

7.3: Machine learning

Methods explained in this section are not only visualizations. Together, they frequently called “classification with learning”, “supervised classification”, “machine learning”, or just “classification”. All of them are based on the idea of *learning*:

... He scrambled through and rose to his feet. ... He saw nothing but colours—colours that refused to form themselves into things. Moreover, he knew nothing yet well enough to see it: you cannot see things till you know roughly what they are^[1]. His first impression was of a bright, pale world—a watercolour world out

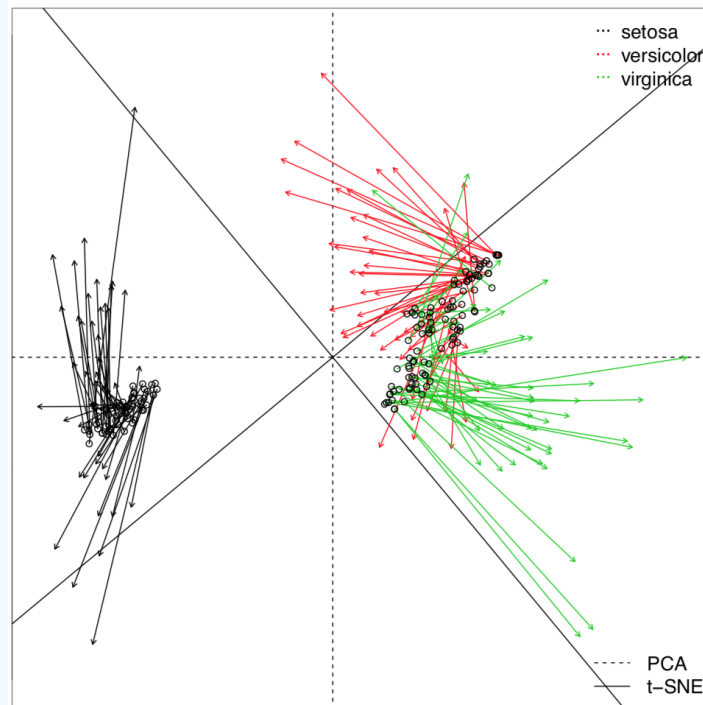


Figure 7.3.1 Procrustes plot which show t-SNE ordination against the target PCA ordination.

of a child's paint-box; a moment later he recognized the flat belt of light blue as a sheet of water, or of something like water, which came nearly to his feet. They were on the shore of a lake or river...

C.S.Lewis. Out of the Silent Planet.

First, small part of data where identity is already known (*training dataset*) used to develop (fit) the model of classification (Figure 7.3.2). On the next step, this model is used to classify objects with unknown identity (*testing dataset*). In most of these methods, it is possible to estimate the quality of the classification and also assess the significance of the every character.

Let us create training and testing datasets from [iris](#) data:

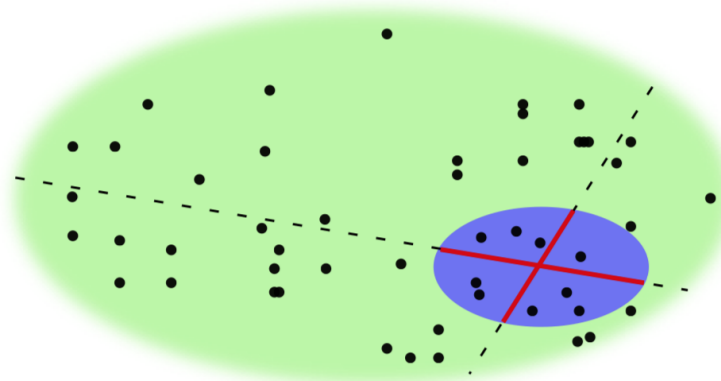


Figure 7.3.2 Graphic representation of the statistical machine learning. Blue is a training dataset, red lines is classification model, green is a testing dataset, dashed lines show prediction (estimation) process.

