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:** developing a rule to *allocate* a new object into one of a number of known groups. We use such classification rules to classify objects into pre-defined classes. Here the emphasis is on defining the rule to optimally assigning objects to classes.
* **Discrimination:** finding the features that *separate* known groups in a multivariate sample. This can be either done graphically or algebraically. We try to find *discriminants* (features) whose numeric values can separate the classes as much as possible.

A classification rule is 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.

Our response variable here is Class (1, 2, or 3), representing the three cultivars. Note that is no ordering to the cultivars. The values 1, 2, and 3 are just category placeholders.

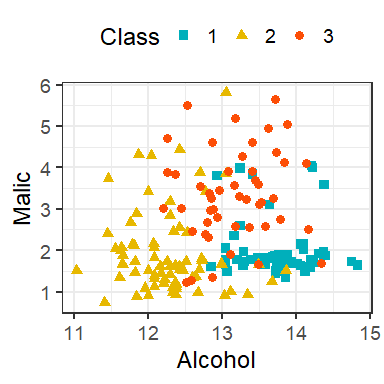
# Read the data  
wines <- read\_table("data/Wines.txt")  
#convert our response to a factor  
wines$Class <- as.factor(wines$Class)

A snapshot of the full data is shown below. 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 (1, 2, or 3) based on the predictors (13 variables).

#frequency of each class of wine  
freq <- table(wines$Class) |>  
 data.frame()  
names(freq) <- c("Cultivar", "Freq")  
freq |>  
 kable()

| Cultivar | Freq |
| --- | --- |
| 1 | 59 |
| 2 | 71 |
| 3 | 48 |

For this demonstration, we will only consider two predictor variables, Alcohol and Malic. However, the techniques discussed hereafter can be applied to any number of predictors. The figure below shows the three classes on a scatterplot of Alcohol vs. Malic.



### -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:

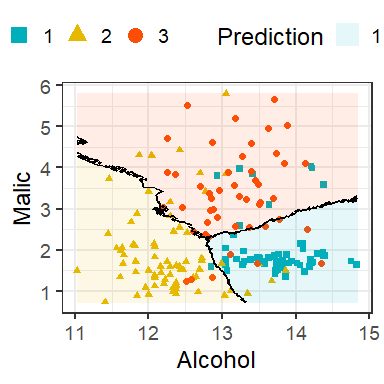
* Identify the observations in the training data such the their values are “nearest” to .
* Predict the class label corresponding to as the class having the majority vote, that is, having the most number of points among the neighbors obtained form previous step.

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 . We can use the knn() function in the class package, or use caret with method = "knn". Note that caret does both regression and classification. It automatically determines the problem depending on whether the response is numeric or categorical (factor). We have already converted the Class variable in the wines data to a 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"))

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



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

The lines in the figure are called the decision boundaries of this classifier!

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 . Then for a data point , KNN estimates the conditional probability that the class label is given as

for each . Thus, for each of the classes, we compute the proportion of the neighbors belonging to that class. We classify to the class that has the highest estimated probability.

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

where is the prediction from a classifier. Notice that depends on , and depends on both and the training set. We want a classifier that minimizes the expected prediction error. It can be shown that the optimal classifier is the one that predicts a new observation by such that

if is maximum among

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.

The Bayes rate is analogous to the irreducible error that we encountered in the regression setting. Even if we knew everything about the relationships between our ’s and , we can’t ever beat this error rate!

Unfortunately, we can not directly use the Bayes classifier since we do not know the distribution of . Different classifiers use different estimators of such conditional distributions – KNN uses proportion of points in the nearest neighbors belonging to each class as the estimator.

Thus, 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 :

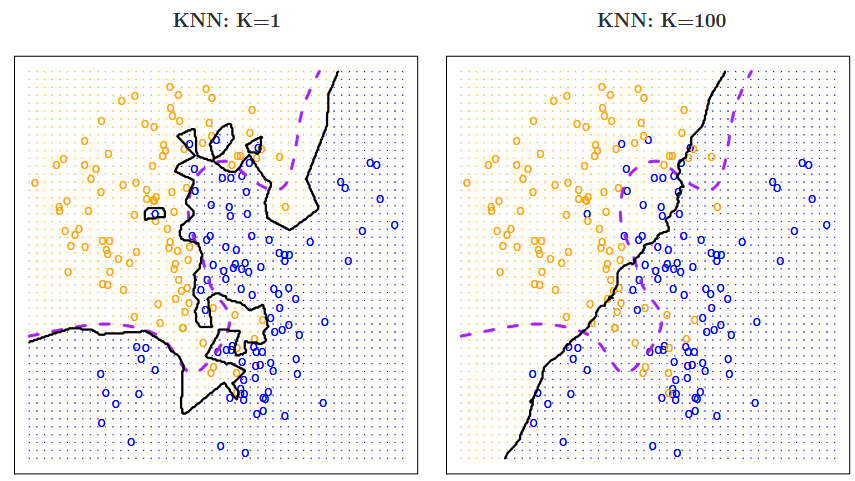
* *Directly estimating/modeling* 
  + An example of direct estimation of is the KNN classification technique, where the conditional probability is estimated by taking a majority vote from nearest point to .
  + Another example is *logistic* regression model, where the conditional probability is modeled 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.

