

MS&E 226: Fundamentals of Data Science

Lecture 5: Classification

Ramesh Johari

Classification problems

Overview

Thus far in our study of prediction, we have focused on the *regression* problem: Predict *continuous* outcomes.

In this lecture we (briefly) cover the *classification* problem: Predict *discrete* outcomes.

Specifically, we will focus on *binary classification*, where the outcome is zero or one.

Example: Spam filtering

Suppose that you are running a mail server.

Given an incoming message your goal is to decide: Is this spam or not?

Typical covariates:

- ▶ Counts of key words, phrases, etc.
- ▶ Indicator for origin domain of e-mail
- ▶ Indicators for links/domains present in the e-mail
- ▶ Indicator for whether sender is in your address book

Example: Medical diagnostics

Suppose that given a patient, your goal is to determine whether this person is likely to suffer an adverse medical event (e.g., a heart attack).

Typical covariates:

- ▶ Weight, body mass index, obesity indicator
- ▶ Age
- ▶ Gender
- ▶ Cholesterol levels
- ▶ Indicator for family history
- ▶ Results of medical tests (blood tests, imaging, etc.)

Example: Hiring in a labor market

Suppose that, given a job opening and an applicant, your goal is to determine whether the applicant will be hired for the job opening.

Typical covariates:

- ▶ Indicators for skills of the applicant
- ▶ Indicators for skills required on the job
- ▶ Years of education, work experience of the applicant
- ▶ Years of education, work experience required in the job opening
- ▶ *Interactions* between the preceding covariates
- ▶ Detail in applicant's profile
- ▶ Who made the first contact: applicant or employer

Classification: Formalism

Formally, classification problems look a lot like what we've studied so far:

- ▶ There is a dataset with n observations; Y_i is the outcome in the i 'th observation, and $\mathbf{X}_i$ is the covariate vector corresponding to the i 'th observation.

Classification: Formalism

Formally, classification problems look a lot like what we've studied so far:

- ▶ There is a dataset with n observations; Y_i is the outcome in the i 'th observation, and $\mathbf{X}_i$ is the covariate vector corresponding to the i 'th observation.
- ▶ For each i , $Y_i \in \{0, 1\}$ (or some other binary set). We refer to zeroes as "negative" outcomes, and ones as "positive" outcomes.

Classification: Formalism

Formally, classification problems look a lot like what we've studied so far:

- ▶ There is a dataset with n observations; Y_i is the outcome in the i 'th observation, and $\mathbf{X}_i$ is the covariate vector corresponding to the i 'th observation.
- ▶ For each i , $Y_i \in \{0, 1\}$ (or some other binary set). We refer to zeroes as "negative" outcomes, and ones as "positive" outcomes.
- ▶ Using the data, fit a model $\hat{f}$ (a "classifier"). Given a covariate vector $\vec{X}$, $\hat{f}(\vec{X}) \in \{0, 1\}$.
- ▶ The model is evaluated through some measure of its prediction error on new data (generalization error). Common example is expected 0-1 loss on a new sample:

$$\mathbb{E}_Y[\mathbb{I}\{Y \neq \hat{f}(\vec{X})\} | \hat{f}, \vec{X}] = \mathbb{P}_Y(Y \neq \hat{f}(\vec{X}) | \hat{f}, \vec{X}).$$

0-1 loss and “accuracy”

The 0-1 loss function is a measure of *accuracy*: How well, on average, does the classifier make predictions?

Another way to say it: A good classifier minimizes the number of samples on which the classifier makes a classification mistake.

Is this always what we want?

Classification: Visualization

Train-validate-test

The train-validate-test methodology carries over to classification in a straightforward way:

Given a dataset, split it into *training*, *validation*, and *test* sets.