(`iris.unknown` is of course the fake unknown so to use it properly, we must specify `iris.unknown[, -5]`. On the other hand, species information will help to create misclassification table (confusion matrix, see below).)

Learning with regression

Linear discriminant analysis

One of the simplest methods of classification is the linear discriminant analysis (LDA). The basic idea is to create the set of linear functions which “decide” how to classify the particular object.

Training resulted in the hypothesis which allowed almost all plants (with an exception of seven *Iris virginica*) to be placed into the proper group. Please note that LDA does not require scaling of variables.

It is possible to check LDA results with inferential methods. Multidimensional analysis of variation (MANOVA) allows to understand the relation between data and model (classification from LDA):

Important here are both p-value based on Fisher statistics, and also the value of Wilks’ statistics which is the *likelihood ratio* (in our case, the probability that groups are *not different*).

It is possible to check the relative importance of every character in LDA with ANOVA-like techniques:

(This idea is applicable to other classification methods too.)

... and also visualize LDA results (Figure 7.3.3):

(Please note 95% confidence ellipses with centers.)

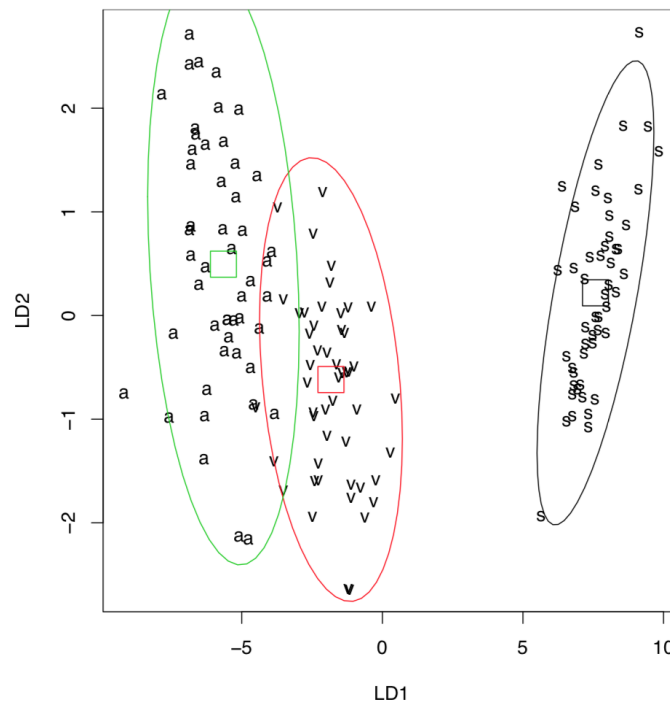


Figure 7.3.3 Graphical representation of the linear discriminant analysis results. 95% confidence ellipses and their centers added with `Ellipses()` function.

To place all points on the plot, we simply used all data as training. Note the good discrimination (higher than in PCA, MDS or clustering), even between close *Iris versicolor* and *I. virginica*. This is because LDA frequently overestimates the differences between groups. This feature, and also the parametricity and linearity of LDA made it less used over the last years.

With LDA, it is easy to illustrate one more important concept of machine learning: *quality of training*. Consider the following example:

Misclassification error here almost two times bigger! Why?

Well, using `sample()` (and particular `set.seed()` value) resulted in biased training sample, this is why our second model was trained so poorly. Our first way to sample (every 5th iris) was better, and if there is a need to use `sample()`, consider to sample each species *separately*.

Now, return to the default random number generator settings:

Please note that it is widely known that while LDA was developed on biological material, this kind of data rarely meets two key assumptions of this method: (1) multivariate normality and (2) multivariate homoscedasticity. Amazingly, even Fisher’s *Iris* data

with which LDA was invented, does not meet these assumptions! Therefore, we do not recommend to use LDA and keep it here mostly for teaching purposes.