How would the decision boundary look like for the Bayes classifier? For a two class problem, the Bayes classifier assigns the most probable class to a new data point. Thus for a new , it will be assigned to class 1 if and assigned to class 2 otherwise. Equivalently, will be assigned to class 1 if . Thus the decision boundary of the Bayes classifier is the set of all such that .

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!).

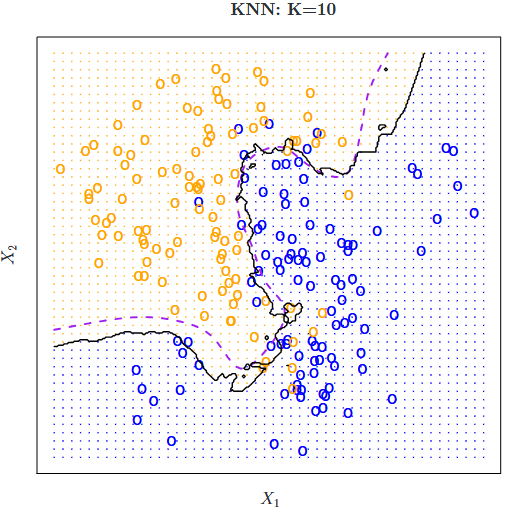


Impact of K on decision boundaries of KNN classifiers. The Bayes dicision boundary is shown using purple dashed line. Image adapted from Introduction to Statistical Learning.

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. Although it follows the Bayes boundary closely, it is overly flexible (uses local features) and tries to discover patterns that do not conform to the Bayes boundary. This is an example of overfitting a classification problem.

In contrast, for a large value of (such as ), the decision boundary is much smoother but does not capture the shape of the Bayes boundary. (Again we see the bias-variance trade-off here, even though we are in the classification setting.) Large values of result in a non-flexible (uses global features but averages over local ones) classifier that perhaps captures the overall trend of the Bayes boundary, but misses the details. In fact, as grows, the decision boundary will get closer to a straight line.

Therefore, 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

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.

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

A snapshot of the full data is shown below.

wines[1:4, ] |>  
 kable()

| Class | Alcohol | Malic | Ash | Alcal | Mg | Phenol | Flav | Nonf | Proan | Color | Hue | Abs | Proline |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 14.23 | 1.71 | 2.43 | 15.6 | 127 | 2.80 | 3.06 | 0.28 | 2.29 | 5.64 | 1.04 | 3.92 | 1065 |
| 1 | 13.20 | 1.78 | 2.14 | 11.2 | 100 | 2.65 | 2.76 | 0.26 | 1.28 | 4.38 | 1.05 | 3.40 | 1050 |
| 1 | 13.16 | 2.36 | 2.67 | 18.6 | 101 | 2.80 | 3.24 | 0.30 | 2.81 | 5.68 | 1.03 | 3.17 | 1185 |
| 1 | 14.37 | 1.95 | 2.50 | 16.8 | 113 | 3.85 | 3.49 | 0.24 | 2.18 | 7.80 | 0.86 | 3.45 | 1480 |

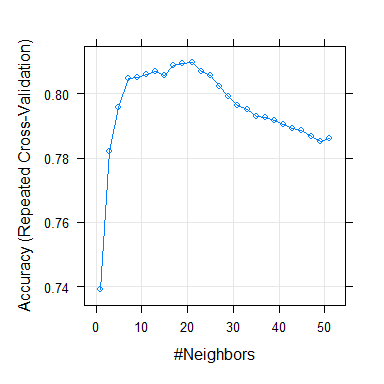
The goal is to find a *rule* that can assign a specimen of wine to its cultivar. In other words, we want to predict the classes (cultivar) based on the predictors (13 variables).

# classes of wine  
freq |>  
 kable()

| Cultivar | Freq |
| --- | --- |
| 1 | 59 |
| 2 | 71 |
| 3 | 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.

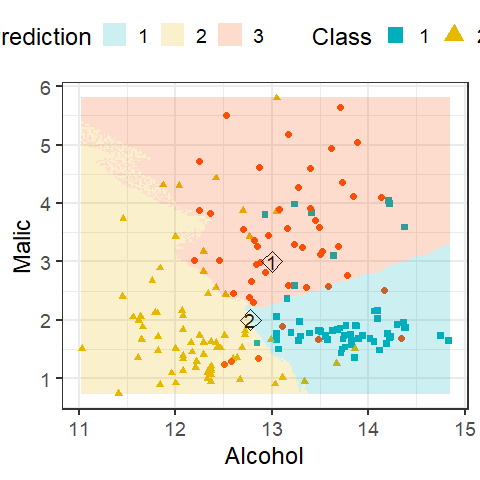
#best value of k  
fit$bestTune$k

[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")  
names(pred\_prob) <- c("Class 1", "Class 2", "Class 3")  
rownames(pred\_prob) <- c("New Obs 1", "New Obs 2")  
pred\_prob |>  
 kable()

|  | Class 1 | Class 2 | Class 3 |
| --- | --- | --- | --- |
| New Obs 1 | 0.1428571 | 0.0476190 | 0.8095238 |
| New Obs 2 | 0.3809524 | 0.4285714 | 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 uncertainlty 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 ,

