Classification Tasks & Logistic Regression

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.

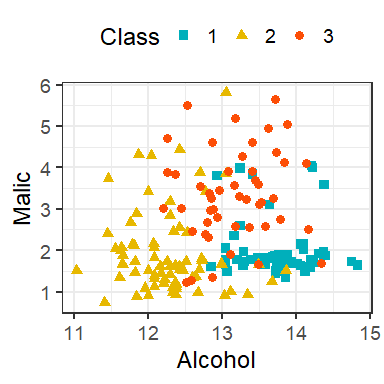
# Read the data  
wines <- read\_table("data/Wines.txt")  
#convert our response to a factor  
wines$Class <- as.factor(wines$Class)

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  
table(wines$Class)

1 2 3   
59 71 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.



Scatterplot of Alcohol vs. Malic in the wine data.

### -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.

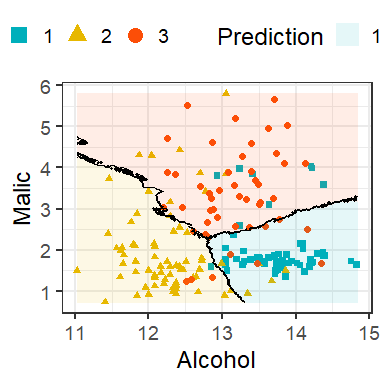
The process and concepts around train/test sets, tuning, etc. all follow through here. We’ll go through the example more explicitly soon. First, let’s just look at some KNN decision boundaries.

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"))  
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.

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.

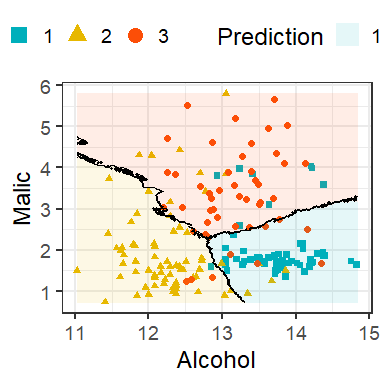
Thus, it is natural to try to estimate/model the conditional probabilities using the data, and use them to create classifiers.

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

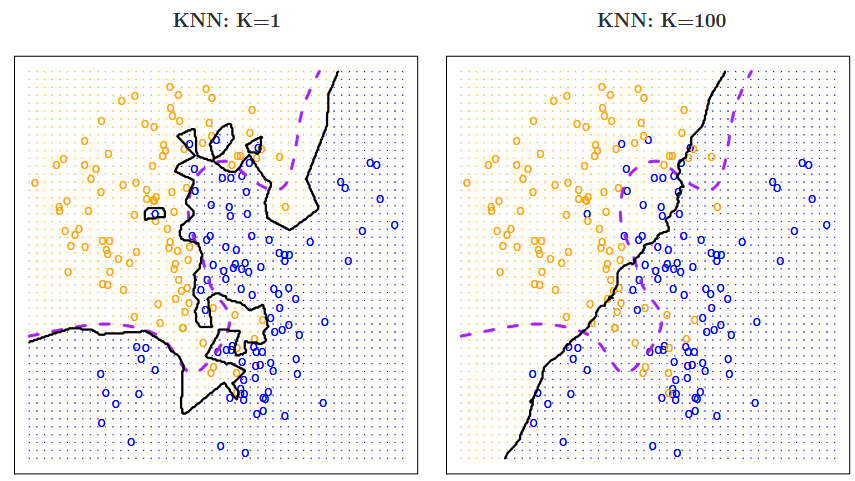
As with KNN regression, the hyperparameter determines how flexible the KNN method is. However, the idea of flexibility is subtle in this case. The boundary that separates the regions of the predictor space is effectively the classification rule for that classifier. The *decision boundary*, is the boundary of the regions.



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

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!).

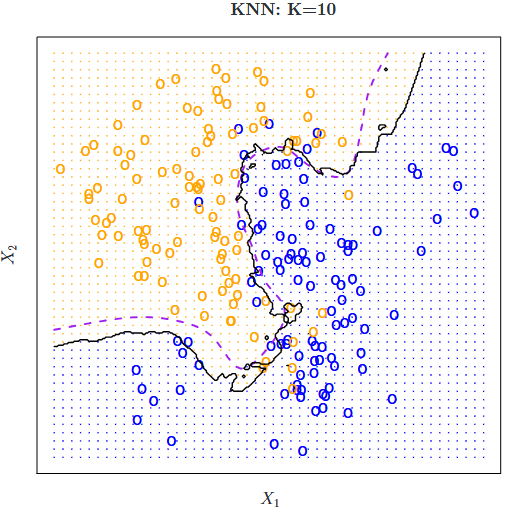


Impact of K on decision boundaries of KNN classifiers. The Bayes dicision boundary is shown using purple dashed line. Image adapted from *Introduction to Statistical Learning*.

