MS&E 226: Fundamentals of Data Science

Lecture 7: 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.

all 0 or 1

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:

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

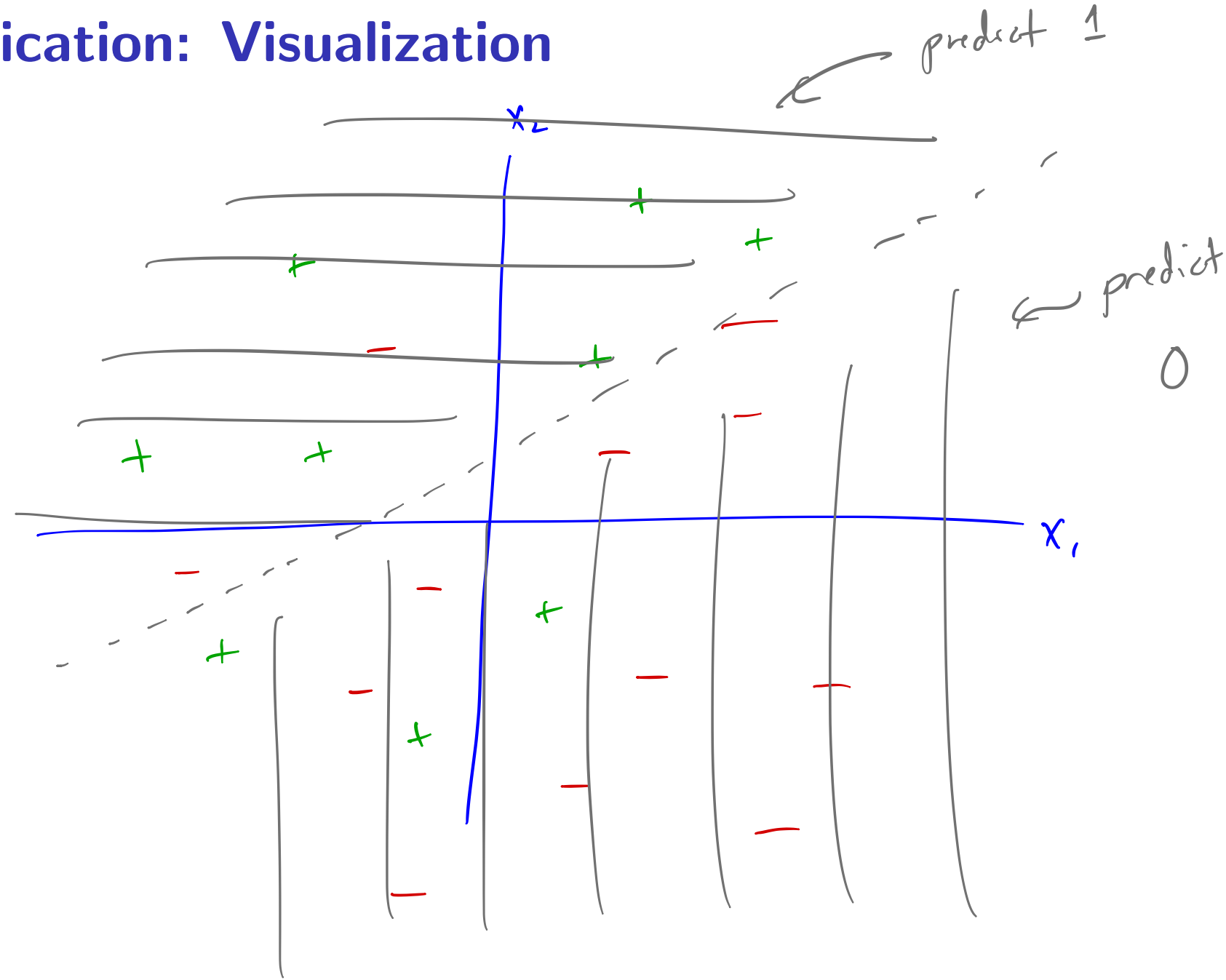
0-1 loss and "accuracy"

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 classifer 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 $\hat{f}^{(1)}, \ldots, \hat{f}^{(L)}$.
- *Validation*: Compute the average 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.

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

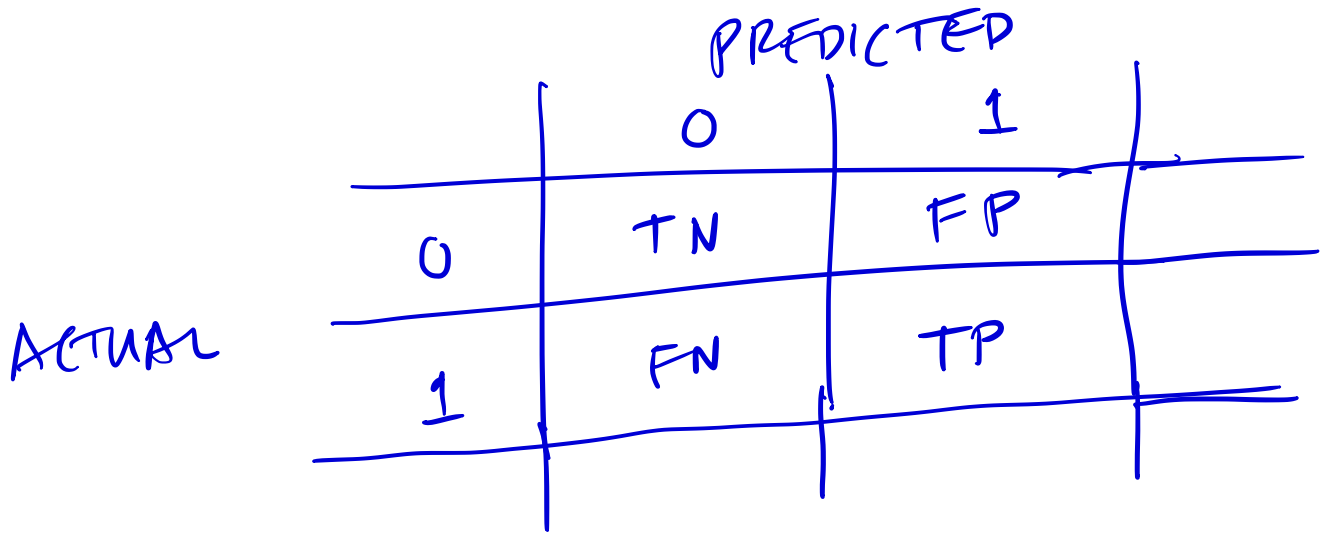
Here:

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

[1] Be 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* $= (\mathsf{TP} + \mathsf{TN})/n$.
- *Mean 0-1 loss* $= (\mathsf{FP} + \mathsf{FN})/n = 1 -$ accuracy.
- *True positive rate (TPR)* $= \mathsf{TP}/(\mathsf{FN} + \mathsf{TP})$.
- *False positive rate (FPR)* $= \mathsf{FP}/(\mathsf{TN} + \mathsf{FP})$.
- *True negative rate (TNR)* $= \mathsf{TN}/(\mathsf{TN} + \mathsf{FP})$.
- *False negative rate (FNR)* $= \mathsf{FN}/(\mathsf{FN} + \mathsf{TP})$.
- *Sensitivity* $= \mathsf{TPR}$.
- *Specificity* $= \mathsf{TNR}$.
- *Precision* $= \mathsf{TP}/(\mathsf{TP} + \mathsf{FP})$.
- *Recall* $=$ sensitivity $= \mathsf{TPR}$.
- *Type I error rate* $= \mathsf{FPR}$.
- *Type II error rate* $= \mathsf{FNR}$.
- *False discovery rate* $= \mathsf{FP}/(\mathsf{TP} + \mathsf{FP})$

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).

