = (1 - z)^2$ (can be set to 0 for $z > 1$, just like hinge loss).

0-1 loss

$$L_{01}(z) = \begin{cases} 1 & \text{if } z \leq 0 \\ 0 & \text{if } z > 0 \end{cases}$$

$$\boxed{\text{Margin: } z(x) = c(x)\,\hat{s}(x)}$$

e.g., $+1 * 1.2$ (correct)
e.g., $+1 * -0.7$ (incorrect)

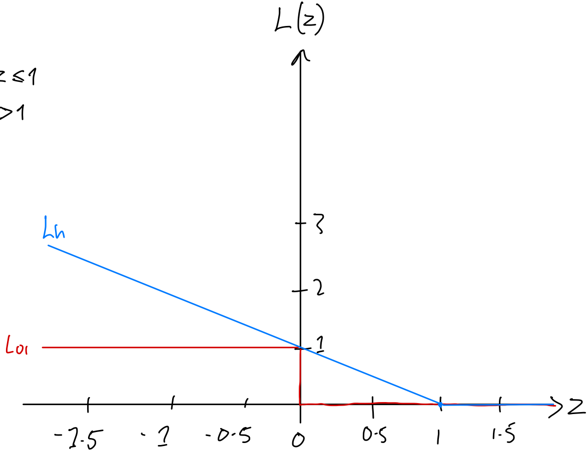
Ignores magnitude of the margin

Hinge loss

$$L_h(z) = \begin{cases} (1-z) & \text{if } z \leq 1 \\ 0 & \text{if } z > 1 \end{cases}$$

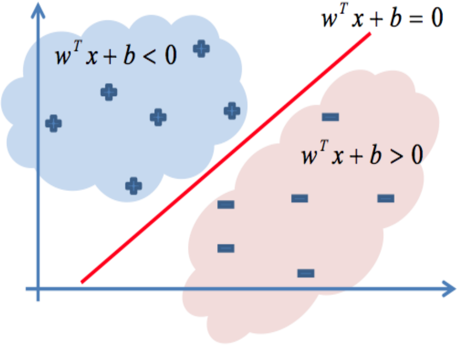
e.g., $+1 * 2.4$ (correct)
$+1 * -1.7$ (incorrect)

$L_h = 1 - (-1.7) = 2.7$
$1 - (-0.7) = 1.7$



Review: Linear classification

- Example: a two-class classifier "separates" instances in feature space: $f(x) = sign(w^T x + b)$



Issues in linear classification

Issues in linear classification

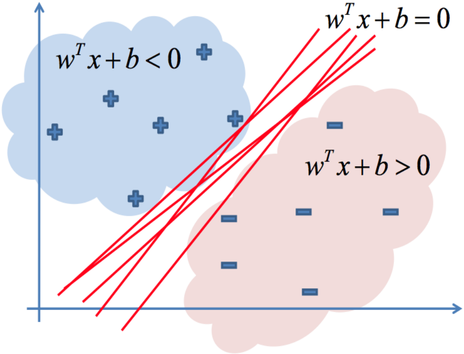
- A linear model defines a hyperplane in feature space

Issues in linear classification

- A linear model defines a hyperplane in feature space
- This decision boundary can be used for classification

Issues in linear classification

- Many possible linear decision boundaries: which one to choose ?



Issues in linear classification

Is there an optimal linear classification learning method ?

Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data

Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data
- learn the *maximum margin* separating hyperplane

Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data
- learn the *maximum margin* separating hyperplane
- unique solution

Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data
- learn the *maximum margin* separating hyperplane
- unique solution
- minimises empirical risk

Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data
- learn the *maximum margin* separating hyperplane
- unique solution
- minimises empirical risk
- is there a way to trade-off minimising risk with model complexity?

Issues in linear classification

Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data
- learn the *maximum margin* separating hyperplane
- unique solution
- minimises empirical risk
- is there a way to trade-off minimising risk with model complexity?
- answer: yes, under Vapnik's framework for statistical learning

Issues in linear classification

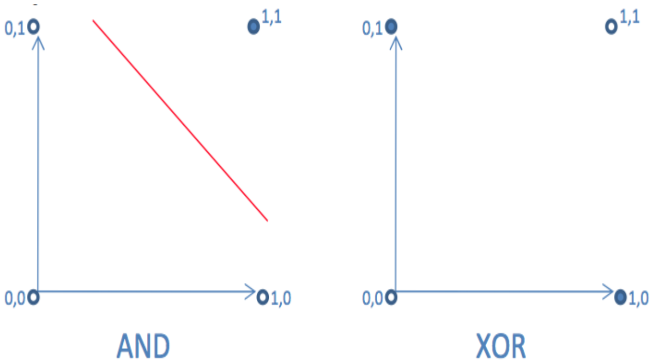
Is there an optimal linear classification learning method ?