Recursive partitioning

To replace linear discriminant analysis, multiple methods with similar background ideas were invented. Recursive partitioning, or *decision trees* (regression trees, classification trees), allow, among other, to make and visualize the sort of discrimination key where every step results in splitting objects in two groups (Figure 7.3.4):

We loaded first the `tree` package containing `tree()` function (`rpart` is another package which makes classification trees). Then we again used the whole dataset as training data. The plot shows that all plants with petal length less than 2.45 cm belong to *Iris setosa*, and from the rest those plants which have petal width less than 1.75 cm and petal length more than 4.95 cm, are *I. versicolor*; all other irises belong to *I. virginica*.

In fact, these trees are result of something similar to “hierarchical discriminant analysis”, and it is possible to use them for the supervised classification:

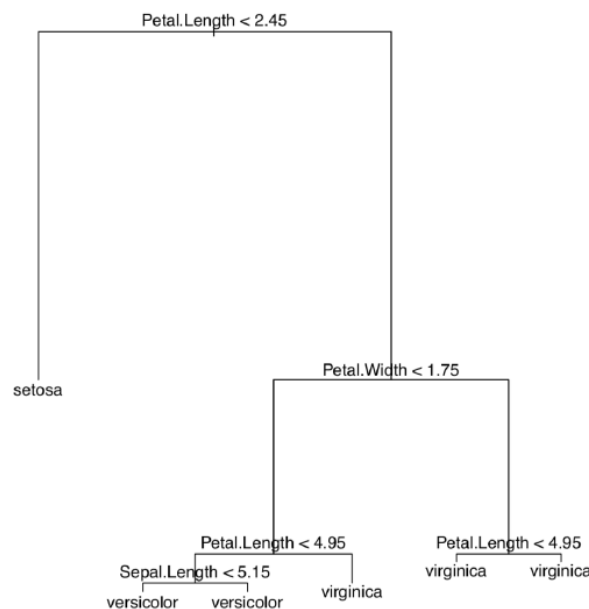


Figure 7.3.4 Classification tree for the *iris* data from `tree` package.

Try to find out which characters distinguish species of horsetails described in `eq.txt` data file. File `eq_c.txt` contains the description of characters.

Package `party` offers sophisticated recursive partitioning methods together with advanced tree plots (Figure 7.3.5): (For species names, we used one-letter abbreviations.)

Ensemble learning

Random Forest

The other method, internally similar to regression trees, rapidly gains popularity. This is the *Random Forest*. Its name came from the ability to use numerous decision trees and build the complex classification model. Random Forest belongs to *bagging ensemble methods*; it uses bootstrap (see in Appendix) to multiply the number of trees in the model (hence “forest”). Below is an example of Random Forest classifier made from the *iris* data:

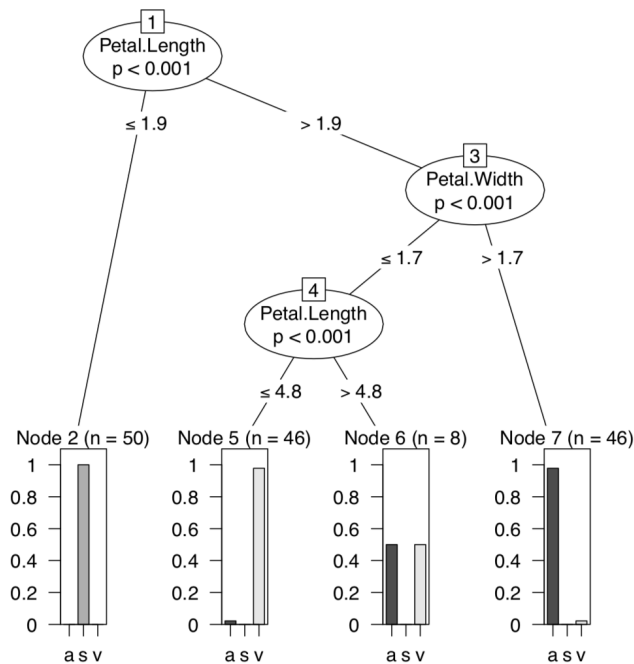


Figure 7.3.5 Classification tree for the *iris* data from *party* package.