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.

print(wines, n = 4)

# A tibble: 178 x 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>

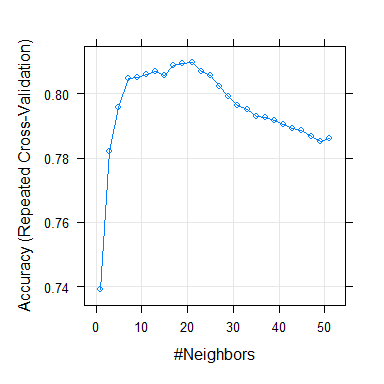
The goal is to find a *rule* that can assign a specimen of wine to its cultivar. In other words, we want to predict the classes (cultivar) based on the predictors (13 variables).

# 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.

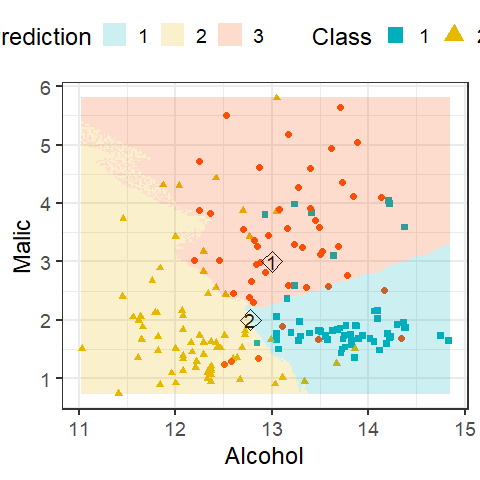
fit$bestTune$k

[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 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 ,

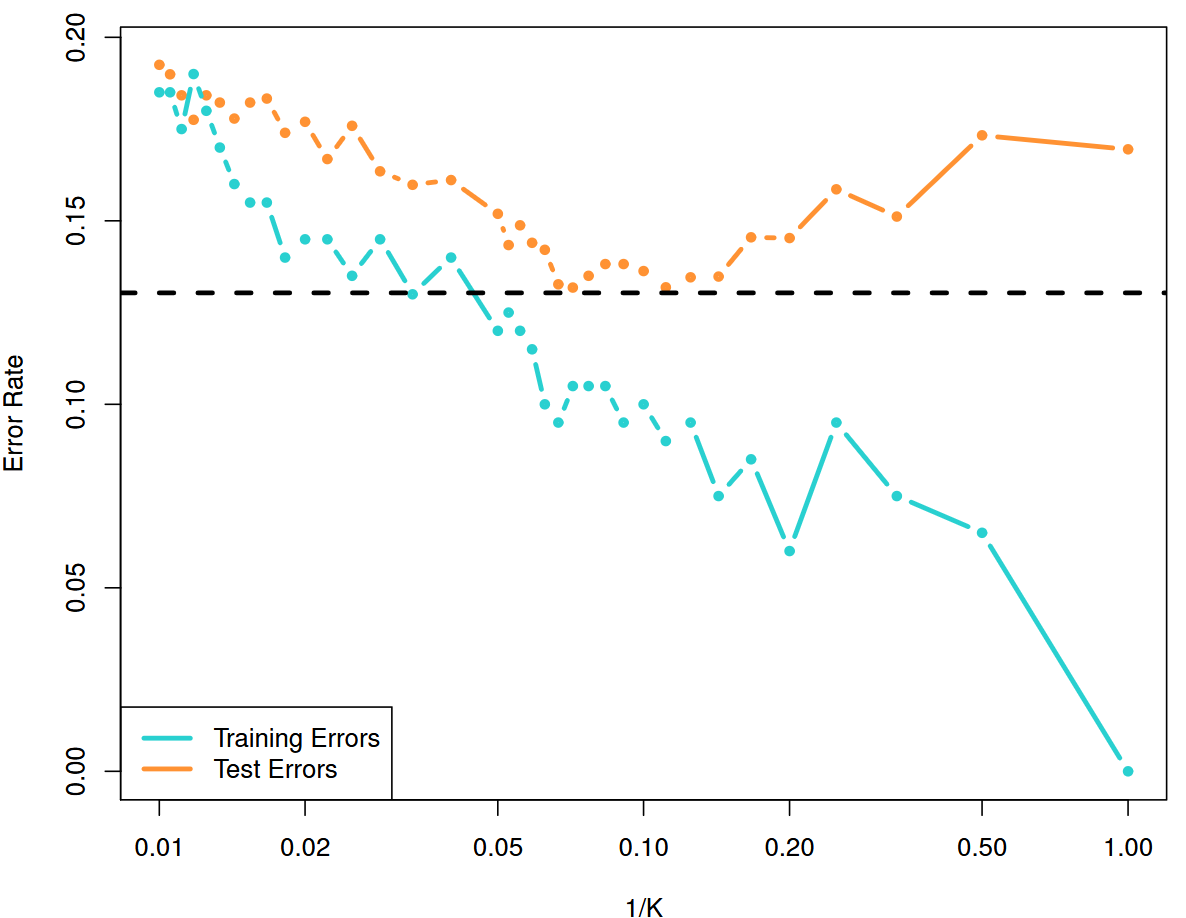
We can compute *training* and *test accuracy* depending on whether is the training or testing set.

**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)  
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.

# Logistic Regression

The logistic regression model arises from the desire to model the posterior probabilities of each of the classes as functions of the data vector without actually specifying any distribution of .

