GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

LECTURE 9: SUPPORT VECTOR MACHINES AND DECISION TREES

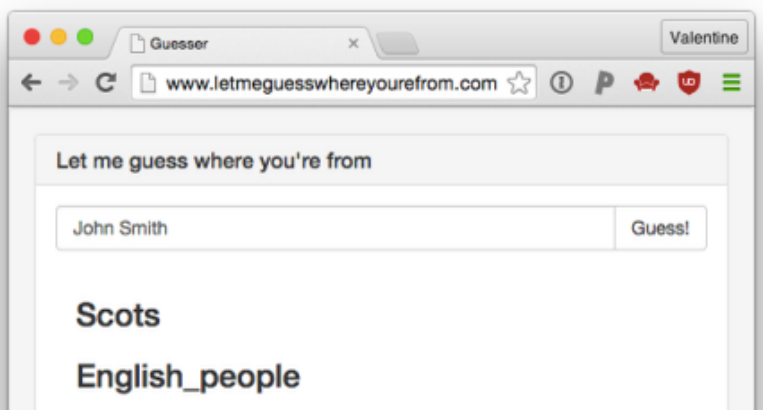
Francesco Mosconi
DAT16 SF // August 26, 2015

DATA SCIENCE IN THE NEWS

DATA SCIENCE IN THE NEWS

Let me guess where you're from

Let me guess where you're from
On the website letmeguesswhereyourefrom.com you can enter a name, and an algorithm will print five countries the name seems come from. Try it out!



http://nxn.se/post/127065307170/let-me-guess-where-youre-from

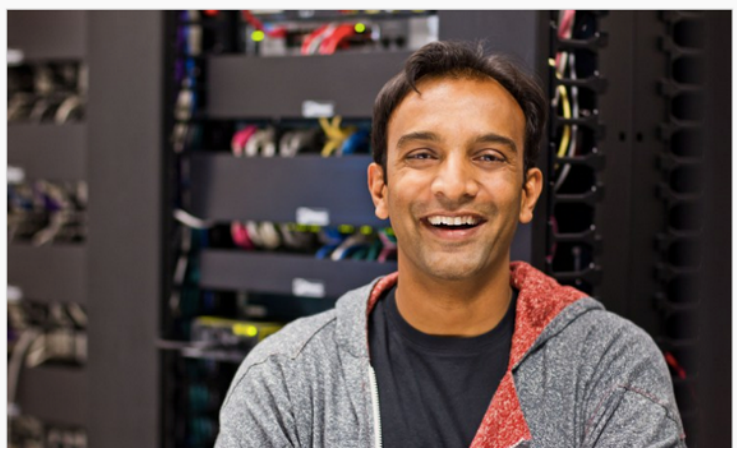
DATA SCIENCE IN THE NEWS

America's Chief Data Scientist DJ Patil Is Coming To Disrupt SF 2015

Posted Aug 19, 2015 by **Sarah Buhr** (@sarahbuhr)

1,154 SHARES

Next Story



CrunchBase

TechCrunch –

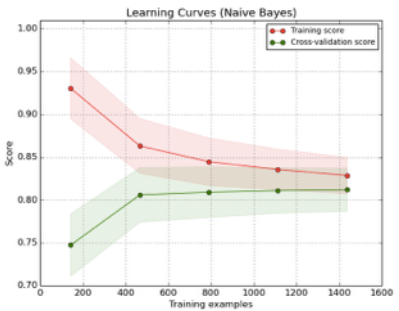
FOUNDED
2005

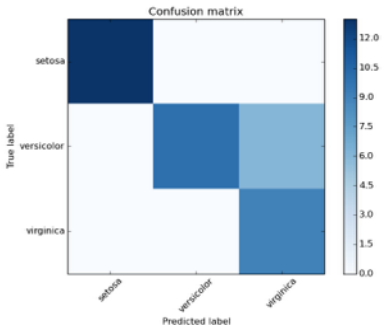
OVERVIEW
TechCrunch is a leading technology media property, dedicated to obsessively profiling startups, reviewing new Internet products, and breaking tech news. Founded in June 2005, TechCrunch and its network of websites now reach over 12 million unique visitors and draw more than 37 million page views per month. The TechCrunch community includes more than 2 million friends and followers on Twitter, Facebook, ...

http://techcrunch.com/2015/08/19/americas-deputy-chief-technology-officer-dj-patil-is-coming-to-disrupt-sf-2015/

LAST TIME:

I. IMBALANCED CLASSES
II. ERROR RATES
III. ROC CURVES
IV. EVALUATION METRICS
V. COST FUNCTION
VI. VALIDATION CURVES AND LEARNING CURVES





QUESTIONS?

WHAT WAS THE MOST INTERESTING THING YOU LEARNT?

WHAT WAS THE HARDEST TO GRASP?

AGENDA

I. SUPPORT VECTOR MACHINES (SVM)
II. LAB ON SVM
III. DECISION TREES
IV. LAB ON DECISION TREES

KEY OBJECTIVES

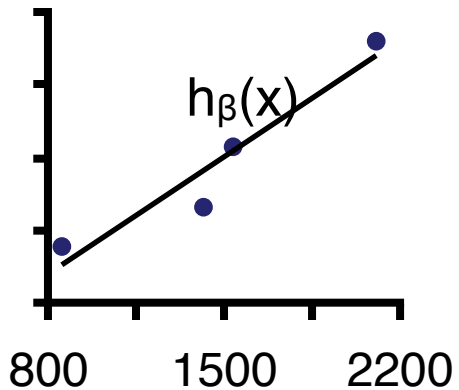
- KNOW WHAT SVMS ARE AND HOW THEY ARE CONSTRUCTED
- UNDERSTAND SLACK VARIABLES AND HOW THEY EXPAND SVMS
- KNOW WHAT A KERNEL FUNCTION IS AND HOW IT IS USED BY SVMS
- KNOW WHAT DECISION TREES ARE
- DESCRIBE HUNT'S ALGORITHM AND HOW IT IS USED TO CONSTRUCT A TREE
- UNDERSTAND THE ADVANTAGES OF DECISION TREES

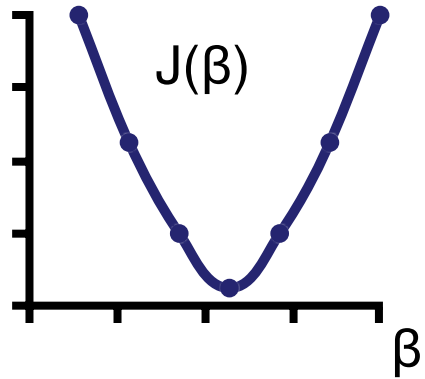
SUPPORT VECTOR MACHINES

Remember last time:

Training set : (x, y)

Hypothesis Function : $h_\beta(x)$



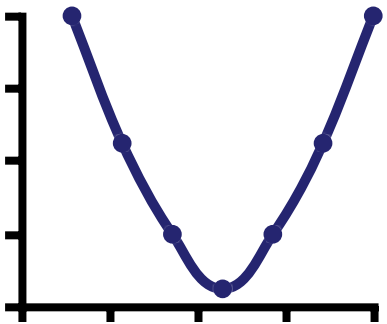


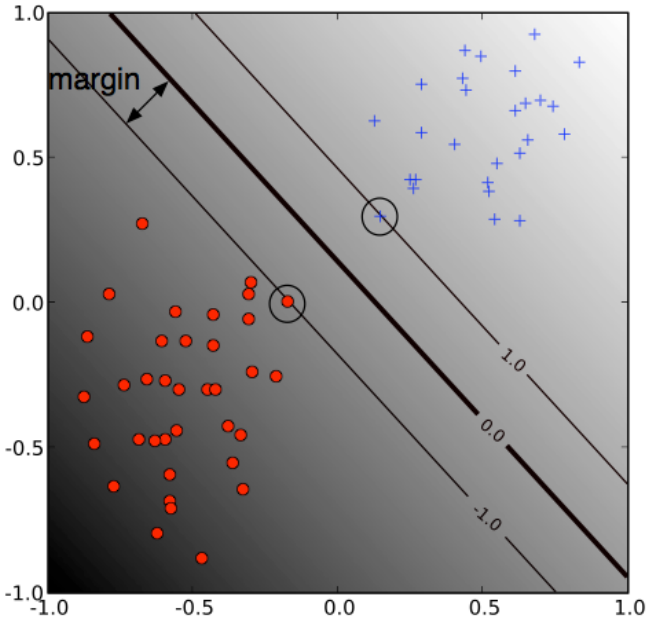
Cost Function: J(β)

GOAL

Find the values for the parameters β that minimize the cost function over the set of training data

Support vector machines are binary linear classifier whose decision boundary is explicitly constructed to minimize generalization error.





Decision boundary is derived using geometric reasoning
(as opposed to the algebraic reasoning we've used to derive other classifiers).

generalization error *is equated with the geometric concept of* **margin**, *which is the region along the decision boundary that is free of data points.*

SUPPORT VECTOR MACHINES

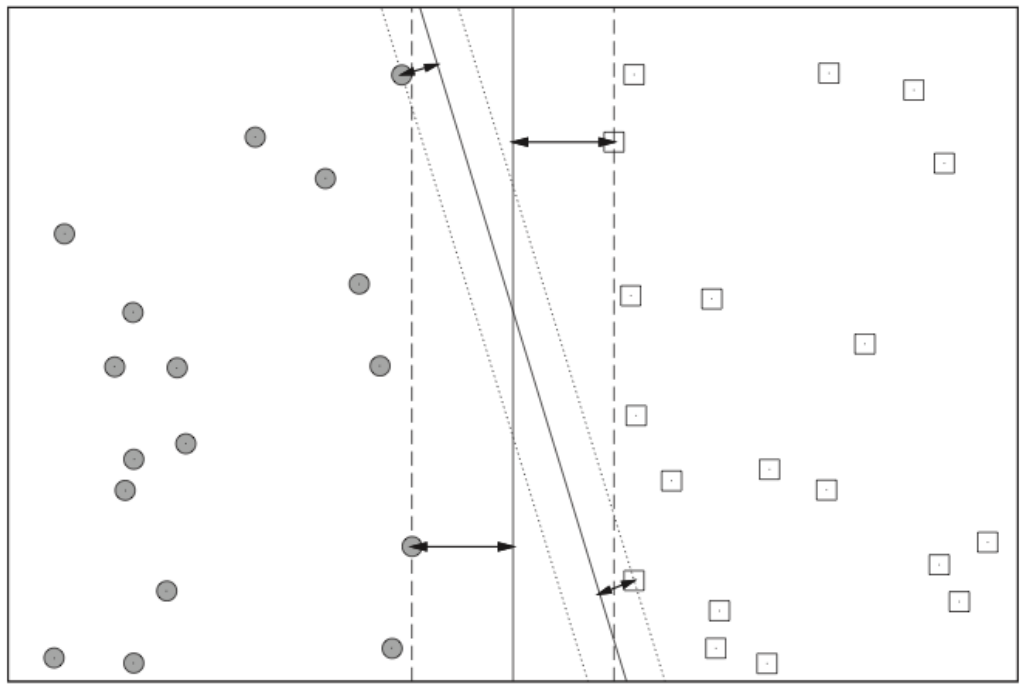
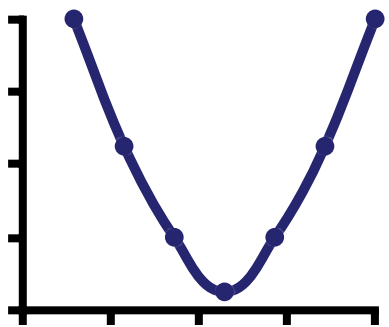
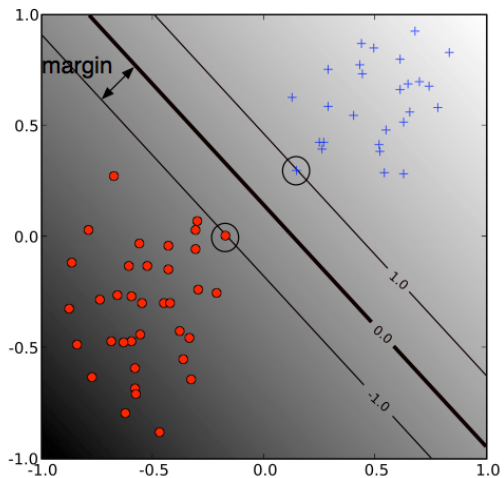


FIGURE 18-4. Two decision boundaries and their margins. Note that the vertical decision boundary has a wider margin than the other one. The arrows indicate the distance between the respective support vectors and the decision boundary.

source: Data Analysis with Open Source Tools, by Philipp K. Janert. O'Reilly Media, 2011.

The goal of an SVM is to create the linear decision boundary with the ***largest margin****. This is commonly called the* **maximum margin hyperplane (MMH)***.*





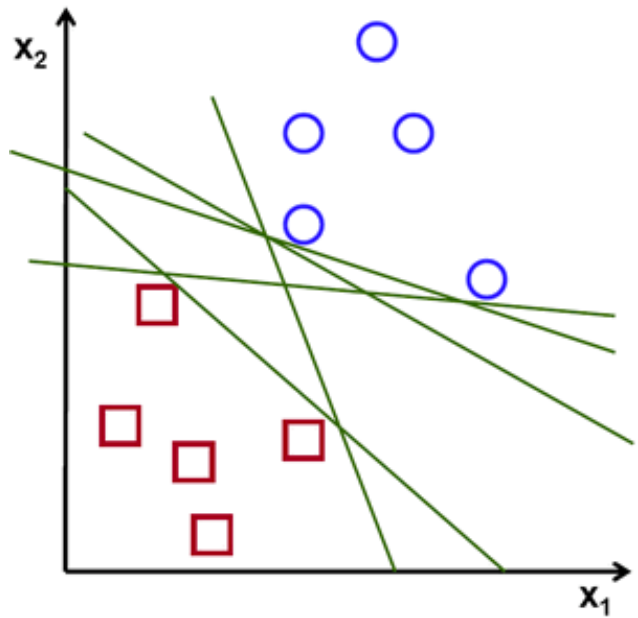
NOTE

A *hyperplane* is just a high-dimensional generalization of a line.

MAXIMUM MARGIN HYPERPLANES (MMH)

Q: How is the decision boundary (MMH) derived?

so many possible hyperplanes, which one to choose?



Q: How is the decision boundary (MMH) derived?
A: By the discriminant function,

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} + b.$$

such that w is the weight vector and b is the bias.

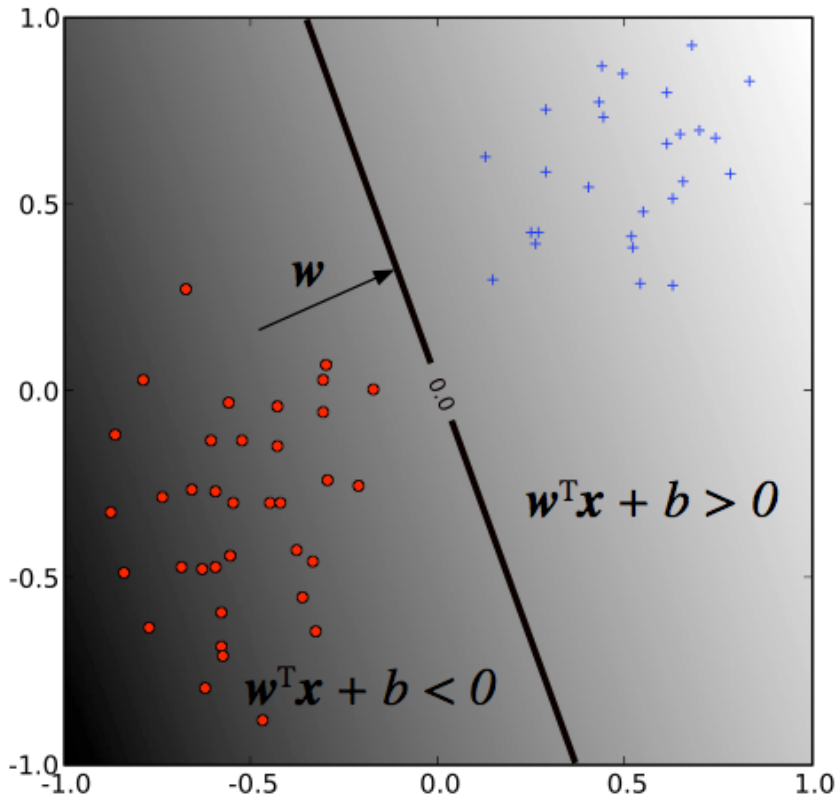
Q: How is the decision boundary (MMH) derived?
A: By the discriminant function,

$$f(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b.$$

such that w is the weight vector and b is the bias.

The sign of f(x) determines the (binary) class label of a record x.

MAXIMUM MARGIN HYPERPLANES



source: http://pyml.sourceforge.net/doc/howto.pdf

NOTE

The weight vector determines the *orientation* of the decision boundary.

The bias determines its *translation* from the origin.

As we said before, SVM solves for the decision boundary that minimizes generalization error, or equivalently, that has the maximum margin.

Q: Why are these the same thing?

As we said before, SVM solves for the decision boundary that minimizes generalization error, or equivalently, that has the maximum margin.

Q: Why are these the same thing?
A: Because using the MMH as the decision boundary minimizes the probability that a small perturbation in the position of a point produces a classification error.

MAXIMUM MARGIN HYPERPLANES

As we said before, SVM solves for the decision boundary that minimizes generalization error, or equivalently, that has the maximum margin.

Q: Why are these the same thing?
A: Because using the MMH as the decision bou
minimizes the probability that a small perturbation in the position of a point produces a classification error.

NOTE

Intuitively, the wider the margin, the clearer the distinction between classes.

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a convex *objective function.*

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a convex *objective function.*

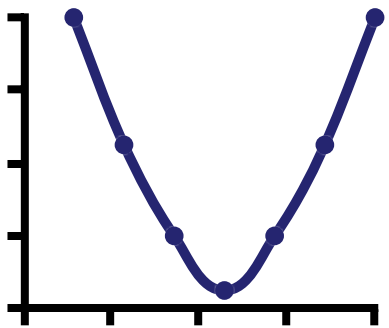
Remember what convex means in this context…?

MAXIMUM MARGIN HYPERPLANES

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a convex *objective function.*

Nice! Convex optimization problems are guaranteed to give global optima *(and they're easy to solve numerically too).*

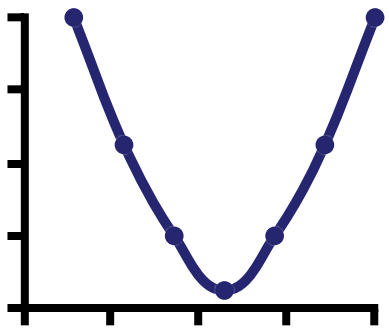


MAXIMUM MARGIN HYPERPLANES

Selecting the MMH is a straightforward exercise in analytic geometry (we won't go through the details here).

In particular, this task reduces to the optimization of a convex *objective function.*

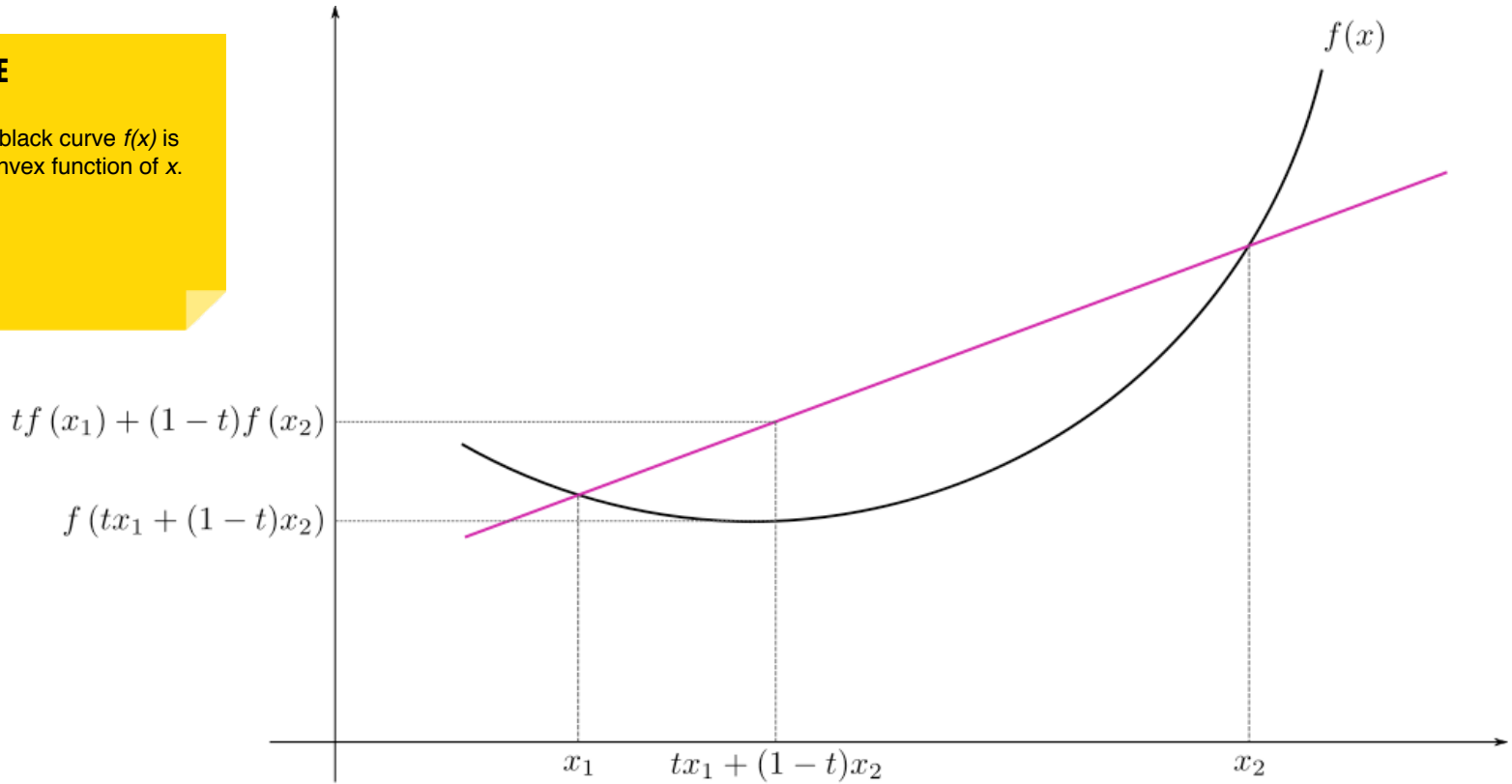
Nice! Convex optimization problems are guaranteed to give global optima *(and they're easy to solve numerically too).*



CONVEX FUNCTIONS

NOTE

The black curve *f(x)* is a convex function of *x*.

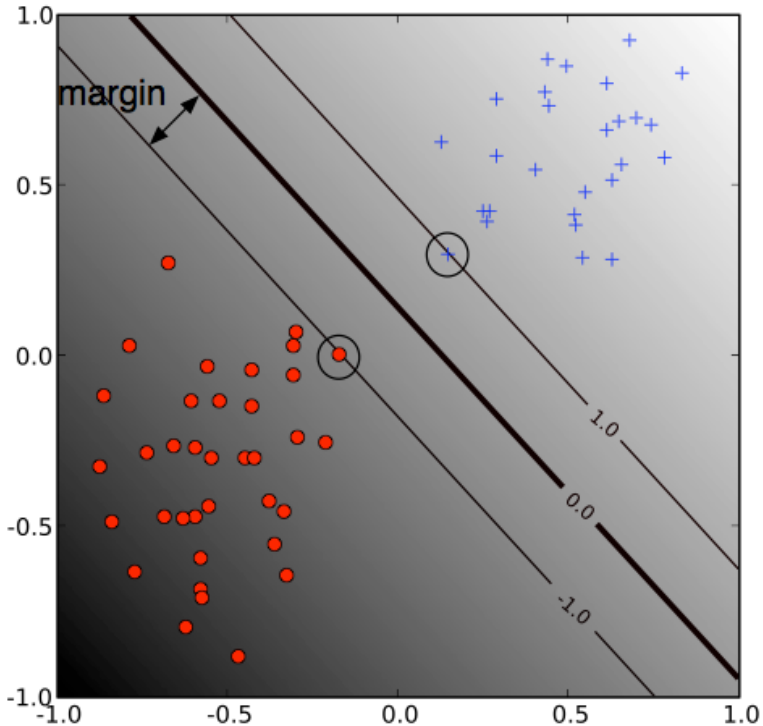


source: http://en.wikipedia.org/wiki/File:ConvexFunction.svg

MAXIMUM MARGIN HYPERPLANES

Notice that the margin depends only on the points that are nearest to the decision boundary.

These points are called the support vectors*.*



MAXIMUM MARGIN HYPERPLANES

All of the decision boundaries we've seen so far have split the data perfectly; eg, the data are (perfectly) linearly separable*, and therefore the training error is 0.*

The optimization problem that this SVM solves is:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{w}||^2 \\ \text{subject to:} \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 \quad i = 1, \ldots, n\,. \end{aligned}$$

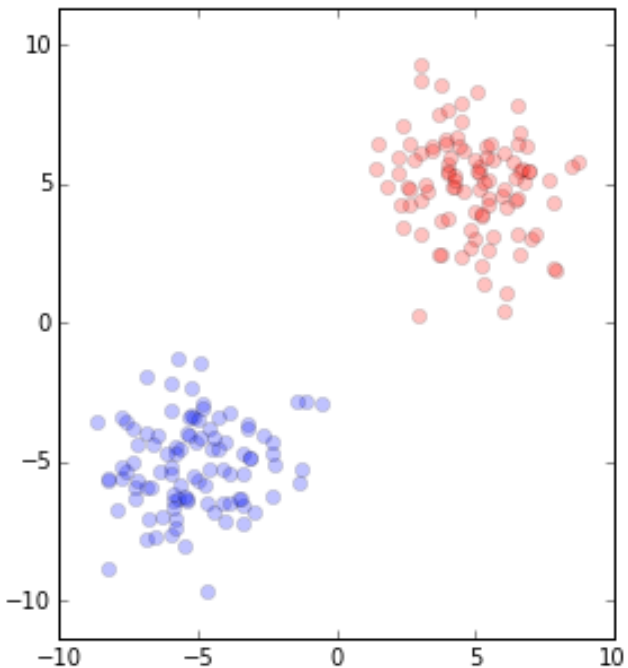
NOTE

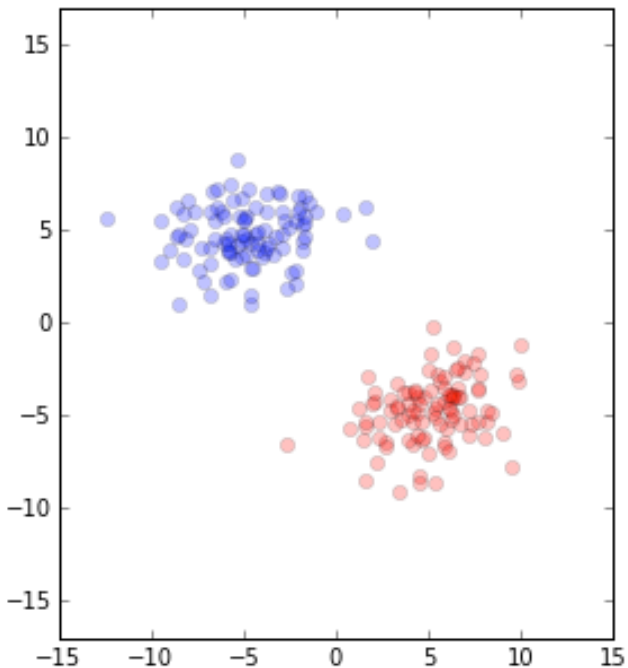
This type of optimization problem is called a *quadratic program.*

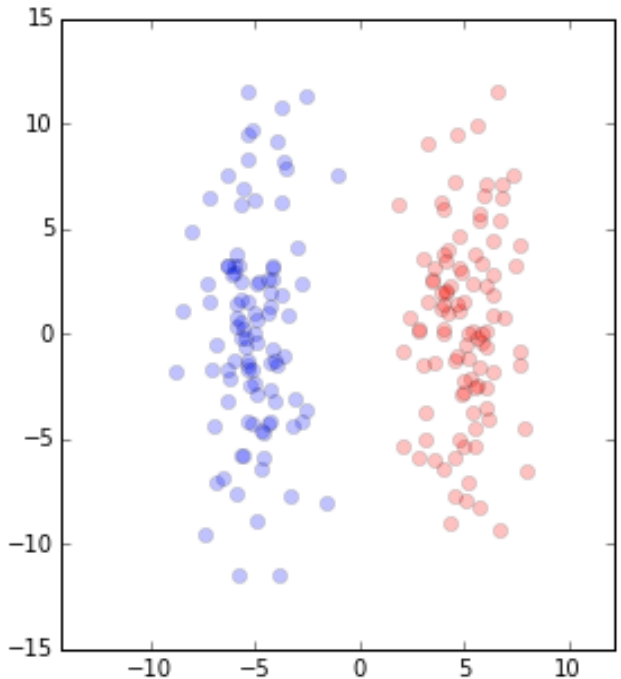
The result of this qp is the *hard margin classifier* we've been discussing.

MAXIMUM MARGIN HYPERPLANES

You do it: find the MMH

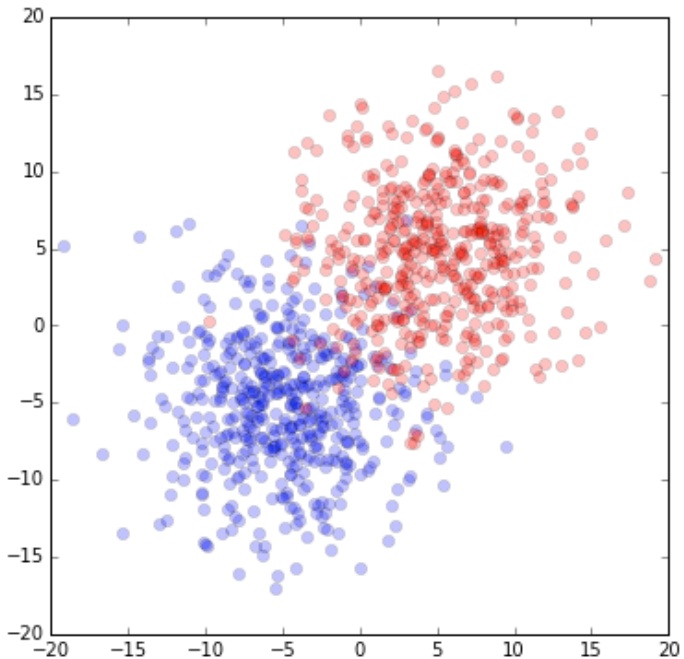






MAXIMUM MARGIN HYPERPLANES

What if data is not perfectly separable with a linear boundary?

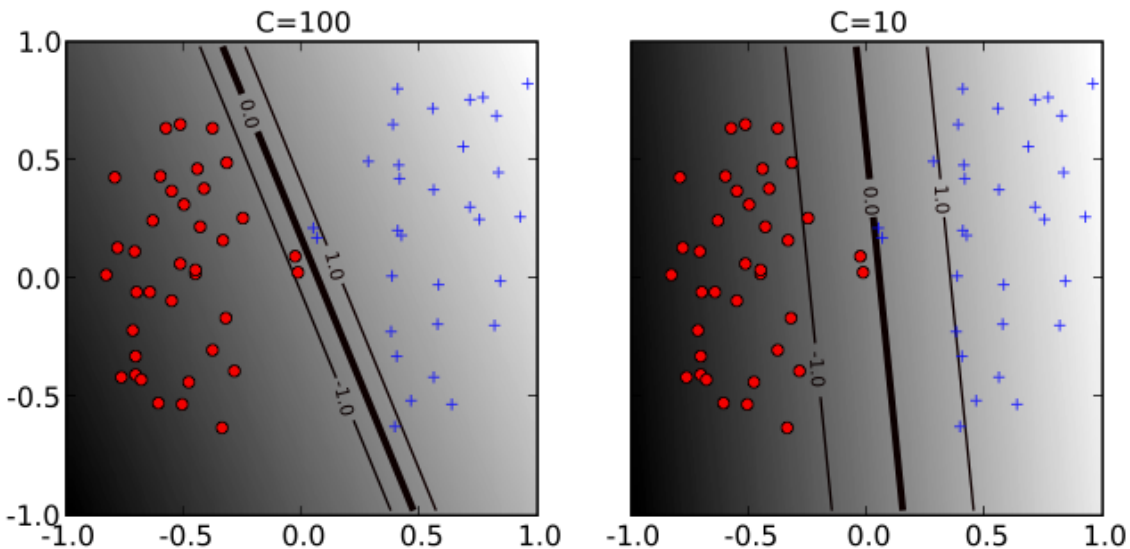


?

SLACK VARIABLES

SLACK VARIABLES

We can trade some training error in order to get a larger margin (remember the bias-variance trade-off)



source: http://pyml.sourceforge.net/doc/howto.pdf

SLACK VARIABLES

Slack variables ξ_i generalize the optimization problem to permit some misclassified training records (which come at a cost C).

The resulting soft margin classifier *is given by:*

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

(This is another example of bias-variance tradeoff)

SLACK VARIABLES

The soft-margin optimization problem can be rewritten as:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \textstyle\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \textstyle\sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

SLACK VARIABLES

The soft-margin optimization problem can be rewritten as:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

NOTE

This is called the *dual formulation* of the optimization problem.

(reached via Lagrange multipliers)

The soft-margin optimization problem can be rewritten as:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

Notice that this expression depends on the features x_i only via the inner product

$$\langle x_i, x_j \rangle = x_i^T x_j$$

The inner product is an operation that takes two vectors and returns a real number.

The fact that we can rewrite the optimization problem in terms of the inner product means that we don't actually have to do any calculations in the feature space K.

The inner product is an operation that takes two vectors and returns a real number.

The fact that we can rewrite the optimization problem in terms of the inner product means that we don't actually have to do any calculations in the feature space K.

This will have a very powerful consequence as we shall shortly see.

Quick recall exercise. Calculate these dot products:

$$v = \begin{bmatrix} 1 & 3 & 9 & 2 \end{bmatrix} \qquad v.w = ?$$

$$u = \begin{bmatrix} 2 & 4 & 6 & 7 \end{bmatrix} \qquad v.u = ?$$

$$w = \begin{bmatrix} 2 & 3 & 6 & 5 \end{bmatrix} \qquad w.u = ?$$

Quick recall exercise. Calculate these dot products:

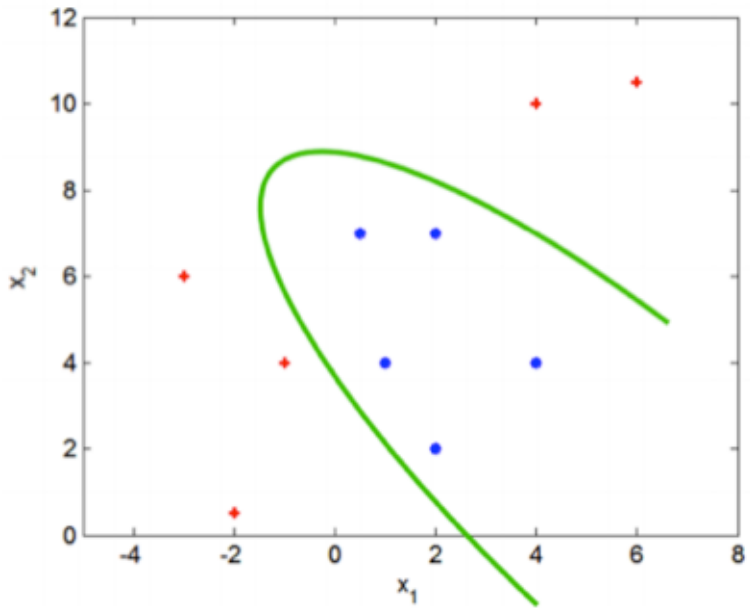
$$v = \begin{bmatrix} 1 & 3 & 9 & 2 \end{bmatrix} \qquad v.w = 75$$

$$u = \begin{bmatrix} 2 & 4 & 6 & 7 \end{bmatrix} \qquad v.u = 82$$

$$w = \begin{bmatrix} 2 & 3 & 6 & 5 \end{bmatrix} \qquad w.u = 87$$

NONLINEAR CLASSIFICATION

Suppose we need a more complex classifier than a linear decision boundary allows.



Suppose we need a more complex classifier than a linear decision boundary allows.

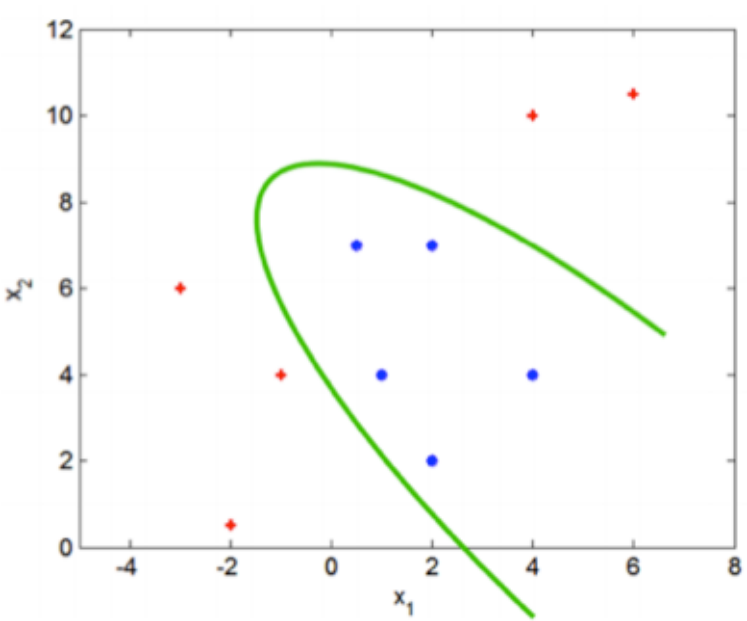
One possibility is to add nonlinear combinations of features to the data, and then to create a linear decision boundary in the enhanced (higher-dimensional) feature space.

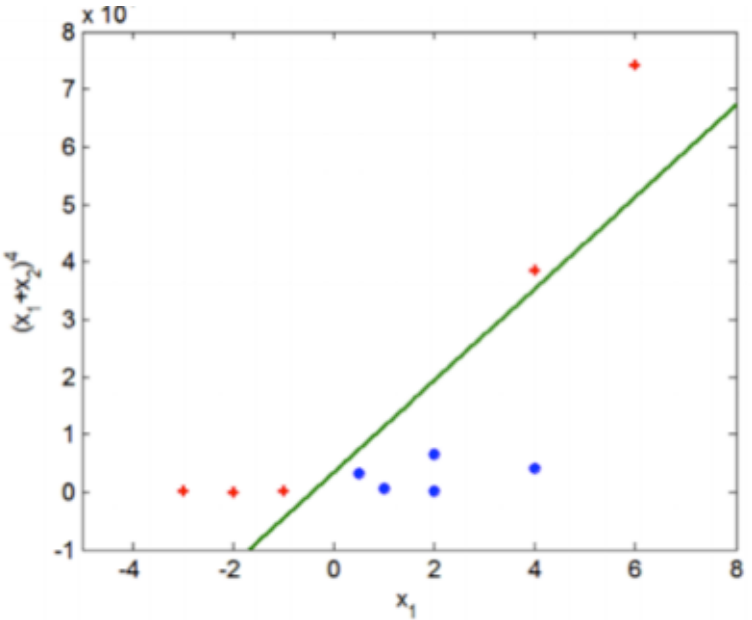
Suppose we need a more complex classifier than a linear decision boundary allows.

One possibility is to add nonlinear combinations of features to the data, and then to create a linear decision boundary in the enhanced (higher-dimensional) feature space.

This linear decision boundary will be mapped to a nonlinear decision boundary in the original feature space.

NONLINEAR CLASSIFICATION





original feature space *K*

higher-dim feature space *K'*

The logic of this approach is sound, but there are a few problems with this version.

The logic of this approach is sound, but there are a few problems with this version.

In particular, this will not scale well, since it requires many high-dimensional calculations.

The logic of this approach is sound, but there are a few problems with this version.

In particular, this will not scale well, since it requires many high-dimensional calculations.

It will likely lead to more complexity (both modeling complexity and computational complexity) than we want.

Let's hang on to the logic of the previous example, namely:

Let's hang on to the logic of the previous example, namely:
- remap the feature vectors x_i into a higher-dimensional space K'
- create a linear decision boundary in K'
- back out the nonlinear decision boundary in K from the result

Let's hang on to the logic of the previous example, namely:
- remap the feature vectors x_i into a higher-dimensional space K'
- create a linear decision boundary in K'
- back out the nonlinear decision boundary in K from the result

But we want to save ourselves the trouble of doing a lot of additional high-dimensional calculations. How can we do this?

Recall that our optimization problem depends on the features only through the inner product $x^T x$:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

Recall that our optimization problem depends on the features only through the inner product $x^T x$:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

We can replace this inner product with a more general function that has the same type of output as the inner product.

Formally, we can think of the inner product as a map that sends two vectors in the feature space K into the real line $\mathbb{R}$

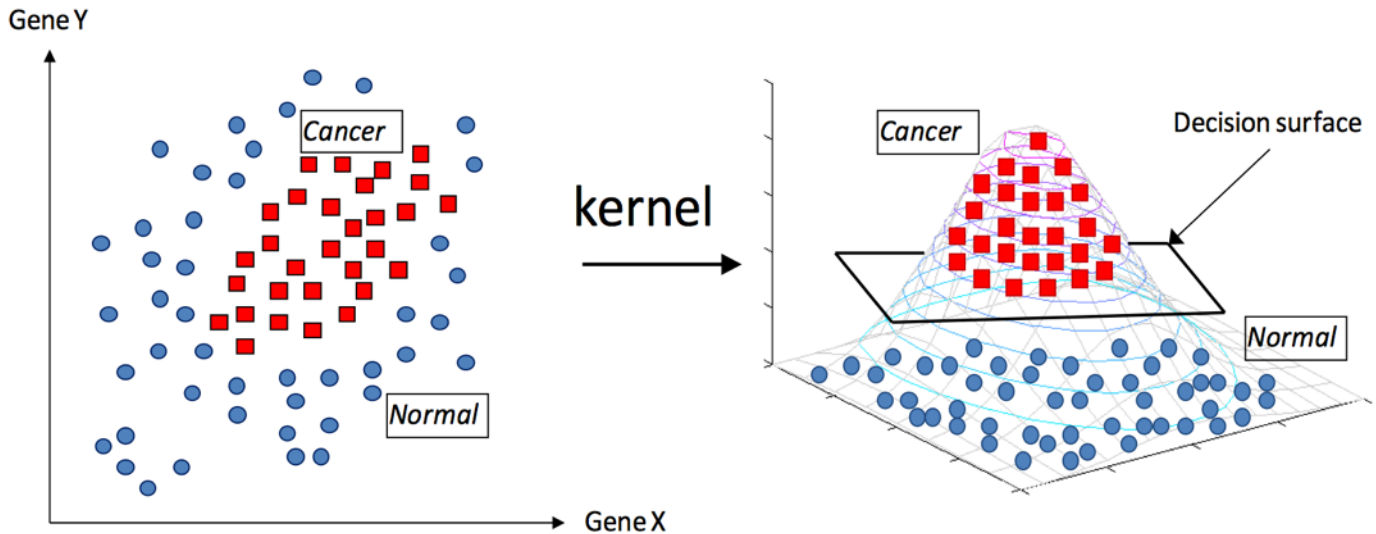
Formally, we can think of the inner product as a map that sends two vectors in the feature space K into the real line $\mathbb{R}$

We can replace this with a generalization of the inner product called a **kernel function** *that maps two vectors in a higher-dimensional feature space K' into* $\mathbb{R}$

Nonlinear *applications of SVM rely on an implicit mapping* Φ *that sends vectors from the original feature space K into a higher-dimensional feature space K'. This is called mapping is called* ***kernel.***

Nonlinear classification in K is then obtained by creating a linear decision boundary in K'.

THE KERNEL TRICK



If a linear decision boundary cannot be found in the original space, we can map into a higher dimensional space and find the separating surface.

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

As long as the kernel function satisfies certain conditions, our conclusions above regarding the MMH continue to hold.

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

As long as the kernel function satisfies certain conditions, our conclusions above regarding the MMH continue to hold.

NOTE

These conditions are contained in a result called *Mercer's theorem*.

The upshot is that we can use a kernel function to implicitly train our model in a higher-dimensional feature space, without incurring additional computational complexity!

As long as the kernel function satisfies certain conditions, our conclusions above regarding the MMH continue to hold.

In other words, no algorithmic changes are necessary, and all the benefits of a linear SVM are maintained.

some popular kernels:

linear kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$

polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + 1)^d$

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x}-\mathbf{x}'||^2)$

some popular kernels:

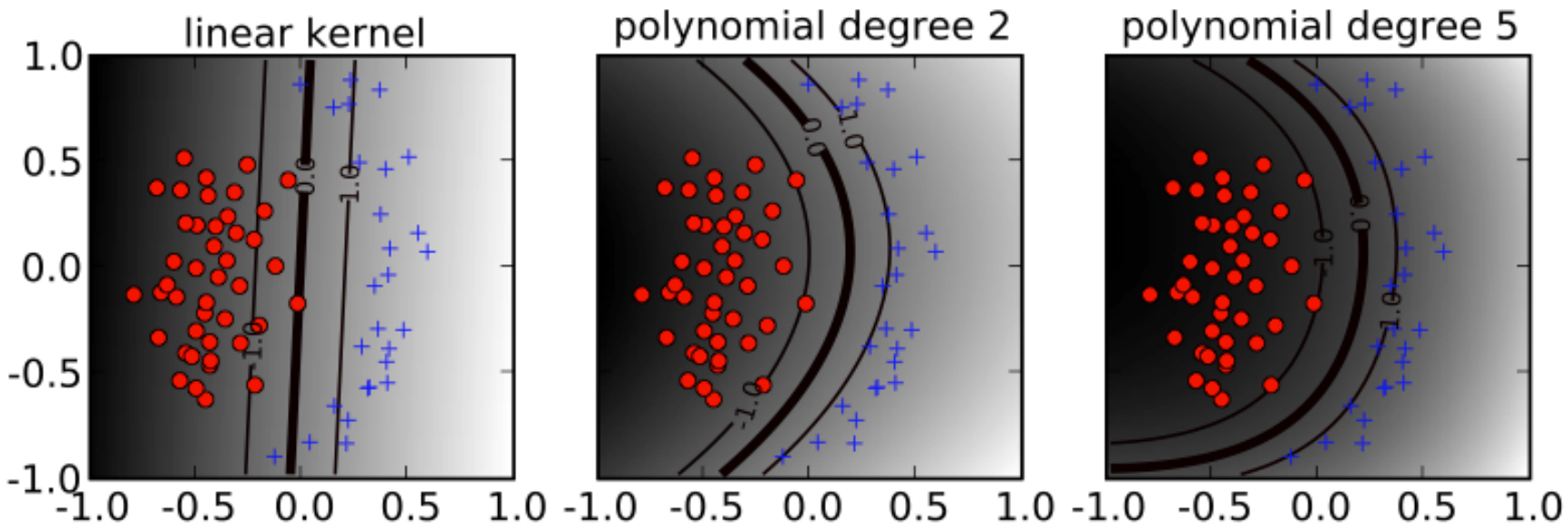
linear kernel $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$

polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + 1)^d$

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x}-\mathbf{x}'||^2)$

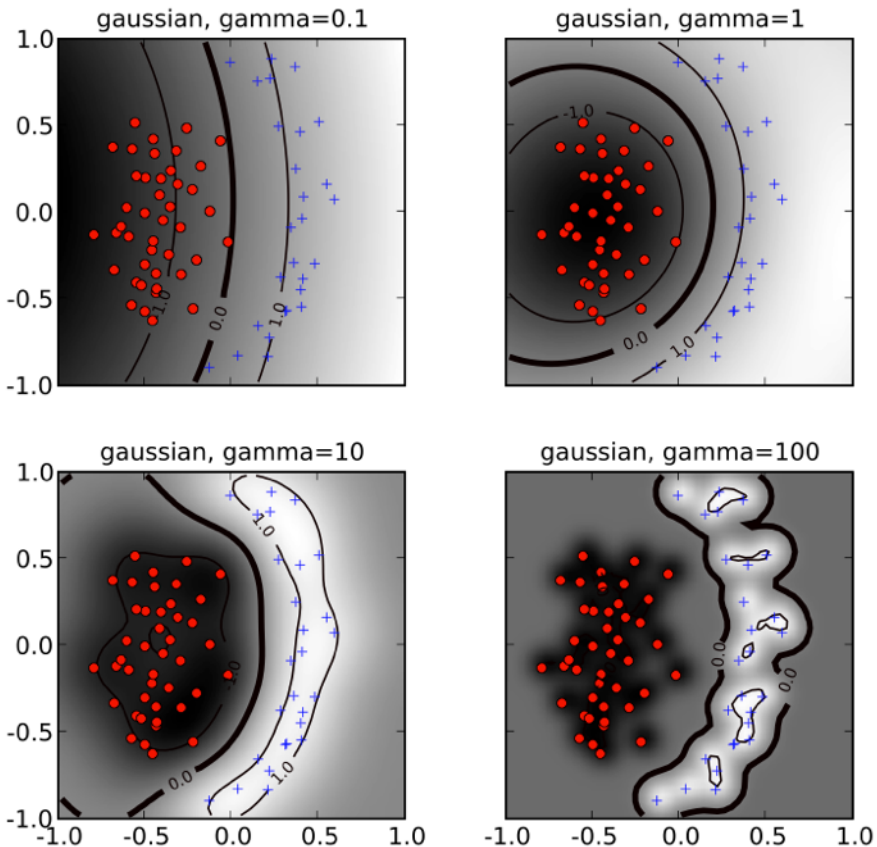
The hyperparameters *d,* γ *affect the flexibility of the decision bdy.*

NONLINEAR CLASSIFICATION – POLYNOMIAL KERNEL



source: http://pyml.sourceforge.net/doc/howto.pdf

NONLINEAR CLASSIFICATION – GAUSSIAN KERNEL



source: http://pyml.sourceforge.net/doc/howto.pdf

SVM STRENGTHS & WEAKNESSES

SVMs (and kernel methods *in general) are versatile, powerful, and popular techniques that can produce accurate results for a wide array of classification problems.*

The main disadvantage of SVMs is the lack of intuition they produce. These models are truly black boxes!

Quick exercise:

discuss with the person next to you what kernels are and how they can be used

LAB SVM

DECISION TREES

Decision trees

non-parametric hierarchical classification technique.

- **non-parametric**: no parameters, no distribution assumptions

- **hierarchical**: consists of a sequence of questions which yield a class label when applied to any record

Decision trees

non-parametric hierarchical classification technique.

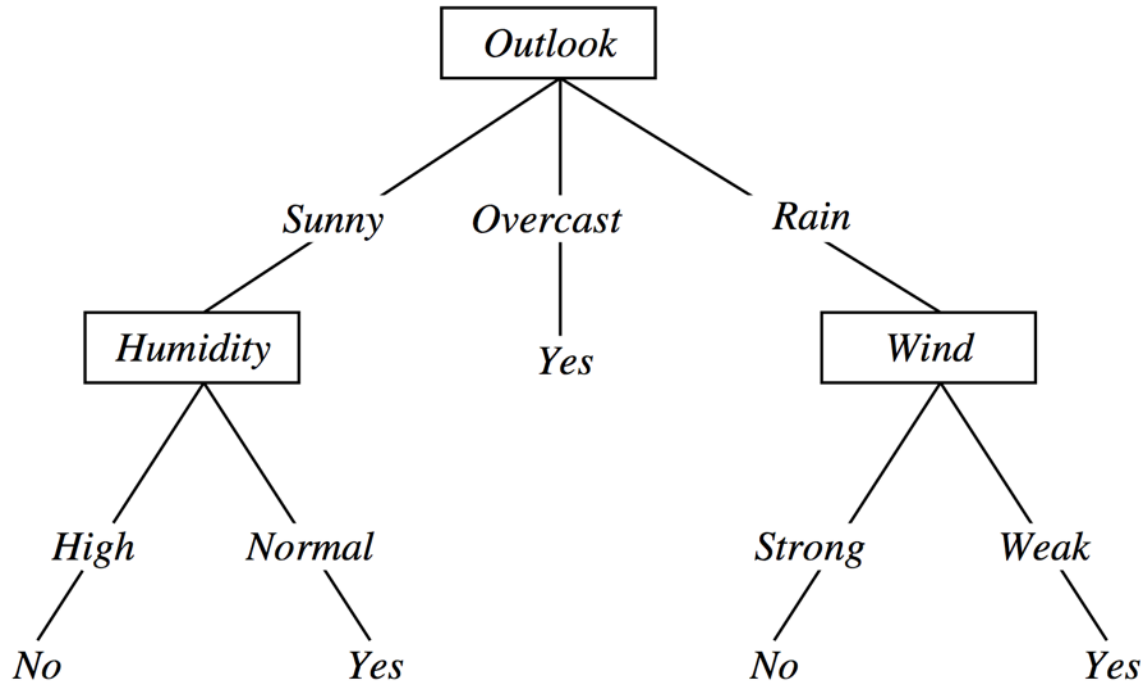
- **non-parametric**: no parameters, no distribution assumptions

- **hierarchical**: consists of a sequence of questions which yield a class label when applied to any record

represented as a **multiway tree**, which is a type of (directed acyclic) graph

Nodes => questions

Edges => answers



Classify an instance: <outlook=Sunny, temp = Hot, humidity=High, wind = Strong>

Top node => root node
0 incoming edges, and 2+ outgoing edges

Internal node => test condition
1 incoming edge, and 2+ outgoing edges

Leaf node => class label
has 1 incoming edge and, 0 outgoing edges

Table 4.1. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

source: http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf

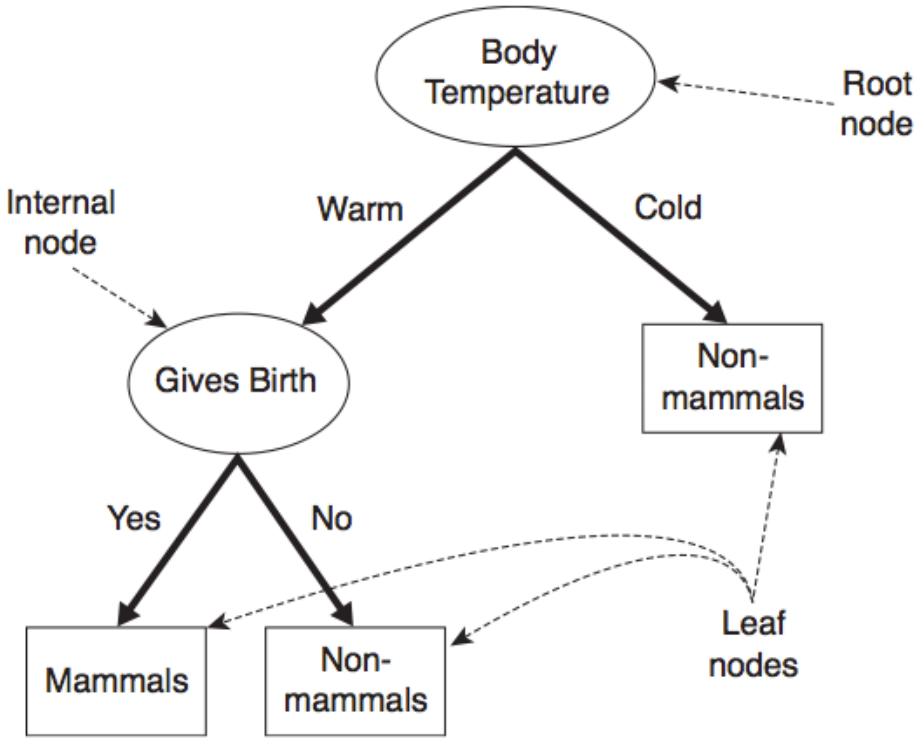


Figure 4.4. A decision tree for the mammal classification problem.

source: http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf

Quick exercise:

what's a decision tree? (3 words)

BUILDING DECISION TREES

How would you build a decision tree?

Hunt's algorithm
greedy recursive algorithm that leads to a local optimum

greedy
algorithm makes locally optimal decision at each step

recursive
splits task into subtasks, solves each the same way

local optimum
solution for a given neighborhood of points

Recursively partition records into smaller & smaller subsets.

The partitioning decision is made at each node according to a **metric** called **purity**

100% pure when all of its records belong to a single class

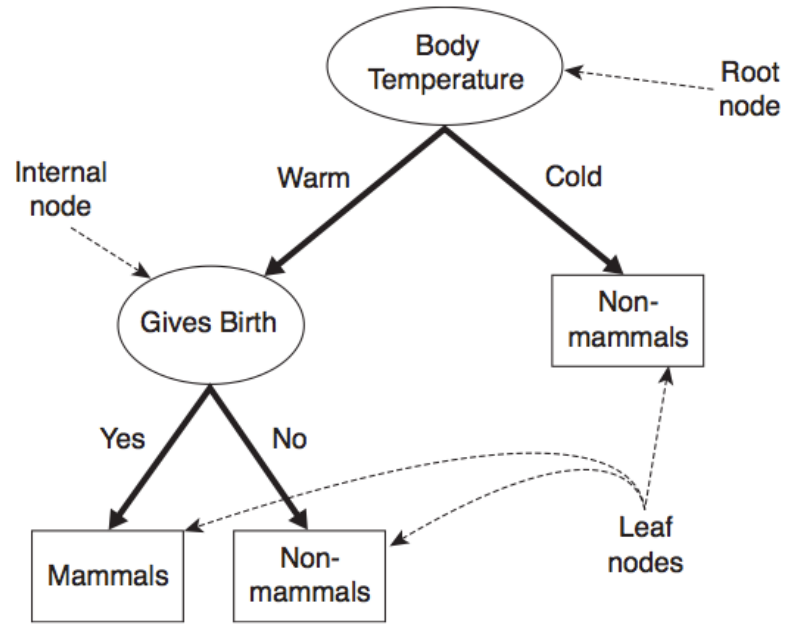


Figure 4.4. A decision tree for the mammal classification problem.

Example: binary classification with classes A, B. Given a set of records D_t at node t:

Example: binary classification with classes A, B. Given a set of records D_t at node t:

I) If all records in D_t belong to class A, then t is a **leaf node** corresponding to class *(Base case)*

Example: binary classification with classes A, B. Given a set of records D_t at node t:

1) If all records in D_t belong to class A, then t is a **leaf node** corresponding to class *(Base case)*

2) If D_t contains records from both A and B do the following:
   i. create test condition to partition the records further
   ii. t is an internal node, with outgoing edges to child nodes
   iii. partition records in D_t to children according to test

Example: binary classification with classes A, B. Given a set of records D_t at node t:

1) If all records in D_t belong to class A, then t is a **leaf node** corresponding to class *(Base case)*

2) If D_t contains records from both A and B do the following:
   i. create test condition to partition the records further
   ii. t is an internal node, with outgoing edges to child nodes
   iii. partition records in D_t to children according to test

3) These steps are then recursively applied to each child node.

Binary splits

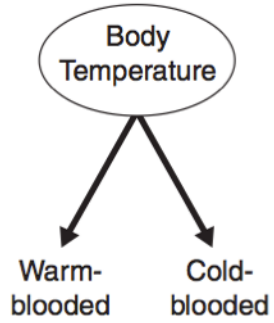
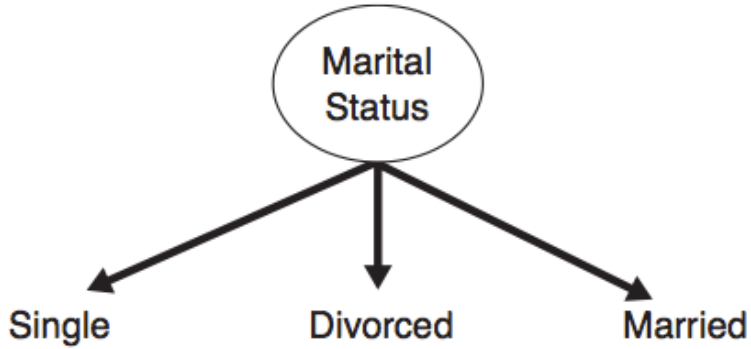


Figure 4.8. Test condition for binary attributes.

Multiway splits



(a) Multiway split

Continous features

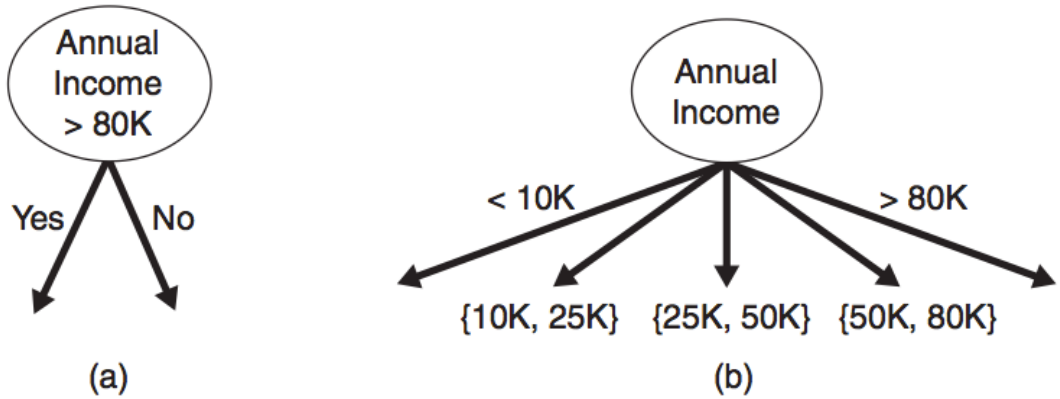


Figure 4.11. Test condition for continuous attributes.

Quick check: find rules to split the data below

Completed	*Time*	*Result*
3	*30*	*Pass*
2	*25*	*Pass*
4	*49*	*Pass*
3	*47*	*Fail*
2	*50*	*Fail*
1	*32*	*Fail*
3	*26*	*Pass*

OPTIMIZATION FUNCTIONS

Q: How do we determine the best split?
A: Recall that no split is necessary (at a given node) when all records belong to the same class.

Therefore we want each step to create the partition with the ***highest possible purity.***

We need an objective function to optimize!

We want our objective function to measure the gain in purity from a particular split.

Table 4.1. The vertebrate data set.

Name	Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
human	warm-blooded	hair	yes	no	no	yes	no	mammal
python	cold-blooded	scales	no	no	no	no	yes	reptile
salmon	cold-blooded	scales	no	yes	no	no	no	fish
whale	warm-blooded	hair	yes	yes	no	no	no	mammal
frog	cold-blooded	none	no	semi	no	yes	yes	amphibian
komodo dragon	cold-blooded	scales	no	no	no	yes	no	reptile
bat	warm-blooded	hair	yes	no	yes	yes	yes	mammal
pigeon	warm-blooded	feathers	no	no	yes	yes	no	bird
cat	warm-blooded	fur	yes	no	no	yes	no	mammal
leopard shark	cold-blooded	scales	yes	yes	no	no	no	fish
turtle	cold-blooded	scales	no	semi	no	yes	no	reptile
penguin	warm-blooded	feathers	no	semi	no	yes	no	bird
porcupine	warm-blooded	quills	yes	no	no	yes	yes	mammal
eel	cold-blooded	scales	no	yes	no	no	no	fish
salamander	cold-blooded	none	no	semi	no	yes	yes	amphibian

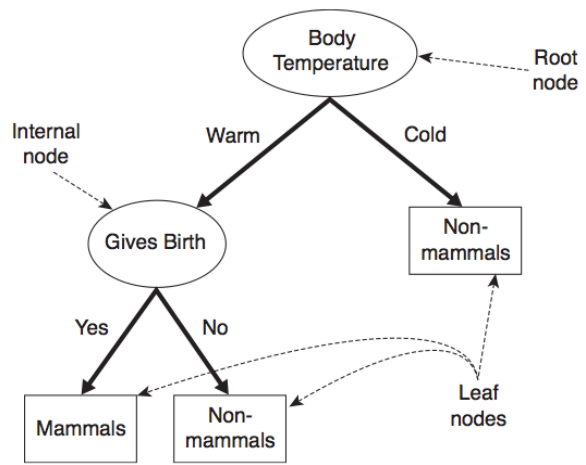


Figure 4.4. A decision tree for the mammal classification problem.

We want our objective function to measure the gain in purity from a particular split.

Therefore we want it to depend on the class distribution over the nodes (before and after the split).

We want our objective function to measure the gain in purity from a particular split.

Therefore we want it to depend on the class distribution over the nodes (before and after the split).

For example, let $p(i|t)$ *be the probability of class* i *at node* t *(eg, the fraction of records labeled* i *at node* t*).*

Then for a binary (0/1) classification problem,

Then for a binary (0/1) classification problem,

The minimum purity partition is given by the distribution:

$$p(0|t) = p(1|t) = 0.5$$

Then for a binary (0/1) classification problem,

The minimum purity partition is given by the distribution:

$$p(0|t) = p(1|t) = 0.5$$

The maximum purity partition is given (eg) by the distribution:

$$p(0|t) = 1 - p(1|t) = 1$$

how to measure the value of information?

*Some measures of **impurity** include:*

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t),$$

$$\text{Gini}(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2,$$

$$\text{Classification error}(t) = 1 - \max_i [p(i|t)],$$

$$\text{Entropy}(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)$$

Use entropy as a measure of impurity or disorder of the data set

$$\text{Entropy}(t) \quad = \quad -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)$$

1. The data set D has 50% positive examples (Pr(*positive*) = 0.5) and 50% negative examples (Pr(*negative*) = 0.5).

2. The data set D has 20% positive examples (Pr(*positive*) = 0.2) and 80% negative examples (Pr(*negative*) = 0.8).

3. The data set D has 100% positive examples (Pr(*positive*) = 1) and no negative examples, (Pr(*negative*) = 0).

source: http://www.cs.uic.edu/~liub/teach/cs583-spring-12/CS583-supervised-learning.ppt

1. The data set D has 50% positive examples (Pr(*positive*) = 0.5) and 50% negative examples (Pr(*negative*) = 0.5).

$$entropy(D) = -0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 1$$

2. The data set D has 20% positive examples (Pr(*positive*) = 0.2) and 80% negative examples (Pr(*negative*) = 0.8).

$$entropy(D) = -0.2 \times \log_2 0.2 - 0.8 \times \log_2 0.8 = 0.722$$

3. The data set D has 100% positive examples (Pr(*positive*) = 1) and no negative examples, (Pr(*negative*) = 0).

$$entropy(D) = -1 \times \log_2 1 - 0 \times \log_2 0 = 0$$

As the data become purer and purer, the entropy value becomes smaller and smaller.

Note that each measure achieves its max at 0.5, min at 0 & 1.

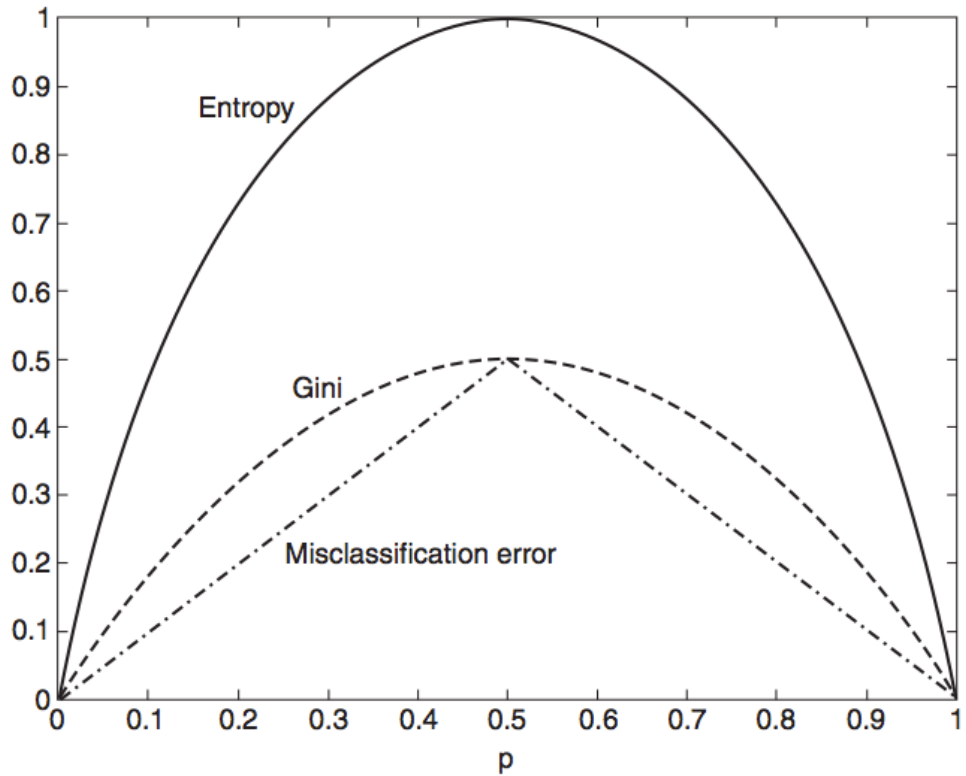


Figure 4.13. Comparison among the impurity measures for binary classification problems.

Note that each measure achieves its max at 0.5, min at 0 & 1.

NOTE

Despite consistency, different measures may create different splits.

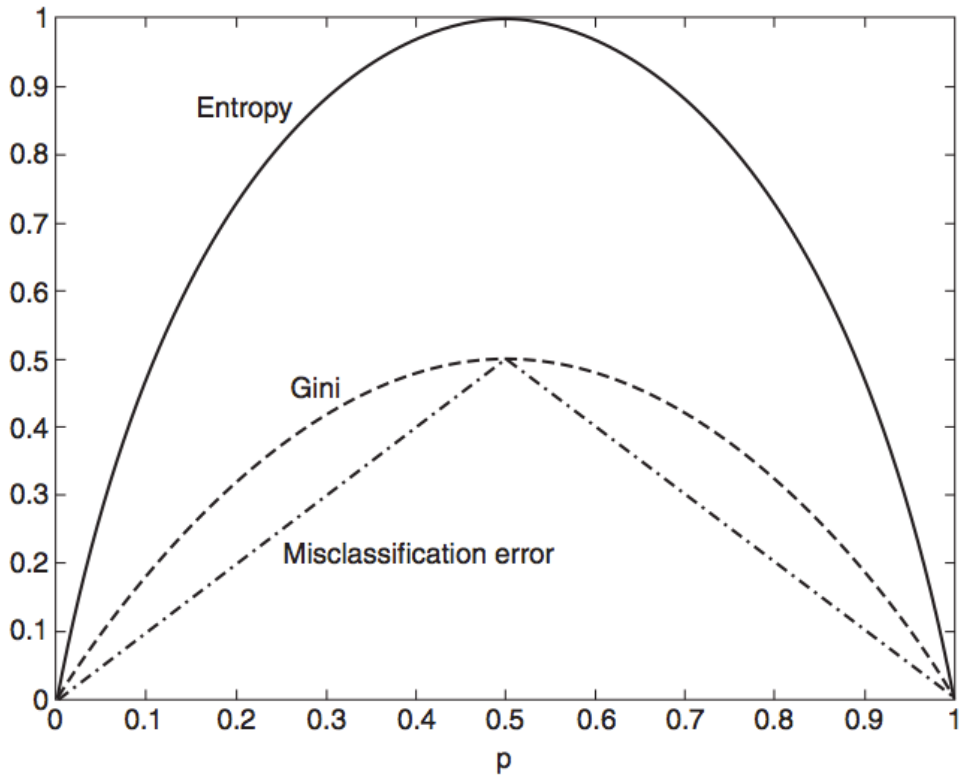


Figure 4.13. Comparison among the impurity measures for binary classification problems.

Impurity measures put us on the right track, but on their own they are not enough to tell us how our split will do.

Impurity measures put us on the right track, but on their own they are not enough to tell us how our split will do.

Q: Why is this true?

Impurity measures put us on the right track, but on their own they are not enough to tell us how our split will do.

Q: Why is this true?
A: We still need to look at impurity before & after the split.

We can make this comparison using the gain*:*

$$\Delta = I(\text{parent}) - \sum_{\text{children } j} \frac{N_j}{N} I(\text{child } j)$$

We can make this comparison using the gain*:*

$$\Delta = I(\text{parent}) - \sum_{\text{children } j} \frac{N_j}{N} I(\text{child } j)$$

(Here I *is the impurity measure,* N_j *denotes the number of records at child node* j*, and* N *denotes the number of records at the parent node.)*

We can make this comparison using the gain*:*

$$\Delta = I(\text{parent}) - \sum_{\text{children } j} \frac{N_j}{N} I(\text{child } j)$$

(Here I *is the impurity measure,* N_j *denotes the number of records at child node* j*, and* N *denotes the number of records at the parent node.)*

When I is the entropy, this quantity is called the ***information gain****.*

Generally speaking, a test condition with a high number of outcomes can lead to overfitting (ex: a split with one outcome per record).

Generally speaking, a test condition with a high number of outcomes can lead to overfitting (ex: a split with one outcome per record).

One way of dealing with this is to restrict the algorithm to binary splits only (CART).

Generally speaking, a test condition with a high number of outcomes can lead to overfitting (ex: a split with one outcome per record).

One way of dealing with this is to restrict the algorithm to binary splits only (CART).

Another way is to use a splitting criterion which explicitly penalizes the number of outcomes (C4.5)

We can use a function of the information gain called the gain ratio *to explicitly penalize high numbers of outcomes:*

$$\text{gain ratio} = \frac{\Delta_{info}}{-\sum p(v_i) log_2 p(v_i)}$$

(Where $\mathrm{p}(\mathrm{v}_\mathrm{i})$ *refers to the probability of label* i *at node* v*)*

We can use a function of the information gain called the gain ratio *to explicitly penalize high numbers of outcomes:*

$$\text{gain ratio} = \frac{\Delta_{info}}{-\sum p(v_i) log_2 p(v_i)}$$

NOTE

This is a form of regularization!

(Where $\mathrm{p}(\mathrm{v}_\mathrm{i})$ *refers to the probability of label* i *at node* v*)*

Quick check: what is information gain?

PREVENTING OVERFITTING

In addition to determining splits, we also need a stopping criterion to tell us when we're done.

In addition to determining splits, we also need a stopping criterion to tell us when we're done.

For example, we can stop when all records belong to the same class, or when all records have the same attributes.

In addition to determining splits, we also need a stopping criterion to tell us when we're done.

For example, we can stop when all records belong to the same class, or when all records have the same attributes.

This is correct in principle, but would likely lead to overfitting.

One possibility is pre-pruning*, which involves setting a minimum threshold on the gain, and stopping when no split achieves a gain above this threshold.*

One possibility is pre-pruning*, which involves setting a minimum threshold on the gain, and stopping when no split achieves a gain above this threshold.*

This prevents overfitting, but is difficult to calibrate in practice (may preserve bias!)

Alternatively we could build the full tree, and then perform pruning *as a post-processing step.*

Alternatively we could build the full tree, and then perform pruning *as a post-processing step.*

To prune a tree, we examine the nodes from the bottom-up and simplify pieces of the tree (according to some criteria).

Complicated subtrees can be replaced either with a single node, or with a simpler (child) subtree.

Complicated subtrees can be replaced either with a single node, or with a simpler (child) subtree.

The first approach is called subtree replacement*, and the second is* subtree raising*.*

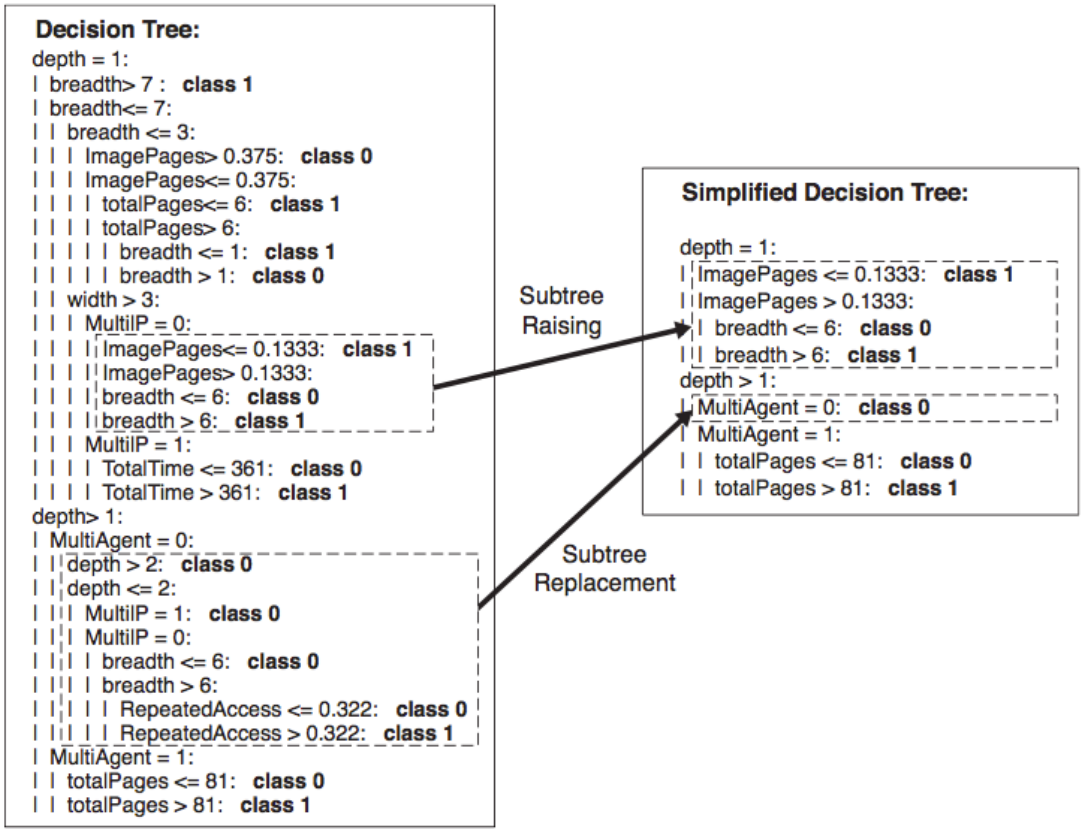


Figure 4.29. Post-pruning of the decision tree for Web robot detection.

LAB: DECISION TREES