- one approach is to define the *empirical risk*, or error on the data
- learn the *maximum margin* separating hyperplane
- unique solution
- minimises empirical risk
- is there a way to trade-off minimising risk with model complexity?
- answer: yes, under Vapnik's framework for statistical learning
    - *structural risk minimization*

Issues in linear classification

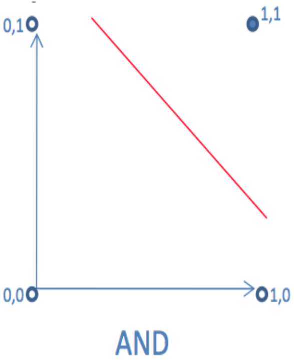
Issues in linear classification

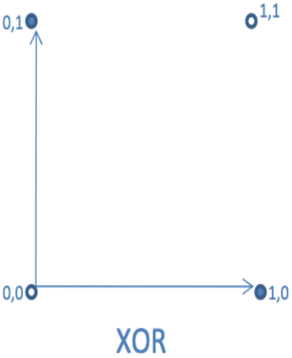
- Recall: may not be possible to learn a *linear* separating hyperplane



Issues in linear classification

- Recall: may not be possible to learn a *linear* separating hyperplane

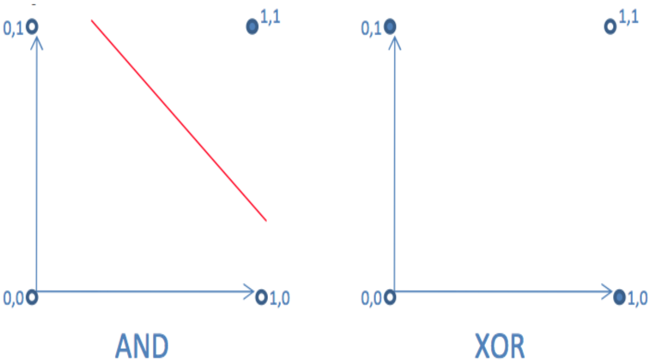




- filled / empty circles are in / out of the target concept

Issues in linear classification

- Recall: may not be possible to learn a *linear* separating hyperplane



- filled / empty circles are in / out of the target concept
- AND is linearly separable – but not XOR

Extending linear classification

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- But there is a simple trick to allow them to do that

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- But there is a simple trick to allow them to do that
  - Nonlinear mapping: map features into new higher-dimensional space consisting of combinations of attribute values

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- But there is a simple trick to allow them to do that
  - Nonlinear mapping: map features into new higher-dimensional space consisting of combinations of attribute values
  - For example: all products with n factors that can be constructed from the features (*feature construction*)

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- But there is a simple trick to allow them to do that
  - Nonlinear mapping: map features into new higher-dimensional space consisting of combinations of attribute values
  - For example: all products with n factors that can be constructed from the features (*feature construction*)
- e.g., for 2 features, all products with $n = 3$ factors

$$y = w_1 x_1^3 + w_2 x_1^2 x_2 + w_3 x_1 x_2^2 + w_4 x_2^3$$

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- But there is a simple trick to allow them to do that
  - Nonlinear mapping: map features into new higher-dimensional space consisting of combinations of attribute values
  - For example: all products with n factors that can be constructed from the features (*feature construction*)
- e.g., for 2 features, all products with $n = 3$ factors

$$y = w_1 x_1^3 + w_2 x_1^2 x_2 + w_3 x_1 x_2^2 + w_4 x_2^3$$

- y is predicted output for instances with two features x_1 and x_2

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- But there is a simple trick to allow them to do that
  - Nonlinear mapping: map features into new higher-dimensional space consisting of combinations of attribute values
  - For example: all products with n factors that can be constructed from the features (*feature construction*)
- e.g., for 2 features, all products with $n = 3$ factors

$$y = w_1 x_1^3 + w_2 x_1^2 x_2 + w_3 x_1 x_2^2 + w_4 x_2^3$$

- y is predicted output for instances with two features x_1 and x_2
- A function in the new higher-dimensional feature space

Two main problems

Two main problems

- Efficiency:

Two main problems

- Efficiency:
  - With 10 features and $n = 5$ have to learn more than 2000 coefficients (weights)

Two main problems

- Efficiency:
  - With 10 features and $n = 5$ have to learn more than 2000 coefficients (weights)
  - Fitting, e.g., a linear regression model, runtime can be cubic in the number of features

Two main problems

- Efficiency:
  - With 10 features and $n = 5$ have to learn more than 2000 coefficients (weights)
  - Fitting, e.g., a linear regression model, runtime can be cubic in the number of features
- Overfitting:

Two main problems

- Efficiency:
  - With 10 features and $n = 5$ have to learn more than 2000 coefficients (weights)
  - Fitting, e.g., a linear regression model, runtime can be cubic in the number of features
- Overfitting:
  - "Too nonlinear" – number of coefficients large relative to number of training instances

Two main problems

- Efficiency:
  - With 10 features and $n = 5$ have to learn more than 2000 coefficients (weights)
  - Fitting, e.g., a linear regression model, runtime can be cubic in the number of features
- Overfitting:
  - "Too nonlinear" – number of coefficients large relative to number of training instances
  - *Curse of dimensionality* applies . . .