Usually, logistic regression models are used mostly as a tool for data analysis and inference, where the main goal is to understand the role of the predictors in explaining the outcome.

## Model

For simplicity, we start with the case where we have two classes (1 and 2), and suppose for an item, we have covariate . We can model posterior probabilities of the 2 classes via linear combinations of , such that the probabilities sum to one. We assume that has a Bernoulli distribution, and model the posterior probabilities as follows:

Here is the vector of coefficients of the covariates. Another way to write the same model is using *log-odds*:[[1]](#footnote-54)

$$
log\left[\frac{P(Y = 1 | \X)}{P(Y = 2 | X)}\right] = \beta\_0 + X\_{i1}\beta\_1 + \ldots + X\_{ip}\beta\_p.
$$

The expression is called the *odds* of being 1 vs being 2. The parameters quantifies the impact of the covariates to the prediction of class labels. (Note that the logistic regression is not just a classifier; it is a more general regression model.) The group used in the denominator (class 2 in our formulation above) is called the *reference group*. The choice of reference group is arbitrary as the estimates of the posterior probabilities are same.

## Odds and log-odds

Let us understand the concepts of *odds* and *log-odds* in more detail. Let us revisit wines data with classes (1 and 2) and only one covariate, $X = `Alcohol`$. The odds are defined as

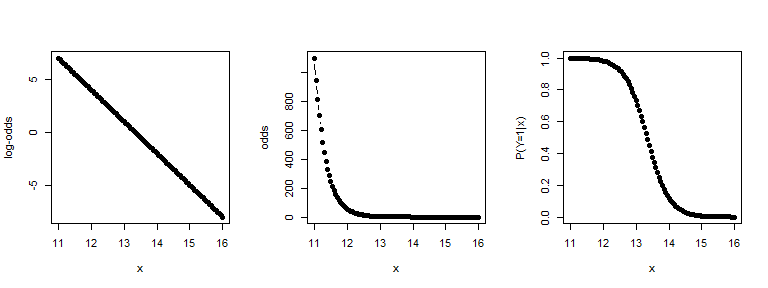
Thus, if leads to odds . In contrast leads to odds . Thus greater odds relate to higher posterior probability of class 1 (since class 2 is the reference class).

In terms of log-odds, we have the model

This model implies that with one unit increase in (Alcohol), the log-odds will change by units. This change in log-odds does *not* depend on the value of , that is, whether goes from 10 to 11, or from 13 to 14, the change in log-odds stays the same, . Equivalently, due to one unit increase in , the *odds* gets multiplied by . Specifically,

In terms of probabilities , we note that the relation between the posterior probabilities and odds (and ) is *not* linear:

Thus one unit increase in , or equivalently unit change in odds, does not result in a constant amount of change in . The actual change in depends on both the starting value of as well as the change in . For example, with and , Figure shows the plots of log-odds, odds, and over a grid of values of .



Plots of plots of log-odds, odds, and over a grid of values of .

## Classification

The model parameters can be estimated directly by maximum likelihood, solution is obtained numerically by iteratively reweighted least squares.[[2]](#footnote-60) It follows that the posterior probabilities can be estimated by

$$
\widehat P(Y = 1 | \X = \x) = \frac{exp(\widehat\beta\_0 + x\_1\widehat\beta\_1 + \ldots x\_p\widehat\beta\_p)}{1 + exp(\widehat\beta\_0 + x\_1\widehat\beta\_1 + \ldots x\_p\widehat\beta\_p)};
$$

$$
\widehat P(Y = 2 | \X = \x) = 1 - \widehat P(Y = 1 | \x),
$$

where ’s are the estimates of the regression coefficients. We can predict the class for a item with covariate $\x$ using the estimated probability that as follows:

The item is classified in group if $\widehat P(Y=1|\x) \geq \widehat P(Y=2|\x)$, otherwise in group .

Logistic regression can be performed using the glm() function in base R. For demonstration purposes, let us consider only two classes (1 and 2), and two covariates Alcohol and Proline. We create the small wines data – the only change is that now we explicitly change Class to a factor.

# wine data for classes 1 and 2  
wine\_new <- wines[wines$Class == 1 | wines$Class == 2, ]  
wine\_new$Class <- as.factor(wine\_new$Class)  
wine\_new$Class <- relevel(wine\_new$Class, ref = 2)

Note that we called the function relevel() with the argument "ref = 2". This is done to set class 2 as the reference group. Now we can perform logistic regression using the glm() function.

# Logistic regression  
wine\_glm = glm(Class ~ Proline + Alcohol,   
 family = binomial(),   
 data = wine\_new)

The first part Class ~ Proline + Alcohol is specifying Class as response and Proline and Alcohol as covariates. The statement family = binomial() is used to perform logistic regression.

The estimated coefficients are as follows.

wine\_coef <- wine\_glm$coefficients  
wine\_coef

(Intercept) Proline Alcohol   
-69.02763830 0.01369522 4.45310922

We can interpret as the *log-odds* when both Proline and Alcohol levels are zero. Thus we have the corresponding odds of . Keep in mind that in our dataset, zero values for Alcohol and Proline are not present, so such an interpretation is purely mechanical.

