Machine Learning
Learning Theory

dsai.asia

Asian Data Science and Artificial Intelligence Master's Program

Co-funded by the
Erasmus+ Programme
of the European Union

Readings

Readings for these lecture notes:

- Bishop, C. (2006), *Pattern Recognition and Machine Learning*, Springer, Chapter 3.
- Hastie, T., Tibshirani, R., and Friedman, J. (2016), *Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, Chapter 7.
- Ng, A. (2017), *Learning Theory*. Lecture note sets 4, 5 for CS229, Stanford University.

Outline

Introduction

Machine learning is one of those areas where practical rules of thumb tend to emerge before the theoretical principles that might lead to those rules of thumb.

In this series, we cover some of the most important theoretical principles that underly or seek to explain our practical heuristics:

- The bias-variance tradeoff
- How are training set error and test set error related?
- When we use a high variance model that is prone to overfitting, how can we control it?

Outline

Bias-Variance tradeoff

Test set error

In machine learning, now you see that the general approach is to fit a model $h()$ to a training dataset then test the model on a test dataset.

Consider the linear regression case. After estimating $h_{\boldsymbol{\theta}}$, we compute the mean squared error over a test set:

$$\mathbb{E}_{(\mathrm{x},y)\sim\text{test set}}(h_{\boldsymbol{\theta}}(\mathrm{x})-y)^2.$$

What are the possible cause of the test set error being too high?

- Overfitting: the model is too closely tailored to the examples in the training set and can't generalize to the test set.
- Underfitting: the model doesn't capture the link between the features x and the target y.
- Noise: the data are inherently noisy, and no learning method is going to give us lower error on a test set.

Bias-Variance tradeoff

Randomness of $h_{\boldsymbol{\theta}}()$

To formalize this intuition, we first suppose that the training and test data are all sampled from a distribution

$$y = f(\mathrm{x}) + \varepsilon,$$

where $\mathbb{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma^2$.

We use a training set to estimate $h_{\boldsymbol{\theta}}()$ then apply to each pattern j in the test set.

Our prediction of the $y^{(j)}$ for $\mathrm{x}^{(j)}$ (note that $y^{(j)}$ is equal to $f(\mathrm{x}^{(j)}) + \varepsilon^{(j)}$) is $h_{\boldsymbol{\theta}}(\mathrm{x}^{(j)})$.

Clearly, $\mathrm{x}^{(j)}$ is fixed when we compute $h_{\boldsymbol{\theta}}(\mathrm{x}^{(j)})$, but $h_{\boldsymbol{\theta}}(\mathrm{x}^{(j)})$ itself is random, since it depends on the values $\varepsilon^{(i)}$ in the training set.

For linear regression, we define the bias of $h_{\boldsymbol{\theta}}()$ as

$$\mathbb{E}(h_{\boldsymbol{\theta}}(x) - f(x))$$

and the variance of $h_{\boldsymbol{\theta}}()$ as

$$\mathbb{E}\left((h_{\boldsymbol{\theta}}(x) - f(x))^2\right).$$

There are various formal notions of bias and variance for classifiers, but there is no general agreement on which one is best.

Mean squared error on the test data set can be decomposed as

$$\begin{aligned}
\text{Test MSE} &= \mathbb{E}\left((y - h_{\boldsymbol{\theta}}(\mathrm{x}))^2\right) \\
&= \mathbb{E}\left((f(\mathrm{x}) + \varepsilon - h_{\boldsymbol{\theta}}(\mathrm{x}))^2\right) \\
&= \mathbb{E}(\varepsilon^2) + \mathbb{E}\left((f(\mathrm{x}) - h_{\boldsymbol{\theta}}(\mathrm{x}))^2\right) \qquad (1) \\
&= \sigma^2 + \left(\mathbb{E}(f(\mathrm{x}) - h_{\boldsymbol{\theta}}(\mathrm{x}))\right)^2 + \mathrm{Var}\left(f(\mathrm{x}) - h_{\boldsymbol{\theta}}(\mathrm{x})\right) \qquad (2) \\
&= \sigma^2 + \left(\text{Bias } h_{\boldsymbol{\theta}}(\mathrm{x})\right)^2 + \mathrm{Var}\left(h_{\boldsymbol{\theta}}(\mathrm{x})\right)
\end{aligned}$$

How to get Equation 1?

