

Lecture 2: Supervised learning introduction

CSI 5v93: Introduction to machine learning

Baylor University
Computer Science Department

Dr. Greg Hamerly
<http://cs.baylor.edu/~hamerly/>

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 1/24

General announcements

- First assignment: due January 20th (see the website).
- Get the book ASAP if you don't have it.

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 2/24

Questions?

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 3/24

Chapter 2: Overview of supervised learning

- 2.1 – Introduction
- 2.2 – Variable types and terminology
- 2.3 – Two simple approaches to prediction: Least squares and nearest neighbors
- 2.4 – Statistical decision theory
- 2.5 – Local methods in high dimensions
- 2.6 – Statistical models, supervised learning, and function approximation
- 2.7 – Structured regression models
- 2.8 – Classes of restricted estimators
- 2.9 – Model selection and the bias-variance tradeoff

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 4/24

Least squares (linear regression – 2.3.1)

A linear model is one which is linear *in the inputs*.

Examples:

- linear model: $f(x) = 3x$
- linear model: $f(x, y) = 3x - 10y$
- nonlinear model: $f(x, y) = x^2 - 4\sqrt{y} + xy$
- NOTE! linear model: $f(x, x^2, y, xy) = x^2 - xy + x - y$

Linear models are popular because:

- simple to understand
- simple to find
- stable (high bias)

Linear models make strong assumptions about the relationship between the inputs and the output.

Linear model

Given a vector of inputs $X = (X_1, X_2, \dots, X_p)$, predict the output Y with the model

$$(1) \quad \hat{Y} = \hat{\beta}_0 + \sum_{j=1}^p X_j \hat{\beta}_j$$

Using linear algebra's inner-product (aka dot product) notation we can write this as:

$$(2) \quad \hat{Y} = X^T \hat{\beta}$$

where here β is a vector of all the coefficients, and X^T is the transpose of the vector X .

Two-dimensional example (on the board).

Finding the coefficients

We defined our model: the output (Y) equals the inputs (X) times their respective coefficients (β).

Here, the learning problem is finding the appropriate coefficients β for a given training set. How should we do this?

Our first step is to define an *error function*, which is a common thing in machine learning.

The error function will tell us how well our estimate $\hat{\beta}$ is doing at describing the training data.

Least squares error function

The most common method for finding coefficients of a linear function is *least squares*.

Error function:

$$(3) \quad \text{RSS}(\beta) = \sum_{i=1}^n (y_i - x_i^T \beta)^2$$

We want to *minimize* this error function for a fixed training set.

Minimization should remind you of calculus: take the derivative with respect to β , set to zero. . .

Remember that β is a vector.

Least squares error function

We can rewrite the last equation as:

$$(4) \quad \text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$$

Taking the derivative with respect to β , setting to zero, we get:

$$(5) \quad \mathbf{X}^T (\mathbf{y} - \mathbf{X}\beta) = 0$$

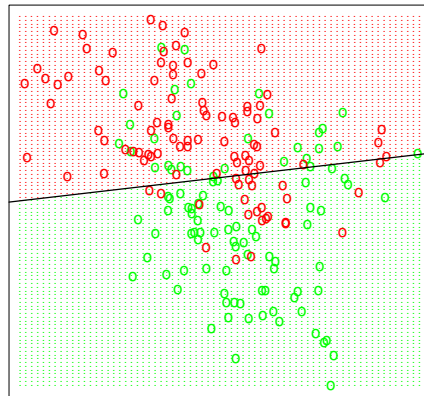
If $\mathbf{X}^T \mathbf{X}$ is nonsingular, then the unique solution is given by:

$$(6) \quad \hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

We will derive this later, see section 3.2 in your book.

Linear regression application

Linear Regression of 0/1 Response



Input: 2-dimensions. Output: **GREEN** or **RED** (coded as 0/1).

Note: we have used a *regression* model for a *classification* problem. Predict **RED** if $\hat{Y} > 0.5$, or **GREEN** if $\hat{Y} \leq 0.5$.

The linear model makes strict assumptions. If the data does not fit a linear model we have misclassifications.

Nearest neighbors (2.3.2)

Instead of using a linear model, let's use the points near the query to classify it.

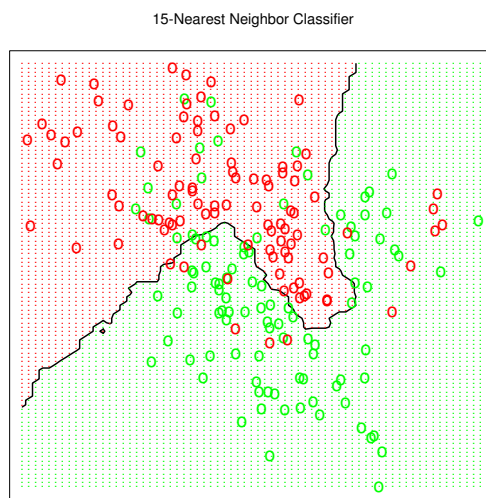
$$(7) \quad \hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

Where the $N_k(x)$ is the neighborhood of k points closest to x .

This is an average of the nearest neighbors.

Here we don't assume a linear model, or any model. We do have to choose k , however.

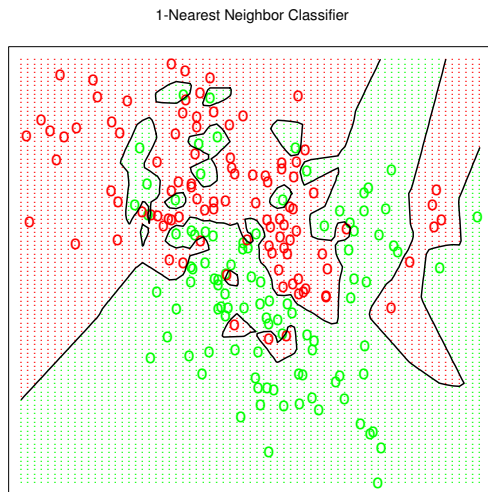
15-Nearest neighbor classifier



We still make misclassifications, but we make fewer than the linear model.

Note the shape of the boundary, which is determined by the red/green points.

1-Nearest neighbor classifier

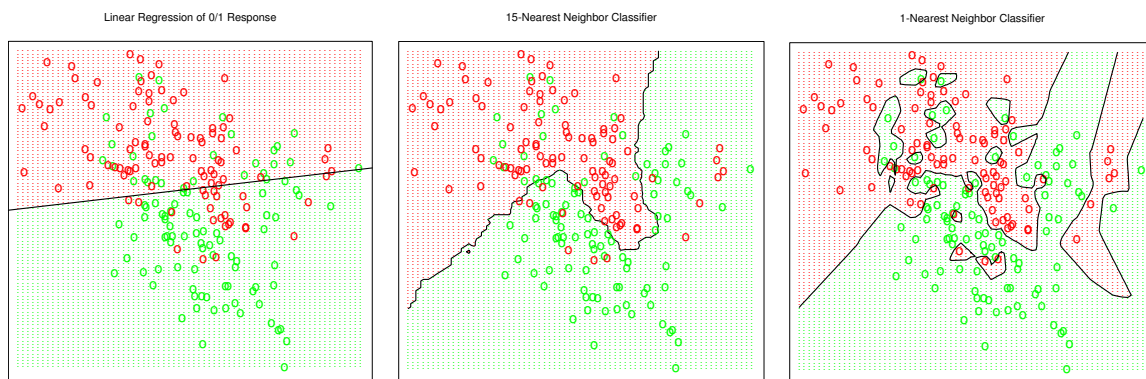


We can get *perfect* performance on the training set if we use $k = 1$.

However, this does not generalize well.

Number of parameters to estimate

Small # parameters \longrightarrow large # parameters



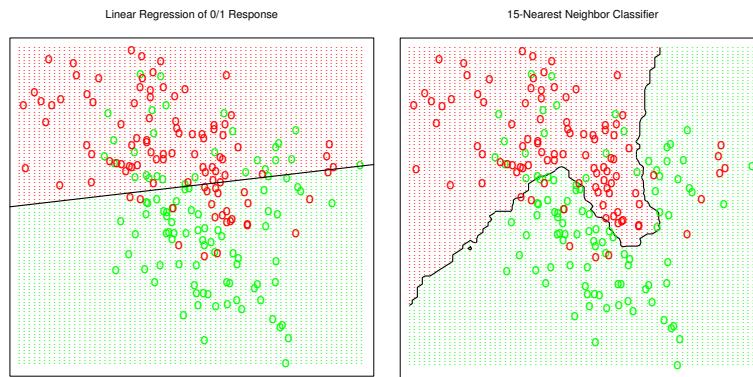
For linear regression, we estimate $d + 1$ parameters, where here $d = 2$.

For nearest-neighbors, we estimate roughly N/k parameters.

More parameters \longrightarrow more flexibility \longrightarrow harder to learn \longrightarrow need lots of data.

Machine learning must deal with the tradeoff between model flexibility and model stability/learnability. This is often called the *bias-variance tradeoff*.

Bias and variance



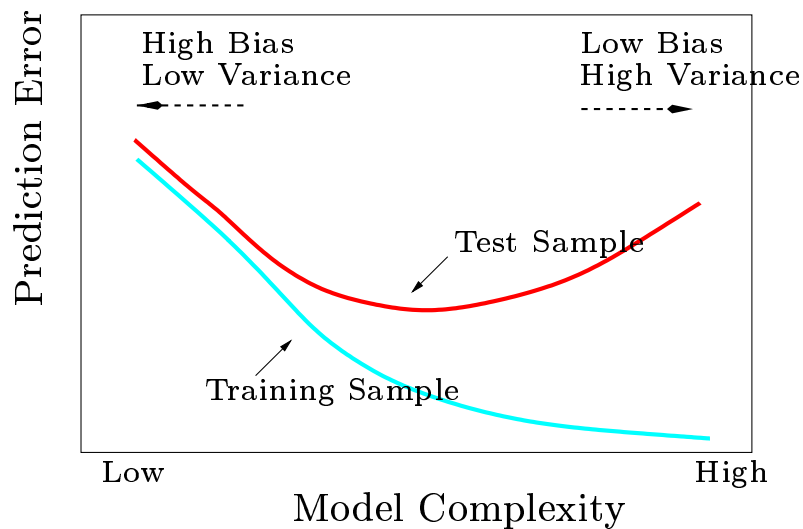
Every model has some bias and some variance.

A strict model (like linear regression) has high bias and low variance. Better with smaller training sets.

A loose model (like nearest neighbor) has low bias and high variance. Better with larger training sets.

The bias and variance trade off depending on the model – we will see more later.

Bias-variance and model complexity (2.9)



Another way to look at bias-variance and model complexity. On the left we are *underfitting*, on the right we are *overfitting*.

Quick sidebar: expectations

Expectation: weighted mean of a variable, where weights come from a probability distribution:

Expectations for continuous variable x and discrete variable y :

$$(8) \quad E[f(x)] = \int_{-\infty}^{\infty} f(x) P(x) dx$$

$$(9) \quad E[g(y)] = \sum_y g(y) P(y)$$

Expectations are linear:

$$(10) \quad E[ax + by] = aE[x] + bE[y]$$

You should be familiar with expectations.

Statistical decision theory (2.4)

Given: $X \in \mathbb{R}^d$, $Y \in \mathbb{R}$, and joint distribution $P(X, Y)$.

We want a function $f(X)$ that predicts Y given X .

We need a *loss function* $L(Y, f(X))$ for penalizing prediction errors. The simplest and easiest is squared-error loss:

$$(11) \quad EPE(f) = E[(Y - f(X))^2]$$

$$(12) \quad = \int (y - f(x))^2 \Pr(dx, dy)$$

See your book for derivations, but this is minimized when

$$(13) \quad f(x) = E[Y|X = x]$$

Thus the minimum-error solution is when we take the weighted mean of the output variable (Y) at $X = x$.

Averaging over data

Nearest neighbor methods average over training data in a neighborhood near the input point:

$$(14) \quad \hat{f}(x) = \text{Ave}(y_i | x_i \in N_k(x))$$

This is an approximation to computing the expected output for the given input.

Linear regression also averages over the training data.

Both compute averages, but have dramatic differences:

- least-squares assumes $f(x)$ is well-approximated by a globally linear function
- k -nearest neighbors assumes $f(x)$ is well-approximated by a locally constant function

Classifiers and Bayes classifier

For a classifier, we define the loss function as $L(k, l)$ is the cost of classifying an observation of true class \mathcal{G}_k as \mathcal{G}_l

If we assume that the losses are 0 or 1:

- $L(k, l) = 0$ if $k = l$
- $L(k, l) = 1$ if $k \neq l$

then we obtain the *Bayes classifier*:

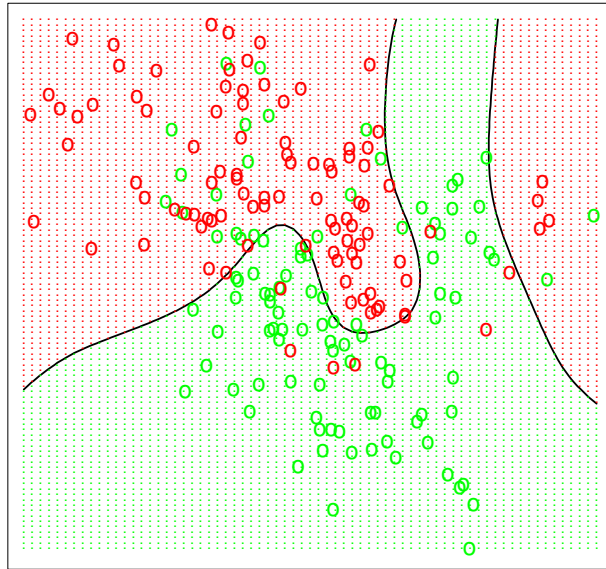
$$(15) \quad \hat{G}(X) = \mathcal{G}_k \text{ if } \Pr(\mathcal{G}_k | X = x) = \max_{g \in \mathcal{G}} \Pr(g | X = x)$$

The Bayes classifier always chooses the most probable class.

The error rate of the Bayes classifier is called the Bayes error rate.

Optimal Bayes classifier

Bayes Optimal Classifier



The classifier shown here comes from the known distribution that generated the data, not from the data itself. Therefore it is optimal.

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 21/24

Dimensionality problems (2.5)

Even though the k -nearest neighbor classifier seems like it will do everything we need, it has problems when the input dimension is high.

This is known as the “curse of dimensionality”, and it affects every learning algorithm (though some worse than others).

Basic idea: in higher dimensions, everything looks far away.

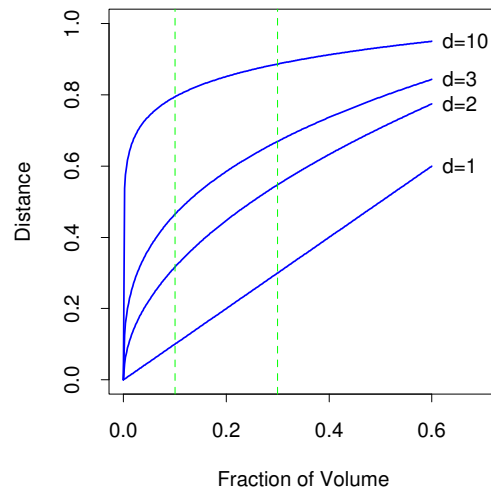
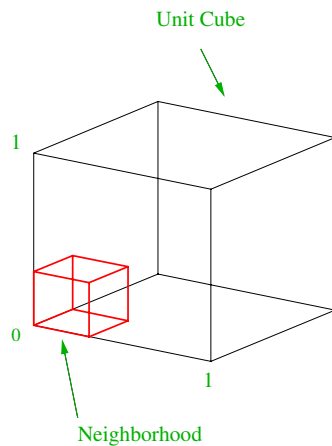
Example explanation: the Euclidean distance formula is

$$(16) \quad d(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$$

As d increases, there are more added terms. Distances get large.

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 22/24

The curse: another example



Right graph shows the required sidelength to capture the given fraction of volume in a unit hypercube for different dimensions.

For $d = 10$, to capture just 10% of the volume requires a box with sidelength 80%!

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 23/24

2-minute journal

Please write a response to the following on a piece of paper and hand it in immediately. Please make it anonymous (no names). Write about:

- major points you learned today
- areas not understood or requiring clarification

CSI 5v93: Introduction to machine learning, Lecture 2 – p. 24/24