CS 5785 – Applied Machine Learning – Lec. 2

Profs. Serge Belongie, Cornell Tech
Scribes: Mor Cohen, Nicolas J., Praveen G., Daniel S.,
Shreyas K, Francesco P., and Mervin F.

August 29, 2019

1 Linear Regression

Last lecture we learned about kNN, which is a *lazy learning* or memory-based method, not dependent on a model fit. In lazy learning algorithms, one defers prediction until query time. In contrast to this, *eager learning* algorithms, work by summarizing the data using a model or function at training time, and then only rely on the summary model at query time. To illustrate this, we'll look at a method that is somewhat less lazy than kNN, a method that uses a model, albeit a simple one: linear model fit by least squares.

1.1 The Linear Model

The model looks like this:

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

with vector of inputs $X = (X_1, X_2, \ldots, X_p)^\top$ (a p dimensional column vector) and $\hat{\beta}_0$ the intercept or bias. The hat on symbols indicates a prediction.

This equation can be shortened by changing the structure of X and concatenating all the $\hat{\beta}_i$ into one vector. The 1 at the front of this new X absorbs the bias term $\hat{\beta}_0$:

$$X = (1, X_1, X_2, \ldots, X_p)^\top$$
$$\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \ldots \hat{\beta}_i \ldots \hat{\beta}_p)^\top$$

These changes of notation enable us to write the following:

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

This is simply the dot product (or inner product) between X and $\hat{\beta}$. Here we have a single outcome variable, so $\hat{Y}$ is a scalar, but this can be generalized to multiple outputs $Y_1, \ldots, Y_K$ [HTF §3.2.4].

Let $f(X) = X^\top \hat{\beta}$ represent some linear function over the $(p+1)$-dim. space. We're going to use this function for classification by applying a threshold to $f(X)$.

1.2 Fitting with Least Squares

How do we fit this model? In this context, *fit* means "estimate β." The most popular method for doing this is *least squares*. In this method, we pick β to minimize the *residual sum of squares (RSS)*:[1]

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - X_i^\top \beta)^2$$

In other words, we are looking for the best compromise in β over all the data points.

We can write this sum in matrix-vector form as:

$$RSS(\beta) = (y - X\beta)^\top (y - X\beta) = \|y - X\beta\|^2$$

where $X \in \mathbb{R}^{N \times (p+1)}$, a matrix where each row is an input vector, and $y \in \mathbb{R}^N$ is a vector of outputs:

$$X = \begin{bmatrix} 1 & X_1^1 & X_2^1 & \cdots & X_p^1 \\ 1 & X_1^2 & X_2^2 & \cdots & X_p^2 \\ \vdots & & \vdots & & \vdots \\ 1 & X_1^N & X_2^N & \cdots & X_p^N \end{bmatrix} \qquad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix}$$

Here we have N data points and $(p+1)$ dimensions. The matrix X is known as the *design matrix*.

We want to find a β vector that will best return the desired values of y. How do we solve for β? Optimizing with respect to β we have,

$$\begin{aligned} \nabla_\beta SSR(\beta) &= \nabla_\beta \|y - X\beta\|^2 \\ 0 &= X^\top (y - X\beta) \end{aligned}$$

also known as "the normal equations." Provided $X^\top X$ is non singular, we have:

$$\hat{\beta} = (X^\top X)^{-1} X^\top y = X^+ y$$

where $X^+ = (X^\top X)^{-1} X^\top$ denotes the *pseudoinverse* of X.

The fitted value for the ith input x_i is:

$$\hat{y}_i = x_i^\top \hat{\beta}$$

which is simply the dot product of the input vector with the fitted coefficients. Importantly, this applies not just for the input points, it applies for an arbitrary input x_0, too! Thus, the linear model *generalizes* over the plane (space, etc.). How *well* it generalizes, however, is a different story – we've made some huge simplifying assumptions here, but it's still noteworthy that we're seeing some actual generalization.

[1] Also referred to as the *sum of squared residuals (SSR)*.

1.2.1 Why Least Squares? (Optional)

It is worth considering why least squares is such a popular way of defining a "best" fit, because it may appear somewhat arbitrary. Why not minimize the l_1-norm or the ∞-norm? Although we might rule out the l_1-norm because it lacks differentiability at the minimum, that criticism doesn't carry over to most other norms.