Contents

Linear classifiers in dual form

An observation about the Perceptron training rule:

Linear classifiers in dual form

An observation about the Perceptron training rule:

Every time an example $\mathbf{x}_i$ is misclassified, add $y_i\mathbf{x}_i$ to the weight vector.

Linear classifiers in dual form

An observation about the Perceptron training rule:

Every time an example $\mathbf{x}_i$ is misclassified, add $y_i\mathbf{x}_i$ to the weight vector.

This leads to a different perspective on training linear classifiers.

Linear classifiers in dual form

Linear classifiers in dual form

- After training has completed, each example has been misclassified zero or more times. Denoting this number as α_i for example $\mathbf{x}_i$, the weight vector can be expressed as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Linear classifiers in dual form

- After training has completed, each example has been misclassified zero or more times. Denoting this number as α_i for example $\mathbf{x}_i$, the weight vector can be expressed as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

- In the dual, instance-based view of linear classification we are learning instance weights α_i rather than feature weights w_j. An instance $\mathbf{x}$ is classified as

$$\hat{y} = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} \right)$$

Linear classifiers in dual form

- After training has completed, each example has been misclassified zero or more times. Denoting this number as α_i for example $\mathbf{x}_i$, the weight vector can be expressed as

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

- In the dual, instance-based view of linear classification we are learning instance weights α_i rather than feature weights w_j. An instance $\mathbf{x}$ is classified as

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}\right)$$

- During training, the only information needed about the training data is all pairwise dot products: the n-by-n matrix $\mathbf{G} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$ containing these dot products is called the **Gram matrix**.

Perceptron training in dual form

Algorithm DualPerceptron(D) // perceptron training in dual form

Input: labelled training data D in homogeneous coordinates

Output: coefficients α_i defining weight vector $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$

$\alpha_i \leftarrow 0$ for $1 \leq i \leq |D|$
converged←false
while *converged* = false **do**
    converged←true
    for $i = 1$ to $|D|$ **do**
        if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j \leq 0$ **then**
            $\alpha_i \leftarrow \alpha_i + 1$
            converged←false
        end
    end
end

Perceptron Training from Initialisation

Note: both Perceptron & Dual Perceptron start with initial parameters set to zero!

Perceptron (primal) [lecture "Classification"]

first example: $\langle x_1, y_1 \rangle$

check prediction (line 6):

$$y_1 \, w \cdot x_1 = 0$$

MISTAKE!

update (line 7):

now weights $w \neq 0$

and training continues ...

Perceptron (dual) [last slide]

first example: $\langle x_1, y_1 \rangle$

check prediction (line 6):

$$y_1 \sum_{j=1}^{n} \alpha_j \, y_j \, x_j \cdot x_1 = 0$$

MISTAKE!

update (line 7):

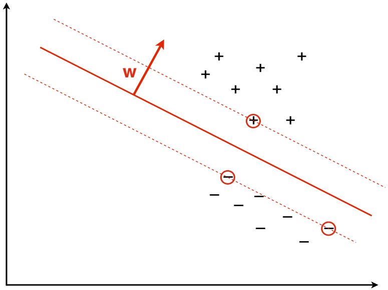
now $\alpha_1 \neq 0$

and training continues ...

So we see that for both versions of the training procedure, although the parameters (either w or $\alpha_1, \alpha_2, \ldots, \alpha_n$) are initialised to zero and will misclassify the first example, they are then updated to be non-zero.

Contents

Support vector machine

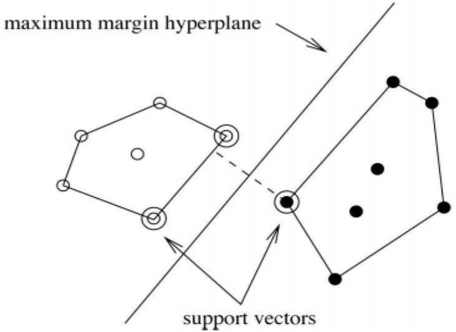


The decision boundary learned by a support vector machine maximises the margin, which is indicated by the dotted lines. The circled data points are the support vectors.

