Classification II

Introduction to Computational BioStatistics with R

Alexey Fedoseev

November 13, 2025

Classification Approaches

There are lots of classification approaches which one might use.

- **Decision trees**: analyze the features of the data and make ‘decisions’ about how to ‘split’ the data into uniform groups.
- **Logistic regression**: like linear regression, but now we fit a “yes/no” function to the data.
- **Naive Bayes**: a type of probabilistic analysis.
- **kNN**: k Nearest Neighbours; use the k nearest neighbours to a data point to predict the category of a new data point.
- **Support Vector Machines**: essentially a linear model of the data, used for separate groups.
- **Neural networks**: a weird algorithmic approach to using functions to categorize data.

Today we will go over logistic regression and support vector machines.

Fitting a Line to Categorical Data

We have the following data and we want to know how well we can predict the Outcome column:

x	Outcome
-2.5	No
-1.5	No
-0.5	No
0.5	Yes
1.5	Yes
2.5	Yes

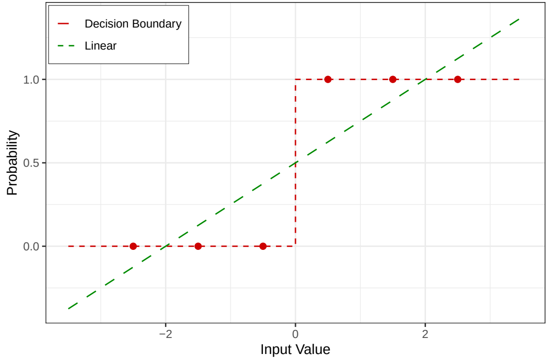
We can replace ‘No’ with 0, ‘Yes’ with 1, and fit a line. Let’s do this.

Logistic Regression

A simple but flawed approach is to apply linear regression to the 0/1 outcome, then classify based on whether the prediction exceeds 0.5.

However, linear regression can produce predictions outside [0,1], making it unsuitable for probabilities, and its assumptions may not hold for classification boundaries.

We also introduced a numerical relationship between the Yes and No outcomes, which may not exist.



The Logistic (Sigmoid) Function

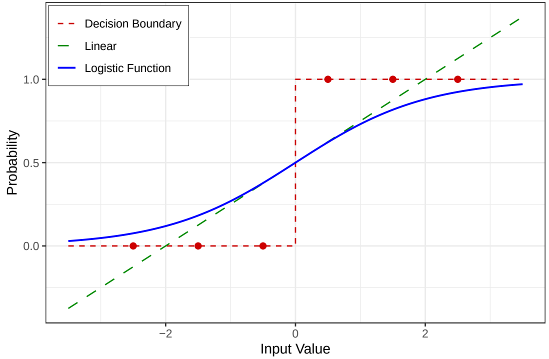
Logistic regression uses the sigmoid function to map linear predictions to probabilities:

$$p = \frac{1}{1 + e^{-\mathbf{x} \cdot \beta}}$$

- p is the predicted probability of the positive class (“Yes”).
- x represents the input features (measurements).
- β are the model coefficients (weights) learned from the data.

Outputs are between 0 and 1.

Provides probabilistic predictions for binary classification.



Logistic regression, example

Logistic regression in R is usually performed using a glm.

- You will need to install the ‘mlbench’ package to get this data.
- The ‘complete.cases’ function returns a boolean vector, indicating which rows have complete data.
- We specify “family = ‘binomial’ ” when we want to perform logistic regression.

```r
> data(BreastCancer, package = 'mlbench')
>
> bc <- BreastCancer[complete.cases(BreastCancer),]
>
> ind <- sample(c(TRUE, FALSE), nrow(bc),
+   replace = TRUE, prob = c(0.7, 0.3))
>
> train.d <- bc[ind,]
> test.d <- bc[!ind,]
>
> model <- glm(Class ~ Cl.thickness + Cell.size +
+   Cell.shape, family = 'binomial',
+   data = train.d)
```

Logistic regression, example, continued

Predictions using logistic regression need some postprocessing.

- Use the “type = ‘response’ ” flag when using predict with logistic regression.
- The returned values are probabilities of ‘success’. These must be converted to a classification.
- The ifelse function applies a conditional to each element, and returns the first argument for a TRUE value, the second for FALSE.

```
> pred <- predict(model, newdata = test.d,
+   type = 'response')
> new.pred <- ifelse(pred > 0.5,
+   'malignant', 'benign')
> conf <- table(test.d$Class, as.factor(new.pred))
> conf
           benign malignant
benign        122         4
malignant       4        60
>
> sum(diag(conf)) / sum(conf)
[1] 0.9578947
```

Evaluating binary classifiers

Binary classification is a common and important enough special case that its confusion matrix elements have special names, and various quality measures are defined.