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 consider which type of error is more consequential to you.[2]

[2] Note 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:

Building a classifier

Optimal prediction

In regression, we saw that if we knew the population model, then:

- Given a new covariate vector $\vec{X}$, the best prediction we could make (in terms of squared error) was the *conditional expectation* $f(\vec{X}) = \mathbb{E}[Y|\vec{X}]$.
- Even if we did so, there is still some error remaining in our predictions: the *irreducible error*.

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

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.*

Approximating 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.

One example we explore: k-*nearest-neighbor* classification.

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]

This approximates the Bayes classifier by *local averaging*.

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

Adding a threshold

More generally, let 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 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`)

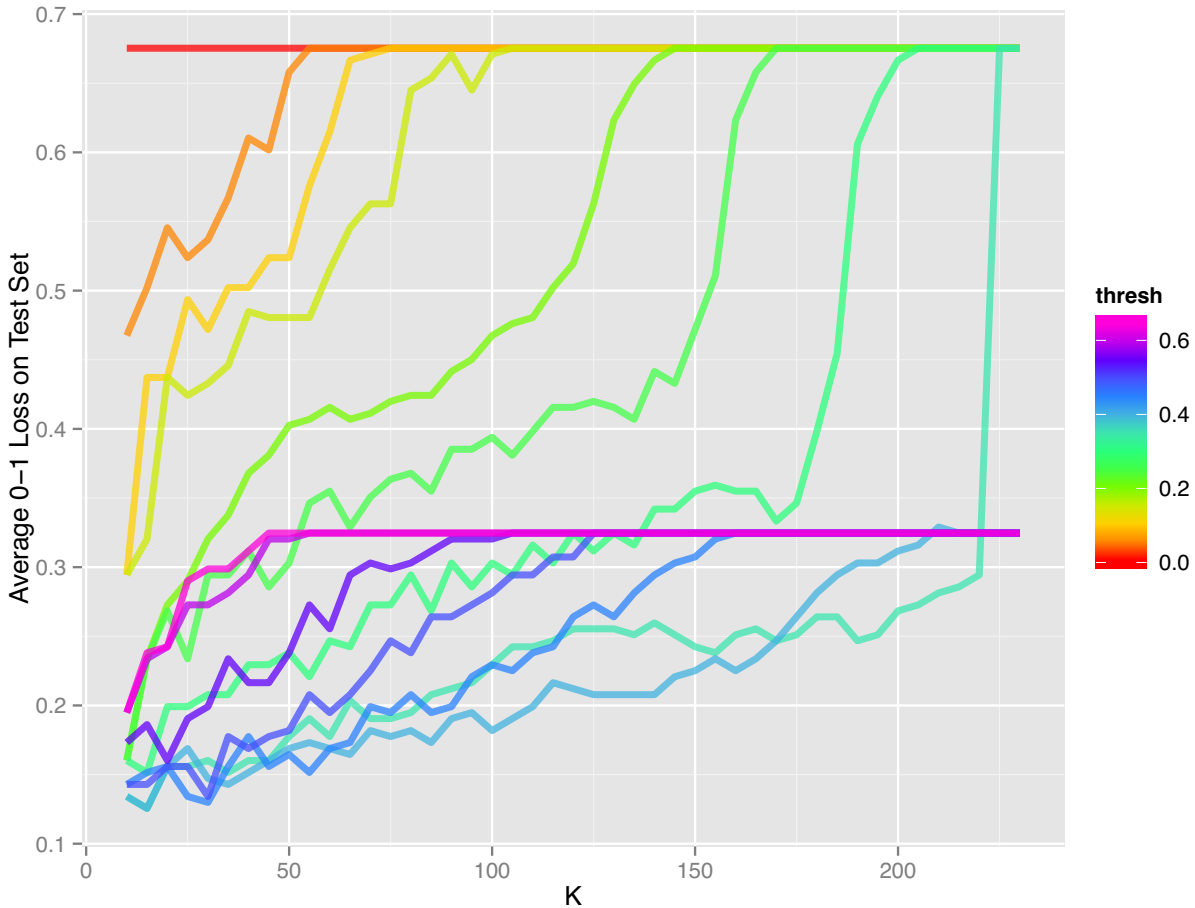
Example: The CORIS dataset

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.

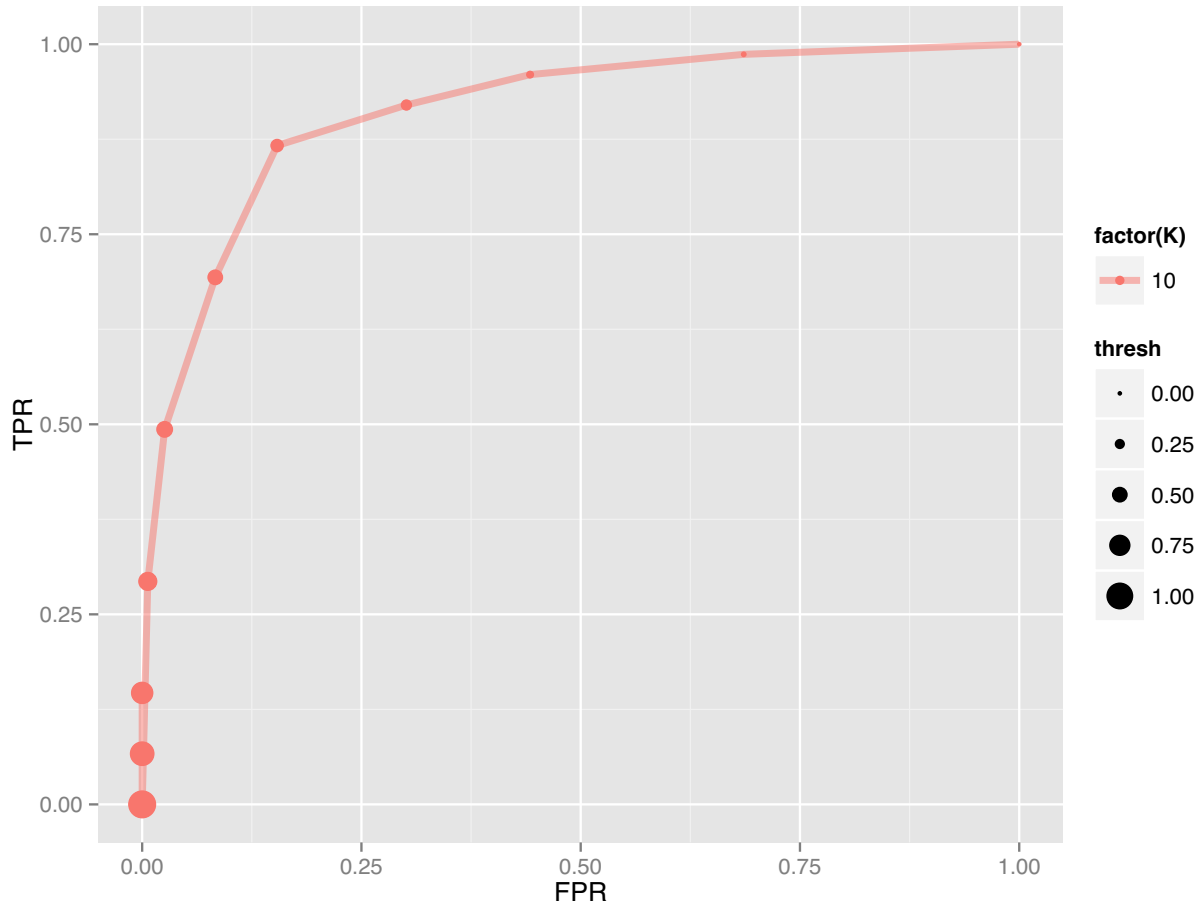
Example: The CORIS dataset

Prediction error (0-1 loss) on test set:



Example: The CORIS dataset

FPR vs. TPR for the choice $k = 10$, as threshold L/k varies:



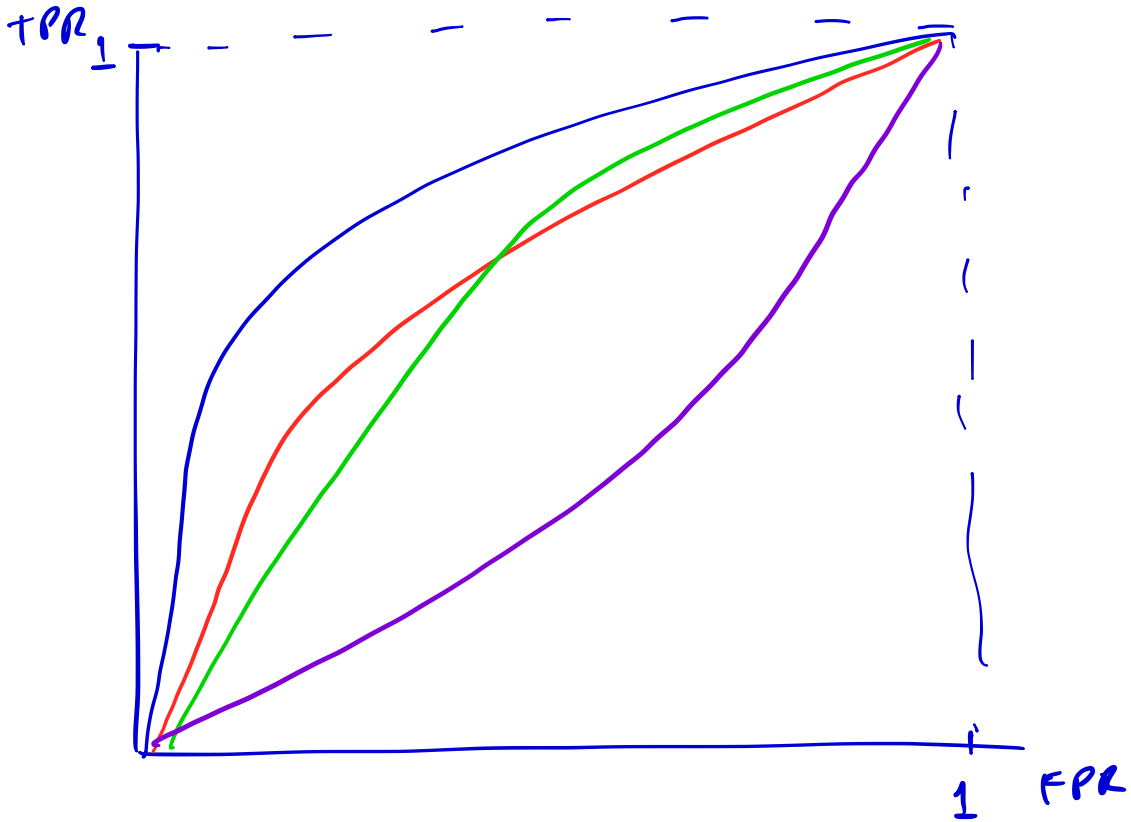
The ROC curve

The graph of FPR vs. TPR 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.
- The closer the AUC is to 1, the better the classifier is on *both* types of errors.

More on ROC curves



Model scoring and selection for classification

Cross validation

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 0-1 loss (or any other objective derived from the confusion matrix).

Model complexity scores

The scores we developed (C_p, AIC, BIC) are also developed only for *squared error loss*.

In practice, AIC and BIC are often used even in the case of 0-1 loss.

There is no formal justification for this practice, except that both scores provide a heuristic penalty for excessive model complexity.

Bias, variance, and model complexity

Note that the bias-variance decomposition we computed only applies for squared error loss.

There are versions of this decomposition that have been developed for 0-1 loss as well, though one has to be careful in interpreting them.[4]

In general, just as there is often a "bias-variance tradeoff" in regression, something similar is true in classification:

- More "complex" models (e.g., lower k in k-NN classification) tend to overfit the training data, and thus have higher variance, but have lower bias.
- Less "complex" models (e.g., higher k in k-NN classification) tend to underfit the training data, and thus have lower variance, but have higher bias.

[4] See P. Domingos (2000), "A unified bias-variance decomposition", ICML.