Neural Networks

Lecture 23:
Linear Support Vector Machines

Getting good generalization on big datasets

- If we have a big data set that needs a complicated model, the full Bayesian framework is very computationally expensive.

- Is there a frequentist method that is faster but still generalizes well?

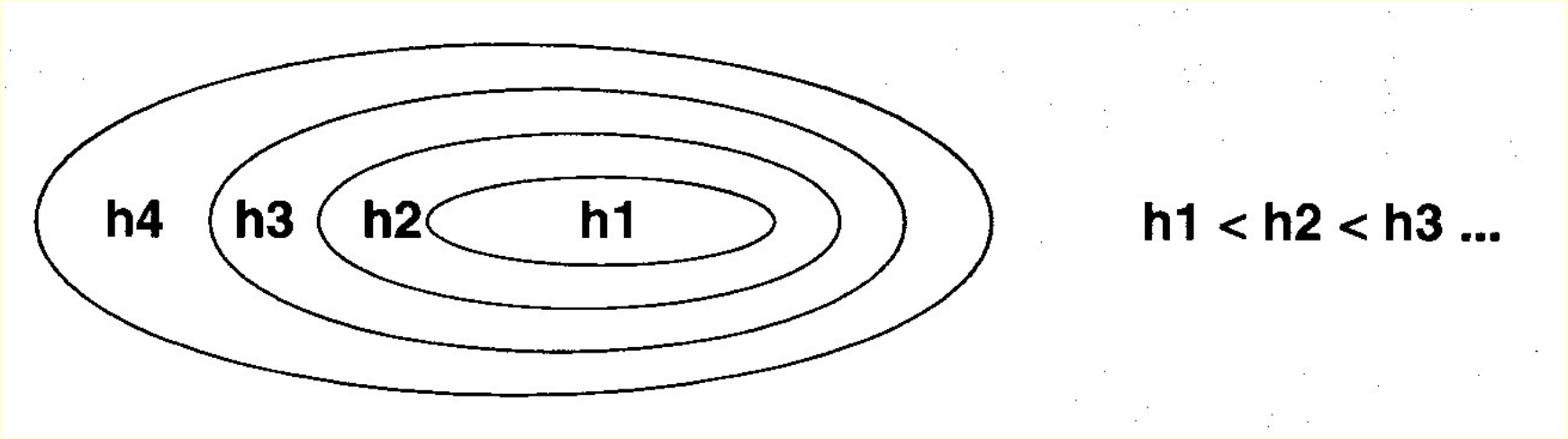
Preprocessing the input vectors

- Instead of trying to predict the answer directly from the raw inputs we could start by extracting a layer of "features".
  - Sensible if we already know that certain combinations of input values would be useful (e.g. edges or corners in an image).
- Instead of learning the features we could design them by hand.
  - The hand-coded features are equivalent to a layer of non-linear neurons that do not need to be learned.
  - If we use a very big set of features for a two-class problem, the classes will almost certainly be linearly separable.
    - But surely the linear separator will give poor generalization.

Is preprocessing cheating?

- Its cheating if we use a carefully designed set of task-specific, hand-coded features and then claim that the learning algorithm solved the whole problem.
  - The really hard bit is done by designing the features.
- Its not cheating if we learn the non-linear preprocessing.
  - This makes learning much more difficult and much more interesting (e.g. backpropagation)
- Its not cheating if we use a very big set of non-linear features that is task-independent.
  - Support Vector Machines do this.
  - They have a clever way to prevent overfitting
  - They have a clever way to use a huge number of features without requiring nearly as much computation as seems to be necessary

A hierarchy of model classes



- Some model classes can be arranged in a hierarchy of increasing complexity.
- How do we pick the best level in the hierarchy for modeling a given dataset?

A way to choose a model class

- We want to get a low error rate on unseen data.
  - This is called “structural risk minimization”

- It would be really helpful if we could get a guarantee of the following form:

  Test error rate =< train error rate + f(N, h, p)

  Where N = size of training set,

  h = measure of the model complexity,

  p = the probability that this bound fails

  We need p to allow for really unlucky test sets.

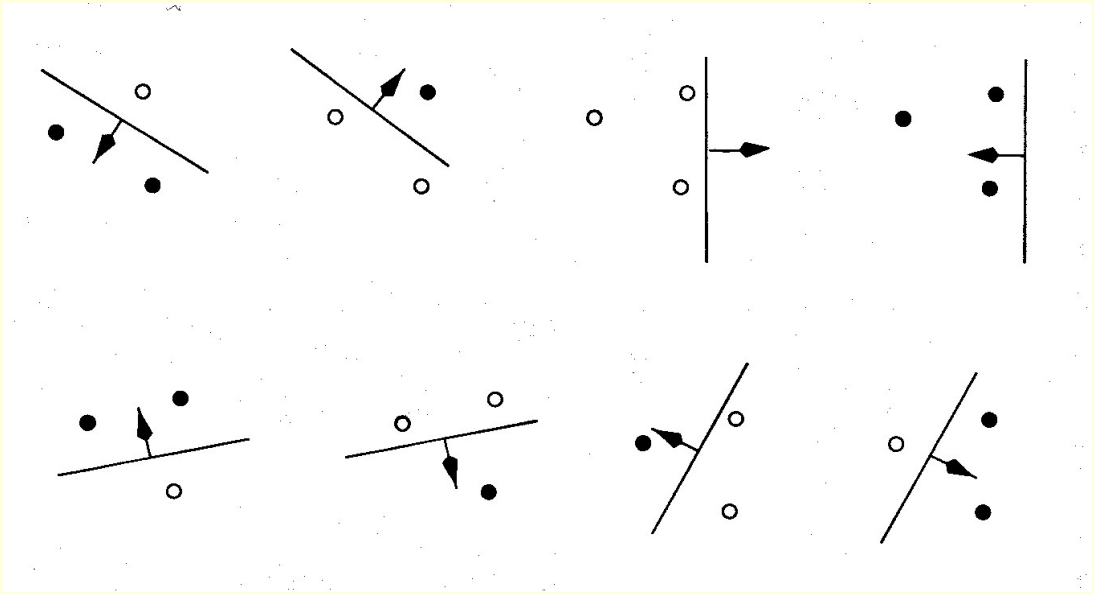
- Then we could choose the model complexity that minimizes the bound on the test error rate.

A weird measure of model complexity

- Suppose that we pick n datapoints and assign labels of + or – to them at random. If our model class (e.g. a neural net with a certain number of hidden units) is powerful enough to learn any association of labels with data, its too powerful!
- Maybe we can characterize the power of a model class by asking how many datapoints it can learn perfectly for all possible assignments of labels.
  - This number of datapoints is called the Vapnik-Chervonenkis dimension.
  - Creationism has infinite VC dimension.

An example of VC dimension

- Suppose our model class is a hyperplane.
- In 2-D, we can find a plane (i.e. a line) to deal with any labeling of three points. A 2-D hyperplane shatters 3 points

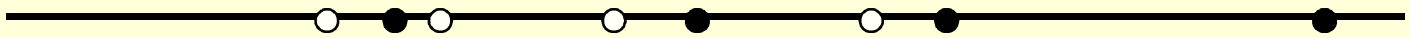


- But we cannot deal with some of the possible labelings of four points. A 2-D hyperplane line does not shatter 4 points.

Some examples of VC dimension

- The VC dimension of a hyperplane in 2-D is 3.
  - In k dimensions it is k+1.
- Its just a coincidence that the VC dimension of a hyperplane is almost identical to the number of parameters it takes to define a hyperplane.
- A sine wave has infinite VC dimension and only 2 parameters! By choosing the phase and period carefully we can shatter any random collection of one-dimensional datapoints (except for nasty special cases).

$$f(x) = a \sin(b\ x)$$



The probabilistic guarantee

$$E_{test} \leq E_{train} + \left(\frac{h + h\log(2N/h) - \log(p/4)}{N} \right)^{\frac{1}{2}}$$

where N = size of training set

h = VC dimension of the model class

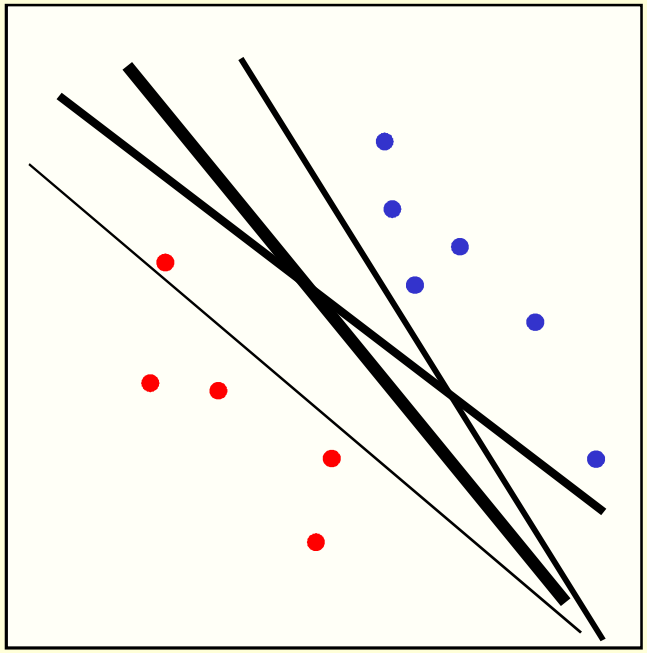
p = upper bound on probability that this bound fails

So if we train models with different complexity, we should pick the one that minimizes this bound

Actually, this is only sensible if we think the bound is fairly tight, which it usually isn't. The theory provides insight, but in practice we still need some witchcraft.

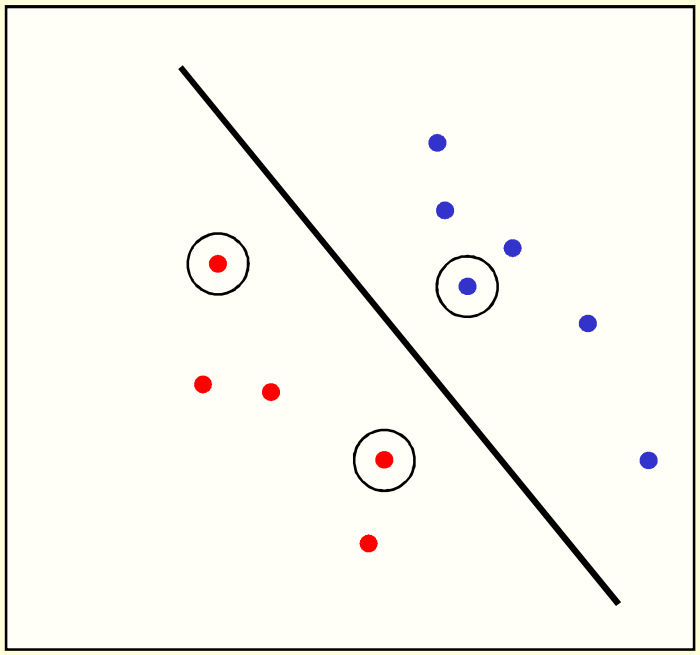
Preventing overfitting when using big sets of features

- Suppose we use a big set of features to ensure that the two classes are linearly separable. What is the best separating line to use?
- The Bayesian answer is to use them all (including ones that do not quite separate the data.)
- Weight each line by its posterior probability (i.e. by a combination of how well it fits the data and how well it fits the prior).
- Is there an efficient way to approximate the correct Bayesian answer?



Support Vector Machines

- The line that maximizes the minimum margin is a good bet.
  - The model class of “hyper-planes with a margin of m” has a low VC dimension if m is big.
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called “support vectors”.
  - It will be useful computationally if only a small fraction of the datapoints are support vectors.



The support vectors are indicated by the circles around them.

Training a linear SVM

- To find the maximum margin separator, we have to solve the following optimization problem:

$$\mathbf{w}.\mathbf{x}^c + b > +1 \quad \textit{for positive cases}$$

$$\mathbf{w}.\mathbf{x}^c + b < -1 \quad \textit{for negative cases}$$

$$\textit{and} \quad \|\mathbf{w}\|^2 \ \textit{is as small as possible}$$

- This is tricky but it's a convex problem. There is only one optimum and we can find it without fiddling with learning rates or weight decay or early stopping.
  - Don't worry about the optimization problem. It has been solved. Its called quadratic programming.
  - It takes time proportional to N^2 which is really bad for very big datasets.

Testing a linear SVM

- The separator is defined as the set of points for which: $\mathbf{w}.\mathbf{x} + b = 0$

$$so\ if\quad \mathbf{w}.\mathbf{x}^c + b > 0\quad say\ its\ a\ positive\ case$$

$$and\ if\quad \mathbf{w}.\mathbf{x}^c + b < 0\quad say\ its\ a\ negative\ case$$

A Bayesian Interpretation

- Using the maximum margin separator often gives a pretty good approximation to using all separators weighted by their posterior probabilities.

What to do if there is no separating plane

- Use a much bigger set of features.
  - This looks as if it would make the computation hopelessly slow, but in the next lecture we will see how to use the “kernel” trick to make the computation fast even with huge numbers of features.
- Extend the definition of maximum margin to allow non-separating planes.
  - This can be done by using “slack” variables

Introducing slack variables

- Slack variables are constrained to be non-negative. When they are greater than zero they allow us to cheat by putting the plane closer to the datapoint than the margin. So we need to minimize the amount of cheating. This means we have to pick a value for lamba (this sounds familiar!)

$$\mathbf{w}.\mathbf{x}^c + b \geq +1 - \xi^c \quad \textit{for positive cases}$$

$$\mathbf{w}.\mathbf{x}^c + b \leq -1 + \xi^c \quad \textit{for negative cases}$$

$$\textit{with} \quad \xi^c \geq 0 \quad \textit{for all } c$$

$$\textit{and} \quad \frac{\|\mathbf{w}\|^2}{2} + \lambda \sum_c \xi^c \quad \textit{as small as possible}$$

A picture of the best plane with a slack variable