- ▶ *Training*: Use the training data to build different candidate classifiers $\hat{f}^{(1)}, \dots, \hat{f}^{(L)}$.
- ▶ *Validation*: Compute the average target loss (e.g., 0-1 loss) of each of the L classifiers on the validation data. Choose the classifier $\hat{f}^*$ with the lowest loss (i.e., lowest misclassification rate, or highest accuracy).
- ▶ *Test*: Make predictions using $\hat{f}^*$ on the test set to get an unbiased estimate of the generalization error.

Cross validation

Similarly, cross validation carries over in a straightforward way.

As noted previously, cross validation is a completely general technique that can be applied regardless of the model class or loss function being used.

In particular, K -fold CV can be used in exactly the same way for evaluation and selection of classifiers with any chosen loss criterion (e.g., 0-1 loss).

False positives and false negatives

Example: Fraud detection

Suppose that you are asked to build a fraud detection algorithm for PayPal.

Data: Transactions, including account info, profile info, etc. of both parties, as well as amount and type of transaction

Outcomes: For each transaction, either zero (not fraud) or one (fraud)

Suppose: 0.3% of transactions are fraud.

Your job: *build a "good" classifier*. Is accuracy the right objective?

Beyond accuracy

Because of the issue identified in the preceding example, we should distinguish between misclassification of *positive* samples and *negative* samples.

For this purpose we typically use the *confusion matrix* of a classifier on a test set.¹

		Predicted		Total
		0	1	
Actual	0	TN	FP	TN + FP
	1	FN	TP	FN + TP
	Total	TN + FN	TP + FP	n

Here:

- ▶ TN = # of *true negatives*; FP = # of *false positives*
- ▶ FN = # of *false negatives*; TP = # of *true positives*

¹Be careful! One source of "confusion" about confusion matrices is that sometimes the truth indexes the rows, and sometimes the truth indexes the columns.

Confusion matrices

How to interpret confusion matrices:

Other metrics

Nearly every metric of interest in binary classification can be derived from confusion matrices:

- ▶ *Accuracy* = $(TP + TN)/n$.
- ▶ *Mean 0-1 loss* = $(FP + FN)/n = 1 - \text{accuracy}$.
- ▶ *True positive rate (TPR)* = $TP/(FN + TP)$.
- ▶ *False positive rate (FPR)* = $FP/(TN + FP)$.
- ▶ *True negative rate (TNR)* = $TN/(TN + FP)$.
- ▶ *False negative rate (FNR)* = $FN/(FN + TP)$.
- ▶ *Sensitivity* = TPR.
- ▶ *Specificity* = TNR.
- ▶ *Precision* = $TP/(TP + FP)$.
- ▶ *Recall* = sensitivity = TPR.
- ▶ *F1 score* = $2 \times \text{Precision} \times \text{Recall} / (\text{Precision} + \text{Recall})$.
- ▶ *Type I error rate* = FPR.
- ▶ *Type II error rate* = FNR.
- ▶ *False discovery proportion (FDP)* = $FP/(TP + FP)$.

False positives and false negatives

Except for accuracy and 0-1 loss, all these metrics are just different ways of measuring the two kinds of error that can be made by the classifier:

1. Misclassifying a true negative example as a positive (false positive, or Type I error).
2. Misclassifying a true positive example as a negative (false negative, or Type II error).

The *false discovery rate (FDR)* is the expected FDP. By convention, FDP is zero when no positives are predicted.

Choosing your objective: A tradeoff

Note that:

1. It is easy to design a classifier with no false positives. (How?)
2. It is easy to design a classifier with no false negatives. (How?)

On the other hand, in general there is “no free lunch”: it is generally not possible to ensure *both* no false positives and no false negatives.

In general, for “good” classifiers, *reducing false positives comes at the expense of increasing false negatives*.

Therefore in designing a classifier it is important to carefully *choose your objective*: i.e., to consider which type of error is more consequential to you.²

²Note that this sometimes leads to objectives that are *weighted sums* of the entries of the confusion matrix.

Example: Fraud detection

As an example, suppose that you build a classifier for fraud detection on a dataset with $n = 50,000$ with the following confusion matrix:

		Predicted		
		Not Fraud	Fraud	Total
Actual	Not Fraud	49,603	247	49,850
	Fraud	22	128	150
	Total	49,625	375	50,000

Example: Fraud detection

As an example, suppose that you build a classifier for fraud detection on a dataset with $n = 50,000$ with the following confusion matrix:

		Predicted		
		Not Fraud	Fraud	Total
Actual	Not Fraud	49,603	247	49,850
	Fraud	22	128	150
	Total	49,625	375	50,000

- Note that this classifier has *lower* accuracy than one that just always predicts "Not Fraud."
- Would you use this classifier? What if reducing the false negatives to zero also meant increasing false positives to ≈ 1000 ?

Example: Fraud detection

Analysis:

A note on class imbalance

Class imbalance occurs when one outcome label is much more common than the other.

This is often treated as an algorithmic issue, so many common procedures for “balancing” the training data have been developed:

- ▶ Undersampling the majority class;
- ▶ Oversampling the minority class; or
- ▶ Generating synthetic minority-class examples.

A key word of caution though: These algorithmic approaches implicitly change the *objective* you are optimizing!

Balancing classes changes the objective

In our fraud example, only 0.3% of transactions are fraudulent. Minimizing 0-1 loss means minimizing:

$$0.997 \text{ FPR} + 0.003 \text{ FNR.}$$

Suppose instead we *balance* the training data to contain equal numbers of fraudulent and nonfraudulent transactions. This makes unweighted 0-1 loss equal to:

$$0.5 \text{ FPR} + 0.5 \text{ FNR.}$$

In the original population, this assigns each missed fraud approximately 332 *times the cost of a false alarm*. Is that the specific objective you want?

That tradeoff should be chosen *deliberately*, based on the consequences of the two errors; not implicitly, due to a hidden algorithmic procedure.

An example: k -nearest-neighbor classification

k -nearest-neighbor classification

Basic version:

- ▶ Given a covariate vector $\vec{X}$, find the k nearest neighbors.
- ▶ Take a *majority vote* to determine the classification.³

k -NN is an example of an *instance-based* learning algorithm (or “lazy” algorithm):

Computation is done at the time a new prediction is made (the “instance”).

³Ties are typically broken at random among the furthest neighbors, or just by choosing k odd.

Adding a threshold

More generally, let $\text{NN}(k, L)$ denote the k -nearest neighbor algorithm with a *threshold* L :

- ▶ Require at least L of the k nearest neighbors to have label 1 to assign label 1; otherwise assign label 0.
- ▶ Basic algorithm corresponds to $\text{NN}(k, k/2)$.
- ▶ What happens if $L = 0$?
- ▶ What happens if $L > k$?

Example: The CORIS dataset

462 South African males evaluated for heart disease.

Outcome variable: Coronary heart disease (`chd`).

Covariates:

- ▶ Systolic blood pressure (`sbp`)
- ▶ Cumulative tobacco use (`tobacco`)
- ▶ LDL cholesterol (`ldl`)
- ▶ Adiposity (`adiposity`)
- ▶ Family history of heart disease (`famhist`)
- ▶ Type A behavior (`typea`)
- ▶ Obesity (`obesity`)
- ▶ Current alcohol consumption (`alcohol`)
- ▶ Age (`age`)

Example: The CORIS dataset

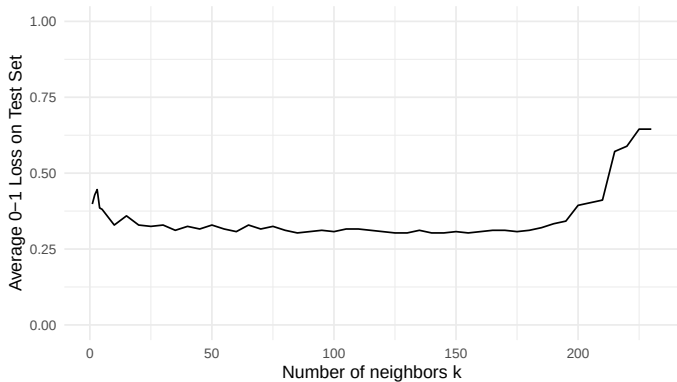
Steps:

- ▶ Import the data
- ▶ (Random) split into train and test
- ▶ Train $NN(k, L)$ for range of k , and for $L = ak, a \in [0, 1.05]$
- ▶ Record TP, FP, TN, FN on test set.

If these results guide our choice of k or L , this set serves as *validation* data; evaluating the winner requires a separate test set.

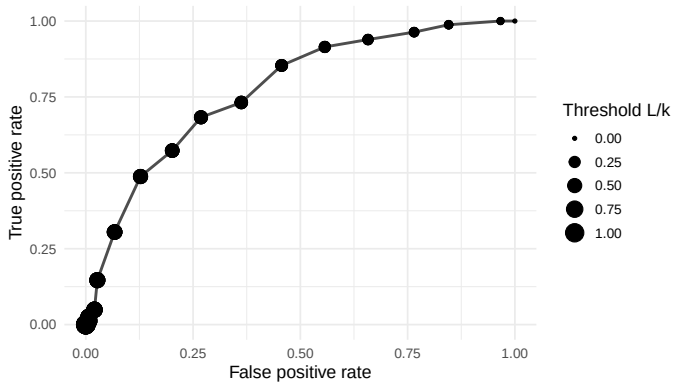
Example: The CORIS dataset

Prediction error (0-1 loss) on test set with threshold $L/k = 0.32$:



Example: The CORIS dataset

TPR vs. FPR for the choice $k = 20$, as threshold L/k varies:



The ROC curve

The graph of TPR vs. FPR as the classifier's threshold is varied is called the *ROC curve* (for “receiver operating characteristic”).

The *area under the curve* (*AUC*) is often reported as a measure of the quality of the classifier:

- ▶ Better classifiers are further “up and to the left” in the graph.
- ▶ A perfect classifier would have zero FPR and unit TPR $\equiv$ top left corner.
- ▶ Therefore a perfect classifier should have AUC 1.
- ▶ Higher AUC summarizes better overall ranking, but ROC curves can cross: it need not give a better tradeoff at the operating point you care about.

More on ROC curves

Appendix: The Bayes classifier [*]

Optimal prediction [*]

In regression, we saw that if we knew the population model, then the *squared-error-minimizing* prediction was the *conditional expectation of the outcome given the covariates*.

What is the analogue of conditional expectation for classification with *0-1 loss* as the objective?

The Bayes classifier [*]

Suppose we want to minimize 0-1 loss (i.e., maximize accuracy). Note that:

$$\mathbb{E}_Y[\mathbb{I}\{Y \neq \hat{f}(\vec{X})\}|\vec{X}] = \mathbb{P}_Y(Y \neq \hat{f}(\vec{X})|\vec{X}).$$

How do we minimize this?

- ▶ If Y is more likely to be 0 than 1 (given $\vec{X}$), then we should set $\hat{f}(\vec{X}) = 0$; and vice versa.
- ▶ In other words: If $\mathbb{P}_Y(Y = 0|\vec{X}) > 1/2$, set $\hat{f}(\vec{X}) = 0$; otherwise if $\mathbb{P}_Y(Y = 1|\vec{X}) > 1/2$, set $\hat{f}(\vec{X}) = 1$.

This is called the *Bayes classifier*.

The Bayes classifier [*]

The Bayes classifier is unattainable as a predictive model, for the same reason the conditional expectation is unattainable: *we don't know the population model.*

In general, good classification models (with respect to 0-1 loss) are those that approximate the Bayes classifier well.

This observation serves as a foundation for starting to think about analogs of "bias" and "variance" in the context of classification.