Support vector machines

- Support vector machines (*machine* $\equiv$ *algorithm*) learn linear classifiers
- Can avoid overfitting – learn a form of decision boundary called the *maximum margin hyperplane*
- Fast for mappings to nonlinear spaces
  - employ a mathematical trick to avoid the actual creation of new "pseudo-attributes" in transformed instance space
  - i.e., the nonlinear space is created *implicitly*

Training a support vector machine



Training a support vector machine

- learning problem: fit maximum margin hyperplane, i.e., a kind of linear model
- for a linearly separable two-class dataset the maximum margin hyperplane is the classification surface which
  - correctly classifies all examples in the data set
  - has the greatest *separation* between classes
- "convex hull" of instances in each class is tightest enclosing convex polygon
- for a linearly separable two-class data set convex hulls do not overlap
- maximum margin hyperplane is orthogonal to shortest line connecting convex hulls, intersects with it halfway
- the more "separated" the classes, the larger the margin, the better the generalization

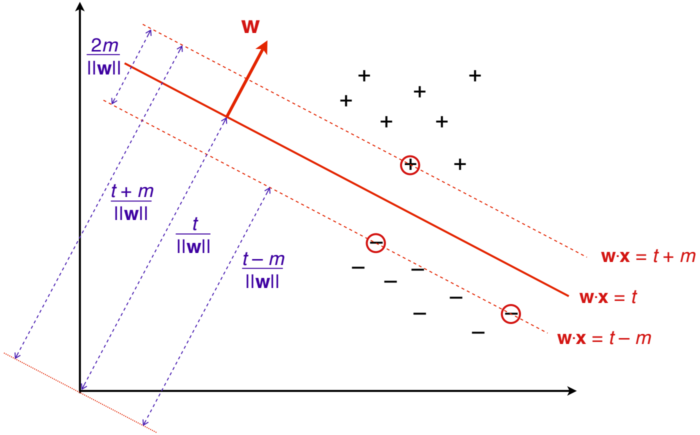
Support vectors

- Instances closest to maximum margin hyperplane are *support vectors*
- Important observation: support vectors define maximum margin hyperplane!
    - All other instances can be deleted without changing position and orientation of the hyperplane!

Finding support vectors

- Determining parameters is a constrained quadratic optimization problem
  - standard algorithms, or
  - special-purpose algorithms are faster, e.g. Platt's sequential minimal optimization (SMO), or LibSVM
- Note: all this assumes separable data!

Support vector machine



The geometry of a support vector classifier. The circled data points are the support vectors, which are the training examples nearest to the decision boundary. The support vector machine finds the decision boundary that maximises the margin $m/||\mathbf{w}||$.

Maximising the margin

Since we are free to rescale t, $||\mathbf{w}||$ and m, it is customary to choose $m = 1$. Maximising the margin then corresponds to minimising $||\mathbf{w}||$ or, more conveniently, $\frac{1}{2}||\mathbf{w}||^2$, provided of course that none of the training points fall inside the margin.

This leads to a quadratic, constrained optimisation problem:

$$\mathbf{w}^*, t^* = \underset{\mathbf{w},t}{\arg\min} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

Using the method of Lagrange multipliers, the dual form of this problem can be derived.

Deriving the dual problem

Adding the constraints with multipliers α_i for each training example gives the Lagrange function

$$
\begin{aligned}
\Lambda(\mathbf{w}, t, \alpha_1, \ldots, \alpha_n) &= \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i - t) - 1) \\
&= \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i) + \sum_{i=1}^{n} \alpha_i y_i t + \sum_{i=1}^{n} \alpha_i \\
&= \frac{1}{2}\mathbf{w} \cdot \mathbf{w} - \mathbf{w} \cdot \left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right) + t \left(\sum_{i=1}^{n} \alpha_i y_i \right) + \sum_{i=1}^{n} \alpha_i
\end{aligned}
$$

- By taking the partial derivative of the Lagrange function with respect to t and setting it to 0 we find $\sum_{i=1}^{n} \alpha_i y_i = 0$.
- Similarly, by taking the partial derivative of the Lagrange function with respect to $\mathbf{w}$ and setting to 0 we obtain $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ – the same expression as we derived for the perceptron.

Deriving the dual problem

- For the perceptron, the instance weights α_i are non-negative integers denoting the number of times an example has been misclassified in training. For a support vector machine, the α_i are non-negative reals.
- What they have in common is that, if $\alpha_i = 0$ for a particular example $\mathbf{x}_i$, that example could be removed from the training set without affecting the learned decision boundary. In the case of support vector machines this means that $\alpha_i > 0$ only for the support vectors: the training examples nearest to the decision boundary.

These expressions allow us to eliminate $\mathbf{w}$ and t and lead to the dual Lagrangian

$$
\begin{aligned}
\Lambda(\alpha_1, \ldots, \alpha_n) &= -\frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right) \cdot \left(\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i\right) + \sum_{i=1}^{n} \alpha_i \\
&= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i
\end{aligned}
$$

SVM in dual form

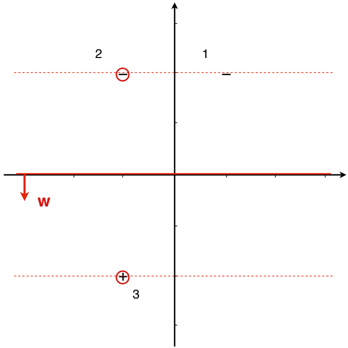
The dual optimisation problem for support vector machines is to maximise the dual Lagrangian under positivity constraints and one equality constraint:

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1,\ldots,\alpha_n}{\arg\max} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