The estimated value of can be interpreted as the amount log-odds will *change* due to one unit *increase* in Proline while *keeping the Alcohol level fixed*. Thus keeping Alcohol level fixed, one unit increase in Proline level is associated with unit change in log-odds. Equivalently, odds will change by a multiplicative factor of (in other words, increase by percent). Similar interpretation can be given for .

Suppose we have a new sample with Proline = 600 and Alcohol = 13. So here and . We can compute the estimated posterior probabilities as

$$
\widehat P(Y = 2 | \x = (600,13)) = 1 - \widehat P(Y = 1 | x\_1 = 600, x\_2 = 13) = 0.9488304.
$$

Thus the new sample will be classified to class 2.

In R, we can simply use the predict() function to compute the probabilities shown above. By default, predict() gives the probability of *non-reference class*, class 1 in our example.

newx <- data.frame(Proline = 600,  
 Alcohol = 13)  
predict(wine\_glm,   
 newdata = newx,  
 type = "response")

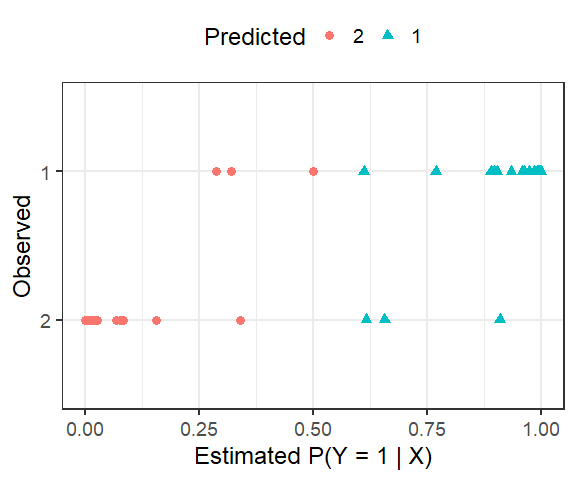
1   
0.05116964

We can also view the estimated posterior probabilities of the training set using predict() or using the $fitted.values component of the fit. The probabilities for the *non-reference group* is computed by default. We have also included the prediction of the whole dataset. Figure shows the posterior probabilities along with the true class labels.

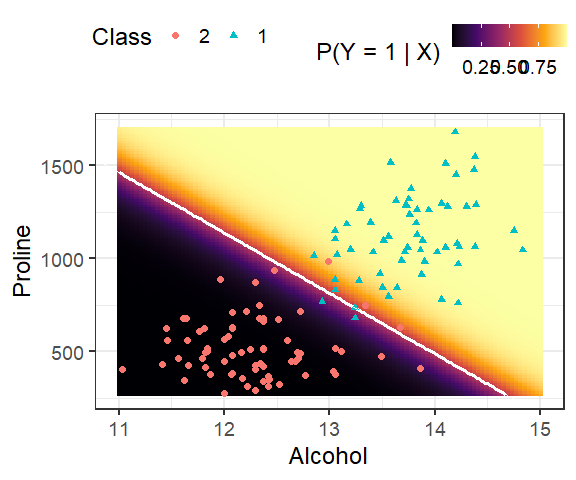
# Training set estimation of P(Y = 1)  
post.prob.1 = wine\_glm$fitted  
# Training set estimation of P(Y = 2)  
post.prob.2 = 1 - post.prob.1  
# Predicted groups  
Y.hat = as.factor(ifelse(post.prob.1>post.prob.2, 1, 2))  
Y.hat <- relevel(Y.hat, ref = 2)  
df <- data.frame("Class\_1" = post.prob.1,  
 "Class\_2" = post.prob.2,  
 "Predicted" = Y.hat,  
 "Observed" = wine\_new$Class)  
head(df, 4)

Class\_1 Class\_2 Predicted Observed  
1 0.9998671 1.329346e-04 1 1  
2 0.9842244 1.577560e-02 1 1  
3 0.9969940 3.006042e-03 1 1  
4 0.9999998 2.424491e-07 1 1

As an example, for the first wine sample (1st row), and . Thus this particular sample will be classified to group 1. Figure shows the decision boundary of the classifier, that is, all values of (Alcohol, Proline) that has the same posterior probability of being in class 1 or 2, or equivalently, odds of 1 and log-odds of 0. Formally, we have the boundary is given by all the solutions of the linear equation (the estimated formula of log-odds):



Esrimated posterior probabilities of class 1 along with the true class labels.



Decision boundary of a 2-class logistic regression based classifier.

We can use the function errormatrix() in the klaR package to obtain the training confusion matrix or, alternatively, we can also use the function confusionMatrix() in the caret package for a detailed output.]

# Confusion matrix  
err <- klaR::errormatrix(true = wine\_new$Class,   
 predicted = Y.hat,   
 relative = TRUE)  
round(err, 3)

predicted  
true 1 2 3 -SUM-  
 1 0.949 0.051 0 0.051  
 2 0.042 0.958 0 0.042  
 3 0.000 0.000 0 0.000  
 -SUM- 0.500 0.500 0 0.046

Ideally, we should use cross-validation, training-testing sets to estimate the accuracy, as we have learned before.

