Classification Tasks

Arnab Maity - Modified by Justin Post

library(MASS)  
library(klaR)  
library(tidyverse)  
library(caret)  
library(rsample)  
library(ISLR2)  
library(knitr)  
library(AppliedPredictiveModeling)  
library(kableExtra)  
library(nnet)  
library(glmnet)

# Classification Tasks

The problems of separating two or more groups/classes, and allocating new objects in previously defined classes are called classification and discrimination.

* **Classification:**

   
 

* **Discrimination:**

   
 

A classification rule is often based on the features that separate the groups, so the two goals often (but not always) overlap.

## Classification Example

To give some context, let’s consider a data set and the familiar model KNN model used as a classifier.

Consider the wines data set available at the [UCI machine learning repository](https://archive.ics.uci.edu/ml/datasets/wine). The dataset contains

* quantities of 13 constituents found in each of the three types (cultivars) of wines
* response variable is Class (1, 2, or 3)

# Read the data  
wines <- read\_table("data/Wines.txt")  
#convert our response to a factor  
wines$Class <- as.factor(wines$Class)

A snapshot of the full data is shown below.

#frequency of each class of wine  
table(wines$Class)

1 2 3   
59 71 48

Goal:

Here we will only consider two predictor variables: Alcohol and Malic but the techniques we discuss apply to any number of predictors.

The figure below shows the three classes on a scatterplot of Alcohol vs. Malic.

|  |
| --- |
| Scatterplot of Alcohol vs. Malic in the wine data. The color and shapes represent the different cultivars of wine (Class). |

### -Nearest Neighbors

Suppose we have training data for , where is categorical variable denoting class label of . For a given predictor , KNN classifier predicts the class label as follows:

   
   
   
 

The process and concepts around train/test sets, tuning, etc. all follow through here. We’ll go through the example more explicitly soon. First, let’s just look at some KNN decision boundaries.

Let us start with a KNN classifier with .

* Use caret with method = "knn"
* This does both regression and classification tasks!
* It automatically determines the problem depending on whether the response is numeric or categorical (factor)

## 30-NN Classifier / with no tuning done  
fit <- train(Class ~ Alcohol + Malic,  
 data = wines,  
 method = "knn",  
 tuneGrid = data.frame(k = 30),  
 trControl = trainControl(method = "none"))  
fit

k-Nearest Neighbors   
  
178 samples  
 2 predictor  
 3 classes: '1', '2', '3'   
  
No pre-processing  
Resampling: None

Now let’s plot the classifications made by the model:

|  |
| --- |
| Decision boundary of 30-NN classifier of the wines data. |

Formally, we can think of the KNN process as estimating the conditional probability of Y given X, .

* Suppose that has categorical classes.
  + Denote the values can take on as .
* Suppose denotes the indices of the points whose values are nearest to .
* For a data point , KNN estimates the conditional probability that the class label is given as

   
   
   
   
   
 

Here we can see that the KNN classifier gives us a rule to allocate objects to classes, but **does not give us any discriminants**.

Now that we’ve seen an example, we need to understand how evaluation of a classifier differs from evaluation of a regression model.

# Bayes classifier

The motivation behind estimating the conditional probabilities is from minimizing test error rate. Test error rate differs in the classification setting as compared to the regression setting!

Similar to regression, given a new independent test point with label , we can define expected prediction error for classification as

   
   
   
 

* depends on
* depends on both and the training set

We want a classifier that minimizes this expected prediction error.

* It can be shown that the optimal classifier is the one that predicts a new observation by such that

The optimal classifier is called the *Bayes classifier*.

**Bayes Classifier:** Classifies an observation to the most probable class using the discrete conditional distribution of

**Bayes error rate:** misclassification error rate of the Bayes classifier. For a given , Bayes error is . The overall Bayes error rate is .

Every classification problem has these two (unknown) items.

   
   
   
 

It is natural to try to estimate/model the conditional probabilities using the data, and use them to create classifiers. There are two major approaches for obtaining estimates of :

* *Directly estimating/modeling* 
  + An example of direct estimation of is the KNN classification technique, where the conditional probability is estimated by taking a majority vote from nearest point to .
  + Another example is *logistic* regression model, where the conditional probability is modeled using transformations of linear combinations of of the form:
  + Therefore it is sufficient to estimate the coefficients to obtain estimates of .
* *Generative models* where we model the distribution of for and use **Bayes Theorem** to obtain a classifier

## kNN and the Bayes Classifier

As with KNN regression, the hyperparameter determines how flexible the KNN method is. However, the idea of flexibility is subtle in this case.

* The boundary that separates the regions of the predictor space is effectively the classification rule for that classifier.
* The *decision boundary*, is the boundary of the regions.

|  |
| --- |
| Decision boundary of 30-NN classifier of the wines data. |

How would the decision boundary look like for the Bayes classifier? We can’t know for this problem!

* For a two class problem, the Bayes classifier assigns the most probable class to a new data point.