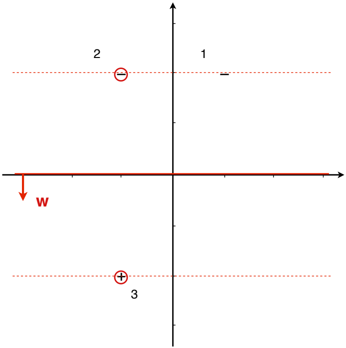
$$\text{subject to} \;\; \alpha_i \geq 0, \;\; 1 \leq i \leq n \;\; \text{and} \;\; \sum_{i=1}^{n}\alpha_i y_i = 0$$

A maximum-margin classifier

A maximum-margin classifier

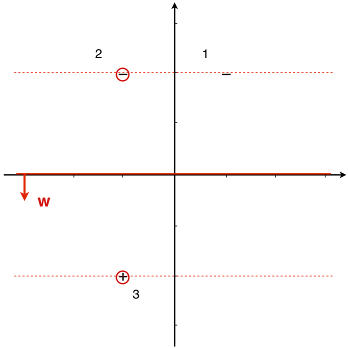


A maximum-margin classifier



A maximum-margin classifier built from three examples, with $\mathbf{w} = (0, -1/2)$ and margin 2.

A maximum-margin classifier



A maximum-margin classifier built from three examples, with $\mathbf{w} = (0, -1/2)$ and margin 2.

The circled examples are the support vectors: they receive non-zero Lagrange multipliers and define the decision boundary.

A maximum-margin classifier

A maximum-margin classifier

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \qquad \mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

A maximum-margin classifier

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \qquad \mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix $\mathbf{X}'$ on the right incorporates the class labels; i.e., the rows are $y_i\mathbf{x}_i$. The Gram matrix is (without and with class labels):

A maximum-margin classifier

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \qquad \mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix $\mathbf{X}'$ on the right incorporates the class labels; i.e., the rows are $y_i \mathbf{x}_i$. The Gram matrix is (without and with class labels):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & -5 \\ 3 & 5 & -3 \\ -5 & -3 & 5 \end{pmatrix} \qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{pmatrix}$$

A maximum-margin classifier

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ -1 & 2 \\ -1 & -2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} -1 \\ -1 \\ +1 \end{pmatrix} \qquad \mathbf{X}' = \begin{pmatrix} -1 & -2 \\ 1 & -2 \\ -1 & -2 \end{pmatrix}$$

The matrix $\mathbf{X}'$ on the right incorporates the class labels; i.e., the rows are $y_i\mathbf{x}_i$. The Gram matrix is (without and with class labels):

$$\mathbf{X}\mathbf{X}^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & -5 \\ 3 & 5 & -3 \\ -5 & -3 & 5 \end{pmatrix} \qquad \mathbf{X}'\mathbf{X}'^{\mathrm{T}} = \begin{pmatrix} 5 & 3 & 5 \\ 3 & 5 & 3 \\ 5 & 3 & 5 \end{pmatrix}$$

The dual optimisation problem is thus

$$\begin{aligned} & \underset{\alpha_1,\alpha_2,\alpha_3}{\arg\max} -\frac{1}{2}\left(5\alpha_1^2 + 3\alpha_1\alpha_2 + 5\alpha_1\alpha_3 + 3\alpha_2\alpha_1 + 5\alpha_2^2 + 3\alpha_2\alpha_3 + 5\alpha_3\alpha_1 + 3\alpha_3\alpha_2 + 5\alpha_3^2\right) \\ & \qquad + \alpha_1 + \alpha_2 + \alpha_3 \\ = & \underset{\alpha_1,\alpha_2,\alpha_3}{\arg\max} -\frac{1}{2}\left(5\alpha_1^2 + 6\alpha_1\alpha_2 + 10\alpha_1\alpha_3 + 5\alpha_2^2 + 6\alpha_2\alpha_3 + 5\alpha_3^2\right) + \alpha_1 + \alpha_2 + \alpha_3 \end{aligned}$$

subject to $\alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0$ and $-\alpha_1 - \alpha_2 + \alpha_3 = 0$.

A maximum-margin classifier

A maximum-margin classifier

- Using the equality constraint we can eliminate one of the variables, say α_3, and simplify the objective function to

$$\underset{\alpha_1,\alpha_2,\alpha_3}{\arg\max} -\frac{1}{2}\left(20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2\right) + 2\alpha_1 + 2\alpha_2$$

A maximum-margin classifier

- Using the equality constraint we can eliminate one of the variables, say α_3, and simplify the objective function to

$$\underset{\alpha_1,\alpha_2,\alpha_3}{\arg\max} -\frac{1}{2}\left(20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2\right) + 2\alpha_1 + 2\alpha_2$$

- Setting partial derivatives to 0 we obtain $-20\alpha_1 - 16\alpha_2 + 2 = 0$ and $-16\alpha_1 - 16\alpha_2 + 2 = 0$ (notice that, because the objective function is quadratic, these equations are guaranteed to be linear).