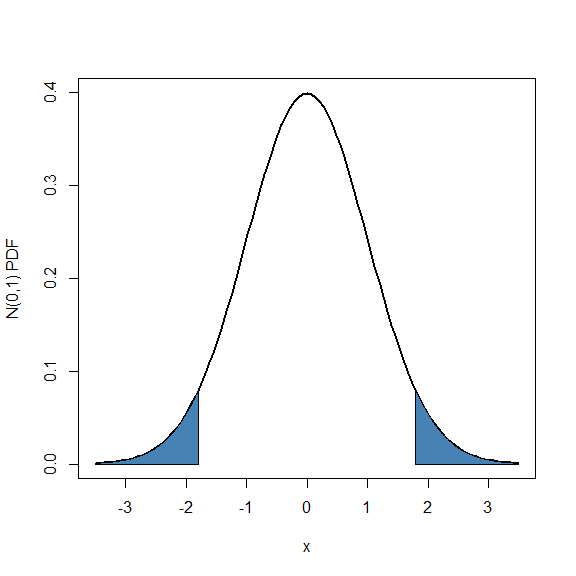
## Hypothesis testing and confidence intervals

We can get a summary of the fit as follows.

# testing each beta coefficient  
summary(wine\_glm)

Call:  
glm(formula = Class ~ Proline + Alcohol, family = binomial(),   
 data = wine\_new)  
  
Deviance Residuals:   
 Min 1Q Median 3Q Max   
-2.19220 -0.03196 -0.00558 0.02786 1.57716   
  
Coefficients:  
 Estimate Std. Error z value Pr(>|z|)   
(Intercept) -69.027638 22.757675 -3.033 0.00242 \*\*  
Proline 0.013695 0.004362 3.140 0.00169 \*\*  
Alcohol 4.453109 1.592916 2.796 0.00518 \*\*  
---  
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
  
(Dispersion parameter for binomial family taken to be 1)  
  
 Null deviance: 179.11 on 129 degrees of freedom  
Residual deviance: 19.72 on 127 degrees of freedom  
AIC: 25.72  
  
Number of Fisher Scoring iterations: 9

The summary of the fit produces -tests for coefficient of each covariate[[3]](#footnote-68); it seems both alcohol and proline are associated with the group labels. Formally, the test statistic is essentially same as in linear regression: suppose we want to test whether Alcohol (the second predictor with coefficient ) has any association with . Thus we test for vs. . The test statistic is



Two-tailed p-value for a z-test.

Using large sample theory it can be shown that approximately follows a distribution. Thus we reject for large positive or large negative values of . Equivalently, the p-value can be computed as

where denotes random variable. Figure shows the p-value of a two-sided -test.

We can produce large sample confidence intervals for as

where is the quantile of the distribution. For example, a confidence intervals for is

We can interpret this intervals as follows: with every unit increase in level of (Proline in our example), we can expect an *increase in log-odds* by an amount of 0.0051456 to 0.0222448. Equivalently, every unit increase in level of , *odds* will be changed by a factor of 1.0051589 to 1.0224941 (in other words, increase in odds will be between 0.516 percent and 2.249 percent.)

## Logistic regression with multiple classes

We can extend logistic regression presented for two classes to the case of multiple classes; the regression method is called *Multinomial Logistic Regression*. Suppose we have classes, and we take the -th class as the reference. The log-odds of classes vs the reference class are modeled as follows:

Some algebra shows that the corresponding posterior probabilities are as follows:

We can similarly build a classification rule as follows.

An item with covariate is predicted to be in class if the estimated probability is larger than the other posterior probabilities.

We can use the multinom() function in the nnet library to perform multinomial logistic regression. Let us consider the wine data with all the three classes.

# Convert Class in wines data to a fcator  
# and relevel to make 3 as reference  
wines$Class <- as.factor(wines$Class)  
wines$Class <- relevel(wines$Class, ref=3)  
# multinomial logistic regression  
multilogit <- multinom(Class ~ Proline + Alcohol,   
 data = wines,   
 maxit = 200, trace=FALSE)  
# summary  
summary(multilogit)

Call:  
multinom(formula = Class ~ Proline + Alcohol, data = wines, maxit = 200,   
 trace = FALSE)  
  
Coefficients:  
 (Intercept) Proline Alcohol  
1 -45.59355 0.017412051 2.348672  
2 39.00287 -0.004735415 -2.824180  
  
Std. Errors:  
 (Intercept) Proline Alcohol  
1 0.012260974 0.003256728 0.19427257  
2 0.006280407 0.001748203 0.08287512  
  