Here results are similar to LDA but Random Forest allows for more. For example, it can clarify the importance of each character (with function `importance()`), and reveal classification distances (proximities) between all objects of training subset (these distances could be in turn used for clustering). Random Forest could also visualize the multidimensional dataset (Figure 7.3.6): (We applied several tricks to show convex hulls and their centroids.)

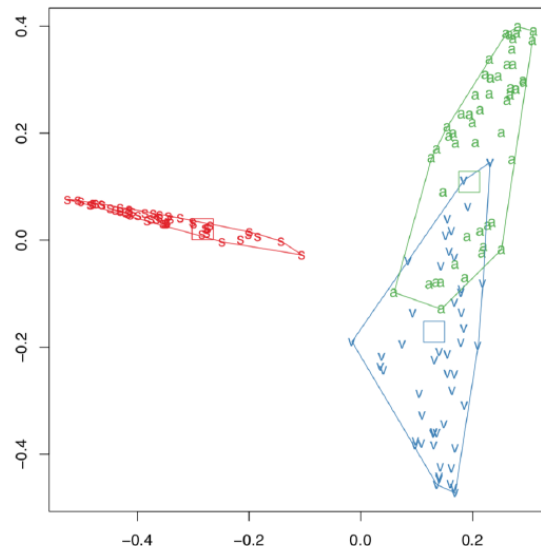


Figure 7.3.6 Visualization of *iris* data with the help of “Random Forest”. Hulls and their centroids added with `Hulls()` function.

Package *ranger* implements even faster variant of Random Forest algorithm, it also can employ parallel calculations.

Gradient boosting

There are many weak classification methods which typically make high misclassification errors. However, many of them are also ultra-fast. So, is it possible to combine many weak learners to make the strong one? Yes! This is what boosting methods do. Gradient boosting employs multi-step optimization and is now among most frequently using learning techniques. In R, there are several gradient boosting packages, for example, *xgboost* and *gbm*:

(Plot is purely technical; in the above form, it will show the marginal effect (effect on membership) of the 1st variable. Please make it yourself. “Membership trick” selects the “best species” from three alternatives as `gbm()` reports classification result in fuzzy form.)

Learning with proximity

k-Nearest Neighbors algorithm (or kNN) is the “lazy classifier” because it does not work until unknown data is supplied:

kNN is based on *distance calculation* and “voting”. It calculates distances from every unknown object to the every object of the training set. Next, it considers several (5 in the case above) nearest neighbors with known identity and finds which id is prevalent. This prevalent id assigned to the unknown member. Function `knn()` uses Euclidean distances but in principle, any distance would work for kNN.

To illustrate idea of nearest neighbors, we use *Voronoi decomposition*, the technique which is close to both kNN and distance calculation:

The plot (Figure 7.3.7) contains multiple cells which represent neighborhoods of training sample (big dots). This is not exactly what kNN does, but idea is just the same. In fact, Voronoi plot is a good tool to visualize any distance-based approach.

Depth classification based on how close an arbitrary point of the space is located to an implicitly defined center of a multidimensional data cloud:

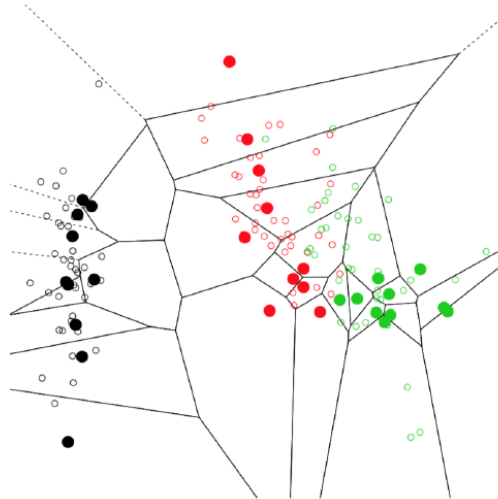


Figure 7.3.7 Visualization of training data points neighborhoods with Voronoi decomposition.

Learning with rules

Naïve Bayes classifier is one of the simplest machine learning algorithms which tries to classify objects based on the probabilities of previously seen attributes. Quite unexpectedly, it is typically a good classifier:

Note that Naïve Bayes classifier could use not only numerical like above, but also nominal predictors (which is similar to correspondence analysis.)

Apriori method is similar to regression trees but instead of classifying objects, it researches *association rules* between classes of objects. This method could be used not only to find these rules but also to make classification. Note that measurement iris data is less suitable for association rules than nominal data, and it needs *discretization* first:

(Rules are self-explanatory. What do you think, does this method performs better for the nominal data? Please find it out.)

Learning from the black box

Famous SVM, *Support Vector Machines* is a *kernel* technique which calculates parameters of the hyper-planes dividing multiple groups in the multidimensional space of characters:

Classification, or prediction *grid* often helps to illustrate the SVM method. Data points are arranged with PCA to reduce dimensionality, and then classifier predicts the identity for the every point in the artificially made grid (Figure 7.3.8). This is possible to perform manually but `Gradd()` function simplifies plotting:

And finally, *neural networks*! This name is used for the statistical technique based on some features of neural cells, *neurons*. First, we need to prepare data and convert categorical variable *Species* into three logical *dummy variables*:

Now, we call *neuralnet* package and proceed to the main calculation. The package “wants” to supply all terms in the model explicitly:

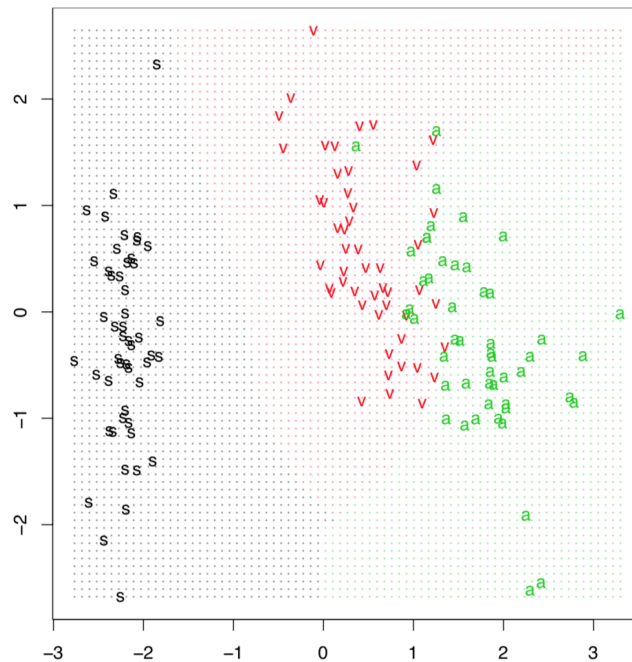


Figure 7.3.8 Classification grid which illustrates the SVM algorithm. Data points are arranged with PCA.

(Note use of `set.seed()`, this is to make your results similar to presented here.)

Now predict (with `compute()` function) and check misclassification:

Results of neural network prediction are fuzzy, similar to the results of fuzzy clustering or regression trees, this is why `which.max()` was applied for every row. As you see, this is one of the lowest misclassification errors.

It is possible to plot the actual network:

The plot (Figure 7.4.1) is a bit esoteric for the newbie, but hopefully will introduce into the method because there is an apparent multi-layered structure which is used for neural networks decisions.

References

1. Emphasis mine.

This page titled 7.3: Machine learning is shared under a [Public Domain](#) license and was authored, remixed, and/or curated by Alexey Shipunov via source content that was edited to the style and standards of the LibreTexts platform.