A maximum-margin classifier

- Using the equality constraint we can eliminate one of the variables, say α_3, and simplify the objective function to

$$\underset{\alpha_1,\alpha_2,\alpha_3}{\arg\max} -\frac{1}{2}\left(20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2\right) + 2\alpha_1 + 2\alpha_2$$

- Setting partial derivatives to 0 we obtain $-20\alpha_1 - 16\alpha_2 + 2 = 0$ and $-16\alpha_1 - 16\alpha_2 + 2 = 0$ (notice that, because the objective function is quadratic, these equations are guaranteed to be linear).
- We therefore obtain the solution $\alpha_1 = 0$ and $\alpha_2 = \alpha_3 = 1/8$. We then have $\mathbf{w} = 1/8(\mathbf{x}_3 - \mathbf{x}_2) = \begin{pmatrix} 0 \\ -1/2 \end{pmatrix}$, resulting in a margin of $1/||\mathbf{w}|| = 2$.

A maximum-margin classifier

- Using the equality constraint we can eliminate one of the variables, say α_3, and simplify the objective function to

$$\underset{\alpha_1,\alpha_2,\alpha_3}{\arg\max} -\frac{1}{2}\left(20\alpha_1^2 + 32\alpha_1\alpha_2 + 16\alpha_2^2\right) + 2\alpha_1 + 2\alpha_2$$

- Setting partial derivatives to 0 we obtain $-20\alpha_1 - 16\alpha_2 + 2 = 0$ and $-16\alpha_1 - 16\alpha_2 + 2 = 0$ (notice that, because the objective function is quadratic, these equations are guaranteed to be linear).
- We therefore obtain the solution $\alpha_1 = 0$ and $\alpha_2 = \alpha_3 = 1/8$. We then have $\mathbf{w} = 1/8(\mathbf{x}_3 - \mathbf{x}_2) = \begin{pmatrix} 0 \\ -1/2 \end{pmatrix}$, resulting in a margin of $1/||\mathbf{w}|| = 2$.
- Finally, t can be obtained from any support vector, say $\mathbf{x}_2$, since $y_2(\mathbf{w} \cdot \mathbf{x}_2 - t) = 1$; this gives $-1 \cdot (-1 - t) = 1$, hence $t = 0$.

Noise

Noise

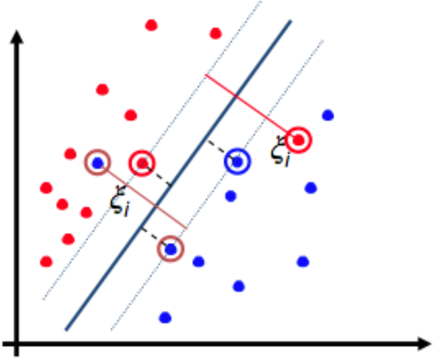
- So far we have assumed that the data is separable (in original or transformed space)

Noise

- So far we have assumed that the data is separable (in original or transformed space)
- Misclassified examples may break the separability assumption

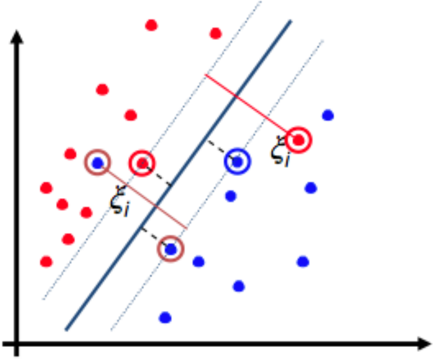
Noise

- So far we have assumed that the data is separable (in original or transformed space)
- Misclassified examples may break the separability assumption



Noise

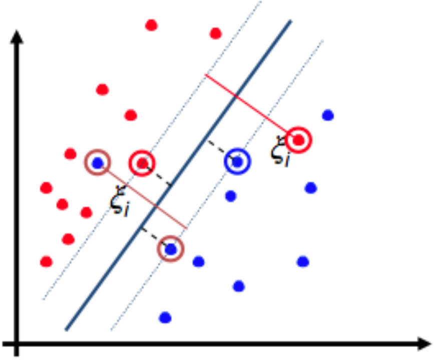
- So far we have assumed that the data is separable (in original or transformed space)
- Misclassified examples may break the separability assumption



- Introduce "slack" variables ξ_i to allow misclassification of instances

Noise

- So far we have assumed that the data is separable (in original or transformed space)
- Misclassified examples may break the separability assumption



- Introduce "slack" variables ξ_i to allow misclassification of instances
- This "soft margin" allows SVMs to handle noisy data

Allowing margin errors

Allowing margin errors

The idea is to introduce *slack variables* ξ_i, one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary.

Allowing margin errors

The idea is to introduce *slack variables* ξ_i, one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary.

We will call these *margin errors*.

Allowing margin errors

The idea is to introduce *slack variables* ξ_i, one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary.

We will call these *margin errors*.