Residual Deviance: 135.2578   
AIC: 147.2578

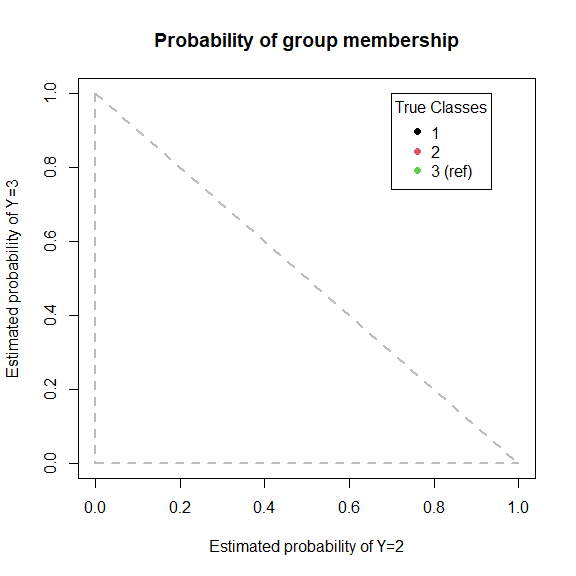
The option “trace = FALSE” in multinom() function suppresses the printing of convergence steps. The option “maxit = 200” sets the upper bound of the number of iterations to be performed to find the solutions.

The estimated posterior probabilities are as follows.

# estimated posterior probabilities  
probs <- multilogit$fitted.values  
head(probs)

3 1 2  
1 1.705991e-03 0.9982906 3.365146e-06  
2 2.430299e-02 0.9747533 9.437503e-04  
3 2.603414e-03 0.9973369 5.972627e-05  
4 8.947561e-07 0.9999991 1.665497e-10  
5 7.479590e-01 0.1367403 1.153007e-01  
6 2.248876e-06 0.9999978 7.798500e-10

Warning in plot.xy(xy, type, ...): supplied color is neither numeric nor  
character



Estimated posterior probailities for the wine data using multinomial logistic regression.

Each row shows the estimated probability of group membership for the corresponding wine sample. For example, the first wine sample (first row), has a 99.82% probability of being in group 1. We can also visualize the posterior probabilities as shown in Figure .

We can also hypothesis testing for each using the standard errors reported in the summary output.

# Extract the coefficients and se  
mlogit\_sum <- summary(multilogit)  
coef <- mlogit\_sum$coefficients  
se <- mlogit\_sum$standard.errors  
# z-statistics  
z <- coef/se  
z

(Intercept) Proline Alcohol  
1 -3718.591 5.346487 12.08957  
2 6210.246 -2.708733 -34.07754

# p-value  
pv <- 2\*pnorm(abs(z), lower.tail = FALSE)  
pv

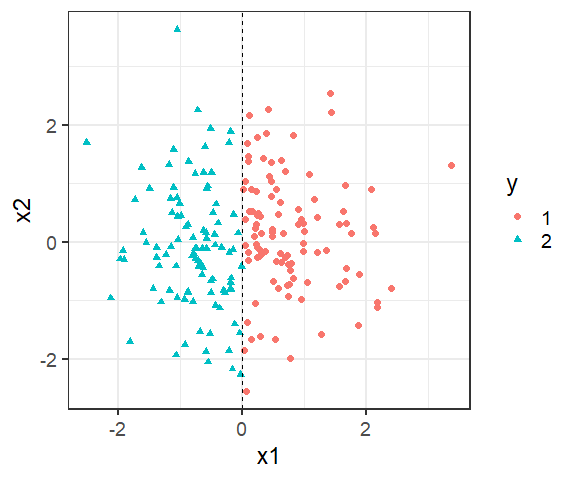
(Intercept) Proline Alcohol  
1 0 8.967795e-08 1.199076e-33  
2 0 6.754064e-03 1.587078e-254

We see for both classes 1 and 2, the two predictor variables are significant at any reasonable test level.

## Issues to consider

There are some situations where logistic regression might not perform well. One such situation is *complete (or quasi-complete) separation* of the data.

This situation happens when the outcome variable separates a predictor completely. This leads to perfect prediction of the outcome by the predictor. Consider the following data set with binary response and two predictors and . Figure shows relationship between and the two predictors. In such a case, logistic regression may produce unreasonable over-inflated estimates of regression coefficients.



Simulated data set. The response separates completely, but not .

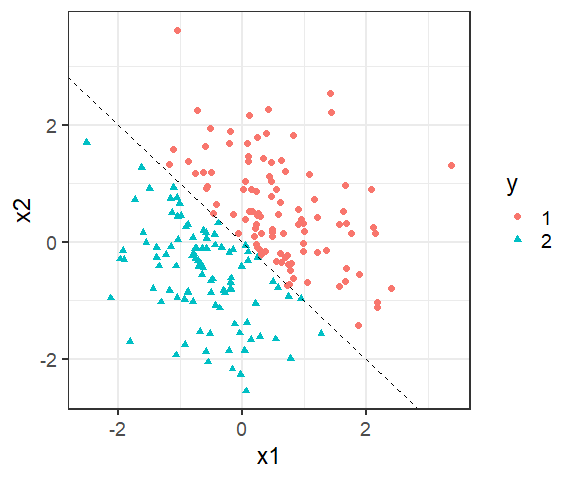
glm(y ~ x1 + x2, family = binomial())

Warning: glm.fit: algorithm did not converge

Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

Call: glm(formula = y ~ x1 + x2, family = binomial())  
  
Coefficients:  
(Intercept) x1 x2   
 6.801 -851.898 -5.579   
  
Degrees of Freedom: 199 Total (i.e. Null); 197 Residual  
Null Deviance: 277.2   
Residual Deviance: 4.285e-07 AIC: 6

In general, if there is a linear combination that is completely separated by , logistic regression will fail to produce reasonable results. Figure shows one such example where the data is completely separated by the line .