One of the key features that least squares has is that the error nicely decomposes into recognizable quantities. Similar to the analysis of expected prediction error (EPE) from the previous lecture, we can analyze the expected error of a given model $f(x)$ as follows,

$$\begin{aligned}
E[(f(X)-Y)^2] &= E_X[E[(f(X)-Y)^2|X]] \\
&= E_X[f(X)^2 - 2f(X)E[Y|X] + E[Y^2|X]] \\
&= E_X[(f(X)-E[Y|X])^2 + E[Y^2|X] - E[Y|X]^2] \\
&= E_X[(f(X)-E[Y|X])^2] + E[var[Y|X]] \\
&= E_X[f(X)-E[Y|X]]^2 + (E[f(X)^2] - E[f(X)]^2) + E_X[var[Y|X]] \\
&= E_X[f(X)-E[Y|X]]^2 + var[f(X)] + E_X[var[Y|X]]
\end{aligned}$$

The three final terms correspond to the squared *bias* of the model with respect to the data ($E_X[f(x)-E[Y|X]]^2$), the *variance* of the model itself ($var[f(X)]$), and the inherent noise in the data ($E_X[var[Y|X]]$). Clearly, the choice of model $f(x)$ cannot have any effect on this last component. Minimizing least squares, therefore, puts squared bias on equal terms with model variance. We will talk more about the notion of a *bias-variance trade-off* later.

1.3 The "Linear Classifier"

Let's now bring this back to the classification problem from last lecture, with the response Y coded as 0 for blue and 1 for orange; see HTF Fig. 2.1. We use the convention

$$\hat{G} = \begin{cases} \text{orange, if} & \hat{Y} > 0.5 \\ \text{blue, if} & \hat{Y} \leq 0.5 \end{cases}$$

The *linear decision boundary* in $\mathbb{R}^2$ is given by $\{x : x^\top \hat{\beta} = 0.5\}$ and data points in $\mathbb{R}^2$ get the orange label for $\{x : x^\top \hat{\beta} > 0.5\}$

How did we do? We can see the linear model makes a lot of mistakes. Avoidable? Yes and no. One aspect of this approach that should strike us as odd is the binary coding of Y in the face of fitting what amounts to a linear ramp in this case. Isn't this a strange representation? We get wildly different contributions to the RSS from pts. at varying distance from the decision boundary.

In this sense, linear regression on a 0/1 response is not very natural. Soon we'll learn about a more natural way to solve this problem using something called *logistic regression*.

This method is high bias, low variance method. As described by P. Domingos, "Bias is a learner's tendency to consistently learn the same wrong thing. Variance is the tendency to learn random things irrespective of the real signal." See Figure 1 for an illustration of this idea using darts.

FIGURE 2.1

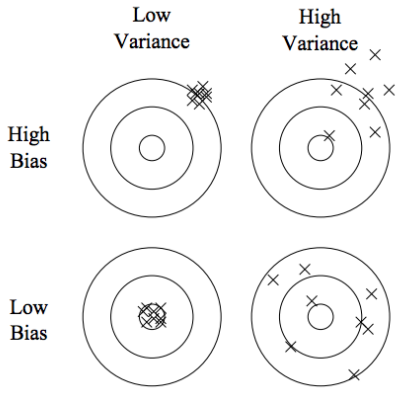


Figure 1: Bias and Variance in dart throwing. [P. Domingos 2012]

1.4 Nonlinear Models (Optional)

The least squares approach is not limited to fitting linear models, but nonlinear models can also be fit by augmenting the feature data with combinations of individual features. For example, if one had two observed features z_1 and z_2, a quadratic feature set would take the form

$$X = \begin{pmatrix} 1 \\ z_1 \\ z_2 \\ z_1^2 \\ z_1 z_2 \\ z_2^2 \end{pmatrix}$$

1.5 Encoding Categorical Features

In some problems you'll encounter categorical features encoded as integers, such as *Yale*, *Columbia* and *Cornell* encoded as 0, 1 and 2, respectively. Multiplying that value by a weight in β (as one does in a linear classifier) isn't meaningful. This is commonly handled by using *one-hot* or *one-of-K* encoding. For this case, the encoding for these three schools would be the length-3 vectors $[1, 0, 0]^\top$, $[0, 1, 0]^\top$ and $[0, 0, 1]^\top$, respectively.

2 Performance Evaluation

Before we dig further into more classification models, we need clarify how we intend to rate the performance of our classifiers. In addition to standardized performance metrics, these measurement involve a bunch of tools used to keep track of what you're doing, how well you're doing and also for debugging purposes. They become our bread and butter for communicating results to others and we need to understand them well.

2.1 Confusion Matrix

A confusion matrix, or contingency table, is used to quantify the discrepancies between the ground truth and the ouput of a classifier. In the confusion matrix each column represents the instances in a predicted class, while each row represents the instances in an actual class. The (i, j)th entry in the matrix equals the number of observations known to be in class i, but predicted by our classifier to be in class j. Using this representation, every entry on the diagonal is the number of correctly classified entities of that class. A heavier diagonal implies a more accurate classifier. Normalizing the matrix, dividing each entry by the sum of all entries (the total number of entities tested) can help calculating the overall error and success rates. The sum of the diagonal of the normalized confusion matrix equals the success rate of the classifier, and the sum of all other cells equals the error rate.

2.1.1 2×2 Confusion Matrix

In the case of a **True-False** classifier there are only two classes and the resulting confusion matrix would is 2×2, a *binary* confusion matrix; see Figure 2. An

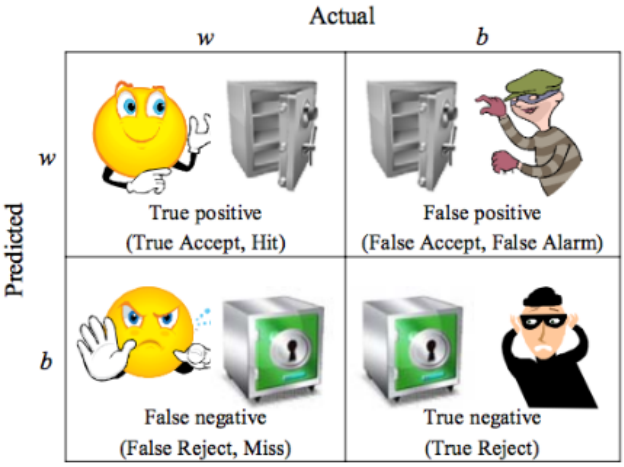


Fig. 4. 2×2 contingency table (confusion matrix)

Figure 2: Contingency Table (Cha & Tappert)

TABLE 9.1. *Test data confusion matrix for the additive logistic regression model fit to the spam training data. The overall test error rate is* 5.5%.

	Predicted Class	
True Class	`email` (0)	`spam` (1)
`email` (0)	58.3%	2.5%
`spam` (1)	3.0%	36.3%

(unnormalized)

example of such a classifier would be an email vs. spam classifier; see HTF Table 9.1.

True-positives (TP) and true-negatives (TN) represent correct classifications and False-positives (FP) and false-negatives (FN) represent incorrect classification. FP is also called a *type I* error and FN is also called a *type II* error. The type of error that matters most depends on the problem we are trying to solve. In case of biometrics (e.g., does this fingerprint belong to person x?), type I errors are really bad because it means we let the wrong person get past security, while if you have to submit your fingerprint twice to get access it is not that bad. In the case of credit card fraud, in which case the credit provider may be wary of alienating customers, and type II errors become more important. Since the different errors have different significance under different situations, one number is not enough to describe the performance of a classifier. Therefore, be wary if someone tells you "our system is 99% accurate!" Be a stickler: what do they mean by "accurate?" Is this a true positive rate? Perhaps it is simply meaningless.

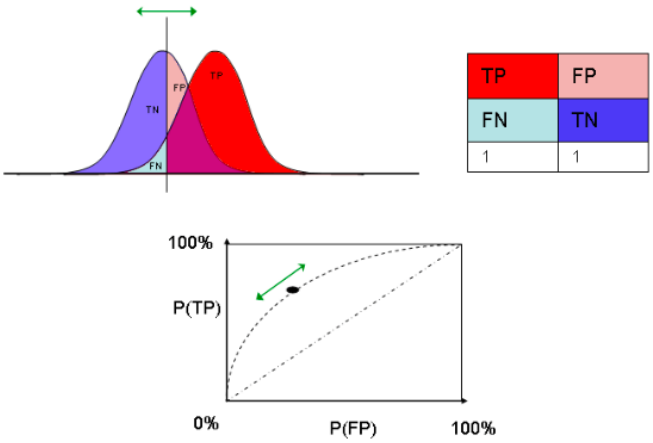


Figure 3: ROC curve. Top Left: The x axis represents distance or score. The vertical line represents the classifier threshold setting: scores above the threshold are declared positives. Top Right: 2-class (binary) confusion matrix corresponding to the indicated choice of threshold. Bottom: ROC Curve, with dot representing choice of threshold. Diagonal line represents the ROC curve for random chance. (Wikipedia)

2.2 Receiver Operating Characteristic (ROC) Curve

The ROC curve is a graphical means of representing a binary confusion matrix as a function of classifier threshold; see Figure 3. Remember from §1.3 that when attempting to fit a linear classifier we get a number representing the score of the classified entity (some function of a distance, for example) and then we **threshold** it to make a decision. The threshold of the blue/orange problem discussed earlier a depicted in HTF Fig. 2.1. was 0.5. Once you commit to a threshold you can create the confusion matrix; every threshold has a corresponding confusion matrix. The ROC curve captures the tradeoff expressed in the confusion matrix as the threshold is swept over a wide range.

The ROC curve plots the **True Positive Rate** (TPR), or sensitivity, against the **True Negative Rate** (TNR), or specificity.

$$TPR = \frac{TP}{TP + FN}$$

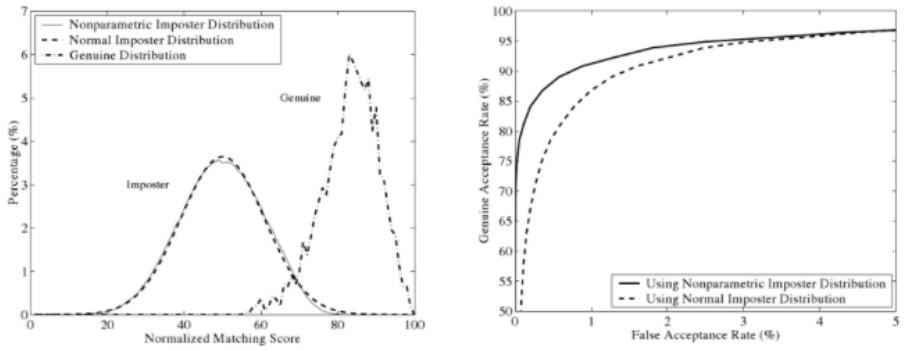
$$TNR = \frac{TN}{FP + TN}$$

Some version of ROC curves use the **False Positive Rate** (FPR) instead of TNR. When reading an ROC curve remember to always check which convention is used.

$$FPR = \frac{FP}{FP + TN}$$

Note: $TNR = 1 - FPR$

Accuracy (ACC) is the number of entities classified correctly over the total number of entities tested. If we denote the total number of positive entities by



[Maltoni et al. Handbook of Fingerprint Recognition]

Figure 4: Integrating scores of genuine and impostor pair matches to produce the ROC curve. As an example, a genuine score would be the similarity between two voiceprints from Alice, while an impostor scores would capture the similarity between a voiceprint form Alice and a voiceprint from Bob. In general, we expect the distribution of impostor scores to be largely to the left of the genuines, but some overlap is inevitable. (Maltoni et al.)

P and the total number of false entities by N we would get:

$$ACC = \frac{TP + TN}{P + N}$$

In the example from Wikipedia[2] (Fig. 3) $P = N = 100$ creating a *balanced* example.

2.2.1 Area Under the Curve

A generic metric for comparing the performance of one algorithm to another across all possible thresholds is to compute the Area Under the [ROC] Curve (AUC or AUROC). This is generic because it summarizes the overall performance, but individual algorithms with lower AUCs could still be more appropriate in certain application contexts.

2.2.2 ROC Curve Example: Biometric Identification

Let's look at an example from the field of biometrics (fingerprint recognition, voice recognition, etc.). In this setting we can examine the distribution of *genuine* and *impostor* scores (or distances) arising from comparisons between all possible pairs of *same* and *different* people, respectively.[3] Figure 4 demonstrates the matching scores of both *genuine* and *impostor* pairs and the relationships between them. We then generate the ROC curve by sweeping over the range of possible score thresholds, calculating the TPR and FPR at each threshold and plotting these two values against each other.

[2] `http://en.wikipedia.org/wiki/Receiver_operating_characteristic`

[3] Example: Matching two different impressions of the same finger gives us a *genuine* match score. Matching fingerprints from two different people gives us an *impostor* match score.

Suppose we set our threshold so that we obtain a 1% FNR and a 0.01% FPR. The way to read this is that we incur (or pay the price of) a 1% rejection rate in exchange for a one-in-ten-thousand false positive rate. The latter is the same as the probability of guessing someone's 4 digit PIN.

2.3 Precision and Recall

For datasets that are very skewed (not balanced) TP, FP, TN and FN are not good representations of the performance of the algorithms. In such cases we use:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F - Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Note: F-Score is the harmonic mean of Precision and Recall.

Document retrieval is an example of a problem with a skewed dataset. Imagine you type in a search query and some algorithm computes a similarity score over a set of stored documents. The threshold you sweep here are the number of results presented to the user. In this scenario:

$$Precision = \frac{|\{\text{Relevant Documents}\} \cap \{\text{Retrieved Documents}\}|}{|\{\text{Retrieved Documents}\}|}$$

$$Recall = \frac{|\{\text{Relevant Documents}\} \cap \{\text{Retrieved Documents}\}|}{|\{\text{Relevant Documents}\}|}$$

Precision and Recall can be plotted together too, similar to the ROC curve. *See Davis & Goadrich ICML06 to learn more about the connections between ROC curves and precision ancall.*