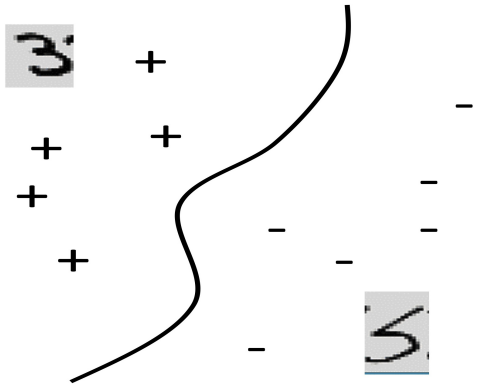
Fundamentals of Learning Course Notes
Cynthia Rudin

Important Problems in Data Mining

1. Classification

   - Input: $\{(x_i, y_i)\}_{i=1}^n$ "examples," "instances with labels," "observations"
   - $x_i \in \mathcal{X} \subset \mathbf{R}^p, y_i \in \{-1, 1\}$ "binary"



   - Output: $f : \mathcal{X} \to \mathbf{R}$ and use $sign(f)$ to classify.
   - Applications: automatic handwriting recognition, speech recognition, biometrics, document classification, detecting fraudulent credit transactions

2. Conditional probability estimation

   - Input: $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathcal{X}, y_i \in \{-1, 1\}$
   - Output: $f : \mathcal{X} \to [0, 1]$ as "close" to $p(y = 1|x)$ as possible.
   - Applications: estimate probability of failure, probability to default on loan

3. Regression

   - Input: $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathcal{X}, y_i \in \mathbf{R}$
   - Output: $f : \mathcal{X} \to \mathbf{R}$
   - Applications: predicting an individual's income, predict house prices, predict demand for energy, predict test scores

4. Ranking - between classification and regression.

- Input: $\{x_i\}_{i=1}^n$, $x_i \in \mathcal{X}$, and also we have a set of pairs (i, k) that are labeled that i should be ranked above k, that is, we should have $f(x_i) > f(x_k)$ for these special pairs.
- Output: $f : \mathcal{X} \to \mathbf{R}$ such that $f(x_i) > f(x_k)$ for the specified (i, k) pairs as often as possible.
- Applications: Search engines use ranking methods. Useful also for predictive maintenance applications.

5. Finding patterns (correlations) in large datasets
   - Input: $\{x_i\}_{i=1}^n$, $x_i \in \mathcal{X}$, (labels can be included optionally)
   - e.g. (Diapers $\to$ Beer). Use frequent itemset mining algorithms.
6. Clustering - grouping data into clusters that "belong" together - objects within a cluster are more similar to each other than to those in other clusters. This is unsupervised.
   - Input: $\{x_i\}_{i=1}^n$, $x_i \in \mathcal{X}$
   - Output: $f : \mathcal{X} \to \{1, \ldots, K\}$ (K clusters)
   - Applications: clustering consumers for market research, clustering genes into families, image segmentation (medical imaging)
7. Density estimation. This is unsupervised.
   - $\{x_i\}_i$, $x_i \in \mathcal{X}$
   - Output: $f : \mathcal{X} \to [0, 1]$ as "close" to $p(x)$ as possible.
   - Applications: anomaly detection (anomalous mechanical behavior of a piece of equipment)

Rule mining, clustering, and density estimation are **unsupervised methods** (no ground truth), whereas classification, ranking, and conditional probability estimation are **supervised methods** (there is ground truth). In all of these problems, we do not necessarily assume we know the distribution (or even the form of the distribution) that the data are drawn from.

Training and Testing (in-sample and out-of-sample) for *supervised learning.*

Training: training data are input, and model f is the output.

$$\{(x_i, y_i)\}_{i=1}^n \Longrightarrow \boxed{\text{Algorithm}} \Longrightarrow f.$$

Testing: Evaluate the model. You want to predict y for a new x, where (x, y) comes from the *same* distribution as $\{(x_i, y_i)\}_{i=1}^n$.

That is, $(x, y) \sim D(\mathcal{X}, \mathcal{Y})$ and each $(x_i, y_i) \sim D(\mathcal{X}, \mathcal{Y})$.

Problems arise in practice when the training data and test data are not drawn from the same distribution.

To judge the quality of predictions, we need to know how to answer: How well does $f(x)$ match the label y?

Measure goodness of f using a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbf{R}$.

For instance

$$\begin{aligned} \ell(f(x), y) &= (f(x) - y)^2 \quad \text{least squares loss, or} \\ \ell(f(x), y) &= \mathbf{1}_{[\text{sign}(f(x)) \neq y]} \quad \text{(mis)classification error} \end{aligned}$$

$$\begin{aligned} R^{\text{test}}(f) &= \mathbf{E}_{(x,y)\sim D} \ell(f(x), y) \\ &= \int_{(x,y)\sim D} \ell(f(x), y) dD(x, y). \end{aligned}$$

R^{test} is also called the **true risk** or the **test error**.

We can't calculate it.

We want R^{test} to be small. If so, that would mean $f(x)$ is a good predictor of y.

How can we ensure $R^{\text{test}}(f)$ is small?

Look at how well f performs (on average) on $\{(x_i, y_i)\}_i$.

$$R^{\text{train}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i).$$

R^{train} is also called the **empirical risk** or **training error**. For example,

$$R^{\text{train}}(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{[\text{sign}(f(x_i)) \neq y_i]}.$$

(How many handwritten digits did f classify incorrectly?)

Say our algorithm constructs f so that $R^{\text{train}}(f)$ is small. If $R^{\text{train}}(f)$ is small, hopefully $R^{\text{test}}(f)$ is too. But that doesn't always happen.

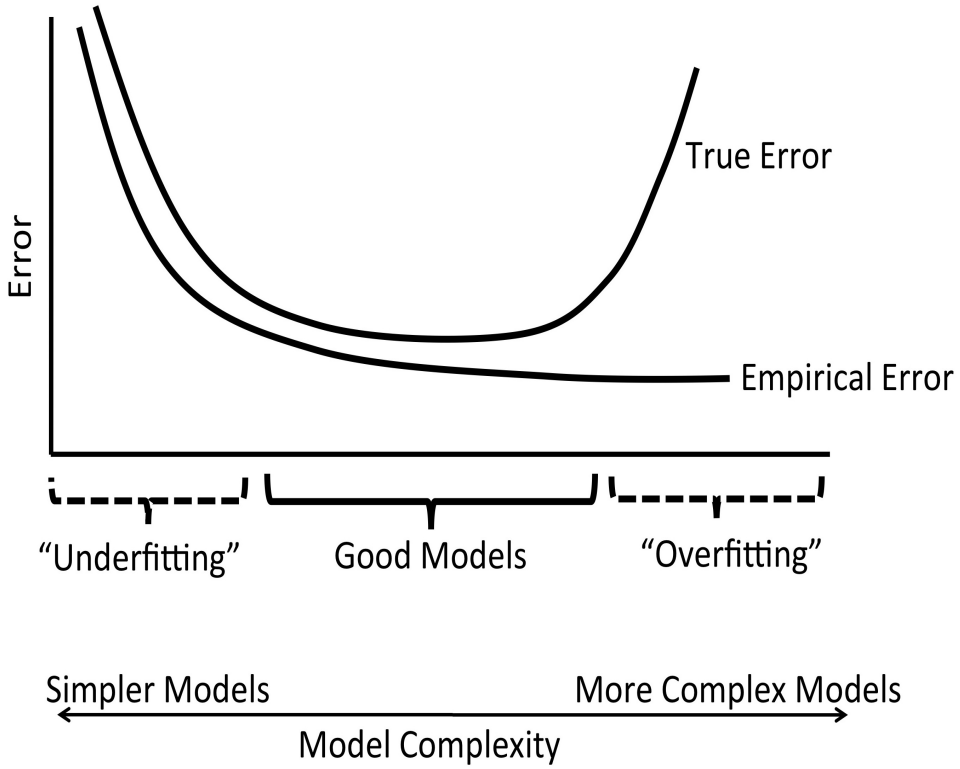
We would like a guarantee on how close R^{train} is to R^{test}. When would it be close to R^{test}?

- If n is large.
- If f is "simple."

Illustration

In one of the figures in the illustration, f: was overfitted to the data, modeled the noise, "memorized" the examples, and didn't give us much other useful information. In other words, it doesn't "generalize," i.e., predict. We didn't "learn" anything!

Computational Learning Theory, a.k.a. Statistical Learning Theory, a.k.a., learning theory, and in particular, Vapnik's **Structural Risk Minimization** (SRM) addresses generalization. Here's SRM's classic picture:



Structural Risk Minimization says that we need models to be somewhat simple in order to learn/generalize (avoid overfitting).

Statistical learning theory addresses how to construct probabilistic guarantees on the true risk. In order to do this, it quantifies classes of "simple models," discussed formally later.

What can we use as a definition of a "simple" model in practice?

There are many definitions. For instance:

- "simple" linear models have small coefficients.

$$f(x) = \sum_j \lambda_j x^{(j)} \text{ where } \|\boldsymbol{\lambda}\|_2^2 = \sum_j \lambda_j^2 < C.$$

- "simple" Bayesian models might be models that have large values of a Bayesian prior.
- "simple" models could be models that are very smooth, and can't change too fast as we move around the space $\mathcal{X}$.

- "simple" models have small values of a "simplicity function" which we generally call a regularization term.

Later we will learn some formal definitions of simplicity.

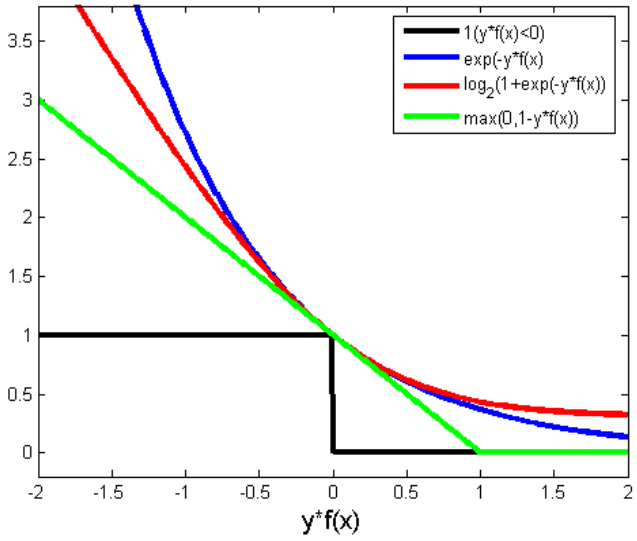
This expression below is kind of omnipresent. This form captures many algorithms: SVM, boosting, ridge regression, LASSO, and logistic regression.

Regularized Learning Expression:

$$\sum_i \ell(f(x_i), y_i) + CR^{\text{reg}}(f)$$

In the regularized learning expression, the loss $\ell(f(x_i), y_i)$ is often misclassification error $\mathbf{1}_{[y_i \neq \text{sign}(f(x_i))]} = \mathbf{1}_{[y_i f(x_i) \leq 0]}$ for classification problems.
Note that minimizing $\sum_i \mathbf{1}_{[y_i f(x_i) \leq 0]}$ with respect to f is computationally hard. This is why we often minimize upper bounds for it that are convex and easier to minimize. Here are some of them:

- "logistic loss" $\log\left(1 + e^{-y_i f(x_i)}\right) \Longleftarrow$ logistic regression
- "hinge loss" $\max(0, 1 - y_i f(x_i)) \Longleftarrow$ SVM
- "exponential loss" $e^{-y_i f(x_i)} \Longleftarrow$ AdaBoost



In the regularized learning expression, we define a couple of options for $R^{\text{reg}}(f)$. Usually f is linear, $f(x) = \sum_j \lambda_j x^{(j)}$. We choose $R^{\text{reg}}(f)$ to be either:

- $\|\boldsymbol{\lambda}\|_2^2 = \sum_j \lambda_j^2 \Longleftarrow$ ridge regression, SVM
- $\|\boldsymbol{\lambda}\|_1 = \sum_j |\lambda_j| \Longleftarrow$ LASSO, approximately AdaBoost

The method called ridge regression is simply the squared loss with ℓ_2 regularization:

$$\frac{1}{n}\sum_i (y_i - f(x_i))^2 + C\|\boldsymbol{\lambda}\|_2^2.$$

Support vector machines simply use the hinge loss and the ℓ_2 norm:

$$\frac{1}{n}\sum_i \max(0, 1 - y_i f(x_i)) + C\|\boldsymbol{\lambda}\|_2^2.$$

AdaBoost uses the exponential loss without regularization, (although it acts as if it uses ℓ_1 regularization in some cases):

$$\frac{1}{n}\sum_i e^{-y_i f(x_i)}.$$

Regularized logistic regression uses the logistic loss with either the ℓ_1 or ℓ_2 norm.

Since the difference between these algorithms in theory is somewhat small, they tend also to perform similarly in practice on many datasets if they use the same features. The catch is that they use very different kinds of features.