Simulated data set. The response separates the data completely – the boundary (dashed line) is . Negative values of corresponds to , and positive values corresponds to .

glm(y ~ x1 + x2, family = binomial())

Warning: glm.fit: algorithm did not converge

Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

Call: glm(formula = y ~ x1 + x2, family = binomial())  
  
Coefficients:  
(Intercept) x1 x2   
 14.54 -1578.34 -1550.26   
  
Degrees of Freedom: 199 Total (i.e. Null); 197 Residual  
Null Deviance: 277.1   
Residual Deviance: 2.051e-06 AIC: 6

Although the examples above shows complete separation using continuous predictors, it is more like to happen when using categorical predictors coded by dummy variables. Small sample size might contribute to this problem as well. In such situations, applying other classification methods (e.g., LDA) is preferred.

Since logistic regression deals with binary outcome, often it requires more sample size that linear regression. Multinomial logit regression requires even more sample size than binary logistic regression due to the fact that it estimates parameters for multiple classes.

In presence of categorical predictor, it might happen that there are some combination of predictor and response values that are not present in the data. In such a case, logistic fit may become unstable, or might even fail to converge.

As a practical example of perfect separability, consider the wines data with two classes, “1” and “2”, but with all 13 predictors. Note that glm did not converge, and produces extremely inflated standard errors.

wines <- read.table("data/Wines.txt", header = TRUE)  
wines\_all <- glm(as.factor(Class) ~ .,   
 data = wines[1:130,],   
 family = binomial())

Warning: glm.fit: algorithm did not converge

Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

summary(wines\_all)

Call:  
glm(formula = as.factor(Class) ~ ., family = binomial(), data = wines[1:130,   
 ])  
  
Deviance Residuals:   
 Min 1Q Median 3Q Max   
-2.553e-05 -2.110e-08 2.110e-08 2.110e-08 2.919e-05   
  
Coefficients:  
 Estimate Std. Error z value Pr(>|z|)  
(Intercept) 4.358e+02 1.067e+06 0.000 1  
Alcohol -1.975e+01 9.891e+04 0.000 1  
Malic -9.063e+00 5.437e+04 0.000 1  
Ash -5.743e+01 1.170e+05 0.000 1  
Alcal 7.387e+00 1.396e+04 0.001 1  
Mg -3.931e-01 2.462e+03 0.000 1  
Phenol 4.134e+00 2.037e+05 0.000 1  
Flav -1.271e+01 8.464e+04 0.000 1  
Nonf 1.805e+01 7.368e+05 0.000 1  
Proan 1.448e+01 7.417e+04 0.000 1  
Color 3.624e+00 4.266e+04 0.000 1  
Hue 1.966e+01 5.242e+05 0.000 1  
Abs -3.509e+01 9.601e+04 0.000 1  
Proline -8.307e-02 2.426e+02 0.000 1  
  
(Dispersion parameter for binomial family taken to be 1)  
  
 Null deviance: 1.7911e+02 on 129 degrees of freedom  
Residual deviance: 4.4477e-09 on 116 degrees of freedom  
AIC: 28  
  
Number of Fisher Scoring iterations: 25

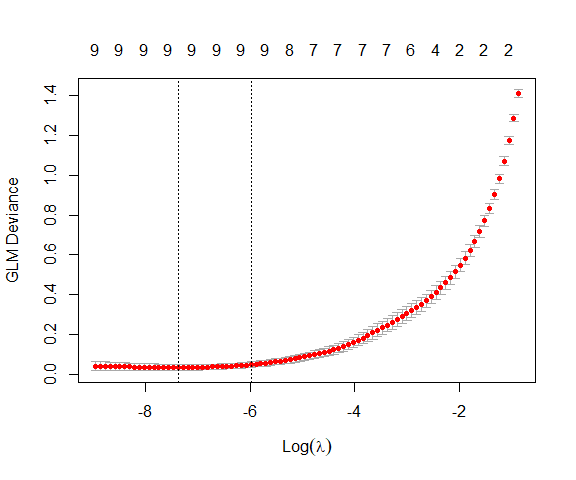
# High-Dimensional Problems

When the number of predictors is larger than (or close to) the sample size , the methods described in this section suffer from numerical instability or simply can not be applied to the data.[[4]](#footnote-85) We can apply similar strategies discussed for linear regression here as well: regularization/shrinkage and dimension reduction methods. Some classifier such as naive Bayes are more appropriate in hig-dimensional data than others.

For example, we can extend LDA for high-dimensional data by assuming a *diagonal* covariance matrix (i.e., assuming features are independent in each class). This method is called *Diagonal Linear Discriminant Analysis*. Similar approach can be taken for QDA as well resulting in *Diagonal Quadratic Discriminant Analysis*. Regularized versions of LDA such as *nearest shrunken centroids (NSC)*, *Regularized discriminant analysis (RDA)*, and many other methods are also available in literature. R packages such as sparsediscrim, HDclassif, HiDimDA among others have various classifiers for high-dimensional data.

Like linear regression, we can develop ridge, lasso and elastic net methods for logistic regression as well. All these methods are available in glmnet() package. As before, these methods shrink the regression coefficients towards zero, can be used in high-dimensional setting. We show the lasso based logistic regression fit of the two-class wines data (class 1 and 2) with the penalty parameter chosen by CV below (glmnet() automatically scales the predictors before estimating the regression coefficients, and then outputs the coefficients in the original scale.)

set.seed(1102)  
# CV to choose lambda  
logit\_cv <- cv.glmnet(x = as.matrix(wine\_new[,-1]),   
 y = wine\_new$Class,  
 family = binomial(),  
 alpha = 1)



CV results for logistic regression of wines data with lasso penalty.

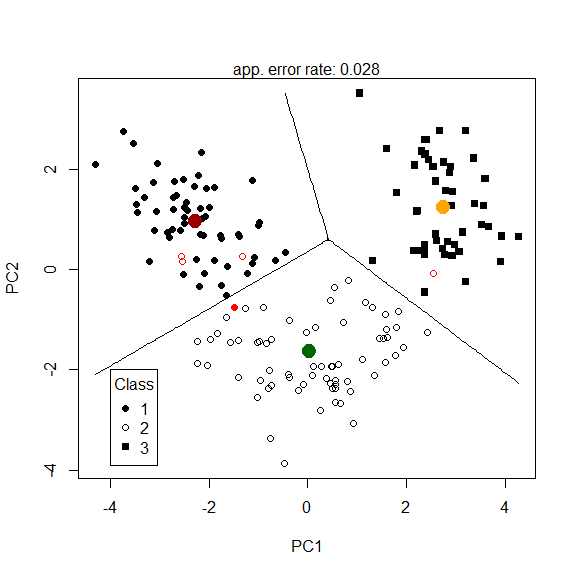
# Final fit with lambda chosen by 1-SE rule  
wine\_lasso <- glmnet(x = as.matrix(wine\_new[,-1]),   
 y = wine\_new$Class,  
 family = binomial(),  
 alpha = 1,  
 lambda = logit\_cv$lambda.1se)

# Estimated coefs  
coef(wine\_lasso)

14 x 1 sparse Matrix of class "dgCMatrix"  
 s0  
(Intercept) -55.12013884  
Alcohol 2.71260772  
Malic 0.90390641  
Ash 5.23709395  
Alcal -0.55394362  
Mg .   
Phenol .   
Flav 0.27146489  
Nonf .   
Proan .   
Color 0.44396146  
Hue -0.55507251  
Abs 2.12543584  
Proline 0.01019674

Dimension reduction method like PCA can still be applied to the predictors before building classifiers. PCA often is useful in providing better visualization of the data. For example, the full wines data has 13 predictors, which are hard to visualize in a plot. Let us perform PCA and plot the first few PCs along with class labels.

# wine data PCA  
wine\_pca <- prcomp(wines[, -1], scale. = TRUE)  
wine\_score <- wine\_pca$x



LDA based classification of wines data using first two PCs.

Figure shows results from LDA using only the first two PCs as predictors. We can see that PC1 and PC2 almost completely separate the three classes. Thus we have reduced dimension from 13 to two. Keep in mid that PCA is an unsupervised technique, and such a nice classification performance may not always happen. Also, even though PC1 and PC2 lead to excellent classification, together they only explain variation of the data. The opposite can be true as well – the first two PCs of some data might explain a large amount of variation but fail to produce good classification results.

summary(wine\_pca)

Importance of components:  
 PC1 PC2 PC3 PC4 PC5 PC6 PC7  
Standard deviation 2.169 1.5802 1.2025 0.9587 0.92369 0.80103 0.74232  
Proportion of Variance 0.362 0.1921 0.1112 0.0707 0.06563 0.04936 0.04239  
Cumulative Proportion 0.362 0.5541 0.6653 0.7360 0.80163 0.85098 0.89337  
 PC8 PC9 PC10 PC11 PC12 PC13  
Standard deviation 0.59039 0.53746 0.5008 0.47517 0.41079 0.32151  
Proportion of Variance 0.02681 0.02222 0.0193 0.01737 0.01298 0.00795  
Cumulative Proportion 0.92018 0.94240 0.9617 0.97907 0.99205 1.00000

1. Thus we are modeling the log-odds of class 1 vs 2 as a linear function of . [↑](#footnote-ref-54)
2. Essentially using a Newton–Raphson algorithm; see for example Hastie, Tibshirani and Friedman (2009), The Elements of Statistical Learning. [↑](#footnote-ref-60)
3. This is actually an approximate (large sample) test. [↑](#footnote-ref-68)
4. Recall same issues in Linear Regression. [↑](#footnote-ref-85)