Thus, we change the constraints to $\mathbf{w} \cdot \mathbf{x}_i - t \geq 1 - \xi_i$ and add the sum of all slack variables to the objective function to be minimised, resulting in the following *soft margin* optimisation problem:

Allowing margin errors

The idea is to introduce *slack variables* ξ_i, one for each example, which allow some of them to be inside the margin or even at the wrong side of the decision boundary.

We will call these *margin errors*.

Thus, we change the constraints to $\mathbf{w} \cdot \mathbf{x}_i - t \geq 1 - \xi_i$ and add the sum of all slack variables to the objective function to be minimised, resulting in the following *soft margin* optimisation problem:

$$\mathbf{w}^*, t^*, \xi_i^* = \operatorname*{arg\,min}_{\mathbf{w}, t, \xi_i} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i \text{and } \xi_i \geq 0, 1 \leq i \leq n$$

Allowing margin errors

Allowing margin errors

- C is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of C means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.

Allowing margin errors

- C is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of C means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.
- If we allow more margin errors we need fewer support vectors, hence C controls to some extent the 'complexity' of the SVM and hence is often referred to as the *complexity parameter*.

Allowing margin errors

- C is a user-defined parameter trading off margin maximisation against slack variable minimisation: a high value of C means that margin errors incur a high penalty, while a low value permits more margin errors (possibly including misclassifications) in order to achieve a large margin.
- If we allow more margin errors we need fewer support vectors, hence C controls to some extent the 'complexity' of the SVM and hence is often referred to as the *complexity parameter*.

C has to be set, e.g., by cross-validation

Sparse data

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of features

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of features
- Example: text classification (most vocabulary words not in any given document)

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of features
- Example: text classification (most vocabulary words not in any given document)
- Example: gene expression (most genes in genome not expressed in any given cell)

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of features
- Example: text classification (most vocabulary words not in any given document)
- Example: gene expression (most genes in genome not expressed in any given cell)
- SVMs a good choice for $p \gg n$ problems, where

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of features
- Example: text classification (most vocabulary words not in any given document)
- Example: gene expression (most genes in genome not expressed in any given cell)
- SVMs a good choice for $p \gg n$ problems, where
  - p number of features or parameters

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e., many feature values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - Just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of features
- Example: text classification (most vocabulary words not in any given document)
- Example: gene expression (most genes in genome not expressed in any given cell)
- SVMs a good choice for $p \gg n$ problems, where
  - p number of features or parameters
  - n number of examples

Contents

The kernel trick

Let $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ be two data points, and consider the mapping $(x, y) \mapsto (x^2, y^2, \sqrt{2}xy)$ to a three-dimensional feature space. The points in feature space corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ are $\mathbf{x}_1' = (x_1^2, y_1^2, \sqrt{2}x_1y_1)$ and $\mathbf{x}_2' = (x_2^2, y_2^2, \sqrt{2}x_2y_2)$. The dot product of these two feature vectors is

$$\mathbf{x}_1' \cdot \mathbf{x}_2' = x_1^2x_2^2 + y_1^2y_2^2 + 2x_1y_1x_2y_2 = (x_1x_2 + y_1y_2)^2 = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$$

That is, by squaring the dot product in the original space we obtain the dot product in the new space *without actually constructing the feature vectors*! A function that calculates the dot product in feature space directly from the vectors in the original space is called a *kernel* – here the kernel is $\boldsymbol{\kappa}(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 \cdot \mathbf{x}_2)^2$.

Kernel trick

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$$

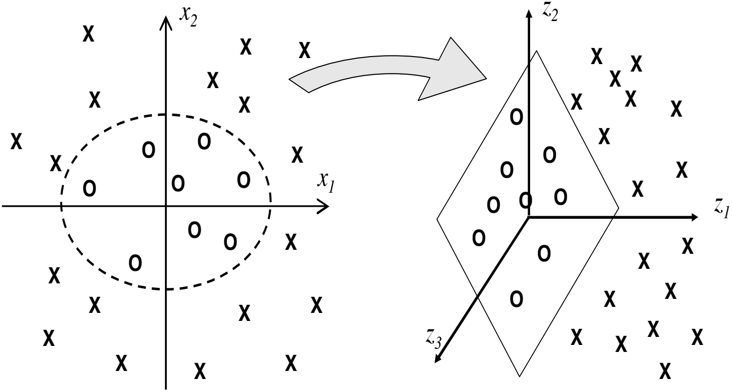


Figure by Avrim Blum, CS Dept, CMU.

'Kernelising' the perceptron

The perceptron algorithm is a simple counting algorithm – the only operation that is somewhat involved is testing whether example $\mathbf{x}_i$ is correctly classified by evaluating $y_i \sum_{j=1}^{|D|} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j$.

