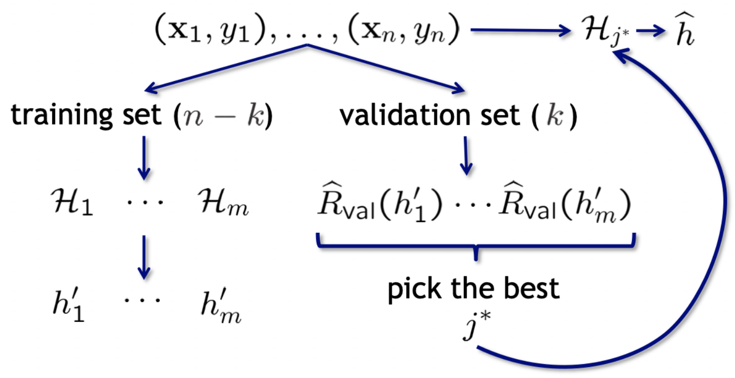
ECE 0402 - Pattern Recognition

Lecture 14

Recap:

We have talked about model selection dilemma:

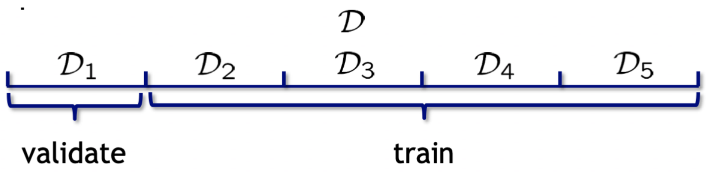
- we need to select appropriate values for the free parameters
- these free parameters usually control the balance between underfitting and overfitting
- but all we have training data to select parameters
- they were left “free” precisely because we don’t want to let the training data influence their selection, as this almost always leads to overfitting.
- **Using validation for model selection**: Suppose we have m models $\mathcal{H}_1, ..., \mathcal{H}_m$:
  - Take the training data (n samples)
  - Split it up into two pieces:
    * training set: $n-k$ samples
    * validation (holdout) set: k samples
    * train m models using the $n-k$ samples : $h'_1, ..., h'_m$.
    * using k samples validation set, get an estimate of risk: $\hat{R}_{val}(h'_1), ..., \hat{R}_{val}(h'_m)$
    * pick the model that gives the best result, lowest $\hat{R}_{val}(h'_{j^*}) \implies j^*$.
    * after picking the model h_{j^*}, re-train this on the entire data set (n-samples) to output $\hat{h}$.


$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$
$\mathcal{H}_{j^*}$
$\widehat{h}$
training set ($n-k$)
validation set (k)
$\mathcal{H}_1 \cdots \mathcal{H}_m$
$\widehat{R}_{\text{val}}(h'_1) \cdots \widehat{R}_{\text{val}}(h'_m)$
$h'_1 \cdots h'_m$
pick the best
j^*

We also talked about more complicated ideas like **“cross-validation”**. Try a different split each time and repeat the whole process (the one depicted in the figure) again, and finally average all of the empirical estimates $\hat{R}_{val}$.

- Leave one out: Train n times on $n-1$ points each. This is ideal, but computationally demanding
- $k-fold$ cross validation: Train k times on $n-\frac{n}{k}$ points each

Notation clarification: in here k is the number of folds $k' = \frac{n}{k}$ is the size of the validation set. Iterate over all 5 choices of validation set, and average. **Remarks**:



5-fold

- For $k-fold$ cross validation, the estimate depends on the particular choice of partition.
- It is common to form several estimates based on different partitions and then average them
- One thing to keep in mind: 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

In practice, cross-validation is a common choice to pick these free parameters in an automatic way. But there are other strategies. I wanna mention just one of them here briefly because it can be useful when you have smaller datasets.

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 result
- Fix $B \geq 1$
- For $b = 1, ..., B$ let $\mathcal{D}_b$ be a subset of size n obtained by <u>sampling with replacement</u> from the full data set $\mathcal{D}$.

  Example: $n = 5$

$$\begin{aligned}
\mathcal{D}_1 &= (x_4, y_4), (x_3, y_3), (x_5, y_5), (x_4, y_4), (x_1, y_1) \\
\mathcal{D}_2 &= (x_1, y_1), (x_2, y_2), (x_5, y_5), (x_5, y_5), (x_2, y_2) \\
&\vdots
\end{aligned}$$

- Define $h_b :=$ model learned base on the data $\mathcal{D}_b$
- And define $\mathcal{D}_b^c := \mathcal{D} \backslash \mathcal{D}_b$
  - In the example this could of been for $\mathcal{D}_1$, (x_2, y_2)
  - for $\mathcal{D}_1$, (x_3, y_3) and (x_4, y_4)
- Set $e_b = \frac{1}{|\mathcal{D}_b^c}| = \sum_{(x_i,y_i)\in\mathcal{D}_b^c} 1_{\{h_b(x_i)\neq y_i\}}$

The **bootstrap error estimate** is given by

$$\hat{R}_B := \frac{1}{B}\sum_{b=1}^{B} e_b$$

Typically, in practice, B must be large, several hundred (say $B \approx 200$) (or more if you can afford it) for the estimate to be accurate. It is very computationally demanding, but also you can get nice confidence intervals in addition to $\hat{R}_B$.

- The bootstrap error estimate $\hat{R}_B$ 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"**. This is really it's name and these weights are actually derived from theory (not a rule of thumb, even though it looks totally like one). And I am sure you can find $5k+$ papers on bootstrap error estimates...

$$0.632\ \hat{R}_B + 0.368\ \hat{R}_{train}$$

- The "balanced" bootstrap chooses $\mathcal{D}_1, ..., \mathcal{D}_B$ such that each input-output pair appears exactly B times.

Linear Methods for Supervised Learning In terms of linear techniques, so far, we have talked about:

- LDA
- Logistic Regression
- Naive Bayes
- PLA
- Maximum margin hyperplanes
- Soft-margin hyperplanes

- Least squares regression
- Ridge regression

Sometimes linear methods (in both regression and classification) just don't work. One way to create "nonlinear estimators" or classifiers is to first transform the data via a nonlinear feature map $\phi : \mathbb{R} \to \mathbb{R}^p$.

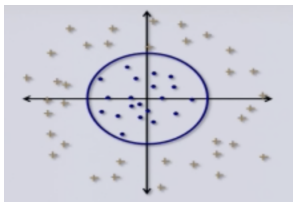
After applying ϕ, we can then try applying a linear method to transformed data $\phi(x_1), ..., \phi(x_n)$. We actually talked about couple of examples of this already. For example, in the case of regression, our model becomes, $f(x) = \beta^T \phi(x) + \beta_0$ where $\beta \in \mathbb{R}^p$. Fitting linear models to nonlinear features...

Example: Suppose $d = 1$ but $f(x)$ is a cubic polynomial, how do we find a least squares estimate of f from training data?

$$\phi_k(x) = x^k \implies A = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

And we have talked about classification long time ago too...

Pictured data set is not linearly separable but can be in a higher dimension with a transform:



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

This dataset is linearly separable after applying such transformation with w=$[-1, 0, 0, 1, 1]^T$

Potential Problems with nonlinear feature maps

Suppose we transform our data via

$$x = \begin{bmatrix} x(1) \\ \vdots \\ x(d) \end{bmatrix} \xrightarrow{\phi} \Phi(x) = \begin{bmatrix} \Phi^{(1)}(x) \\ \vdots \\ \Phi^{(p)}(x) \end{bmatrix}$$

where typically $p \gg d$. The challenges that could arise:

- If $p \gtrapprox n$, then this can lead to an ill-conditioned problem

  - can be mitigated via regularization
  - in addition, when p is very large, this can cause an increase in computational burden
    * (e.g., inverting $p \times p$ matrix)

The "kernel trick"

Shortly, do the mapping without the cost.

There is a clever way to get around this computational challenge by exploiting two facts:

- Many ML algorithms only involve the data through inner products.
- For many interesting feature maps ϕ, the function

$$k(x, x') := \langle \phi(x), \phi'(x) \rangle$$

  has a simple, closed form expression that can be evaluated **without explicitly calculating** $\phi(x)$ and $\phi'(x)$.

Example: Quadratic kernel $k(u, v) = (u^T v)^2$

Suppose $d = 2$ and try to think backwards: "what was the feature map that this kernel is computing an inner product for?"

$$\begin{aligned}
k(\mathrm{u,v}) &= (\mathrm{u}^T\mathrm{v})^2 \\
&= \left([u(1)u(2)] \begin{bmatrix} v(1) \\ v(2) \end{bmatrix} \right)^2 \\
&= (u(1)v(1) + u(2)v(2))^2 \\
&= u(1)^2 v(1)^2 + 2u(1)u(2)v(1)v(2) + u(2)^2 v(2)^2 \\
&= \langle \phi(\mathrm{u}), \phi(\mathrm{v}) \rangle
\end{aligned}$$

$$\phi(\mathrm{u}) = \begin{bmatrix} u(1)^2 \\ \sqrt{2}u(1)u(2) \\ u(2)^2 \end{bmatrix}$$

Now suppose d is arbitrary, and

$$\begin{aligned}
k(\mathrm{u,v}) &= (\mathrm{u}^T\mathrm{v})^2 \\
&= \left(\sum_{i=1}^{d} u(i)v(i)\right)^2 \\
&= \left(\sum_{i=1}^{d} u(i)v(i)\right)\left(\sum_{j=1}^{d} u(j)v(j)\right) \\
&= \sum_{i=1}^{d}\sum_{j=1}^{d} u(i)v(i)u(j)v(j)
\end{aligned}$$

What is the feature mapping in this case? And what is the dimension p of the corresponding feature space?

$$\sum_{i=1}^{d}\sum_{j=1}^{d} u(i)v(i)u(j)v(j) \overset{?}{=} \langle \phi(\mathrm{u}), \phi(\mathrm{v})\rangle$$

$$\phi(\mathrm{u}) = \left[u(1)^2, ..., u(d)^2, ..., \sqrt{2}u(1)u(2), ..., \sqrt{2}u(d-1)u(d)\right]^T$$

$\therefore p = d + \frac{d(d-1)}{2}$

So we can do this for quadratic. what happens if we have cubic kernel?

Example: Cubic kernel $k(u,v) = (u^T v)^3$

In $\mathbb{R}^2$, we can work this out, it is not that much worse, but boring...

$$\begin{aligned}
k(\mathrm{u,v}) &= \big(u(1)v(1) + u(2)v(2) + u(3)v(3)\big)^3 && (1)\\
&= u(1)^3v(1)^3 + 3u(1)^2u(2)v(1)^2v(2) + 3u(1)u(2)^2v(1)v(2)^2 + u(2)^3v(2)^3 && (2)\\
&= \sum_{i=0}^{3}\binom{3}{i} u(1)^{3-i}u(2)^i \cdot v(1)^{3-i}v(2)^i && (3)
\end{aligned}$$

$$\phi(\mathrm{u}) = \left[u(1)^3, \sqrt{3}u(1)^2u(2), \sqrt{3}u(1)u(2)^2, u(2)^3\right]^T$$

and we could do this in d dimensions if we felt like it...But it is probably okay, if we don't...

So let me state what happens in general,

Polynomial Kernels:

$$\begin{aligned}k(\mathrm{u},\mathrm{v}) &= (\mathrm{u}^T\mathrm{v})^m \\ &= \sum_{\substack{\text{partitions}\\ j_1,\ldots,j_d}} \binom{m}{j_1,\ldots,j_d} u(1)^{j_1} v(1)^{j_1} \cdots u(d)^{j_d} v(d)^{j_d}\end{aligned}$$

$$\phi(\mathrm{u}) = \left[\ldots, \sqrt{\tbinom{m}{j_1,\ldots,j_d}} u(1)^{j_1} \cdots u(d)^{j_d}, \ldots\right]^T$$

In English, all possible monomials of degree m.

This is one example of a kernel. There are lots of different kernels we could use. The general kind of kernel we will use in this class is called **inner product kernel**.

Definition: A inner product kernel is a mapping

$$k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$$

for which there exists an inner product space $\mathcal{F}$ and a mapping $\phi : \mathbb{R}^d \to \mathcal{F}$ such that

$$k(u,v) = \langle \phi(\mathrm{u}), \phi(\mathrm{v}) \rangle_{\mathcal{F}}$$

for all $u, v \in \mathbb{R}^d$.

Given a function $k(u,v)$, how can we tell when it is an inner product kernel?

- Mercer's theorem
- Positive semidefinite property

Positive semidefinite kernels

We say that $k(u,v)$ is a positive semidefinite kernel

- if k is symmetric.
- for all n and all $x_1, \ldots, x_n \in \mathbb{R}^d$, the **Gram matrix K** is defined by

  $$K(i,j) = k(x_i, x_j)$$

  is positive semidefinite, i.e., $x^T K x \geq 0$ for all x.

Theorem: k is an inner product kernel iff k is a positive semidefinite kernel.

Examples:

- Homogeneous polynomial kernel

$$k(u,v) = (u^T v)^m \qquad m = 1, 2, 3, ...$$

- Inhomogeneous polynomial kernel

$$k(u,v) = (u^T v + c)^m \qquad \begin{array}{l} m = 1, 2, 3, ... \\ c > 0 \end{array}$$

  ϕ maps to the set of all monomials of degree $\leq m$.

- Gaussian/Radial basis function (RBF) kernel:

$$k(u,v) = (2\pi\sigma^2)^{-d/2} \exp\left(-\frac{\|u-v\|^2}{2\sigma^2}\right)$$

  You can just do (it doesn't matter if it is a PDF or not):

$$k(u,v) = \exp\left(-\frac{\|u-v\|^2}{2\sigma^2}\right)$$

  One can show that k is a positive semidefinite kernel. but what is $\mathcal{F}$?

  - $\mathcal{F}$ is **infinite dimensional**!