Model selection

In statistical learning, a ***model*** is a mathematical representation of a function such as a

- classifier
- regression function
- density
- …

In many cases, we have one (or more) “free parameters” that are not automatically determined by the learning algorithm

Often, the value chosen for these free parameters has a significant impact on the algorithm’s output

The problem of selecting values for these free parameters is called ***model selection***

Examples

Method	Parameter
• polynomial regression	polynomial degree d
• ridge regression/LASSO	regularization parameter λ
• robust regression	loss function parameter regularization parameter
• SVMs	margin violation cost C
• kernel methods	kernel choice/parameters
• regularized LR	regularization parameter λ
• k-nearest neighbors	number of neighbors k

Model selection dilemma

We need to select appropriate values for the free parameters

All we have is the training data

We must use the training data to select the parameters

However, these free parameters usually control the balance between ***underfitting*** and ***overfitting***

They were left “free” precisely because we don’t want to let the training data influence their selection, as this almost always leads to overfitting

- e.g., if we let the training data determine the degree in polynomial regression, we will just end up choosing the maximum and doing interpolation

Big picture

For much of this class, we have focused on trying to understand learning via decompositions of the form

$$R(h) = \widehat{R}_n(h) + \underbrace{\text{excess risk}}_{\substack{\text{VC dimension} \\ \text{regularization}}}$$

$$R(h) = \text{bias} + \text{variance}$$

Validation takes another approach:

After we have selected h, why not just try (a little harder) to estimate $R(h)$ directly?

Validation

Suppose that in addition to our training data, we also have a **validation set** $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_k, y_k)$

Use the validation set to form an estimate $\widehat{R}_{\text{val}}(h)$

$$\widehat{R}_{\text{val}}(h) := \frac{1}{k} \sum_{i=1}^{k} e(h(\mathbf{x}_i), y_i)$$

Examples

- Classification: $\widehat{R}_{\text{val}}(h) = \frac{1}{k} \sum_i 1_{\{h(\mathbf{x}_i) \neq y_i\}}(i)$
- Regression: $\widehat{R}_{\text{val}}(h) = \frac{1}{k} \sum_i (h(\mathbf{x}_i) - y_i)^2$

$$\widehat{R}_{\text{val}}(h) = \frac{1}{k} \sum_i |h(\mathbf{x}_i) - y_i|$$

$$\vdots$$

Accuracy of validation

What can we say about the accuracy of $\widehat{R}_{\text{val}}(h)$?

In the case of classification, $e(h(\mathbf{x}_i), y_i) = 1_{\{h(\mathbf{x}_i) \neq y_i\}}(i)$, which is just a Bernoulli random variable

Hoeffding: $\mathbb{P}\left[\left|\widehat{R}_{\text{val}}(h) - R(h)\right| > \epsilon\right] \leq 2e^{-2\epsilon^2 k}$

More generally, we always have

$$\mathbb{E}\left[\widehat{R}_{\text{val}}(h)\right] = \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}\left[e(h(\mathbf{x}_i), y_i)\right] = R(h)$$

$$\text{var}\left[\widehat{R}_{\text{val}}(h)\right] = \frac{1}{k^2}\sum_{i=1}^{k} \text{var}\left[e(h(\mathbf{x}_i), y_i)\right] = \frac{\sigma^2}{k}$$

Accuracy of validation

In either case, this shows us that

$$\widehat{R}_{\mathsf{val}}(h) = R(h) \pm O\left(\frac{1}{\sqrt{k}}\right)$$

Thus, we can get as accurate an estimate of $R(h)$ using a validation set as long as k is large enough

Remember, h is ultimately something we learned from training data

Where is this validation set coming from?

Validation vs training

We are given a data set

$$\begin{tikzcd}[column sep=small] & (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \arrow[dl] \arrow[dr] & \\ (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_k, y_k) & & (\mathbf{x}_{k+1}, y_{k+1}), \ldots, (\mathbf{x}_n, y_n) \end{tikzcd}$$

validation (holdout) set

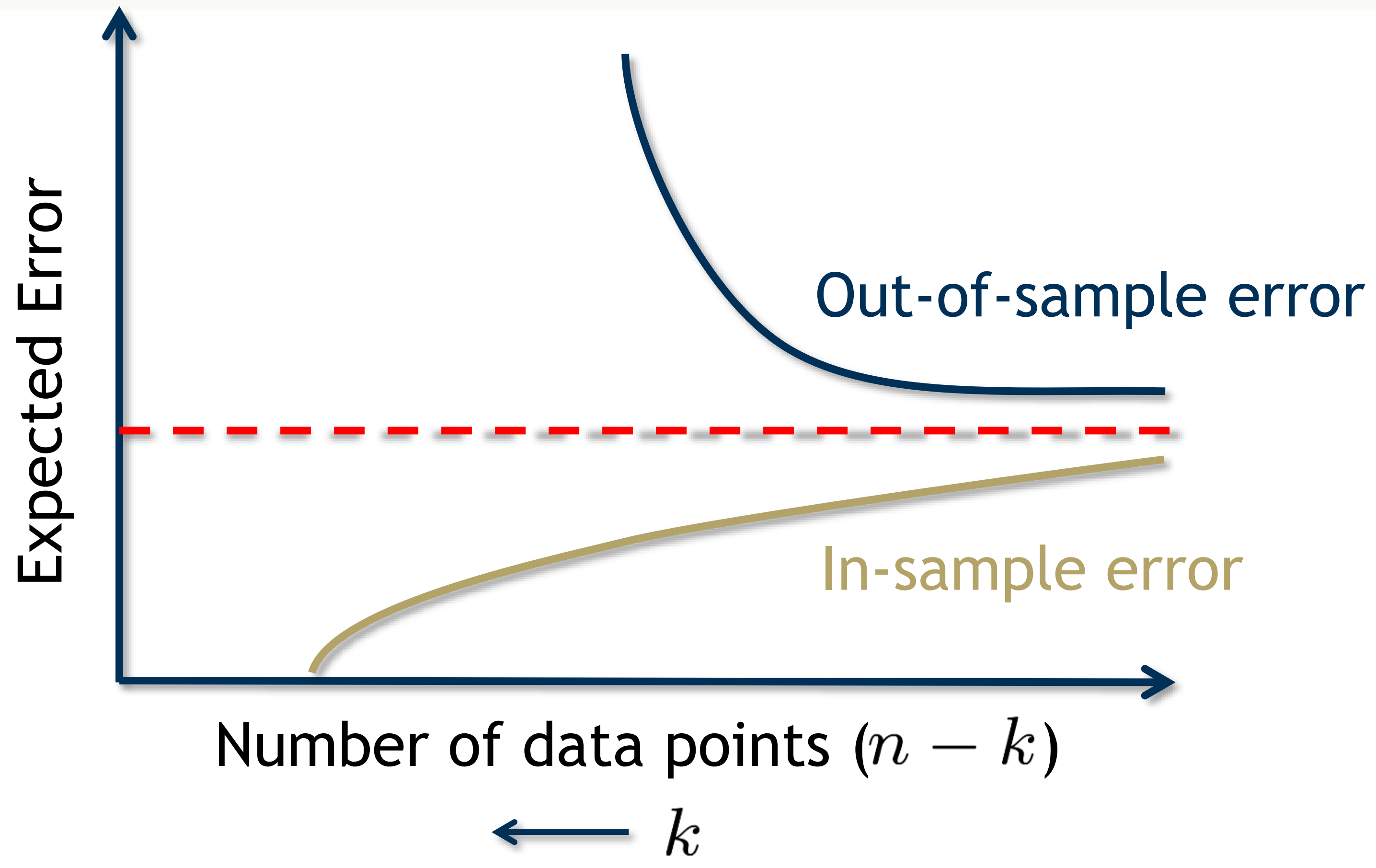
training set

Validation error is $O(1/\sqrt{k})$:

Small k ➡ bad estimate

Large k ➡ accurate estimate, but of what?

Learning curve



Large k lets us say: We are very confident that we have selected a terrible h

Can we have our cake and eat it too?

After we've used our validation set to estimate the error,
re-train on the whole data set

$$\begin{tikzcd}
& (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \arrow[dl] \arrow[dr] \arrow[rr] & & \widehat{h} \\
\text{training set } (n-k) \arrow[d] & & \text{validation set } (k) \arrow[d] & \\
h' & & \widehat{R}_{\text{val}}(h') &
\end{tikzcd}$$

Small k ➡ bad estimate of $R(h')$, but $R(h') \approx R(\widehat{h})$
Large k ➡ good estimate of $R(h')$, but $R(h') \gg R(\widehat{h})$

Rule of thumb: Set $k = n/5$

Validation vs testing

We call this "validation", but how is it any different than simply "testing"?

Typically, $\widehat{R}_{\text{val}}$ is used to make learning choices

If an estimate of $R(h)$ affects learning, i.e., it impacts which h we choose, then it is no longer a ***test*** set

It becomes a ***validation*** set

What's the difference?

- a test set gives us an *unbiased* estimate
- a validation set will have an (overly) *optimistic bias*
  (remember the coin tossing experiments?)

Example

Suppose we have two hypotheses h_1, h_2 and that

$$R(h_1) = R(h_2) = p$$

Next, suppose that our error estimates for h_1, h_2, denoted by $\widehat{R}_{\text{val}}(h_1)$ and $\widehat{R}_{\text{val}}(h_2)$, are distributed according to

$$\widehat{R}_{\text{val}}(h_i) \sim \text{Unif}\,[p - \eta, p + \eta]$$

Pick $h \in \{h_1, h_2\}$ that minimizes $\widehat{R}_{\text{val}}(h)$

It is easy to argue that $\mathbb{E}\left[\widehat{R}_{\text{val}}(h)\right] < p$ ***optimistic bias***

Why? 75 % of the time, $\min\left(\widehat{R}_{\text{val}}(h_1), \widehat{R}_{\text{val}}(h_2)\right) < p$

Using validation for model selection

Suppose we have m models $\mathcal{H}_1, \ldots, \mathcal{H}_m$

$$\begin{tikzcd}
& (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \arrow[dl] \arrow[dr] \arrow[rr] & & \mathcal{H}_{j^*} \arrow[r] & \widehat{h} \\
\text{training set } (n-k) \arrow[d] & & \text{validation set } (k) \arrow[d] & & \\
\mathcal{H}_1 \quad \cdots \quad \mathcal{H}_m \arrow[d] & & \underbrace{\widehat{R}_{\text{val}}(h'_1) \cdots \widehat{R}_{\text{val}}(h'_m)}_{\text{pick the best}} & & \\
h'_1 \quad \cdots \quad h'_m & & j^* \arrow[uuur, bend right] & &
\end{tikzcd}$$

Quantifying the bias

We've seen this before…

For m models $\mathcal{H}_1, \ldots, \mathcal{H}_m$, we use a data set of size k to pick the model that does best out of $\{h'_1, \ldots, h'_m\}$

Back to Hoeffding!

$$R(h'_{j^*}) \leq \widehat{R}_{\text{val}}(h'_{j^*}) + O\left(\sqrt{\frac{\log m}{k}}\right)$$

Or, if the $\mathcal{H}_j$ correspond to a few continuous parameters, we can use the VC approach to argue

$$R(h'_{j^*}) \leq \widehat{R}_{\text{val}}(h'_{j^*}) + O\left(\sqrt{\frac{\#\text{ of parameters}}{k}}\right)$$

Data contamination

We have now discussed three different kinds of estimates of the risk $R(h)$:

$$\widehat{R}_{\text{train}}(h), \widehat{R}_{\text{test}}(h), \widehat{R}_{\text{val}}(h)$$

These three estimates have different degrees of “contamination” that manifests itself as a (deceptively) optimistic bias

- Training set: totally contaminated
- Testing set: totally clean (requires strict discipline)
- Validation set: slightly contaminated

We will return in a bit to the issue of data “contamination”

Validation dilemma

Back to our core dilemma in validation

We would like to argue that

$$R(h) \underset{\substack{\uparrow \\ \text{small } k}}{\approx} R(h') \underset{\substack{\uparrow \\ \text{large } k}}{\approx} \widehat{R}_{\mathsf{val}}(h')$$

All we need to do is set k so that it is simultaneously small and large

Can we do this?

Yes!

Leave one out

We need k to be small, so let's set $k = 1$!

$$\mathcal{D}_j = (\mathbf{x}_1, y_1), \ldots, \cancel{(\mathbf{x}_j, y_j)}, \ldots, (\mathbf{x}_n, y_n)$$

Select a hypothesis h'_j using the data set $\mathcal{D}_j$

Validation error $\widehat{R}_{\text{val}}(h'_j) = e(h'_j(\mathbf{x}_j), y_j) := e_j$

We set k to be too small, so this is a terrible estimate

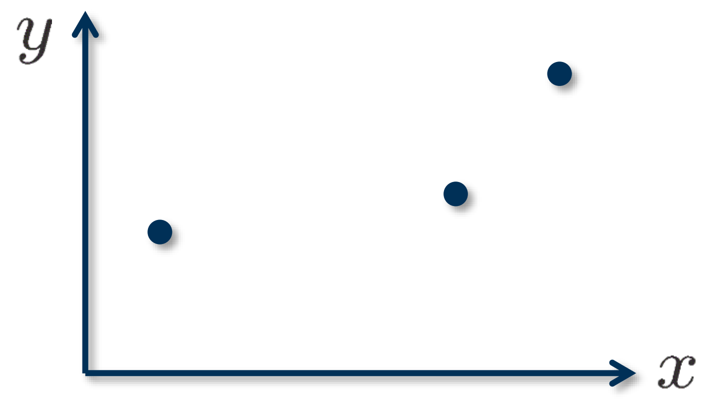
Repeat this for all possible choices of j and average!

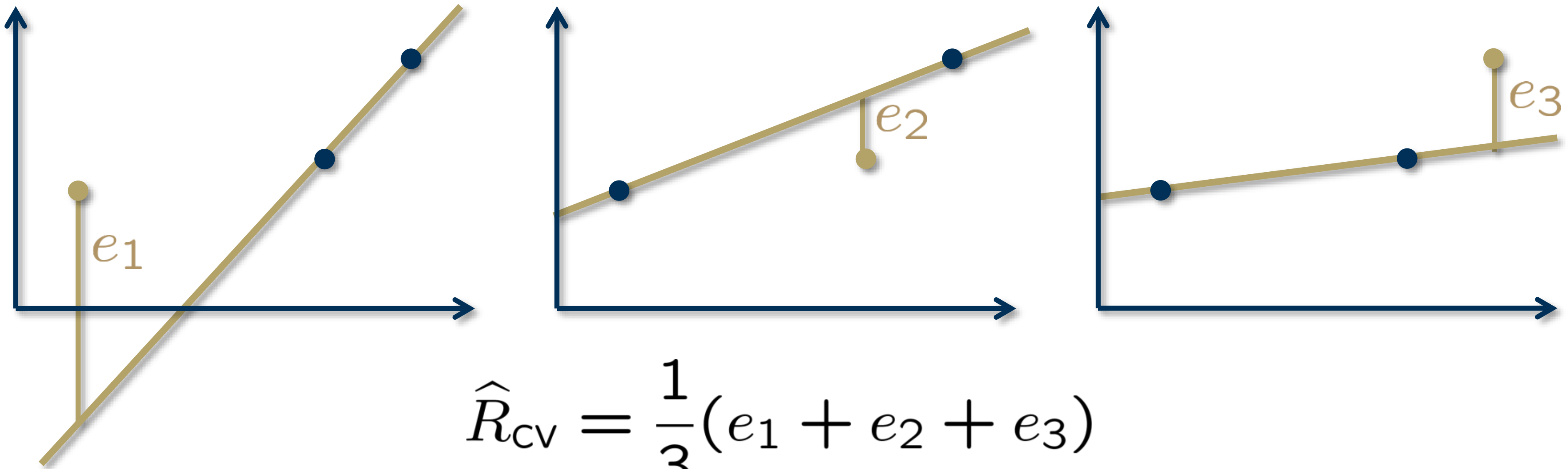
$$\widehat{R}_{\text{CV}} = \frac{1}{n} \sum_{j=1}^{n} e_j$$

This is called the ***leave-one-out cross validation*** error

Example

Fitting a line to 3 data points





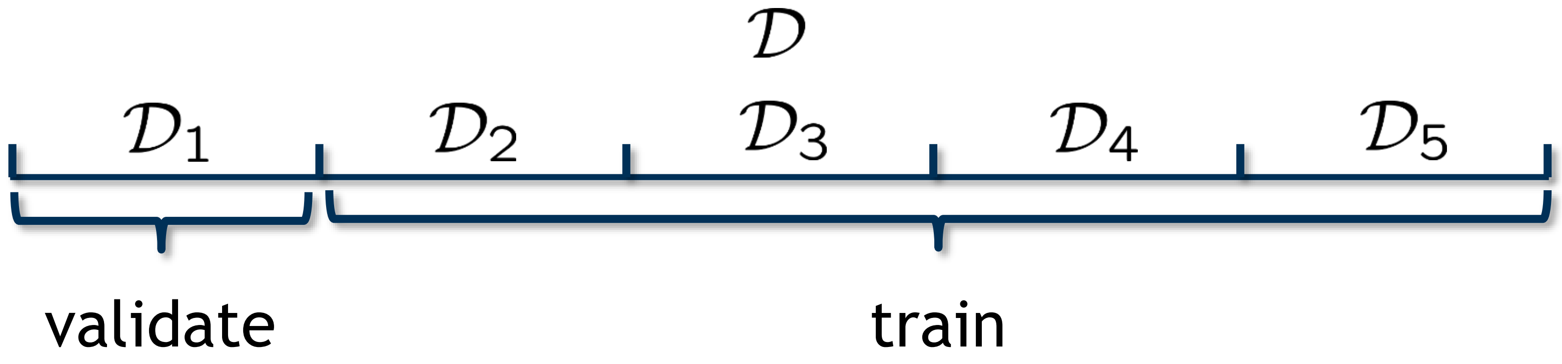
$$\widehat{R}_{\text{CV}} = \frac{1}{3}(e_1 + e_2 + e_3)$$

Leave more out

Leave-one-out: Train n times on $n-1$ points each

k-fold cross validation: Train k times on $n - \frac{n}{k}$ points each

Example: $k = 5$



Iterate over all 5 choices of validation set and average

Common choices are $k = 5, 10$

(Note: On this slide, k is the number of folds and $k' = \frac{n}{k}$ is the size of the validation set)

Remarks

- For k-fold cross validation, the estimate depends on the particular choice of partition

- It is common to form several estimates based on different random partitions and then average them

- When using k-fold cross validation for classification, you should ensure that each of the sets $\mathcal{D}_j$ contain training data from each class in the same proportion as in the full data set
  - "stratified cross validation"

- Most ML toolboxes can do all of this for you for any of the built-in learning methods

The bootstrap

What else can you do when your training set is really small?

You really need as much training data as possible to get reasonable results

Fix $B \geq 1$

For $b = 1, \ldots, B$, let $\mathcal{D}_b$ be a subset of size n obtained by ***sampling with replacement*** from the full data set $\mathcal{D}$

Example: $n = 5$

$$\mathcal{D}_1 = (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4), (\mathbf{x}_5, y_5), (\mathbf{x}_4, y_4), (\mathbf{x}_1, y_1)$$

$$\mathcal{D}_2 = (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_5, y_5), (\mathbf{x}_5, y_5), (\mathbf{x}_2, y_2)$$

$$\vdots$$

The bootstrap error estimate

Define $h_b :=$ model learned based on the data $\mathcal{D}_b$

$$\mathcal{D}_b^c := \mathcal{D} \setminus \mathcal{D}_b$$

Set $e_b = \frac{1}{|\mathcal{D}_b^c|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}_b^c} e(h_b(\mathbf{x}_i), y_i)$

The ***bootstrap*** error estimate is then given by

$$\widehat{R}_{\mathsf{B}} := \frac{1}{B} \sum_{b=1}^{B} e_b$$

Bootstrap in practice

- Typically, B must be large (say, $B \approx 200$) for the estimate to be accurate
- Can be rather computationally demanding
- $\widehat{R}_{\mathrm{B}}$ tends to be ***pessimistic***, so it is common to combine the training and bootstrap error estimates
- A common choice is the ***"0.632 bootstrap estimate"***

$$0.632\widehat{R}_{\mathrm{B}} + 0.368\widehat{R}_{\mathrm{train}}$$

- The "balanced" bootstrap chooses $\mathcal{D}_1, \ldots, \mathcal{D}_B$ such that each input-output pair appears exactly B times
- Can be used to estimate confidence intervals of basically anything, but potentially very computationally demanding

Data snooping

If a data set has affected ***any*** step in the learning process, its ability to assess the outcome has been compromised

This is by far the most common trap that people fall into in practice

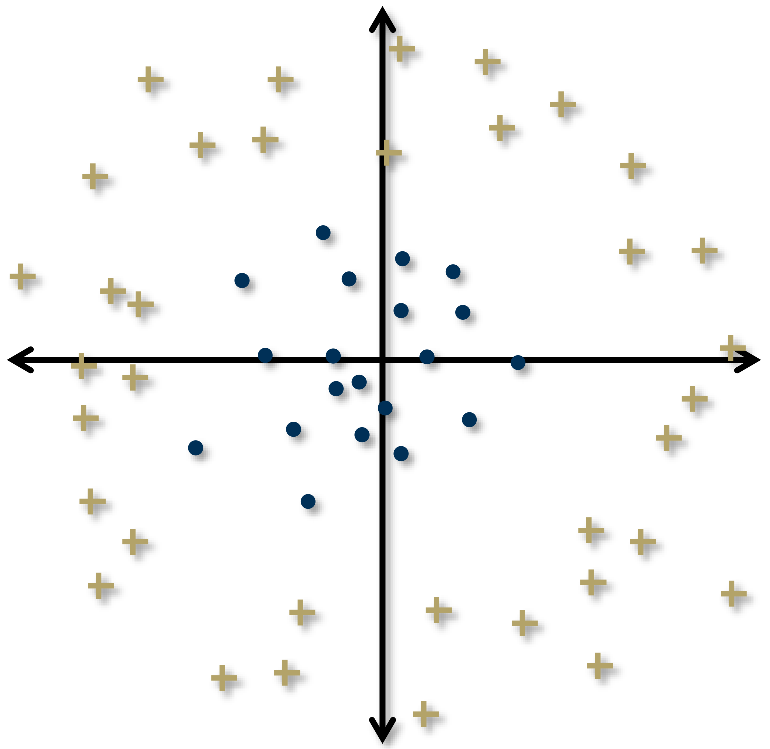
Leads to serious overfitting...

Can be very subtle...

Many ways to slip up...

Example

Suppose we plan to use an SVM with a quadratic kernel on our data set



$$\Phi(\mathrm{x}) = \begin{bmatrix} 1 \\ \cancel{x(1)} \\ \cancel{x(2)} \\ \cancel{x(1)x(2)} \\ x(1)^2 \\ x(2)^2 \end{bmatrix}$$

What is the VC dimension of the hypothesis set in this case?

Reuse of the data set

If you try one model after another ***on the same data set***, you will eventually "succeed"

If you torture the data long enough, it will confess

You need to think about the VC dimension/complexity of the ***total*** learning model

- May include models you only considered ***in your mind***!
- May include models tried by ***others***!

Remedies

- Avoid data snooping (strict discipline)
- Test on new data that no one has seen before
- Account for data snooping

Puzzle: Time-series forecasting

Suppose we wish to predict whether the price of a stock is going to go up or down tomorrow

- Take history over a long period of time
- Normalize the time series to zero mean, unit variance
- Form all possible input-output pairs with
  - input = previous 20 days of stock prices
  - output = price movement on the 21st day
- Randomly split data into training and testing data
- Train on training data only, test on testing data only

Based on the test data, it looks like we can consistently predict the price movement direction with accuracy ~52%

Are we going to be rich?