We have $\mathbb{E}((X_1 + X_2)^2)$ where $X_1 = \varepsilon$ and $X_2 = f(\mathrm{x}) - h_{\boldsymbol{\theta}}(\mathrm{x})$.

$$\mathbb{E}((X_1 + X_2)^2) = \mathbb{E}(X_1^2 + X_2^2 + 2X_1X_2) = \mathbb{E}(X_1^2) + \mathbb{E}(X_2^2) + 2\mathbb{E}(X_1X_2).$$

Since X_1 and X_2 are independent we have $\mathbb{E}(X_1X_2) = \mathbb{E}(X_1)\mathbb{E}(X_2)$. Since $\mathbb{E}(\varepsilon) = 0$, in this special case, we get $\mathbb{E}((X_1 + X_2)^2) = \mathbb{E}(X_1^2) + \mathbb{E}(X_2^2)$.

How to get Equation 2? Start with the definition of variance:

$$\begin{aligned}
\mathrm{Var}(X) &= \mathbb{E}((X - \mathbb{E}(X))^2) \\
&= \mathbb{E}(X^2 - 2X\mathbb{E}(X) + (\mathbb{E}(X))^2) \\
&= \mathbb{E}(X^2) - 2\mathbb{E}(X)\mathbb{E}(X) + \mathbb{E}(X)\mathbb{E}(X) \\
&= \mathbb{E}(X^2) - (\mathbb{E}(X))^2 \\
\mathbb{E}(X^2) &= (\mathbb{E}(X))^2 + \mathrm{Var}(X)
\end{aligned}$$

We cannot predict the noise ε.

The bias term in the MSE is due to underfitting.

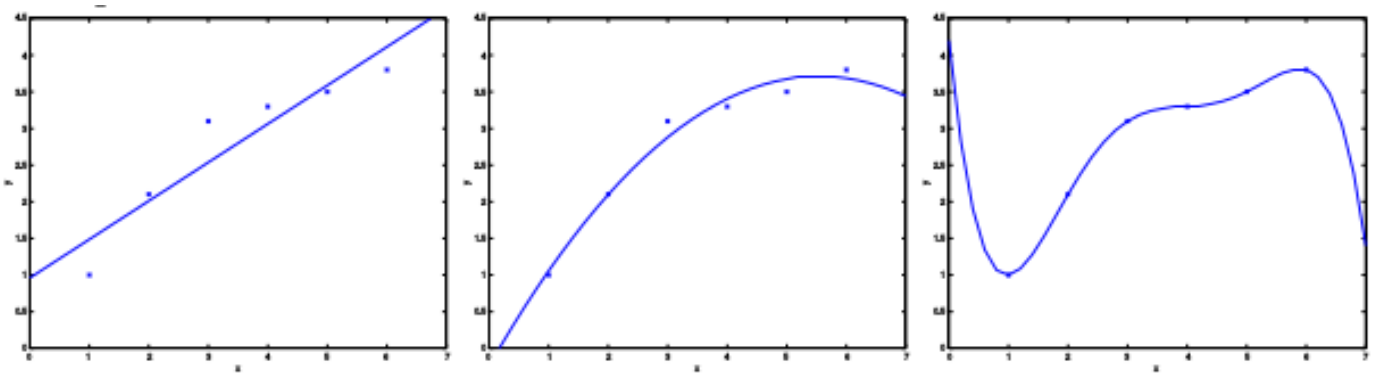
The variance term of the MSE is related to overfitting.

We could say that our grand goal is to find the sweet spot of no underfitting and no overfitting, in which bias and variance are both negligible.

Unfortunately, in practice, when we reduce bias we usually increase variance, and vice versa.

As an example, consider the case of linear regression with a single input variable.

We might try models with degree 1, degree 2, and degree 5 in the input variable:



Ng (2017), CS229 lecture notes

The degree 1 model has high bias, and the degree 5 model has high variance.

Outline

This section is directly from Ng (2017), *Learning Theory*.

To get started in understanding the generalization ability of a learning model, we need some preliminary facts useful for the analysis: the union bound and the Hoeffding inequality.

The union bound

Let $A_1, A_2, \ldots, A_k$ be k different events resulting from a probabilistic experiment that are not necessarily independent.

$$P(A_1 \cup \cdots \cup A_k) \leq P(A_1) + \cdots + P(A_k).$$

The union bound follows from Kolmomgorov's axioms of probability theory and is sometimes considered an axiom itself.

The Hoeffding inequality or the Chernoff bound

1. Let $Z_1, \ldots, Z_m$ be m independent and identically distributed random variables drawn from a Bernoulli(ϕ) distribution (i.e., $P(Z_i = 1) = \phi, P(Z_i = 0) = 1 - \phi$).
2. Let $\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} Z_i$ be the mean of the Z_i.
3. Let $\gamma > 0$ be fixed.