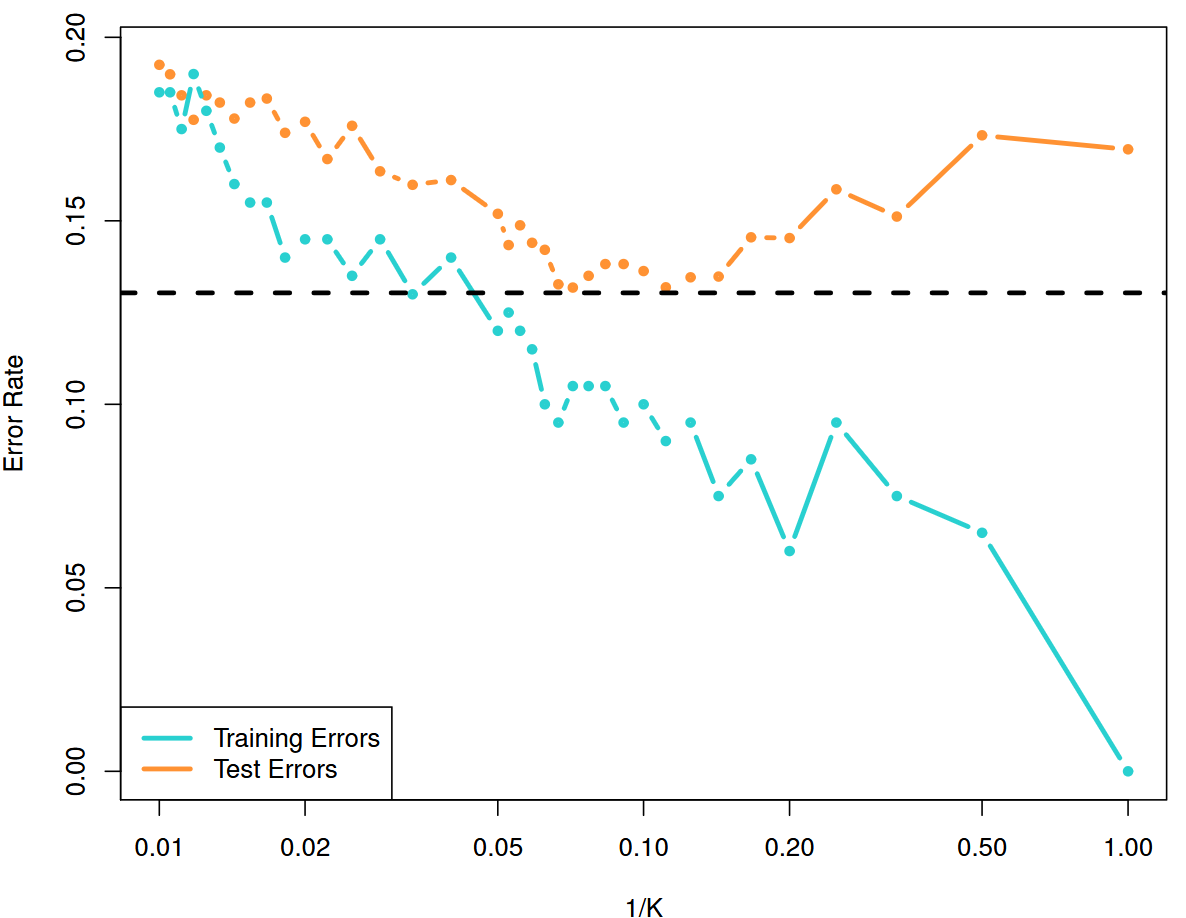
We can compute *training* and *test accuracy* depending on whether is the training or testing set.

**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 percentage of the largest class in the training set.

The NIR represents the accuracy that can be obtained without using any model. Thus for any classifier, the NIR should be the minimum accuracy it should have. Any classifier having accuracy better than NIR might be considered viable.

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)  
#check the best tuning parameter  
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.

conf\_output <- confusionMatrix(test$Class, preds)

The columns of the confusion matrix produced represent the ‘truth’ and the rows represent the predicted value by the model.

conf\_output$table |>  
 kable(row.names = TRUE)

|  | 1 | 2 | 3 |
| --- | --- | --- | --- |
| 1 | 11 | 0 | 6 |
| 2 | 0 | 19 | 2 |
| 3 | 2 | 1 | 11 |

Some metrics on the data are given in the $overall list element.

data.frame(value = conf\_output$overall) |>  
 kable()

|  | value |
| --- | --- |
| Accuracy | 0.7884615 |
| Kappa | 0.6816917 |
| AccuracyLower | 0.6529624 |
| AccuracyUpper | 0.8893885 |
| AccuracyNull | 0.3846154 |
| AccuracyPValue | 0.0000000 |
| McnemarPValue | NaN |

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.