- The key component of this calculation is the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$.
- Assuming bivariate examples $\mathbf{x}_i = (x_i, y_i)$ and $\mathbf{x}_j = (x_j, y_j)$ for notational simplicity, the dot product can be written as $\mathbf{x}_i \cdot \mathbf{x}_j = x_i x_j + y_i y_j$.
- The corresponding instances in the quadratic feature space are $\left(x_i^2, y_i^2\right)$ and $\left(x_j^2, y_j^2\right)$, and their dot product is $\left(x_i^2, y_i^2\right) \cdot \left(x_j^2, y_j^2\right) = x_i^2 x_j^2 + y_i^2 y_j^2$.
- This is almost equal to $(\mathbf{x}_i \cdot \mathbf{x}_j)^2 = (x_i x_j + y_i y_j)^2 = (x_i x_j)^2 + (y_i y_j)^2 + 2 x_i x_j y_i y_j$, but not quite because of the third term of cross-products.
- We can capture this term by extending the feature vector with a third feature $\sqrt{2}xy$.

'Kernelising' the perceptron

This gives the following feature space:

$$\phi(\mathbf{x}_i) = \left(x_i^2, y_i^2, \sqrt{2}x_iy_i\right) \qquad \phi(\mathbf{x}_j) = \left(x_j^2, y_j^2, \sqrt{2}x_jy_j\right)$$
$$\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = x_i^2x_j^2 + y_i^2y_j^2 + 2x_ix_jy_iy_j = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$$

- We now define $\boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$, and replace $\mathbf{x}_i \cdot \mathbf{x}_j$ with $\boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j)$ in the dual perceptron algorithm to obtain the *kernel perceptron*
- This would work for many other kernels satisfying certain conditions.

'Kernelising' the perceptron

Algorithm KernelPerceptron(D, η) // perceptron training algorithm using a ker

Input: labelled training data D in homogeneous coordinates, plus
kernel function $\boldsymbol{\kappa}$

Output: coefficients α_i defining non-linear decision boundary

$\alpha_i \leftarrow 0$ for $1 \leq i \leq |D|$
$converged \leftarrow$ false
while $converged =$ false **do**
    $converged \leftarrow$ true
    for $i = 1$ to $|D|$ **do**
        if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j) \leq 0$ **then**
            $\alpha_i \leftarrow \alpha_i + 1$
            $converged \leftarrow$ false
        end
    end
end

Other kernels

We can define a polynomial kernel of any degree p as $\boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^p$. This transforms a d-dimensional input space into a high-dimensional feature space, such that each new feature is a product of p terms (possibly repeated).

If we include a constant, say $\boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^p$, we would get all lower-order terms as well. So, for example, in a bivariate input space and setting $p = 2$ the resulting feature space is

$$\phi(\mathbf{x}) = \left(x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1\right)$$

with linear as well as quadratic features.

Other kernels

An often-used kernel is the *Gaussian kernel*, defined as

$$\boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right)$$

where σ is a parameter known as the *bandwidth*.

Notice that the soft margin optimisation problem (above) is defined in terms of dot products between training instances and hence the 'kernel trick' can be applied to SVMs:

Other kernels

- The decision boundary learned with a non-linear kernel cannot be represented by a simple weight vector in input space. Thus, in order to classify a new example $\mathbf{x}$ we need to evaluate $y_i \sum_{j=1}^{n} \alpha_j y_j \boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}_j)$ which is an $O(n)$ computation involving all training examples, or at least the ones with non-zero multipliers α_j.
- This is why support vector machines are a popular choice as a kernel method, since they naturally promote sparsity in the support vectors.
- Although we have restricted attention to numerical features here, kernels can be defined over discrete structures, including trees, graphs, and logical formulae, opening the way to extending geometric models to non-numerical data[1].

[1] See, for example, Schölkopf and Smola (2002).

Contents

Summary: Learning with Kernel Methods

Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics

Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics
- Another example of the “optimisation” approach to machine learning

Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics
- Another example of the "optimisation" approach to machine learning
- Kernelisation is a "modular" approach to machine learning

Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics
- Another example of the "optimisation" approach to machine learning
- Kernelisation is a "modular" approach to machine learning
- Algorithms that can be kernelised can learn different model classes simply by changing the kernel, e.g., string kernels for sequence data

Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics
- Another example of the "optimisation" approach to machine learning
- Kernelisation is a "modular" approach to machine learning
- Algorithms that can be kernelised can learn different model classes simply by changing the kernel, e.g., string kernels for sequence data
- SVMs exemplify this – mostly for classification (but also regression, "one-class' classification, etc.)

Summary: Learning with Kernel Methods

- Kernel methods around for a long time in statistics
- Another example of the "optimisation" approach to machine learning
- Kernelisation is a "modular" approach to machine learning
- Algorithms that can be kernelised can learn different model classes simply by changing the kernel, e.g., string kernels for sequence data
- SVMs exemplify this – mostly for classification (but also regression, "one-class' classification, etc.)
- SVMs one of the most widely used "off-the-shelf" classifier learning methods, especially for "small n (examples), large p (dimensionality)" classification problems

Contents

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA.