Support Vector Machines

Adam Richards

Galvanize, Inc

Updated: 10. August 2017

Objectives

Morning

- Decision boundaries, hyperplanes
- Margins
- Maximum margin classifier

Afternoon

- Kernels
- Support vector machines (SVM)
- Parameter tuning (C,ϵ)
- Variants and extensions of SVMs

Support vector machines

Support vector machines

Find a hyperplane that separates classes in feature space

Vladimir Vapnik

If we cannot do this in a satisfying way:

1. We introduce the concept of soft margins
2. We enrich and enlarge the feature space to make separation possible

Some of the neat aspects of SVMs

- Look for this hyperplane in a direct way
- SVMs are a special instance of kernel machines
- Kernel methods exploit information related to inner products
- a kernel trick helps make computation easy
- SVMs make implicit use of a $L1$ penalty

High level overview

The model is similar to linear regression because at it's center is a linear function:

$$\mathbf{w}^T\mathbf{x} + b \tag{1}$$

SVMs do not provide probabilities, just a class prediction.
For example, in the two class problem:

If $\mathbf{w}^T\mathbf{x} + b$ is positive:
    class = 1
or $\mathbf{w}^T\mathbf{x} + b$ is negative:
    class = -1

(?), (?) (pp137)

Hyperplane

- A hyperplane in p dimensions is a flat affine subspace of dimension $p-1$
- An optimal separating hyperplane is a unique solution that separates two classes and maximizes the *margin* between the classes.

$$f(x) = b + w_1 x_1 + w_2 x_2 \ldots w_p x_p = 0 \tag{2}$$

$$f(x) = \mathbf{w}^T \mathbf{x} + b = 0 \tag{3}$$

- If $b = 0$ the hyperplane goes through the origin
- The vector $\mathbf{w} = (w_1, w_2, ..., w_p)$ is known as the normal vector
  It points in a direction orthogonal to the surface of the hyperplane

So what is a hyperplane?

Think of b as β_0, w_1 as β_1 and so on

Hyperplane

- A hyperplane in p dimensions is a flat affine subspace of dimension $p-1$
- An optimal separating hyperplane is a unique solution that separates two classes and maximizes the *margin* between the classes.

$$f(x) = b + w_1 x_1 + w_2 x_2 \ldots w_p x_p = 0 \quad (2)$$

$$f(x) = \mathbf{w}^T \mathbf{x} + b = 0 \quad (3)$$

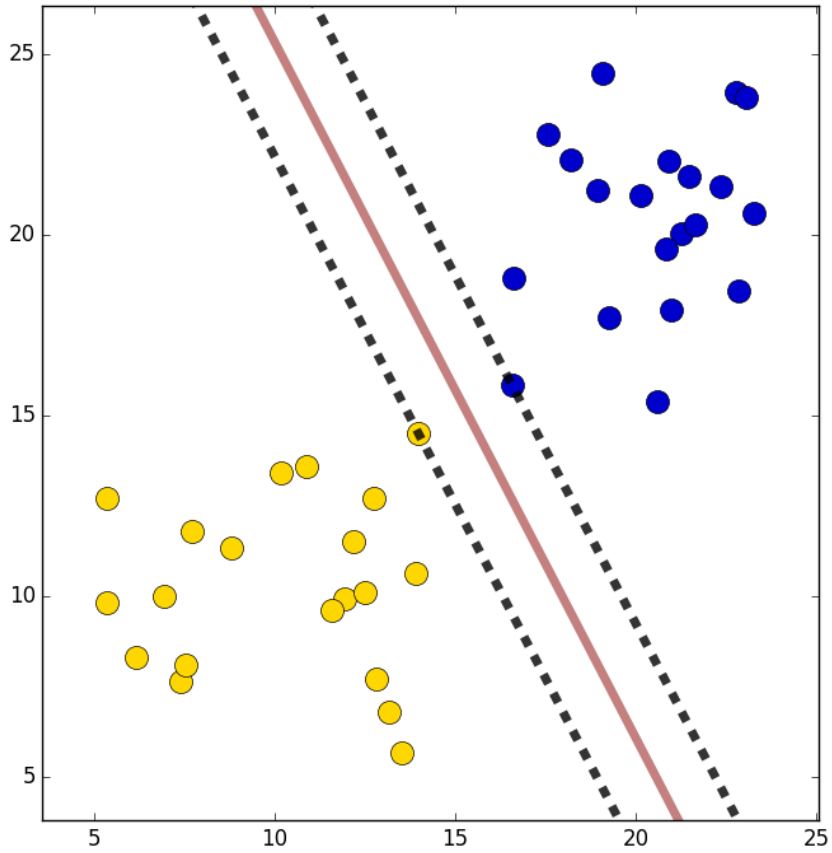
- If $b = 0$ the hyperplane goes through the origin
- The vector $\mathbf{w} = (w_1, w_2, ..., w_p)$ is known as the normal vector
  It points in a direction orthogonal to the surface of the hyperplane

So what is a hyperplane?

Dims	Hyperplane
1D	a point
2D	a line
3D	a plane
pD	$p-1$ dimensional hyperplane

Think of b as β_0, w_1 as β_1 and so on

Hyperplane



Was this the only hyperplane?

Margin

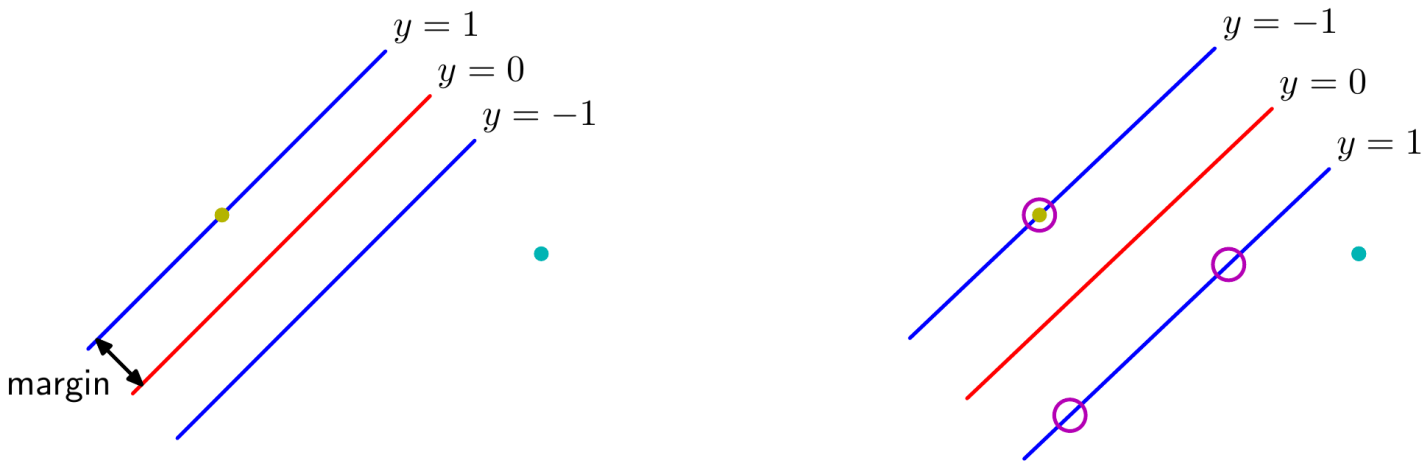
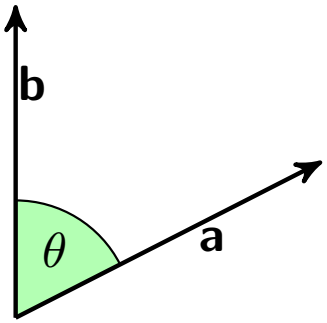


Figure 7.1 The margin is defined as the perpendicular distance between the decision boundary and the closest of the data points, as shown on the left figure. Maximizing the margin leads to a particular choice of decision boundary, as shown on the right. The location of this boundary is determined by a subset of the data points, known as support vectors, which are indicated by the circles.

(?) (Fig. 7.1)

Dot product

The dot product may be defined geometrically or algebraically



$$\mathbf{a}^T\mathbf{b} = \sum a_i b_i$$

$$\mathbf{ab} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos(\theta)$$

```
import numpy as np
a = np.array([[1,2,3,4]]).T
b = np.array([[4,5,6,7]]).T
print(np.dot(a.T,b)[0][0])
print(np.array([a[i]*b[i] for i in range(a.shape[0])]).sum())
```

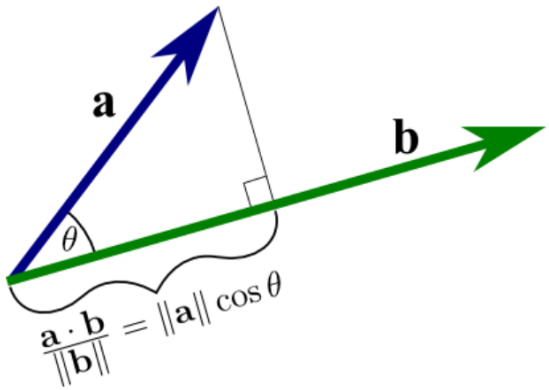
- If **a** and **b** are orthogonal then $\theta = 90$ and $\mathbf{ab} = 0$

Cosine similarity

Given two vectors of attributes, **a** and **b** the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as

$$\frac{\mathbf{ab}}{\|\mathbf{a}\|\,\|\mathbf{b}\|} \text{ or } \frac{\mathbf{ab}}{\|\mathbf{b}\|}$$

when **a** is a unit vector (vector of length 1)

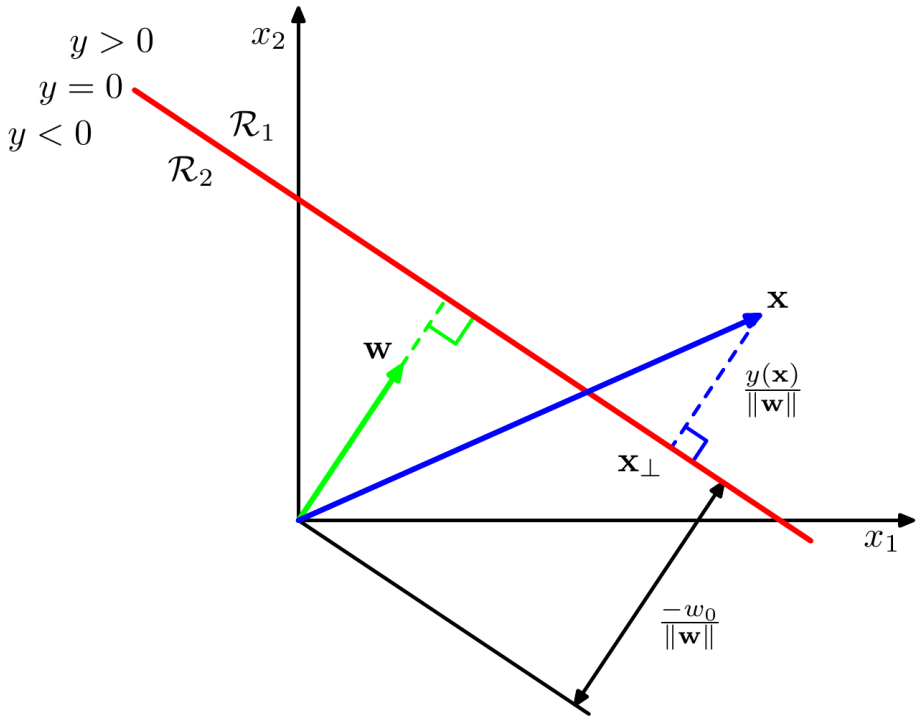



```
cos_sim = np.dot(a.T,b)/(np.linalg.norm(a.T)*np.linalg.norm(b))
```

Check out this tool http://mathinsight.org/dot_pframeroduct

Margin in a bit more detail

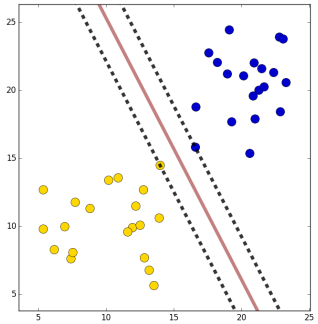
Figure 4.1 Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\mathbf{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\mathbf{x}$ from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$.



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

(?) (Fig. 4.1) (see pp 182 for more details)

Check for understanding



Exercise (Work in groups of 3 or 4) Say $b = -6, w_1 = 0.8, w_2 = 0.6$

1. Draw a similar plot on your table
2. Draw the normal vector
3. Draw a point on the right hand side of the street
4. Orthogonally project the point onto the normal
5. Evaluate the linear function for your point
6. What if my point falls on the hyperplane?

If we evaluate the sum of squares for w_1 and w_2 and they add up to 1 then that means that the normal vector is a **unit vector**. Something neat happens here in that the value that you get is actually the euclidean distance from the hyperplane.

Objectives

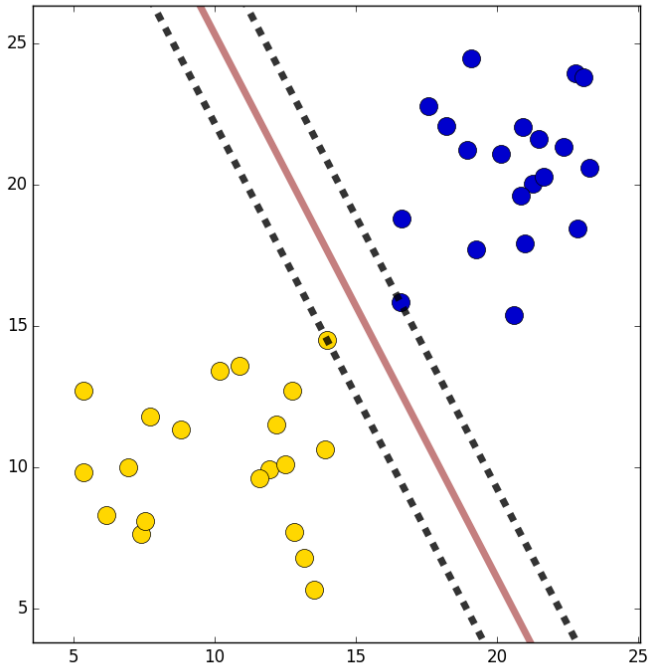
Morning

- ✓ Decision boundaries,hyperplanes
- ✓ Margins
  - Maximum margin classifier

Afternoon

- Kernels
- Support vector machine (SVM)
- Parameter tuning (C,ϵ)
- Variants and extensions of SVMs

If $f(X) = b + w_1 x_1 + ... w_p x_p$ then $f(X) > 0$ for points on one side of the hyperplane and $f(X) < 0$ for points on the other side
If we say that $y_i = 1$ for blue points and $y_i = -1$ for yellow then a separating hyperplane is $y_i \times f(x_i) > 0$ for all $i, f(x) = 0$



So what are we trying to do to get the decision function oriented correctly?

And what do we want w.r.t. the margins?

Maximal Margin Classifier

It is a constrained optimization problem

Maximize M w.r.t $b, w_1, w_2 \ldots w_p$

subject to

$$\sum_{j=1}^{p} w_j^2 = 1 \tag{4}$$

The margin applies for all points

$$y_i \left(b + w_1 x_1 + \ldots w_p x_p\right) \geq M \tag{5}$$

for all $i = 1, \ldots N$

Max margin

Support vectors

The maximum margin hyperplane is defined only by the closest points. Support vectors can be useful, because we only need to use a subset of the inputs to make decisions.

What are some of the potential issues that we might want to keep in mind?

Max margin

Support vectors

The maximum margin hyperplane is defined only by the closest points. Support vectors can be useful, because we only need to use a subset of the inputs to make decisions.

What are some of the potential issues that we might want to keep in mind?

- Outliers can heavily influence which support vectors we choose
- What if the data are not linearly separable?

Objectives

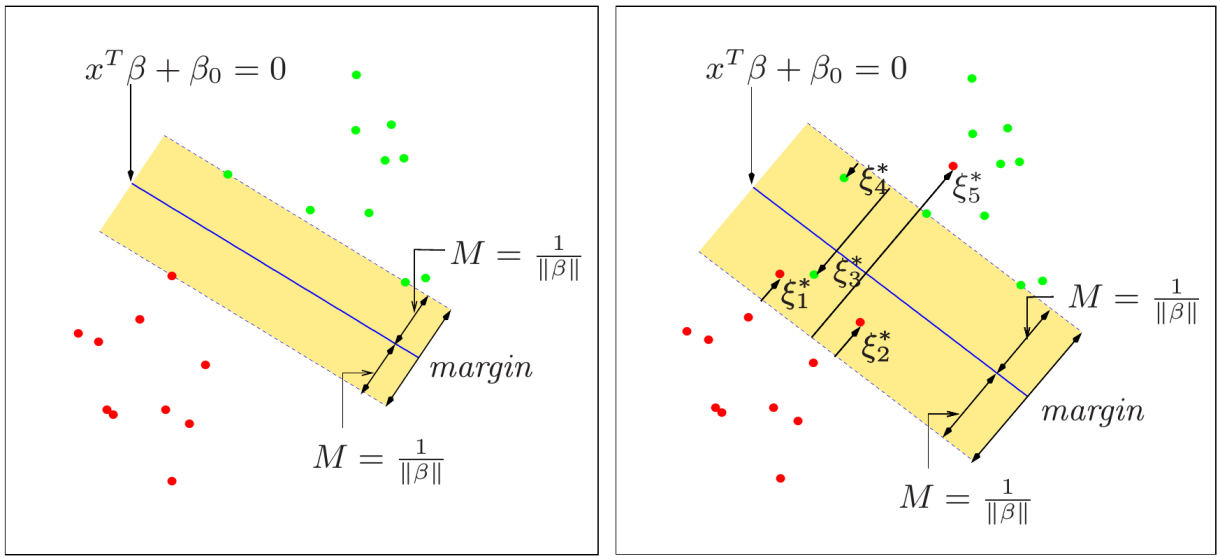
Morning

- ✓ Decision boundaries,hyperplanes
- ✓ Margins
- ✓ Maximum margin classifier

Afternoon

- Kernels
- Support vector machine (SVM)
- Parameter tuning (C,ϵ)
- Variants and extensions of SVMs

Soft margins - relax the idea of a separating hyperplane

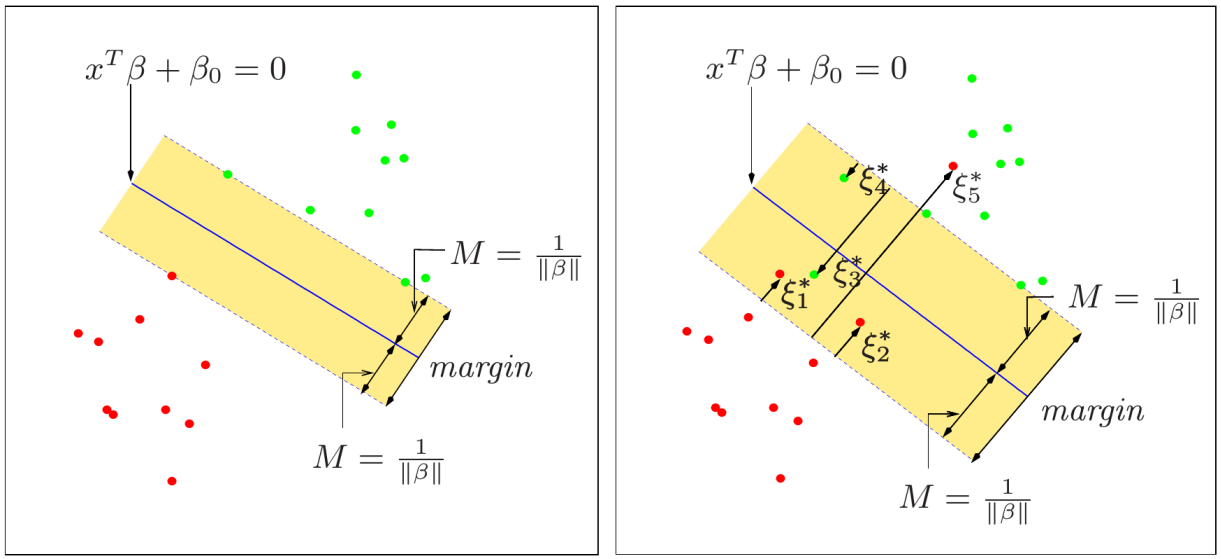


- Points on the correct side have $\xi_j^* = 0$
- Points on the wrong side $\xi_j = M\xi_j^*$
- The margins are maximized with respect to a total budget $\sum \xi_i \leq \text{constant}$

Is there another way to describe $\sum \xi_i$?

(?)

Soft margins - relax the idea of a separating hyperplane



- Points on the correct side have $\xi_j^* = 0$
- Points on the wrong side $\xi_j = M\xi_j^*$
- The margins are maximized with respect to a total budget $\sum \xi_i \leq \text{constant}$

Is there another way to describe $\sum \xi_i$?
The total distance of points on the wrong side of their margin

(?)

Maximal (soft) Margin Classifier

Adjusting the margins is a way of regularizing... This is because the margin is now determined by more than just the closest point.

It is a constrained optimization problem

Maximize M w.r.t $b, w_1, w_2 ... w_p$

subject to Not just the margin now (M) but a budget C

$$\sum_{j=1}^{p} w_j^2 = 1 \tag{6}$$

The margin applies for all points

$$y_i \left(b + w_1 x_1 + ... w_p x_p\right) \geq M(1 - \epsilon_1), \tag{7}$$

$$\epsilon_i \geq 0, \sum N_{i=1} \epsilon_i \leq C \tag{8}$$

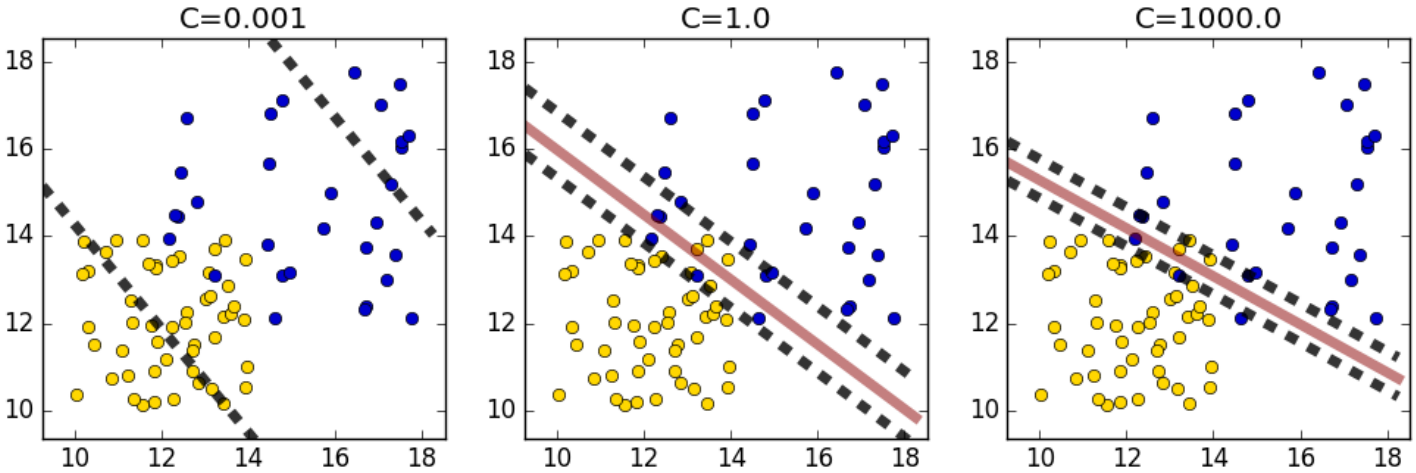
for all $i = 1, ...N$

Modifying the margins

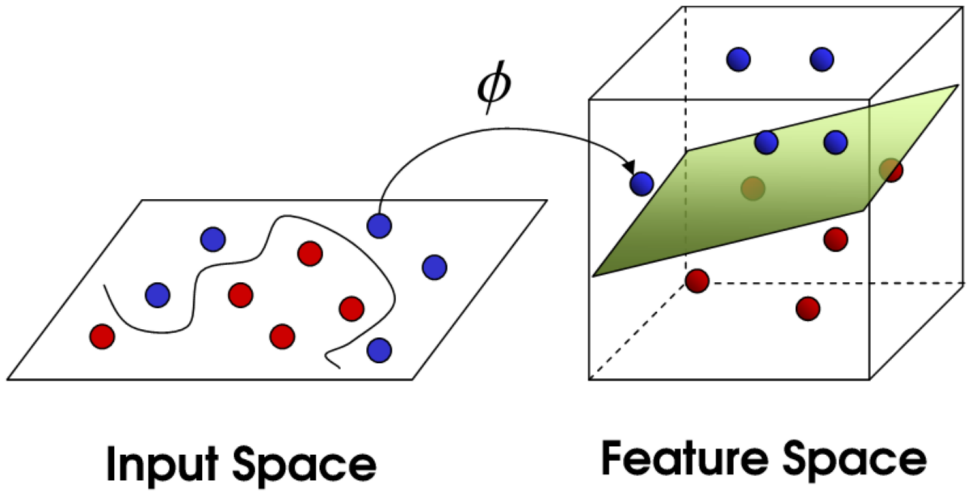
- C is a regularization parameter that modulates the misclassification error penalty
  - Large C - hard margins (accuracy more important)
  - Small C - soft margins (generalization more important)

```
from sklearn import svm
clf = svm.SVC(kernel='linear',C=1.0)
clf.fit(X,y)
```

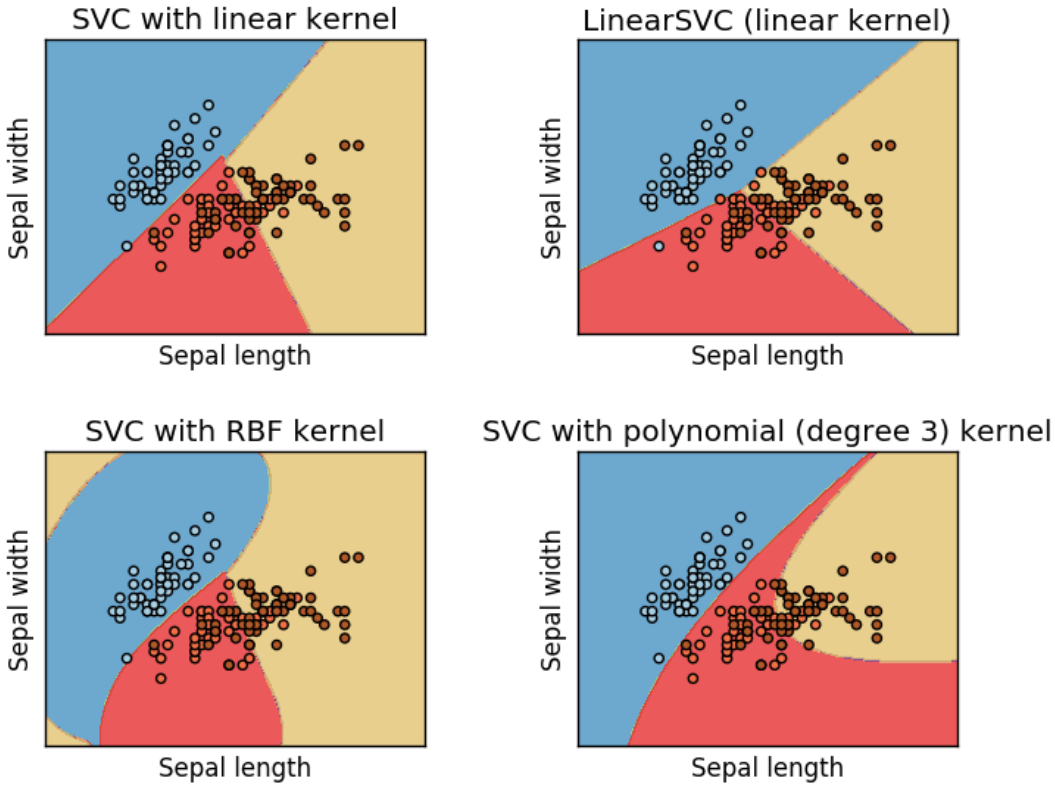
Some programming languages implement this differently so it may actually be $1/C$



- The more points that are involved in the estimation of the margin the more stable the orientation of the margin
- Smaller values of C


ϕ
Input Space
Feature Space

Kernels



http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html

Kernels

- Enlarge the feature space $X_1^2, X_1^3, X_1^2 X_2$
- The decision boundary takes a more complex form (squares, cross-products etc)
- In the enlarged space the decision boundary is linear, but projected back down to two dimensions we get a non-linear function.
- Polynomials get crazy kinda fast
- kernels are the way to account for this

The "Kernel Trick"

Back to the math...

In our optimization problem to maximize the margin, we eventually end up optimizing a vector alpha $\alpha^{(i)}, i \in [1, m]$ in the following equation:

$$\mathcal{L}(\alpha) = \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} (x^{(i)} \cdot x^{(j)})$$

$$\mathcal{L}(\alpha) = \sum_{i=1}^{m} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha^{(i)} \alpha^{(j)} (\phi(x^{(i)}) \cdot \phi(x^{(j)}))$$

The "Kernel Trick"

Creating a kernel...

$$\phi(x^{(i)}) \cdot \phi(x^{(j)}) \in \mathbb{R}$$

... this is just a real number.

What if we never applied ϕ and we never took the dot product, but we instead replaced this whole thing with a "kernel function".

The "Kernel Trick"

Creating a kernel...

$$\phi(x^{(i)}) \cdot \phi(x^{(j)}) \in \mathbb{R}$$

... this is just a real number.

What if we never applied ϕ and we never took the dot product, but we instead replaced this whole thing with a "kernel function".

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)}) \in \mathbb{R}$$

The "Kernel Trick"

Why is this so cool?

- Saves some computation. We never need to compute ϕ.
- Opens new possibilities. A kernel can operate in infinite dimensions!

You can use any $K(x^{(i)}, x^{(j)})$ as long as there **exists** some ϕ such that

$$K(x^{(i)}, x^{(j)}) = \phi(x^{(i)}) \cdot \phi(x^{(j)})$$

... but you don't have to know what ϕ actually **is**!

Another take...
There are certain kernels that can be expressed as an inner product in much higher dimensional space. The inner product gives us the notion of similarity which is related to the inverse of distance function we already know how to compute

$$f(x) = \beta_0 + \sum_{i=1}^{N} a_i k(x, x_i) \tag{9}$$

The Polynomial Kernel

$$K(x^{(i)}, x^{(j)}) = (1 + x^{(i)} \cdot x^{(j)})^d$$

- equivalent to the dot product in the d-order ϕ space
- requires an extra hyper-parameter, d, for "degree"

The RBF Kernel
(Radial Basis Function)

$$K(x^{(i)}, x^{(j)}) = \exp\left(-\gamma||x^{(i)} - x^{(j)}||^2\right)$$

- equivalent to the dot product in the Hilbert space of infinite dimensions
- requires an extra hyper-parameter, γ, "gamma"

What happens when we increase γ?
γ defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. Or we could say that the more we increase γ the more refined the decision boundaries

The RBF Kernel
(Radial Basis Function)

$$K(x^{(i)}, x^{(j)}) = \exp\left(-\gamma||x^{(i)} - x^{(j)}||^2\right)$$

- equivalent to the dot product in the Hilbert space of infinite dimensions
- requires an extra hyper-parameter, γ, "gamma"

What happens when we increase γ?
γ defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. Or we could say that the more we increase γ the more refined the decision boundaries

Kernel overview

Here is yet another take on this since it can be a hard concept to grasp

$$\mathbf{w}^T\mathbf{x} + b = b + \sum_{i=1}^{m} \alpha_i \mathbf{x}^T \mathbf{x}^{(i)} \tag{10}$$

Where $\mathbf{x}^{(i)}$ is a training sample, and α is a vector of coefficients. $\mathbf{x}$ can be replaced with $\phi(\mathbf{x})$ and the dot product can be replaced with function $k(\mathbf{x}, \mathbf{x}^{(i)}$.

$$f(\mathbf{x}) = b + \sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}^{(i)}) \tag{11}$$

Explain what is going on here to your neighbor and how you would make class predictions with this

SVMs are not the only class of machine learning algorithm that can benefit from the kernel trick. The category of algorithms that use this are known as Kernel Machines.

sklearn.svm.LinearSVC

Parameter	Default	Comment
`C`	1.0	Penalty parameter of the error term commonly used
`loss`	'square_hinged'	Loss function
`penalty`	'l2'	Regularization penalty
`multi_class`	'ovr'	One vs rest
`random_state`	'None'	Random state

- sklearn.svm.LinearSVR is available as well
- SVR has an additional parameter ϵ - insensitivity zone
- loss defaults to 'epsilon_insensitive'
- SVR does not have the penalty parameter
- An increase in ϵ means a reduction in requirements for the accuracy
- ϵ also tells us how much point can be on the other side of the margin

sklearn.svm.SVC

Parameter	Default	Comment
`C`	1.0	Penalty parameter of the error term commonly used
`kernel`	'rbf'	linear, poly, rbf, sigmoid, precomputed
`degree`	3.0	work with 'poly'.
`gamma`	'auto'	work with 'rbf', and 'sigmoid','poly'.
`multi_class`	'ovr'	One vs rest
`random_state`	'None'	Random state

- sklearn.svm.SVR is available as well
- SVR has an additional parameter ϵ - insensitivity zone
- An increase in ϵ means a reduction in requirements for the accuracy
- Note that 'coef_' is only available for the linear kernel

Grid Search

```
from sklearn.grid_search import GridSearchCV
svc_rbf = SVC(kernel='rbf')

param_space = {'C':     np.logspace(-3, 4, 15),
               'gamma': np.logspace(-10, 3, 15)}

grid_search = GridSearchCV(svc_rbf, param_space,
                           scoring='accuracy', cv=10)
grid_search.fit(x, y)

print grid_search.grid_scores_
print grid_search.best_params_
print grid_search.best_score_
print grid_search.best_estimator_
```

Objectives

Morning

- ✓ Decision boundaries,hyperplanes
- ✓ Margins
- ✓ Maximum margin classifier

Afternoon

- ✓ Kernels
- ✓ Support vector machines (SVM)
- ✓ Parameter tuning (C,ϵ)
  - Variants and extensions of SVMs

Discussion

- Do you think scaling is important?
- What about class imbalance?
- What do you think a one-vs-rest approach means?
- How about one-vs-one ($K \times K - 1/2$)?
- Which kernels are the most flexible? Is flexible always good?
- `LinearSVC` uses the One-vs-All (also known as One-vs-Rest) multiclass reduction
- `SVC` uses the One-vs-One multiclass reduction.
- `LinearSVC` minimizes the squared hinge loss
- `SVC` minimizes the regular hinge loss.

Bias-Variance tradeoff

Explanation

Bias

Variance

Bias-Variance tradeoff

Explanation

Bias

A high-"bias" model makes many assumptions and prefers to solve problems a certain way.

Variance

Bias-Variance tradeoff

Explanation

Bias

A high-"bias" model makes many assumptions and prefers to solve problems a certain way.

E.g. A linear SVM looks for dividing hyperplanes in the input space *only*.

Variance

Bias-Variance tradeoff

Explanation

Bias

A high-"bias" model makes many assumptions and prefers to solve problems a certain way.

E.g. A linear SVM looks for dividing hyperplanes in the input space *only*.

For complex data, high-bias models often *underfit* the data.

Variance

Bias-Variance tradeoff

Explanation

Bias

A high-"bias" model makes many assumptions and prefers to solve problems a certain way.

E.g. A linear SVM looks for dividing hyperplanes in the input space *only*.

For complex data, high-bias models often *underfit* the data.

Variance

A high-"variance" model makes fewer assumptions and has more representational power.

Bias-Variance tradeoff

Explanation

Bias

A high-"bias" model makes many assumptions and prefers to solve problems a certain way.

E.g. A linear SVM looks for dividing hyperplanes in the input space *only*.

For complex data, high-bias models often *underfit* the data.

Variance

A high-"variance" model makes fewer assumptions and has more representational power.

E.g. An RBF SVM looks for dividing hyperplanes in an infinite-dimensional space.

Bias-Variance tradeoff

Explanation

Bias

A high-"bias" model makes many assumptions and prefers to solve problems a certain way.

E.g. A linear SVM looks for dividing hyperplanes in the input space *only*.

For complex data, high-bias models often *underfit* the data.

Variance

A high-"variance" model makes fewer assumptions and has more representational power.

E.g. An RBF SVM looks for dividing hyperplanes in an infinite-dimensional space.

For simple data, high-variance models often *overfit* the data.

SVMs vs Logistic Regression
(some rules of thumb)

1. Logistic Regression maximizes the *Binomial Log Likelihood* function.
2. SVMs maximize the *margin*.
3. When classes are nearly separable, SVMs tends to do better than Logistic Regression.
4. Otherwise, Logistic Regression (with Ridge) and SVMs are similar.
5. However, if you want to estimate probabilities, Logistic Regression is the better choice.
6. With kernels, SVMs work well. Logistic Regression works fine with kernels but can get computationally too expensive.
7. SVMs consider points near the margin. Logistic regression considers all the points in the data set. Depending on your application this may make a difference.
8. In higher dimensional space SVMs are generally preferred

Resources

- sklearn - maximum margin separating hyperplane
- sklearn - params
- Interpreting parameters
- Intro stat learning video 1 lecture [video]
- Intro stat learning video 2 lecture [video]
- Patrick Winston's Stanford lecture [video]
- If you are interested in the optimization play with this code

References I

Bengio, Y., Goodfellow, I. J., and Courville, A. (2016). *Deep Learning*. MIT Press. Book in preparation for MIT Press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, 20(3):273–297.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition.