2 Overview of Supervised Learning

2.2 Variable Types

- *Quantitative* measurement: implicitly assume an ordering an a similarity between values
- *Qualitative, categorical* or *discrete* measurement: belonging to a finite set of elements, without any explicit ordering among those
- *Ordered categorical* measurement: there exists an ordering between the possible finite set of values, but no metric notation is appropiate

Prediction Tasks

- *Regression*: target is quantitative
- *Classification*: target is qualitative

2.3 Least Squares and Nearest Neighbors

2.3.1 Linear Models and Least Squares

Given a vector of inputs $X^T = (X_1, ..., X_p)$, predict the output Y via the model

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

Including the constant variable 1 in X, include $\hat{\beta}_0$ in the vector of coefficients $\hat{\beta}$, rewrite the model as an inner product

$\hat{Y} = X^T \hat{\beta}$

How do we fit the linear model to a set of training data? One can do so by the method of *least squares.*

$RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$

where $\mathbf{X}$ is a $N \times p$ matrix with each row an input vector, and $\mathbf{y}$ is an N-vector. Differentiating w.r.t β we get the *normal equations*

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

If $\mathbf{X}^T \mathbf{X}$ is non-singular, then the unique solution is given by

$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = 0$

and the fitted value at the i th input x_i is $\hat{y}_i = \hat{y}(x_i) = \hat{x_i}^T \hat{\beta}$.

2.3.2 Nearest-Neighbor Methods

The k-nearest neighbor fit for $\hat{Y}$ is defined as the average of the responses of the closest neighbors:

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

where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample.

For k-nearest-neighbor fits, the error on the training data should be approximately an increasing function of k, and will always be 0 for $k = 1$. Although neastest neighbors has a single parameter k, it is the case that the *effective* number of parameters of k-nearest neighbors is N/k and is generally bigger than p, and decreases with increasing k.

It is also clear that we cannon use sum-of-squared errors on the training set as a criterion for picking k, since we would always pick $k = 1$.

2.4 Statistical Decision Theory

Write the expected (squared) prediction error as

$$\begin{aligned} EPE(f) &= E(Y - f(X))^2 \\ &= \int [y - f(x)]^2 Pr(dx, dy) \end{aligned}$$

By conditioning on X, we can write EPE as

$$EPE(f) = E_X E_{Y|X}([Y - f(X)]^2 | X).$$

The solution is

$$f(x) = E(Y|X = x),$$

the conditional expectation, also known as the *regression* function. Thus the best prediction of Y at any point $X = x$ is the conditional mean, as per the averaged squared error.

The nearest-neighbor methods attempt to directly implement this formula using the training data, taking the average in a neighbourhood of the input:

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

where *Ave* denotes average. Two approximations are happening here:

- expectation is approximated by averagin over sample data;
- conditioning at a point is relaxed to conditioning on some region "close" (in training space) to the target point.

As $N, k \to \infty$, with $k/N \to 0$ (sample size grows faster than k), $\hat{f}(x) \to E(Y|X = x)$. But we do not always have a very large sample, and in those cases a more structure model yields more stable, better performance. Moreover, as the dimension p grows, so does the metric size of k-nearest neighbors. The rate of convergence decreases as the dimension increases.

Looking back at linear regression, which assumes that $f(x) \approx x^T\beta$, we find that by plugging in the liear model into EPE and differentiating, we can solve for β theoretically:

$$\beta = [E(XX^T)]^{-1}E[XY],$$

and the least squares solution previously computed amounts to replacing the expectations by averages over the training data.

What happens if we replace the L_2 loss function with the $L_1 : E|Y - f(X)|$. The solution in this case is the conditional median,

$$\hat{f}(x) = median(Y|X = x),$$

which is a different measure of location.

When the output is a categorical variable G

The same paradigm works here, except we need a different loss function for penalizing prediction errors. Our loss function can be represented by a $K \times K$ matrix $\mathbf{L}$, where $K = card(\mathcal{G})$. $\mathbf{L}$ will be zero on the diagonal and nonnegative elsewhere, where $L(k, l)$ is the price paid for classifying an observation belonging to class $\mathcal{G}_k$ as $\mathcal{G}_l$. Most often we use the *zero-one* loss function. The expected prediction error is

$$EPE = E[L(G, \hat{G}(X))],$$

where the expectation is taken w.r.t the joint distribution $P(G, X)$. Again we condition, and can write EPE as

$$EPE = E_X \sum_{k=1}^{K} L[\mathcal{G}_k, \hat{G}(X)]P(\mathcal{G}_k|X),$$

and again it suffices to minimize EPE pointwise:

$$\hat{G}(x) = argmin_{g \in \mathcal{G}} \sum_{k=1}^{K} L[\mathcal{G}_k, g] P(\mathcal{G}_k | X = x).$$

With the 0-1 loss this simplifies to

$$\hat{G}(x) = argmin_{g \in \mathcal{G}} [1 - P(g|X = x)]$$

or simply

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

Tthis reasonable solution is known as the *Bayes classifier*, and says that we classify to the most probable class, using the conditional distribution $P(G|X)$.

Another way of representing the Bayes classifier, when the target is encoded as a dummy variable, is to set-up a regression task whereby a target is predicted in the $[0, 1]$ range for every dummy target class, and the one with the highest probability is taken as the prediction.

Exercise 2.2

For each class, 10 centers are generated: $ { b_1, b_2, . . . , b_{10} }$ for Blue, and $ { o_1, o_2, . . . , o_{10} }$ for Orange.

The decision boundary for the classifier can be defined by equating the probability of each class conditioned to the data:

$$P(\mathcal{G}_1|X) = P(\mathcal{G}_2|X)$$

Using the Bayes rule, we can write this expression as

$$P(\mathcal{G}_1)P(X|\mathcal{G}_1) = P(\mathcal{G}_2)P(X|\mathcal{G}_2),$$

and since we know that $P(X|\mathcal{G}_k) \sim N([m_1, ..., m_{10}], \mathbf{I}/5)$, and assuming $P(\mathcal{G}_1) = P(\mathcal{G}_2)$, we can write

$$\sum_{i=1}^{5} exp\left(-0.5(\frac{x - b_i}{1/5})^2\right) = \sum_{i=1}^{5} exp\left(-0.5(\frac{x - o_i}{1/5})^2\right)$$

which defines the decision boundary.

2.5 Local Methods in High Dimensions

Intuitively, it might seem that we could always find a sufficiently large training sample that allows us to approximate the theoretically optimal conditional expectation by k-nearest-neighbor averaging, provided we use a sufficiently large neighbourhood. This approach breaks down in high dimensions; this phenomenon is commonly called the *curse of dimensionality.*

As we increase the dimension, the proportion of possible dynamic range of the inputs that we need to consider for a neighborhood becomes larger for including the same amount of sample points in the neighbourhood.

Another consequence of the sparse sampling in high dimensions is that all sample points are close to an edge of the sample. Consider N data points uniformly distributed in a p-dimensional unit ball centered at the origin. The median distance from the origin to the closest data point is given by the expression

$$d(p,N) = \left(1 - \frac{1}{2}^{1/N}\right)^{1/p}$$

A more complicated expression exists for the mean distance to the closest point.

The reason this presents a problem is that prediction is much more difficult near the edges of the reaining samples, where one must extrapolate from neighboring sample points rather than interpolate between them.

Exercise 2.3

We have N points uniformly distributed inside a unit ball centered around the origin.

The cumulative distribution function (CDF) of the distance between a point and the center $y = ||x_i||$ is given by the quotient between the volumes of the smaller sphere of radius y and that of the unit ball, that is, $F(y) = y^p$. The pdf is then $f(y) = py^{p-1}$.

From order statistics, we know that the CDF of the smallest distance Y_1 among all N data points is

$$F_{Y_1}(y) = P(min\{Y_1, ..., Y_n\} \leq y) = 1 - (1 - F(y))^N = 1 - (1 - y^p)^N,$$

and the median distance to the closest data point can then be found by solving

$$\frac{1}{2} = F_{Y_1}(y).$$

This gives

$$d(p,N) = \left(1 - \frac{1}{2}^{1/N}\right)^{1/p} \equiv d_{median}(p,N)$$

From what we have above, we can find the average distance to the closest data point by deriving $ F_{Y_1} (y)$ and computing the expected value:

$$f_{Y_1}(y) = N(1-y^p)^{N-1}py^{p-1}.$$

Appying the definition of the expectation:

$$pN\int_0^1 y(1-y^p)^{N-1}y^{p-1}dy = pN\int_0^1 (1-y^p)^{N-1}y^p dy$$

which we cannot evaluate, but can relate to the *Beta* function:

$$B(a,b) \equiv \int_0^1 t^{a-1}(1-t)^{b-a}dt$$

If we let $y^p = t$, $dy \cdot py^{p-1} = dt$

$$d_{mean}(p,N) \equiv pN\int_0^1 (1-t)^{N-1}t^{1/p}\frac{1}{p}dt$$

$$= N\int_0^1 (1-t)^{N-1}t^{\frac{1}{p}}dt = NBeta(1+\frac{1}{p}, N+\frac{1}{p})$$

Another manifestation of the curse of dimensionality is that the sampling density is proportional to $N^{1/p}$.

Suppose, on the other hand, that we know that the relationship between Y and X is linear,

$$Y = X^T\beta + \epsilon,$$

where $\epsilon \sim N(0,\sigma^2)$ and we fit the model by least squares to the training data. For an arbitrary test point x_0, we have $\hat{y_0} = x_0^T\hat{\beta}$ which can be rewritten as $\hat{y_0} = x_0^T\beta + \sum_{i=1}^N l_i(x_0)\epsilon_i$, where $l_i(x_0)$ is the i-th element of $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}x_0$. Notice that $\hat{y_0}$ is an unbiased estimate of y_0, and the prediction error (assuming the mechanism for generating testing data is the same as for training data) is due to the variance introduced by the noise ϵ. Using this last remark, we find that

$$EPE(x_0) = \sigma^2 + E_{\mathcal{T}}x_0^T(\mathbf{X}^T\mathbf{X})^{-1}x_0\sigma^2$$

Exercise 2.5 a)

First Solution

Remember we assumed that the observations are generated with a linear model plus an additive noise. Write the MSE as

$$MSE = (y_0 - \hat{y_0})^2 = ((y_0 - x_0^T\beta) - (x_0^T\beta - x_0^T\hat{\beta}))^2 = (A - B)^2.$$

Note that the first term depends on the noise introduced at "test" time, while B depends on the noise in the training sample, therefore the cross term of the expression above evaluates to 0. Now we can write

$$EPE = E_{\mathcal{T}}E_{\epsilon}[A^2 - B^2] = A^2 + B^2$$

The first term is nothing but the variance σ^2 introduced by the noise. If we recall that $\hat{y_0}$ is an unbiased estimate of y_0, we realize the second term is the variance of the predictions. Therefore

$$EPE = \sigma^2 + x_0^T(\mathbf{X}^T\mathbf{X})^{-1}x_0\sigma^2$$

and now taking the expectation w.r.t x_0 we end up with

$$EPE(x_0) = \sigma^2 + x_0^T E_{\mathcal{T}}(\mathbf{X}^T\mathbf{X})^{-1}x_0\sigma^2$$

which is what we wanted.

Second Solution

$$MSE = (y_0 - \hat{y_0})^2 = y_0^2 - 2y_0\hat{y_0} + \hat{y_0}^2$$

Since

$$Var(X) = E(X^2) - E^2(X)$$

we can write

$$E_{y_0|x_0}y_0^2 = Var(y_0) + E^2_{y_0|x_0}y_0$$

and similarly

$$E_{\mathcal{T}}\hat{y_0}^2 = Var(\hat{y_0}) + E^2_{\mathcal{T}}\hat{y_0}.$$

Also, since

$$y_0 = x_0^T\beta + \epsilon$$

we get

$$\begin{aligned} E_{y_0|x_0}y_0 &= E_{y_0|x_0}[x_0^T\beta + \epsilon] = x_0^T\beta \\ Var(y_0) &= Var(x_0^T\beta + \epsilon) = \sigma^2 \end{aligned}$$

and also write, taking advantage of the *normal equations* for β:

$$\begin{aligned} \hat{y_0} = x_0^T\hat{\beta} &= x_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\beta + \epsilon) \\ &= x_0^T\beta + x_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\epsilon \\ &= x_0^T\beta + b^T\epsilon. \end{aligned}$$

Recalling the initial expression,

$$\begin{aligned} EMSE &= Var(y_0) + E^2_{y_0|x_0}y_0 - 2E_{y_0|x_0}[y_0]E_{\mathcal{T}}[\hat{y_0}] + E^2_{\mathcal{T}}\hat{y_0} + Var(\hat{y_0}) \\ &= Var(y_0) + (E_{y_0|x_0}y_0 - E_{\mathcal{T}}\hat{y_0})^2 + Var(\hat{y_0}) \\ &= \sigma^2 + (Bias)^2 + Var(\hat{y_0}) \end{aligned}$$

and because the data is assumed to be generated with a linear function, $\hat{y_0}$ is an unbiased estimator for y_0, so the bias term evaluates to 0.

Now, looking at the last term,

$$\begin{aligned} Var(\hat{y_0}) &= E_{\mathcal{T}_X}E_\epsilon[(\hat{y_0} - E_{\mathcal{T}_X}E_\epsilon\hat{y_0})^2] \\ &= E_{\mathcal{T}_X}E_\epsilon[\hat{y_0}^2] \\ &= E_{\mathcal{T}_X}E_\epsilon[(b^T\epsilon)^2] \\ &= E_{\mathcal{T}_X}[b^Tb]\sigma^2 \\ &= E_{\mathcal{T}_X}[x_0^T(\mathbf{X}^T\mathbf{X})^{-1}x_0]\sigma^2. \end{aligned}$$

Putting it all together:

$$EMSE = \sigma^2 + 0^2 + E_{\mathcal{T}_X}[x_0^T(\mathbf{X}^T\mathbf{X})^{-1}x_0]\sigma^2$$

Notice the addition variance incurred due to the randomness inherent in the testing data. There is no bias (since both our model and the data generation mechanism are both linear), and the variance depends on x_0. If N is large and $\mathcal{T}$ were selected at random, and assuming $E(X) = 0$, then $\mathbf{X}^T\mathbf{X} \to NCov(X)$ and

$$E_{x0}EPE(x_0) \sim \sigma^2(p/N) + \sigma^2$$

Here we see that the expected EPE increases linearly as a function of p, with slope σ^2/N. If N is large or σ is small, this growth in variance is negligible. By imposing some heavy restrictions on the class of models being fitted, we have avoided the curse of dimensionality (this assumes that indeed our assumptions are correct; if they are not, bias becomes unavoidable).

Statistical Models, Supervised Learning and Function Approximation

Our goal is to find a useful approximation $\hat{f}(x)$ to the function $f(x)$ that underlies the predictive relationship between the inputs and outputs. We have already seen that L^2 loss leads to the regression $f(x) = E(Y|X = x)$ for a quantitative response. The class of nearest-neighbor methods can be viewed as direct estimates of this conditional expectation, but they can fail when p is high, and when a special structure in the data generation mechanism is known to exist.

2.6.1 A Statistical Model for the Joint Distribution P(X,Y)

We can suppose an additive model

$$Y = f(X) + \epsilon$$

for the obtained targets. Here ϵ represents effects due to unmeasured variables, including measuremenet error.

The assumption that the errors are iid is not strictly necessary, but convenient nonetheless. Simple modifications can be made to avoid the independence assumption; for example, we can have $Var(Y|X = x) = \sigma(x)$, and now both mean and variance depend on X.

So far we have concentrated on the quantitative resonse. Additive error models are typically not used for quaitative outputs G, in this case the target function $p(X)$ *is* the conditional density $P(G|X)$, and this is modeled directly.

2.6.2 Supervised Learning

Supervised learning attempts to learn f by example through a *teacher*. One observes the system under study, both the inputs and outputs, and assembles a *training* set of observations $\mathcal{T} = x_i, y_i, i = 1, ..., N$. The artificial system produces outputs $\hat{f}(x_i)$ in response to the inputs. The learning algorithm has the property that it can modify its input/output relatinship $\hat{f}$ in response to differences $y_i - \hat{f}(x_i)$ between the original and generated outputs.

2.6.3 Function Approximation

While least squares is usually very convenient, it is not the only criterion used and in some cases would not make much sense. A more general principle for estimation is *maximum likelihood estimation*. Suppose we have a random sample $y_i,\ i = 1, ..., N$ from a density $P_\theta(y)$ indexed by some parameters θ. The log-probability of the observed sample is

$$L(\theta) = \sum_{i=1}^{N} log P_\theta(y).$$

The principle of maximum likelihood assumes that the most reasonable values for θ are those for which the probability of the observed samples is largest. Least squares for the additive error model $Y = f_\theta(X) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$, is equivalent to maximum likelihood using the conditional likelihood

$$P(Y|X, \theta) = \mathcal{N}(f_\theta(X), \sigma^2).$$

Another example is the multinomial likelihood for the regression function $P(G|X)$ for a qualitative output G. Suppose we have a model $P(G = \mathcal{G}_k|X = x) = p_{k,\theta}(x),\ k = 1, ..., K$ for the conditional probability of each class given X, indexed by θ. Then the log-likelihood (a.k.a cross-entropy) is

$$L(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{K} y_{i,j}\ log\ p_{j,\theta}(x_i) = \sum_{i=1}^{N} log\ p_{g_i,\theta}(x_i).$$

2.7 Structured Regression Models

Given a set of points, there is an infinite number of functions that can interpolate it. We can limit the solution space of possible classes of models to choose from so that solving the task becomes easier and/or we can solve it more adequately given the structure of the data.

How to decide on the nature of the restrictions is based on considerations outside of the data. These restrictions are sometimes encoded via the parametric representation of f_θ, or may be built into the learning method itself, either implicitly or explicitly. The ambiguity regarding choice of estimator has been transferred to the choice of constraint/class of functions.

The constraints imposed by most learning methods can be described as *complexity* restrictions of one kind or another. This usually means some kind of regular behaviour in small neighborhoods of the input space. That is, for all input points x sufficiently close to each other in some metric, $\hat{f}$ exhibits some special structure such as nearly constant, linear or low-order polynomial behaviour.

The estimator is then obtained by averaging (can be weighted via attention) or polynomial fitting in that neighborhood.

The strngth of the constraint is dictated by the neighborhood size. The larger the size, the stronger the constraint, and the mor sensitive the solution is to the particular choice of constraint.

Some methods, such as kernel and local regression and tre-based methods, directly specify the metric and size of the neighborhood. Other methods such as splines, neural networks and basis-function methods implicitly define neighborhoods of local behaviour.

Any method that attempts to produce locally varying functions in small isotropic neighborhoods will run into problems in high dimensions (the curse of dimensionality). And conversely, all methods that overcome the dimensionality problem have an associated — and often implicit or adaptive — metric for measuring neighborhoods, which basically does not allow the neighborhood to be simultaneously small in all directions.

2.8 Classes of Restricted Estimators

Roughness Penalty and Bayesian Methods

Kernel Methods and Local Regression

Basis Function and Dictionary Methods

2.9 Model Selection and the Bias-Variance Tradeoff

Exercise 2.7

a)

For k-NN, the weights are $l_i = 1/k$ for the k nearest neighbors, and 0 for the rest. For linear regression, the weights l_i are the i-th element of $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}x_0$

b)

$$\begin{aligned} E_{\mathcal{Y}|\mathcal{X}}(f(x_0) - \hat{f}(x_0))^2 &= Var_{\mathcal{Y}|\mathcal{X}}(f(x_0) - \hat{f}(x_0)) + [E_{\mathcal{Y}|\mathcal{X}}[f(x_0) - E_{\mathcal{Y}|\mathcal{X}}\hat{f}(x_0)]]^2 \\ &= Var_{\mathcal{Y}|\mathcal{X}}(\hat{f}(x_0)) + Bias^2(\hat{f}(x_0)) \\ &= \sigma^2 + Bias^2(\hat{f}(x_0)) \end{aligned}$$

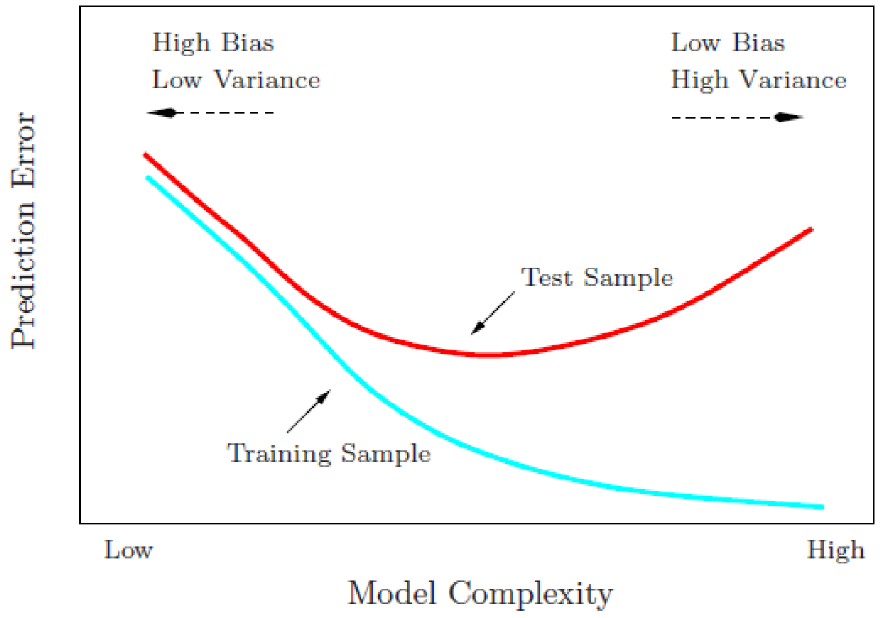

Figure 1: The bias-variance tradeoff illustrated

c)

$$\begin{aligned}
E_{\mathcal{Y},\mathcal{X}}(f(x_0)-\hat{f}(x_0))^2 &= \text{... same development as in exercise 2.5 a)}\\
&= Var_{\mathcal{Y},\mathcal{X}}(f(x_0))\\
&+ E^2_{\mathcal{Y},\mathcal{X}} f(x_0) - 2E_{\mathcal{Y},\mathcal{X}}[f(x_0)]E_{\mathcal{Y},\mathcal{X}}[\hat{f}(x_0)] + E^2_{\mathcal{Y},\mathcal{X}}\hat{f}(x_0)\\
&+ Var_{\mathcal{Y},\mathcal{X}}(\hat{f}(x_0))\\
&= Var_{\mathcal{Y},\mathcal{X}}(f(x_0)) + E_{\mathcal{Y},\mathcal{X}}(f(x_0) - E_{\mathcal{Y},\mathcal{X}}\hat{f}(x_0))^2 + Var_{\mathcal{Y},\mathcal{X}}(\hat{f}(x_0))\\
&= \sigma^2 + (Bias)^2 + E_{\mathcal{T}_X}[x_0^T(\mathbf{X}^T\mathbf{X})^{-1}x_0]\sigma^2
\end{aligned}$$

d)

I don't think one can deduct a relation between bias and variance in the decomposition of the MSE, nor have I been able to find such relation on the internet. My guess is that the authors wanted the reader to realize that if such relation were known, then it would be trivial to strike the perfect balance between variance and bias, yet such relation does not seem to have an explicit formulation, so the variance-bias trade-off remains elusive.

Exercise 2.9

$$\begin{aligned}E[R_{tr}(\hat{\beta})] &= \frac{1}{N}\sum_{i=1}^{N} E[(y_i - \hat{\beta}^T x_i)^2] \\ &= \frac{1}{N}\sum_{i=1}^{N} E[y_i^2 - 2y_i\hat{\beta}^T x_i + (\hat{\beta}^T x_i)^2]\end{aligned}$$

$$\begin{aligned}E[R_{te}(\hat{\beta})] &= \frac{1}{N}\sum_{i=1}^{N} E[(y_i - \hat{\beta}^T x_i)^2] \\ &= \frac{1}{N}\sum_{i=1}^{N} E[\tilde{y_i}^2 - 2\tilde{y_i}\hat{\beta}^T \tilde{x_i} + (\hat{\beta}^T \tilde{x_i})^2]\end{aligned}$$

Since training and testing data are drawn from the same population,

$$E[y_i^2] = E[\tilde{y_i}^2]$$

therefore we can cancel out these terms.

3. Linear Methods for Regression

A linear regression model assumes that the regression function $E(Y|X)$ in the inputs $X_1, ..., X_p$.

3.2 Linear Regression Models and Least Squares

The linear regression model has the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j\beta_j.$$

The linear model assumes that the regression function $E(Y|X)$ is linear in the parameters β_j.

Typically, we estimate the parameters β by *least squares* w.r.t to the training data, i.e. minimize the residual sum of squares

$$\begin{aligned}RSS(\beta) &= \sum_{i=1}^{N}(y_i - f(x_i))^2 \\ &= \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} f(x_{ij}\beta_j))^2 \\ &= (y - X\beta)^T(y - X\beta)\end{aligned}$$

Statistically speaking, **this criterion is reasonable** if the training obesrvations (x_i, y_i) represent independent random draws from their population. Even if the x_i's were not drawn randomly, the criterion is still valid if the y_i's are conditionally independent given the inputs x_i.

Differentiating w.r.t β and assuming the derivative equals 0, we can obtain a closed form solution (assuming that X has full column rank, and hence X^TX is positive definite):

$$\hat{\beta} = (X^TX)^{-1}X^Ty.$$

The pedicted values at x_0 are given by $\hat{f}(x_0) = (1 : x_0)^T\hat{\beta}$; the fitted values at the training inputs are

$$\hat{y} = X\hat{\beta} = X(X^TX)^{-1}X^Ty,$$

where $\hat{y_i} = \hat{f}(x_i)$. Matrix $H = X(X^TX)^{-1}X^T$ is usually called *hat* matrix.

Column vectors $x_0, x_1, ..., x_p$ of X span a subspace of $\mathbb{R}^N$. We minimize $RSS(\beta) = ||y - X\beta||^2$ by choosing $\hat{\beta}$ so that the residual vector $y - \hat{y}$ is orthogonal to this subspace. The hat matrix H computes the orthogonal projection.

It might happen that columns of X are not linearly independent. However, the fitted values $\hat{y} = X\hat{\beta}$ are still the projection of y onto the column space of X; there is just more than one way to express that projection.

In the following, in order to pin down the sampling properties of $\hat{\beta}$, we now assume that the observations y_i are uncorrelated and have constant variance σ^2, and that the x_i are fixed.

The variance-covariance matrix of the least squares parameters is easily computed, and is given by

$$Var(\hat{\beta}) = (X^TX)^{-1}\sigma^2.$$

To be able to draw conclusions about the parameters and the model, we make additional assumptions: that the conditional expectation of Y is linear in $X_1, ..., X_p$, and that the deviations of Y around its expectation are additive and Gaussian. Hence

$$Y = E(Y|X_1, ..., X_p) + \epsilon \quad = \beta_0 + \sum_{j=1}^{p} X_j\beta_j + \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Thus, it is easy to show that

$$\hat{\beta} \sim \mathcal{N}(\beta, (X^T X)^{-1}\sigma^2).$$

Also,

$$(N - p - 1)\hat{\sigma}^2 \sim \sigma^2 \chi^2_{N-p-1}.$$

In addition, $\hat{\beta}$ and $\hat{\sigma}^2$ are statistically independent.

To test the hypothesis that a particular coefficient $\beta_j = 0$, we form the standardized coefficient or Z-*score*

$$z_j = \frac{\beta_j}{\hat{\sigma}\sqrt{v_j}},$$

where v_j is the j-th diagonal element of $(X^T X)^{-1}$. Under the null hypothesis that $\beta_j = 0$, z_j is distributed as t_{N-p-1}. If $\hat{\sigma}$ is replaced with a know σ, then z_j would be normally distributed.

We can also test for the significance of groups of coefficients simultaneously. For example, to test if a categorical variable with k levels can be excluded from a model, we need to test whether the coefficients of the dummy variables used to represent the levels can all be set to zero. Here we use the F statistic,

$$F = \frac{(RSS_0 - RSS_1)/(p_1 - p_0)}{RSS_1/(N - p_1 - 1)},$$

where subindex 1 corresponds to the bigger model with $p_1 + 1$ parameters and subindex 0 corresponds to the nested smaller model with $p_0 + 1$ parameters, having $p_1 - p_0$ parameters constrained to be zero.

11. Neural Networks

11.2 Projection Pursuit Regression (PPR)

The PPR model has the form

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X).$$

This model is additive on the derived features $\omega_m^T X$. The functions g_m are unspecified and estimated along with the directions ω_m using some flexible smoothing method.

Function $g_m(\omega_m^T X)$ (called a *ridge function*) varies only in the direction defined by the vector ω_m. $V = \omega_m^T X$ is the projection of X onto the unit vector ω_m, and we seek ω_m so that the model fits well.

11.3 Neural Networks

Derived features Z_m are created from linear combinations of the inputs, and then the target Y_k is modeled as a function of linear combinations of the Z_m,

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, ..., M \\ T_k &= \beta_{0k} + \beta_k^T Z, k = 1, ..., K \\ f_k(X) &= g_k(T), k = 1, ..., K \end{aligned}$$

where $Z = (Z_1, Z_2, ..., Z_M)$, and $T = (T_1, T_2, ..., T_M)$.

The output function $g_k(T)$ is usually the identity for regression and the *softmax* for classification.

We can think of the values Z_m as a basis expansion of the original inputs X. However, here the parameters of the basis functions are learned from the data.

11.4 Fitting NNs

For regression, we use sum-of-squared errors as our measure of fit

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

For classification we use either squared error or cross-entropy (deviance):

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i),$$

and the corresponding classifier is $G(x) = argmax_k f_k(x)$. With the softmax activation function and the cross-entropy error function, the neural network model is exactly a linear logistic regression model in the hidden units, and all the parameters are estimated by maximum likelihood.

The generic approach to minimizing $R(\theta)$ is by gradient descent, called *back-propagation* in this setting. The gradient can be easily derived using the chain rule for differentiation, with a forward and backward pass over the network, keeping track only of quantities local to each unit.

Backpropagation for squared error loss

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$, and let $z_i = (z_{1i}, z_{2i}, ..., z_{Mi})$. Then we have

$$R(\theta) \equiv \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,$$

with derivatives

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= -2 \sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}. \end{aligned}$$

Given these derivatives, a gradient descent update at the (r+1)st iteration has the form

$$\begin{aligned} \beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\ \alpha_{ml}^{(r+1)} &= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}, \end{aligned}$$

where γ_i is the *learning rate*. Now write the derivatives of R_i as

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= \delta_{ki} z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= s_{mi} x_{il}. \end{aligned}$$

The quantities δ_{ki} and s_{mi} are "errors" from the current model at the output and hidden layer units, respectively. From their definitions, these errors satisfy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

known as the *back-propagation equations*. In the *forward pass*, the current weights are fixed and the predicted values $\hat{f}_k(x_i)$ are computed. In the *backward pass*, the errors δ_{ki} are computed, and then backpropagated via the equation for s_{mi} to give the errors s_{mi}. Both sets of errors are then used to compute the gradients for the udates.

The advantages of back-propagation are its simple, local nature. in the back propagation algorithm, each hidden unit passes and receives information only to and from units that share a connection. Hence it can be implemented efficiently on a parallel architecture.

Exercices 11.3

Cross Entropy Error with Logistic Activation

The outputs are computed by applying the logistic function to the weighted sum of the hidden layer activations, s:

$$\begin{aligned} y_i &= \frac{1}{1+e^{-s_i}} \\ s_i &= \sum_{j=1} h_j w_{ji}. \end{aligned}$$

Now proceed to compute the derivative of the error w.r.t each weight connecting the hidden units to the output units using the chaing rule:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{y_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial w_{ji}}.$$

In turn, examining each factor:

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \frac{-t_i}{y_i} + \frac{1-t_i}{1-y_i} \\ &= \frac{y_i - t_i}{y_i(1-y_i)}, \\ \frac{\partial y_i}{\partial s_i} &= y_i(1-y_i), \\ \frac{\partial s_i}{\partial w_{ji}} &= h_j, \end{aligned}$$

where x_j is the activation of the j node in the hiden layer. Combining things back together,

$$\frac{\partial E}{\partial s_i} = y_i - t_i,$$

and

$$\frac{\partial E}{\partial w_{ji}} = (y_i - t_i)h_j.$$

The above expression gives gradients w.r.t the weights in the last layer. We can apply again the chain rule to derive gradients for the lower layers.

Now we derive the quantity $\frac{\partial E}{\partial s_i^1}$ where j indexes the hidden units, s_j^1 is the weighted input sum at hidden unit j, and $h_j = \frac{1}{1+\exp(-s_j^1)}$ is the activation at unit j.

$$\begin{aligned}
\frac{\partial E}{\partial s_i^1} &= \sum_{i=1}^{K} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial h_j} \frac{\partial h_j}{\partial s_i^1} \\
&= \sum_{i=1}^{K} (y_i - t_i)(w_{ji})(h_j(1-h_j)), \\
\frac{\partial E}{\partial h_j} &= \sum_{i=1}^{K} \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{\partial x_j} \\
&= \sum_{i=1}^{K} \frac{\partial E}{\partial y_i} y_i(1-y_i)w_{ji}.
\end{aligned}$$

Then a weight w_{kj}^1 connecting input unit k to a hidden unit j has gradient

$$\frac{\partial E}{\partial w_{kj}^1} = \frac{\partial E}{\partial s_j^1} \frac{\partial s_j^1}{\partial w_{kj}^1} \quad = \sum_{i=1}^{K} (y_i - t_i) w_{ji} (h_j(1-h_j)) x_k.$$

By recursively computing the gradient of the error w.r.t. the activity of each neuron, we can compute the gradients for all weights in a NN.

Classification with Softmax Output Layer and Cross Entropy Error

The softmax activation of the i-th output unit is

$$y_i = \frac{\exp(s_i)}{\sum_n^N \exp(s_n)},$$

and the cross-entropy loss for multi-class output is

$$E = -\sum_{n}^{N} t_i \log(y_i).$$

Computing the gradient yields

$$
\begin{aligned}
\frac{\partial E}{\partial y_i} &= -\frac{t_i}{y_i} \\
\frac{\partial y_i}{\partial s_k} &= \begin{cases} \dfrac{\exp(s_i)}{\sum_n^N \exp(s_n)} - \left(\dfrac{\exp(s_i)}{\sum_n^N \exp(s_n)}\right)^2, & k = i \\ -\dfrac{\exp(s_i)\exp(s_k)}{(\sum_n^N \exp(s_n))^2} & k \neq i. \end{cases} \\
&= \begin{cases} y_i(1-y_i) & k = i \\ -y_i y_k & k \neq i. \end{cases} \\
\frac{\partial y_i}{\partial s_i} &= \\
&= \sum_n^N \frac{\partial E}{\partial y_n}\frac{\partial y_n}{\partial s_i} \\
&= \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial s_i} - \sum_{k\neq i}\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial s_i} \\
&= -t_i(1-y_i) + \sum_{k\neq i} t_k y_i \\
&= -t_i + y_i \sum_k t_k \\
&= y_i - t_i
\end{aligned}
$$

Notice we arrived at the same expression as in the case with the logistic output activation. The gradient for weights in the top layer is again

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \sum_i^N \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial h_j}\frac{\partial h_j}{\partial s_j^1} \\
&= \sum_i^N (y_i - t_i)(w_{ji})(h_j(1-h_j)).
\end{aligned}
$$

Exercise 11.5

Write a program to fit a single hidden layer neural network via back-propagation and weight decay.

Answer can be found in another of my repositories called MLP-NumPy.