Note that “Positive” here refers to “category 1” and “Negative” means “category 0”.

	Classified Positive (CP)	Classified Negative (CN)
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

One can always get exactly one of FN or FP to be zero (for example, just classify everything positive, then there will never be any false negatives).

But there is usually a tradeoff between false positives and false negatives.

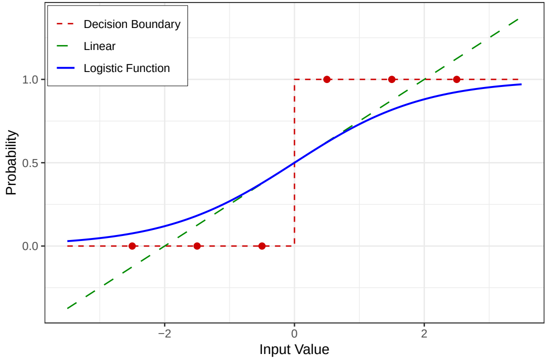
Classification thresholds

In most binary classifiers, there's some equivalent of a threshold you can set. This threshold determines when a given data point moves from one categorization to the other.

For the case of logistic regression, the default threshold is 0.5.

- Set it lower (allow more true, but also false, positives).
- Set it higher (allow more true, but also false, negatives).

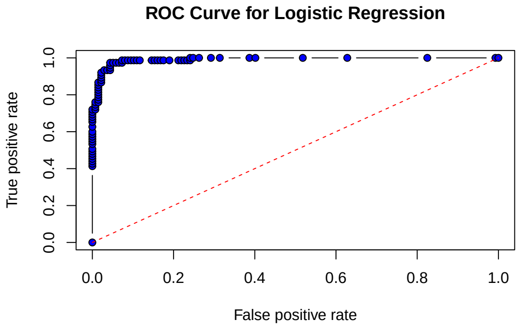
Note that 0.5 is arbitrary; in some fields (medical screening), lower thresholds allow more false positives to minimize missed cases.



ROC curve

By varying the classification threshold, from 1 to 0, we can get a collection of points for the TPR and FPR. Plotting the two measures on either axis gives a ROC (Receiver Operating Characteristic) curve.

- The diagonal line represents random chance.
- We want our curve to be as high above the diagonal as possible.

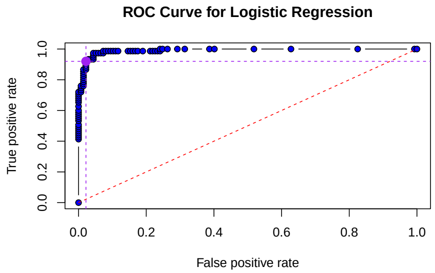


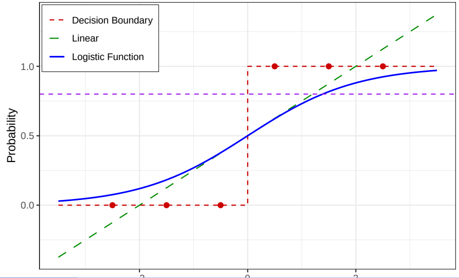
How to Read an ROC Curve

The ROC curve is built by varying the classification threshold from 1 to 0:

- Start at lower-left (threshold = 1): FPR = 0, TPR = 0 – nothing classified as positive.
- Lower to 0.9, 0.8: TPR rises (correct positives), FPR stays 0 (no false positives yet).
- Toward 0.5: TPR higher, FPR starts rising due to noisy data (some false positives).
- Around the curve's corner: Balancing TP and FP.
- At 0.3, 0.2: TPR approaches 1 (most positives caught), FPR higher.
- At threshold = 0: TPR = 1, FPR = 1 – everything classified as positive (all TP, all FP).

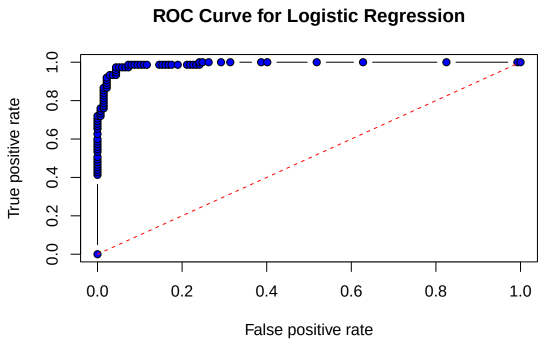
Better curves hug the top-left; diagonal is random (AUC=0.5).





ROC curve, continued

```
> library(ROCR)
> ROC.pred <- prediction(pred,
+   test.d$Class)
> ROC.perf <- performance(ROC.pred,
+   measure = 'tpr',
+   x.measure = 'fpr')
> plot(ROC.perf, type = 'b', pch = 21,
+   bg = 'blue')
> lines(c(0, 1), c(0, 1), lty = 2)
```



Note that your curve will look different from this one, due to randomness.

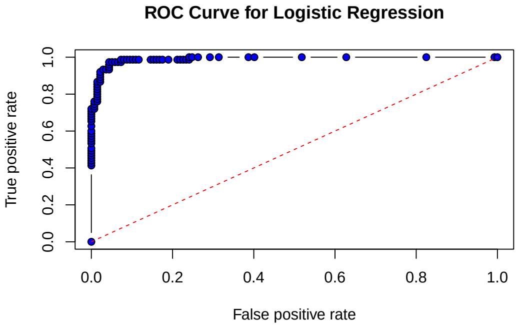
ROC curve, continued more

The quality of a classifier is given by the ROC curve's AUC (area under the curve).

- The worst classifiers will have an AUC near 0.5.
- Good classifiers have an AUC near 1.0.

For the non-binary classification situation, you create "one versus all" ROC curves, with one ROC curve for each category.

```
> ROC.AUC <- performance(ROC.pred,
+   measure = 'auc')
> ROC.AUC@y.values
[[1]]
[1] 0.9786706
```



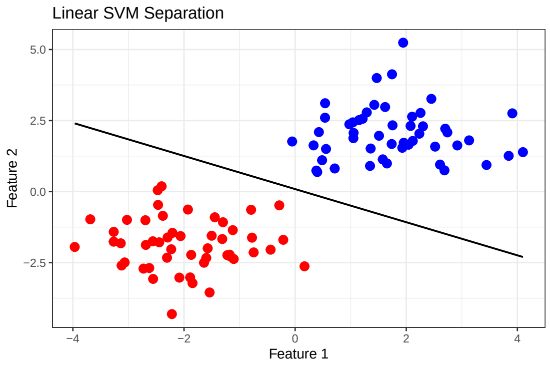
Support Vector Machines

Support Vector Machines (SVM) are geometric algorithms that find the optimal hyperplane to linearly separate data into categories.

Linear SVMs determine a hyperplane which linearly separates the data set into distinct categories.

The goal is to find the plane farthest from all closest points (support vectors) in k-dimensional space.

This is formulated as a minimization problem.



Support Vector Machines, example

To demonstrate the use of Support Vector Machines we will use the heart disease data set.

Note that this dataset does not have a header row and has 13 features.

The hyper-plane which passes through the data will be 12D.

```
> url <- 'https://dataaspirant.com/wp-content/uploads/2017/01/heart_tidy.csv'
> heart.data <- read.csv(url, header = F)
> heart.data$V14 <- as.factor(heart.data$V14)
>
> ind <- sample(c(TRUE, FALSE), nrow(heart.data),
+   replace = TRUE, prob = c(0.7, 0.3))
>
> train.d <- heart.data[ind,]
> test.d <- heart.data[!ind,]
```

Support Vector Machines, example

We use cross-validation to tune the SVM parameters.

Centering and scaling the data is important since SVM is a geometric algorithm sensitive to feature scales.

```
> library(caret)
> svmFit <- train(V14 ~ .,
+   data = train.d, method = 'svmLinear',
+   preProcess = c('center', 'scale'),
+   trControl = trainControl(method = 'cv', number = 10),
+   tuneLength = 10)  # Tries 10 values of the C parameter
```

Parameter C controls how much we penalize misclassifications: higher C for noisy data (stricter), lower C for clean data (wider margin).

Support Vector Machines, example, continued

Support Vector Machines take an optional argument, 'C' (the penalty parameter).

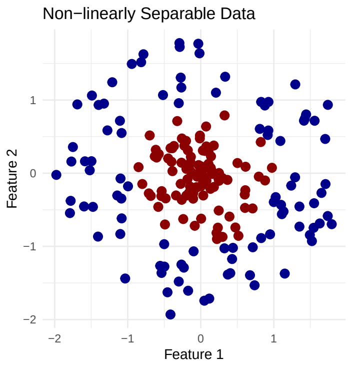
- Large values of C mean that we lack confidence in the data's distribution (noisy data).
- Small values mean the opposite.
- Default value is 1.0.

```
> pred <- predict(svmFit, newdata = test.d)

> confusionMatrix(pred, test.d$V14)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 42  9
         1  5 32

Accuracy : 0.8409
...
```

Nonlinear Support Vector Machines

Linear Support Vector Machines are all well and good if your data are linearly separated. But suppose we have multiple groups and a straight line is not going to separate well between them.

As we can see in the example at right, you can imagine situations where there are clearly defined clusters of data, but fitting linearly is not an option.



Nonlinear Support Vector Machines, continued

We have a technique which is quite good at finding hyperplanes in linearly separated data.

- If the data are not linearly separated, the solution, obviously (!), is to nonlinearly transform the data into a space where it is linearly separable.
- This typically involves adding dimensions to the data which did not previously exist.
- This increases the likelihood of making things linearly separable (Cover's theorem).

Great! But how do we figure out what transformation to apply to the data?

- We don't. We let the SVM kernel do the work for us.
- SVMs use 'kernel functions' to transform the data into the required form.
- There are many types of kernel functions available: linear (which we've already been using), polynomial, radial basis function (RBF), sigmoid, and others.
- Once the hyperplane has been determined in the transformed space, the hyperplane is transformed back into regular data space.

Nonlinear SVM, example

Invoking a nonlinear support vector machine is as simple as specifying "svmRadial" (for the radial basis function kernel).

Because the result has so many dimensions we're not going to try to plot the actual plane.

```
> svmNLFit <- train(V14 ~ .,
+   data = train.d, method = 'svmRadial',
+   preProcess = c('center', 'scale'),
+   trControl = trainControl(method = 'cv',
+                            number = 10),
+   tuneLength = 10)
```

Nonlinear SVM, example, continued

Cross-validation was used to tune the (hidden) hyperparameters.

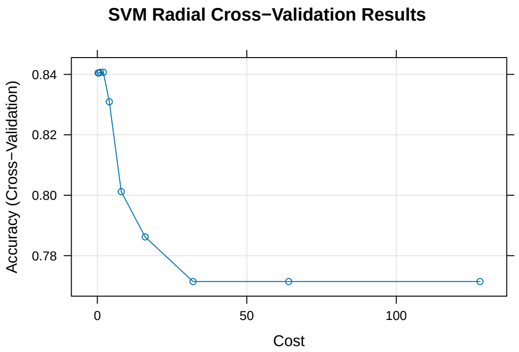
Printing out the model shows how the accuracy changed as a function of the cost (‘penalty’) hyperparameter.

```
> svmNLFit
Support Vector Machines with Radial Basis Function Kernel
...
C        Accuracy   Kappa
0.25   0.8351299  0.6628577
0.50   0.8353680  0.6632606
1.00   0.8256061  0.6434020
...
64.00   0.7579870  0.5100520
128.00  0.7627489  0.5197271

Tuning parameter 'sigma' was held at a value of 0.04483339
Accuracy was used to select the optimal model...
The final model values were sigma = 0.04483339 and C = 0.5
```

Nonlinear SVM, example, continued more

We can plot the cross-validation accuracy as a function of the cost parameter.

```
> plot(svmNLFit)
```



Nonlinear SVM, example, continued even more

As always, we're interested in how well the model does on the test data.

The nonlinear SVM model does slightly better on the test data than the linear SVM, though not much.

```
> NLpred <- predict(svmNLFit, newdata = test.d)
>
> confusionMatrix(NLpred, test.d$V14)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 43  9
         1  4 32

Accuracy : 0.8523
...
```

Summary

You've now seen four classification algorithms: decision trees, logistic regression, kNN and SVM. Some things to remember:

- **Logistic regression**:
    - not prone to over-fitting.
    - can work well with noisy data.
    - assumes (requires) there is a single smooth boundary between categories. This implies categories are linearly or nonlinearly separable without complex shapes. However, models like decision trees can handle jagged boundaries.
- **SVM**:
    - a geometric technique,
    - builds a hyperplane that separates the data into groups.
    - nonlinear SVM projects the data into a higher-dimensional space to build the plane, then returns the plane to regular space.

There are guidelines you can use, but ultimately experience and experimentation is most important.