   
 

The plots below are taken from the book and show the KNN decision boundary and the Bayes classifier decision boundary on some simulated data (hence the ability to know the Bayes classifier!).

|  |
| --- |
| Impact of K on decision boundaries of KNN classifiers. The Bayes dicision boundary is shown using purple dashed line. Image adapted from *Introduction to Statistical Learning*. |

The value of in a KNN classifier determines how smooth or rough the decision boundary is.

* For a small value of (in this example, ), the boundary is extremely rough.
* For a large value of (such as ), the decision boundary is much smoother but does not capture the shape of the Bayes boundary.

We need to tune so that the “optimal” will result in a decision boundary that is not too rough but also sufficiently captures the shape of the Bayes boundary.

The figure below shows one such example with . In practice, we might choose by minimizing the test error rate or equivalently maximizing test accuracy.

|  |
| --- |
| Decision boundary for using simulated data presented in the previous figure. The Bayes decision boundary is shown using purple dashed line. Image adapted from *Introduction to Statistical Learning*. |

# Example: Building a KNN Classifier

Let’s reconsider the wines data set available at the [UCI machine learning repository](https://archive.ics.uci.edu/ml/datasets/wine). The dataset contains quantities of 13 constituents found in each of the three types (cultivars) of wines.

A snapshot of the full data is shown below.

print(wines, n = 4)

# A tibble: 178 × 14  
 Class Alcohol Malic Ash Alcal Mg Phenol Flav Nonf Proan Color Hue  
 <fct> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
1 1 14.2 1.71 2.43 15.6 127 2.8 3.06 0.28 2.29 5.64 1.04  
2 1 13.2 1.78 2.14 11.2 100 2.65 2.76 0.26 1.28 4.38 1.05  
3 1 13.2 2.36 2.67 18.6 101 2.8 3.24 0.3 2.81 5.68 1.03  
4 1 14.4 1.95 2.5 16.8 113 3.85 3.49 0.24 2.18 7.8 0.86  
# ℹ 174 more rows  
# ℹ 2 more variables: Abs <dbl>, Proline <dbl>

The goal is to find a *rule* that can assign a specimen of wine to its cultivar. In other words, we want to predict the classes (cultivar) based on the predictors (13 variables).

# classes of wine  
table(wines$Class)

1 2 3   
59 71 48

We can tune as we did in the regression setting. The code below searches odd values of (to avoid ties) for the optimal value with largest test accuracy. We use 50 times repeated 5-fold CV for tuning.

set.seed(1001)  
## K values for tuning  
kgrid <- expand.grid(k = seq(1,51, by=2))  
## 5-fold CV, repeated, tuning  
tr <- trainControl(method = "repeatedcv",  
 number = 5,  
 repeats = 50)  
## Train the classifier  
fit <- train(Class ~ Alcohol + Malic,  
 data = wines,  
 method = "knn",  
 tuneGrid = kgrid,  
 trControl = tr)  
plot(fit)

|  |
| --- |
| Results for repeated 5-fold CV tuning. |

fit$bestTune$k

[1] 21

## Refit the model with best K  
tuned\_knn\_class <- train(Class ~ Alcohol + Malic,  
 data = wines,  
 method = "knn",  
 tuneGrid = expand.grid(k = fit$bestTune$k),  
 trControl = trainControl(method = "none"))

To estimate the prediction error of the tuned model, we can use any of the methods discussed previously. For example, we could just look at prediction on a test set or we could use the bootstrap as the ‘outer’ loop.

Consider two new unlabeled points. The first with Alcohol = 13 and Malic = 3, and the second with Alcohol = 12.78 and Malic = 2.

|  |
| --- |
| Decision boundary of 21-NN classifier of the wines data with two new unlabeled points. |

We can use the model to predict a class or a class probability with predict.

The default is to give a predicted class.

pred\_class <- predict(tuned\_knn\_class,  
 newdata = new\_dat)  
pred\_class

[1] 3 2  
Levels: 1 2 3

We can specify type = "prob" to do obtain predicted class probabilities.

pred\_prob <- predict(tuned\_knn\_class,  
 newdata = new\_dat,  
 type = "prob")  
pred\_prob

1 2 3  
1 0.1428571 0.04761905 0.8095238  
2 0.3809524 0.42857143 0.1904762

Note that for the first point, has about an probability associated with class 3, and hence we are quite confident about out final class prediction of 3. However, for the second point, probabilities associated with classes 1 and 2 are quite similar ( vs ). So while we are quite confident about the predicted class of the first data point, there is some uncertainty about the second prediction!

# Evaluating a Classifier

## Basic Measures

To evaluate the performance of the classifier, instead of test MSE, we can use *classification accuracy* or *misclassification error rate*. In the definitions below, denotes the *cardinality* of , that is, the number of observations in .

**Accuracy:** the proportion of points correctly classified to their respective classes. With a set of observations ,

   
 

We can compute *training* and *test accuracy* depending on whether is the training or testing set.

**Misclassification error rate:** The proportion of points wrongly classified.

   
 

As before, we can compute *training* and *test error rate*.

As with regression setting, here too we aim to maximize *test* accuracy or minimize test error. Minimizing training error is undesirable since it will lead to overfitting the data.

* For example, consider , a 1-NN classifier. Since each is the closest neighbor to itself, the training error would be zero.
* The figure below shows training and test error rates from a simulation study (figure adapted from the textbook *Introduction to Statistical Learning*).

|  |
| --- |
| Training and test error rates for a KNN classifier based on 200 training and 5000 test observations. The error rates are plotted against 1/K. The black dashed line shows the Bayes error rate. Figure adapted from Introduction to Statistical Learning. |

## Other Metrics and the No Information Rate

There are many other metrics to evaluate a classification technique other than error rate and accuracy. The main criticism of these two criteria are that they provide a global measure, but do not provide much insight into how individual classes are correctly identified.

* For example, accuracy of a classifier does *not* guarantee that it will correctly classify items from *both* the classes correctly of the time!
* Such a criticism is even more relevant when there is class imbalance in the data
  + Say we have a situation where of observations belong to class A, and rest in class B.
  + If we employ a classifier that classifies *every point into class A* regardless of their predictor values. This classifier will have accuracy!
  + This is called the *no information rate (NIR)* of the classification problem.

**No information rate (NIR)**

The NIR represents the accuracy that can be obtained without using any model.

Most other measures to evaluate a classifier can be obtained by cross-tabulating the true and predicted classes of a test set. Such a table is called *confusion matrix*. An example is shown in the table below.

#split the original data  
index <- createDataPartition(wines$Class, p = 0.7, list = FALSE)  
#get the train and test sets  
train <- wines[index,]  
test <- wines[-index,]  
  
## K values for tuning  
kgrid <- expand.grid(k = seq(1,51, by=2))  
## 5-fold CV tuning  
tr <- trainControl(method = "cv",  
 number = 5)  
## Train the classifier  
fit <- train(Class ~ Alcohol + Malic,  
 data = train,  
 method = "knn",  
 tuneGrid = kgrid,  
 trControl = tr)  
fit$bestTune$k

[1] 5

## Refit the model with best K  
tuned\_knn\_class <- train(Class ~ Alcohol + Malic,  
 data = train,  
 method = "knn",  
 tuneGrid = expand.grid(k = fit$bestTune$k),  
 trControl = trainControl(method = "none"))  
#predict on the test set  
preds <- predict(tuned\_knn\_class, test)

With caret the confusionMatrix() function can now be applied to the true labels in the test set and the predictions given by the model.

confusionMatrix(test$Class, preds)

Confusion Matrix and Statistics  
  
 Reference  
Prediction 1 2 3  
 1 11 0 6  
 2 0 19 2  
 3 2 1 11  
  
Overall Statistics  
   
 Accuracy : 0.7885   
 95% CI : (0.653, 0.8894)  
 No Information Rate : 0.3846   
 P-Value [Acc > NIR] : 3.339e-09   
   
 Kappa : 0.6817   
   
 Mcnemar's Test P-Value : NA   
  
Statistics by Class:  
  
 Class: 1 Class: 2 Class: 3  
Sensitivity 0.8462 0.9500 0.5789  
Specificity 0.8462 0.9375 0.9091  
Pos Pred Value 0.6471 0.9048 0.7857  
Neg Pred Value 0.9429 0.9677 0.7895  
Prevalence 0.2500 0.3846 0.3654  
Detection Rate 0.2115 0.3654 0.2115  
Detection Prevalence 0.3269 0.4038 0.2692  
Balanced Accuracy 0.8462 0.9437 0.7440

Some measures we might look at are as follows:

* *sensitivity* (Also called “true positive rate” or “recall”)
* *specificity* (Also called “true negative rate”)
* *Precision*

We can also examine:

* *Cohen’s kappa*: measures the agreement of the classifier to the sample data taking into account any class imbalances, and how much agreement is by chance. Values close to 1 are considered good. The R function to do so is cohen.kappa() in psych library.
* *McNemar’s test*: hypothesis test for agreement between the predictions from an classifier to the observed data using a Chi-squared test. The R function to do so is mcnemar.test().

For a multi-class problem, we can create these measures using a “one-vs-all” approach, that is, by comparing each class vs the remaining combined (class 1 vs not class 1, and so on).

Often, we want a single measure of performance of the classifier rather than the multitude of measures shown above. There are many such options, such as *Youden’s Index*,

which measures the proportions of correct predictions for both the positive and negative events.

## Log-Loss

Most classifiers also produce probabilities of classification. We can use these to understand our model’s performance.

A commonly used metric along these lines is called the **log-loss**

* For a binary classification ( or )

 

|  |
| --- |
| Negative log-loss as a function of the estimated probability and the observed value of y |

Next, we’ll look at some common models for doing classification!