Then

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2e^{-2\gamma^2 m}.$$

Essentially, this says that the average of m Bernoulli(ϕ) variables will be very close to ϕ as long as m is large.

Now, let's restrict ourselves to binary classification with labels $y \in \{0, 1\}$.

We are given a training set $S = \{(\mathrm{x}^{(i)}, y^{(i)})\}_{i \in 1..m}$ of size m drawn i.i.d. from some distribution $\mathcal{D}$.

The training error or empirical risk or empirical error of a hypothesis h is defined as

$$\hat{\varepsilon}(h) = \frac{1}{m} \sum_{i=1}^{m} \delta(h(\mathrm{x}^{(i)}) \neq y^{(i)}).$$

The generalization error is defined as

$$\varepsilon(h) = P_{(\mathrm{x},y)\sim\mathcal{D}}(h(\mathrm{x}) \neq y).$$

The assumption that the training and test distributions are the same is one of the PAC (probably approximately correct) assumptions.

How to determine the best h?

One way, called empirical risk minimization (ERM) says to pick the h minimizing $\hat{\varepsilon}(h)$.

For example, for the linear classifier

$$h_{\boldsymbol{\theta}}(\mathrm{x}) = \delta(\boldsymbol{\theta}^{\top}\mathrm{x} \geq 0),$$

we pick

$$\hat{\boldsymbol{\theta}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \hat{\varepsilon}(h_{\boldsymbol{\theta}}).$$

and output $\hat{h} = h_{\hat{\boldsymbol{\theta}}}$.

Note: algorithms such as logistic regression are not exactly ERM but can be thought of as approximations to ERM.

Now let's generalize beyond linear classifiers.

Any learning algorithm will consider some set of hypotheses $\mathcal{H}$.

Example: linear classifiers are defined by

$$\mathcal{H} = \{h_{\boldsymbol{\theta}} : h_{\boldsymbol{\theta}}(\mathrm{x}) = \delta(\boldsymbol{\theta}^\top \mathrm{x} \geq 0), \boldsymbol{\theta} \in \mathbb{R}^{n+1}\}.$$

Alternatively, $\mathcal{H}$ might be the set of all classifiers represented by a particular neural network architecture.

Once we have $\mathcal{H}$, we can define ERM as determining

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \hat{\varepsilon}(h).$$

Outline

Bounds on generalization error for finite $\mathcal{H}$

We begin by assuming $\mathcal{H} = \{h_1, \ldots, h_k\}$ is finite with k possible hypotheses.

We have k functions mapping $\mathcal{X}$ to $\{0, 1\}$.

ERM selects the h_i minimzing $\hat{\varepsilon}(h_i)$.

What we can do:

- Show that $\hat{\varepsilon}(h)$ is a reliable estimate of $\varepsilon(h)$ for all h.
- Bound the generalization error of $\hat{h}$.

Take a particular hypothesis $h_i \in \mathcal{H}$.

Consider a Bernoulli random variable Z whose distribution is defined as follows:

- Sample $(\mathrm{x}, y) \sim \mathcal{D}$.
- Set $Z = \delta(h_i(\mathrm{x}) \neq y)$.

Z indicates whether h_i misclassifies (x, y).

Next, define $Z_j = \delta(h_i(\mathrm{x}^{(j)}) \neq y^{(j)})$.

Since the training set S was drawn i.i.d. from $\mathcal{D}$ and so was (x, y), we know that Z and Z_j have the same distribution.

The expected value of Z is the probability of misclassification of a randomly drawn example, which is $\varepsilon(h_i)$.

As Z and Z_j have the same distribution, the expected value of Z_j is also $\varepsilon(h_i)$.

We can write the training error as

$$\hat{\varepsilon}(h_i) = \frac{1}{m} \sum_{j=1}^{m} Z_j.$$

We have the mean $\hat{\epsilon}(h_i)$ of m random variables Z_j, all drawn i.i.d. from a Bernoulli distribution with mean $\varepsilon(h_i)$.

Now we can apply the Hoeffding inequality! We obtain

$$P(|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) \leq 2e^{-2\gamma^2 m}.$$

What have we shown?

For our h_i, training error will be close to generalization error with probability that increases with m.

What about all possible h_i?

Well, let A_i denote the event that $|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma$.

For any A_i, we have $P(A_i) \leq 2e^{-2\gamma^2 m}$.

Now, we can use the union bound to bound generalization error for any h_i:

$$\begin{aligned} P(\exists h \in \mathcal{H}.|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) &= P(A_1 \cup \cdots \cup A_k) \\ &\leq \sum_{i=1}^{k} P(A_i) \\ &\leq \sum_{i=1}^{k} 2e^{-2\gamma^2 m} \\ &= 2ke^{-2\gamma^2 m}. \end{aligned}$$

Subtracting both sides from 1, we obtain

$$\begin{aligned} P(\neg\exists h \in \mathcal{H}.|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) &= P(\forall h \in \mathcal{H}.|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| \leq \gamma) \\ &\leq 1 - 2ke^{-2\gamma^2 m}. \end{aligned}$$

Bounds on generalization error for finite $\mathcal{H}$

Using uniform convergence

The previous result is called the uniform convergence result.

The bound holds simultaneously for all $h \in \mathcal{H}$.

We can use this, for example, to answer the question, "Given γ and some $\delta > 0$, find m large enough to guarantee that with probability at least $1 - \delta$, training error will be within γ of generalization error."

To do it, let $\delta = 2ke^{-2\gamma^2 m}$ and solve for m to find that we should use

$$m \geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta}.$$

Now we know how many training examples are needed to make a guarantee about generalization error. Note that m increases in the logarithm of k!!

The training set size m needed to achieve a level of performance is called the algorithm's sample complexity.

Bounds on generalization error for finite $\mathcal{H}$

Using uniform convergence

We can also hold m and δ fixed and solve for γ:

$$|\hat{\varepsilon}(h) - \varepsilon(h)| \leq \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$

[Think about how you could use this fact...]

Bounds on generalization error for finite $\mathcal{H}$

The bound

Now to prove something about the generalization error obtained by picking

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \hat{\varepsilon}(h).$$

We have $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$ for all $h \in \mathcal{H}$.

Let $h^* = \operatorname{argmin}_{h \in \mathcal{H}} \varepsilon(h)$ be the best possible hypothesis in $\mathcal{H}$.

Applying uniform convergence (twice) and the fact that $\hat{h} = \operatorname{argmin}_{h \in H} \hat{\varepsilon}(h)$, we can obtain

$$\begin{aligned} \varepsilon(\hat{h}) &\leq \hat{\varepsilon}(\hat{h}) + \gamma \\ &\leq \hat{\varepsilon}(h^*) + \gamma \\ &\leq \varepsilon(h^*) + 2\gamma \end{aligned}$$

The test error of the minimum risk classifier is at most 2γ worse than the best possible classifier!

Theorem. Let $|\mathcal{H}| = k$, and let any m, δ be fixed. Then with probability at least $1 - \delta$, we have that

$$\varepsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}} \varepsilon(h) \right) + 2\sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$

Prove this by letting $\gamma = \sqrt{\cdot}$ and apply the previous argument.

Bounds on generalization error for finite $\mathcal{H}$

The bound's implications for bias and variance

We have not formally defined bias and variance for a classifier, but the first term in the bound is a rough bias for $\hat{h}$, and the second term is a rough variance.

Suppose we have a hypothesis class $\mathcal{H}$.

If we consider a larger hypothesis class $\mathcal{H}' \supseteq \mathcal{H}$:

- We will improve our bias ($\min_{h \in \mathcal{H}} \varepsilon(h)$ can only get smaller),
- We will increase k, thus increasing the variance.

If we hold γ and δ fixed, obtain a bound on sample complexity.

Corollary. Let $|\mathcal{H}| = k$, and let any δ, γ be fixed. Then for $\varepsilon(\hat{h}) \leq \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$ to hold with probability at least $1 - \delta$, it suffices that

$$\begin{aligned} m &\geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta} \\ &= O\left(\frac{1}{\gamma^2} \log \frac{k}{\delta}\right). \end{aligned}$$

Outline

Bounds on generalization error for infinite $\mathcal{H}$

Motivation

We've seen some interesting results for $|\mathcal{H}| = k$.

But most of the models we've looked at in this class have $|\mathcal{H}| = \infty$.

What can we do??

Bounds on generalization error for infinite $\mathcal{H}$

A not-so-good approach

To begin with, imagine our parameter vector $\boldsymbol{\theta}$ consists of d real numbers represented as double-precision (64 bit) IEEE floating point numbers.

Our hypothesis class thus consists of at most $k = 2^{64d}$ hypotheses.

To guarantee $\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma$ to hold with probability at least $1 - \delta$ we need

$$m \geq O\left(\frac{1}{\gamma^2} \log \frac{2^{64d}}{\delta}\right) = O\left(\frac{d}{\gamma^2} \log \frac{1}{\delta}\right) = O_{\gamma,\delta}(d).$$

So the number of training examples is linear in the number of parameters in the model if we use finite floating point representations of real numbers and manage to implement ERM over this space.

Bounds on generalization error for infinite $\mathcal{H}$

A better approach

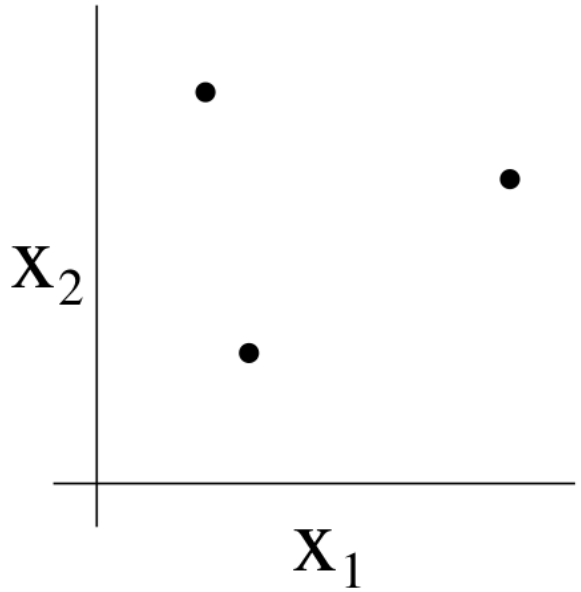
The approach based on finite floating point approximations of $\boldsymbol{\theta} \in \mathbb{R}^d$ depends on the parameterization of the hypothesis.

There is a better approach based on the characteristics feature space.

Definition: Given a set $S = \{\mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(d)}\}$ of points $\mathrm{x}^{(i)} \in \mathcal{X}$, we say that $\mathcal{H}$ shatters S if $\mathcal{H}$ can realize any labeling on S.

Definition: Given a hypothesis class $\mathcal{H}$, we define the Vapnick-Chervonenkis dimension $\mathrm{VC}(\mathcal{H})$ as the largest set that is shattered by $\mathcal{H}$.

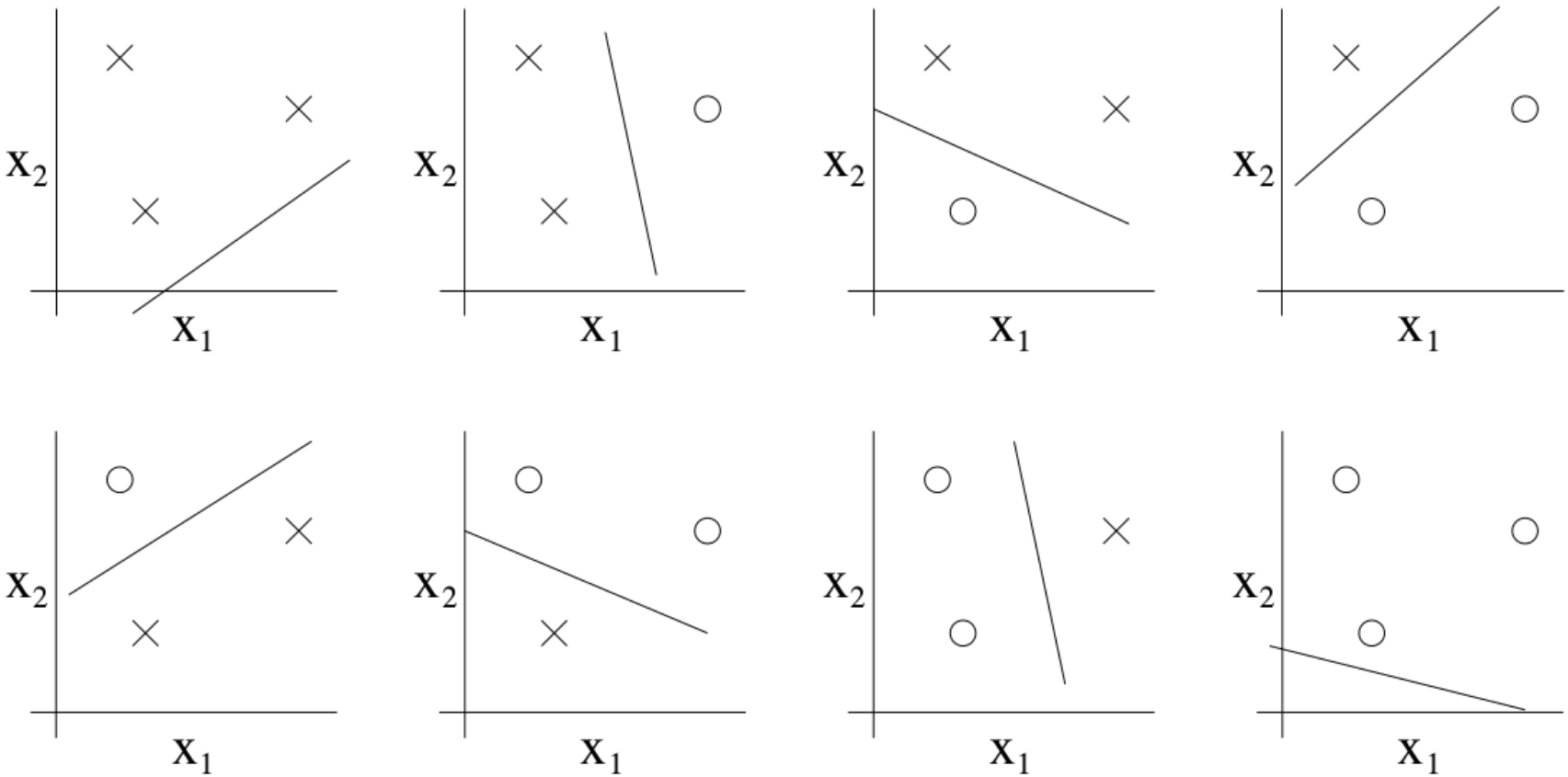
What is the VC dimension of a linear classifier in $\mathbb{R}^2$?



Ng (2017), CS229 lecture notes.

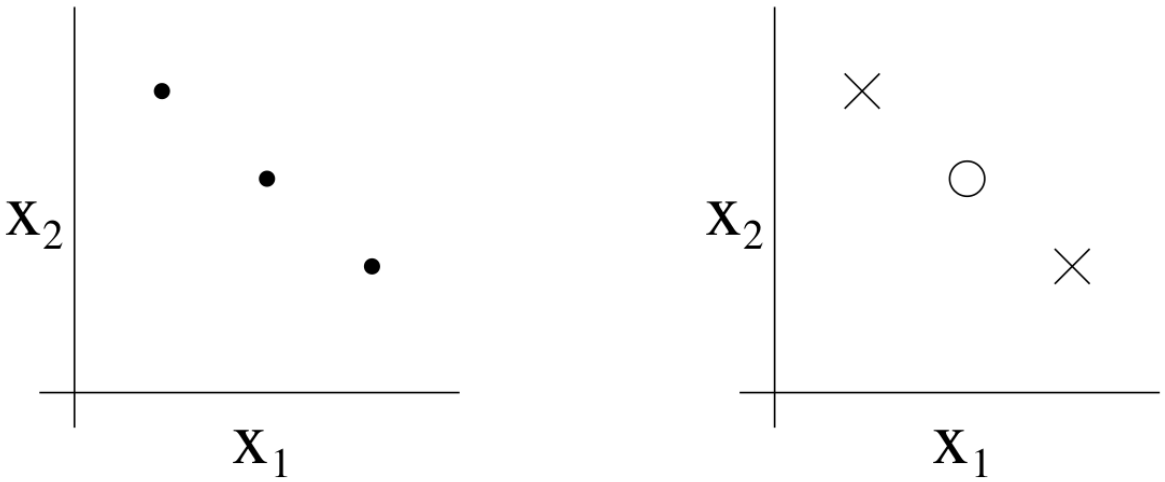
Can these three points be shattered?

The answer is yes:



Ng (2017), CS229 lecture notes.

Note that for VC dimension to be d, we have to be able to shatter some point set of size d, not all point sets:



Ng (2017), CS229 lecture notes.

Theorem (Vapnik). Let $\mathcal{H}$ be given, and let $d = \text{VC}(\mathcal{H})$. Then with probability at least $1 - \delta$, we have that for all $h \in \mathcal{H}$,

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq O\left(\sqrt{\frac{d}{m}\log\frac{m}{d} + \frac{1}{m}\log\frac{1}{\delta}}\right).$$

This also means

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + O\left(\sqrt{\frac{d}{m}\log\frac{m}{d} + \frac{1}{m}\log\frac{1}{\delta}}\right).$$

We have uniform convergence in m so long as d is finite.

This is considered by many the most important theorem in learning theory!

Corollary. For $|\varepsilon(h) - \hat{\varepsilon}(h) \leq \gamma$ to hold for all $h \in \mathcal{H}$, so that $\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma)$ with probability at least $1 - \delta$, it suffices that $m = O_{\gamma,\delta}(d)$.

The number of training examples needed to learn "well" using $\mathcal{H}$ is linear in the VC dimension of $\mathcal{H}$.

For most hypothesis classes, VC dimension is roughly linear in the number of parameters.

This means that for most classifiers, the number of training examples needed is usually roughly linear in the number of parameters of $\mathcal{H}$.

Outline

Conclusion on generalization bounds

Now we've seen the most important results in learning theory.

Generally, the story is encouraging: we can achieve near perfection for most models just by gathering sufficient data.

But the devil is in the details. For example:

- SVMs with finite $\phi(\mathrm{x})$ have finite VC dimension, but if $\phi(\mathrm{x})$ maps x to an infinite dimensional space (like the RBF kernel does) then the VC dimension is infinite.
- We find that the m needed to achieve a particular level of generalization error is linear in $\mathrm{VC}(\mathcal{H})$. But what is the constant of proportionality? (Getting data has a cost, and a factor of 100 would be a lot bigger than 10!)
- CNNs with millions of parameters can be trained effectively with thousands of examples. How is it possible? Are deep learning models cheating Vapnik somehow?

Conclusion on generalization bounds

Deep learning is particularly interesting here.

Neural network architectures' VC dimension is $O(|\mathtt{W}|)$ in the best case.

So a model with millions of parameters needs millions of training examples to guarantee good generalization according to Vapnik.

However, note that Vapnik's theorem is a theorem of sufficiency, not necessity. Techniques we have studied help us cheat Vapnik:

- Stopping training when validation set cost is minimal
- Weight decay and dropout (regularization)

There is a really interesting take from astrophysicists' perspective by Lin, Tegmark, and Roland (2016) at `https://arxiv.org/abs/1608.08225` that provides a interesting theory of how deeper models may better due to the hierarchical structure of real world generative processes.

These are topics needing more research!

Outline

OK, so now we understand that for a hard problem, the "right" model, the one that is guaranteed to generalize to the test set, might be too biased.

We also understand that a less biased model may have VC dimension too high for the number of examples we have and will thus be prone to overfitting (too much variance).

How can we find the "best" model? For example,

- In polynomial regression $h_{\boldsymbol{\theta}}(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_k x^k)$, how to decide the right k?
- In locally-weighted regression, how to choose the bandwidth parameter τ?
- With SVMs, how to choose C (and γ for the RBF kernel)?

Model selection

Cross validation

In each of these cases, we have a set of models $\mathcal{M} = \{M_1, \ldots, M_d\}$ we would like to select from.

From ERM we might train each M_i and pick the M_i with the lowest training error.

But this will give us a high-variance model that doesn't test well.

Our main alternative (using a validation set) is called cross validation:

- Split the training set S into S_{train} and $S_{validation}$ using a ratio such as 70% to 30% or 80% to 20%.
- Train each model M_i on S_{train}.
- Select the M_i with lowest error on $S_{validation}$.
- (Optional) Re-train M_i on all of S.
- Use M_i.

Model selection

k-fold cross validation

When data is scarse, we'd prefer to select the model that is best on all the data, not just 30% of the data!

Then k-fold cross validation makes more sense:

1. Split S into k disjoint subsets $S_1, \ldots, S_k$.
2. For each M_i
   - For $j = 1, \ldots, k$
     1. Train M_i on $S_{train} = S_1, \ldots, S_{j-1}, S_{j+1}, \ldots, S_k$
     2. Test the trained M_i on $S_{validation} = S_j$
3. Select the M_i with the lowest average error on $S_{validation}$ over the k folds.
4. Retrain M_i on all of S.

Typical values of k are 5, 10, and m (leave-one-out cross validation).

Model selection

k-fold cross validation

Be careful about how you split!

- Usually, the partitioning of examples into k folds is random uniform.
- However, sometimes training items are related to each other (e.g. multiple crops of the same object in the same image).
- Randomized partitioning will put some related examples in different folds, leading to overestimated performance.
- In this case, it's necessary to ensure that the related examples are in the same fold.

A special case of model selection is feature selection.

Suppose you have many features, possibly even $n \gg m$.[1]

We know that the VC dimension of a classifier with very large n will be too high.

So we treat the problem as model selection: among the 2^n possible subsets of features, find the subset that gives the best cross validation performance.

Trying all 2^n subsets is impossible unless n is very small. So we need a more efficient approach.

[1] This is possible in some domains such as biological sequence analysis, where we might want to find segments of DNA sequences in a particular genome that code for a particular biological function, or to a lesser extent in text classification.

The forward search method of feature selection:

1. Initialize $\mathcal{F} = \emptyset$.
2. For $j \in \{1, \ldots, n\}$
   1. For $i \in \{1, \ldots, n\} - \mathcal{F}$
      1. Let $\mathcal{F}_i = \mathcal{F} \cup \{i\}$
      2. Train on $\mathcal{F}_i$ with cross validation and obtain the cross validation error
   2. Let $\mathcal{F} = \mathcal{F} \cup \{i\}$, where i is the feature that gave the lowest cross validation error.
3. Select the feature set that gave the lowest cross validation error over all tests.

When we have a method that wraps our learning algorithm, supplying a different feature set on each iteration, this is called wrapper model selection.

Another wrapper method is backward search, in which we start with $\mathcal{F} = \{1, \ldots, n\}$ and iteratively remove the least informative feature.

Wrapper methods work well but take a lot of time, for example, $O(kn^2)$ calls to the optimization algorithm for forward selection with k-fold cross validation.

Filter feature selection methods use a heuristic to decide which subsets of features to try.

Example: we may rank features according to some measure of relatedness to the desired output, then incrementally add features in that order if they improve generalization.

Most common measure of relatedness for discrete features: mutual information $MI(X_i, Y)$ between feature X_i and target Y:

$$MI(X_i, Y) = \sum_{x_i \in X_i} \sum_{y \in Y} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$

[Think about the value of $MI(X_i, Y)$ if X_i and Y are independent or identical and why that would be a good indication of relatedness.]

Outline

We've seen several specific techniques for regularization with different models so far.

Here we'll discuss the general framework of Bayesian statistics and how the Bayesian perspective on parameter estimation can be used for regularization.

The frequentist perspective in statistics is that our parameter vector $\boldsymbol{\theta}$ is constant but unknown.

Maximum likelihood is a general technique to identify such a constant but unknown parameter.

The Bayesian perspective in statistics is that $\boldsymbol{\theta}$ is itself a random variable whose value is unknown.

The frequentist approach of maximum likelihood says that given $S = \{(\mathrm{x}^{(i)}, y^{(i)})\}_{i=1..m}$, we should find

$$\boldsymbol{\theta}_{ML} = \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta}).$$

The Bayesian approach says we should consider the posterior distribution over the parameters

$$\begin{aligned} p(\boldsymbol{\theta} \mid S) &= \frac{p(S \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(S)} \\ &= \frac{\left(\prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}, \boldsymbol{\theta})\right) p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} \left(\prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}, \boldsymbol{\theta}) p(\boldsymbol{\theta})\right) d\boldsymbol{\theta}} \end{aligned}$$

when making a decision.

Both probabilities $p(y^{(i)} \mid \mathrm{x}^{(i)}; \boldsymbol{\theta})$ (frequentist) and $p(y^{(i)} \mid \mathrm{x}^{(i)}, \boldsymbol{\theta})$ (Bayesian) are given by the model.

For logistic regression, for example, we would use $h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)})^{y^{(i)}}(1 - h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}))^{1-y^{(i)}}$ with $h_{\boldsymbol{\theta}}(\mathrm{x}^{(i)}) = \frac{1}{1+e^{\boldsymbol{\theta}^\top \mathrm{x}^{(i)}}}$.

Decision making would treat $\boldsymbol{\theta}$ as random:

$$p(y \mid \mathrm{x}, S) = \int_{\boldsymbol{\theta}} p(y \mid \mathrm{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid S) d\boldsymbol{\theta}$$

[How did we get that??]

Then our answer might be the expected value of y given x:

$$\mathbb{E}[y \mid \mathrm{x}, S] = \int_{y} y p(y \mid \mathrm{x}, S) dy$$

Usually, the full Bayesian predictor on the previous slide is too hard to compute in practice.

Instead, we can go with a single point estimate for $\boldsymbol{\theta}$, leading to the maximum a posteriori (MAP) estimate of $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_{MAP} = \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{m} p(y^{(i)} \mid \mathrm{x}^{(i)}, \boldsymbol{\theta}) p(\boldsymbol{\theta}).$$

Note that this is the same as the ML estimate except for the prior term $p(\boldsymbol{\theta})$. One possible prior is $\boldsymbol{\theta} \sim \mathcal{N}(0, \tau^2 \mathrm{I})$.

This will assign higher scores to models with smaller parameter vectors, reducing the model's variance.

Bayesian logistic regression works well for many problems such as text classification where $n \gg m$.