Week 3 : Classification Algorithms

Ramzi Saouma

June 21, 2020

In classification, we aim to predict a class label from a list of predefined classes. This week, we add to our toolbox of machine learning, a collection of algorithms mostly used to solve classification tasks. We will start with a very simple KNN methodology and end with the more complex decision trees algorithms. Most of the methods we learn this week can also be applied to regression models. However given the topic of this week we will remains within their applications to classifications. In later weeks, when we move to applying our knowledge to real data with the help of python, we will get the chance to see how they work on regression problems.

Outline: Learning Objectives

1. K Nearest Neighbors (KNN)
2. Naive Bayes Classifier
3. Support Vector Machine (SVM)
4. Kernelized Vector Machine
5. Decision Trees

Classification

In mathematical terms a classification is defined as below:

- Given a feature vector X and a qualitative response Y taking values in the class C, the classification task is to build a function $C(X)$ that takes as input the feature vector X and predicts its value for Y; i.e $C(x) \in C$.[1]
- In some cases we will be interested not only in classifying the observation, but to calculate the probabilities that X belong to each of the classes in C.

Classification could be divided into binary classification, which is a special case of having to predict between two distinct classes, and the mutli-class classification, which as the name suggests, is a classification between more than two classes. The binary classification is often used in answering a yes or no question. *Is this email spam?*[2].

[1] Trevor Hastie Gareth James, Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 8th edition, 2017. ISBN 978-1-4614-7138-7

[2] In binary classification we often speak of one class being the positive class and the other class being the negative class. Nonetheless, one must be careful, as positive does not represent having benefit or value, but rather what the objective of the study is. So, when looking for spam, "positive" could mean the spam class. Which of the two classes is called positive is often a subjective matter, and specific to the

K-Nearest Neighbors (KNN)

The KNN is considered one of the simplest machine learning algorithm[3]. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset — its "nearest neighbors." k-Neighbors classification In its simplest version, the k-NN algorithm only considers exactly one nearest neighbor, which is the closest training data point to the point we want to make a prediction for. The prediction is then simply the known output for this training point[4]. Instead of using the single closest neighbor to the observation we are trying to predict we can opt for the point that is the closest to k number of neighbors. (see figure 3)

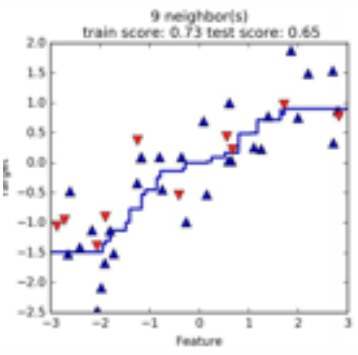

Figure 1:

[3] The KNN algorithm could be used for regression models as we will see later int he course when we apply our knowledge on real data using python. See figure 1

[4] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

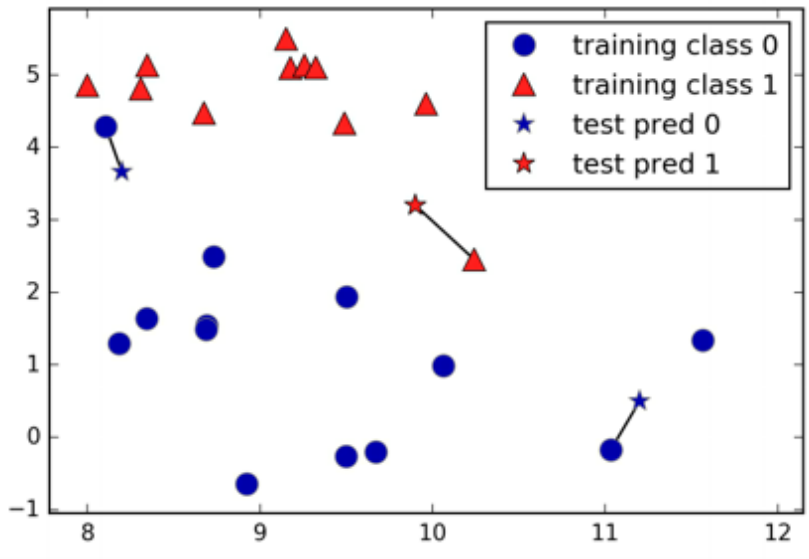

Figure 2: Prediction made with KNN where k=1. Source- Muller2017

While it is much is easier to illustrate the KNN method in a binary classification task, the algorithm could be applied to a multi-class task as well.

In 4 we went one step further to draw the decision boundary, which is the divide between where the algorithm assign class 0 versus class 1. A smoother boundary corresponds to a simpler model. Hence the higher the k parameter the less complex is model.

Naive Bayesian Classification (NBC)[5]

In week one we had a quick preview of NBC when we looked at the Email Spam case. This week we will go a bit deeper into a method that is considered to be fast and efficient without compromising

[5] Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

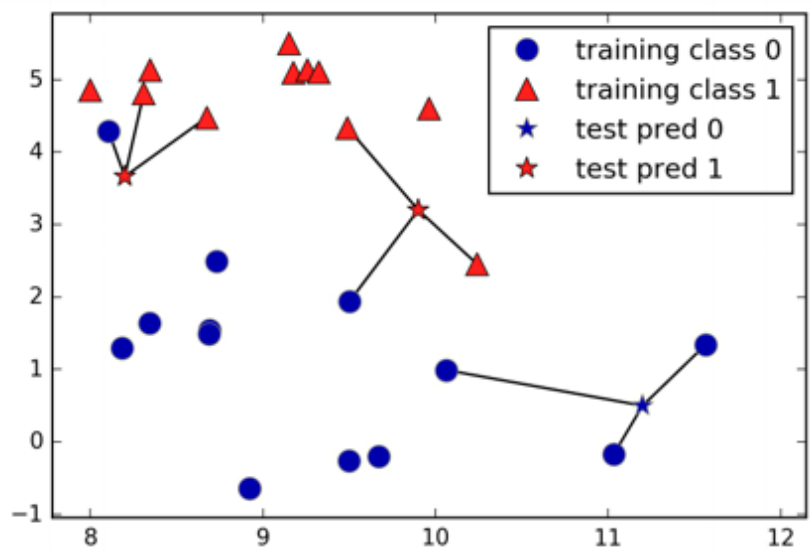

Figure 3: Prediction made with KNN where k=3. Source- Muller2017

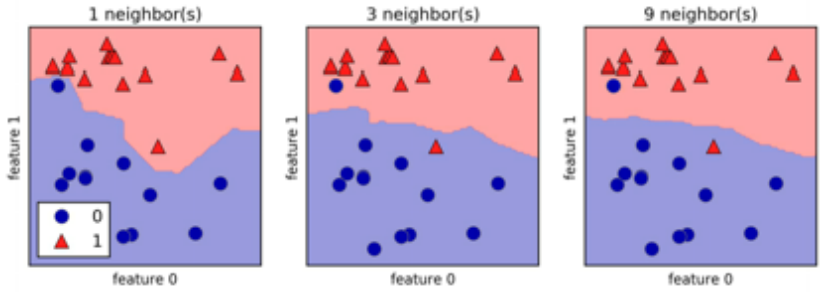

Figure 4: We draw the decision boundaries using KNN with different values of K. Source- Muller2017

on complexity. In NBC we are interested in finding the probability of a label given some observed features, which we can write as $P(L/Features)$. Bayes's Theorem[6] allows to compute the below:

[6] Refer to week 1 for more details

$$P(L|\text{features}) = \frac{P(\text{features}|L)(P(L)}{P(\text{features})} \tag{1}$$

When we want to predict the right label between two classes L_1 and L_2, we just calculate the ratio of the posterior probabilities for each label:

$$\frac{P(L_1|\text{features})}{P(L_2|\text{features})} = \frac{P(\text{features}|L_1)(P(L_1)}{P(\text{features}|L_2)P(L_2)} \tag{2}$$

"All we need now is some model by which we can compute $P(\text{features}|L_i)$ for each label. Such a model is called a generative model because it specifies the hypothetical random process that generates the data. Specifying this generative model for each label is the main piece of the training of such a Bayesian classifier. The general version of such a training step is a very difficult task, but we can make it simpler through the use of some simplifying assumptions about the form of this model. This is where the "naive" in "naive Bayes" comes in: if we make very naive assumptions about the generative model for each label(i.e the underlying probability distribution), we can find a rough approximation of the generative model for each class, and then proceed with the Bayesian classification. Different types of naive Bayes classifiers rest on different naive assumptions about the data"[7].

[7] Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

In this course we will cover 3 types of NBC, each based on different probability distribution assumptions.

1. Bernoulli classifier counts how often each feature of every class is not zero. Lets assume the below simple example:

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

In the above example we observe four observations, with four binary features each. There are two classes, 0 and 1. For class 0 (the first and third data points), the first feature is zero two times and nonzero zero times, the second feature is zero one time and

nonzero one time, and so on. These same counts are then calculated for the data points in the second class. Counting the nonzero entries per class in essence looks like this:

$$\begin{aligned} \text{Class } 0 &\longrightarrow [0,1,0,2] \\ \text{Class } 1 &\longrightarrow [2,0,2,1] \end{aligned}$$

The Bernoulli model is useful if your feature vectors are binary (i.e. zeros and ones). This is typically applied to classification problems where for example the 1s 0s are "word occurs in the document" and "word does not occur in the document" respectively.

2. Gaussian Naive Bayes assumes that each data point is drawn from a simple Gaussian distribution. The assumption is there are no correlation between different features. The model is then fit by simply finding the mean and the standard deviation of the points within each label, which is all you need to define such a distribution.

3. Multinomial Bayes assumes a multinomial distribution for the generative function by taking into account only the mean of each value. This method usually works best with text classification. It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i *isobservedoverthentrials*].

When to Use Naive Bayes[8]

[8] Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5

Given how NBCs makes strong assumptions about the independence of the data, they might often fall behind some of the more complicated model. However they remain solid choices for the below reasons:

- They are extremely fast for both training and prediction
- They provide straightforward probabilistic prediction
- They are often very easily interpreted
- They have very few (if any) adjustable parameters

That makes NBC a good methodology to start your project with as a baseline. If it performs suitably, then congratulations: you have a very fast and easy to interpret classifier for your problem. If it does

not perform well, then you can begin exploring more sophisticated models, with some baseline knowledge of how well they should perform. Naive Bayes classifiers tend to perform especially well in one of the following situations:

- When the naive assumptions actually match the data (very rare in practice)
- For very well-separated categories, when model complexity is less important
- With high dimensional features and where complexity is less important

Support Vector Machines (SVM)

Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression. In this section, we will develop the intuition behind support vector machines and their use in classification problems.

In the previous section when we discussed NBC models, we saw how a simple model that is based on selected distributions can use generative models to probabilistically determine labels for new points. That was an example of generative classification; here we will consider instead discriminatory classification: instead of modeling each class, we attempt to draw a line or curve (in two dimensions) or manifold (in multiple dimensions) that separates each class from the others.

Let's consider the data we see in figure 5 where the two classes are visually distinct:

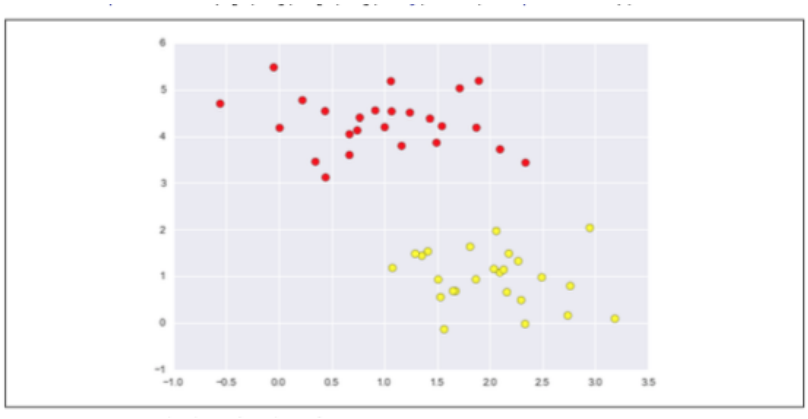

Figure 5: Simple Data for classification. Source- VanDerPlas 2017

A linear discriminative classifier would attempt to draw a straight line separating the two sets of data, and thereby create a model for

classification. For two-dimensional data like that shown here, this is a task we could do by hand. But immediately we see a problem: there is more than one possible dividing line that can perfectly discriminate between the two classes!

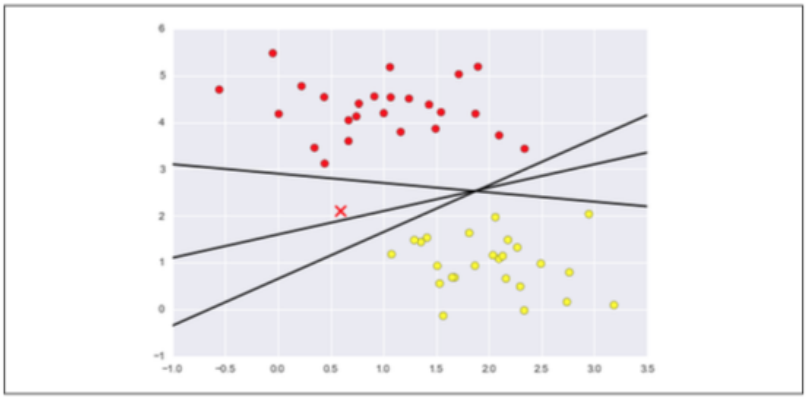

Figure 6: We fit three perfectly fitting lines. Source- VanDerPlas 2017

These are three very different separators that, nevertheless, perfectly discriminate between these samples. Depending on which you choose, a new data point (e.g., the one marked by the "X" in figure 6) will be assigned a different label! Evidently our simple intuition of "drawing a line between classes" is not enough, and we need to think a bit deeper.

Support vector machines offer one way to improve on this. The intuition is this: rather than simply drawing a zero-width line between the classes, we can draw around each line a margin of some width, up to the nearest point. Here is an example of how this might look (see figure 7):

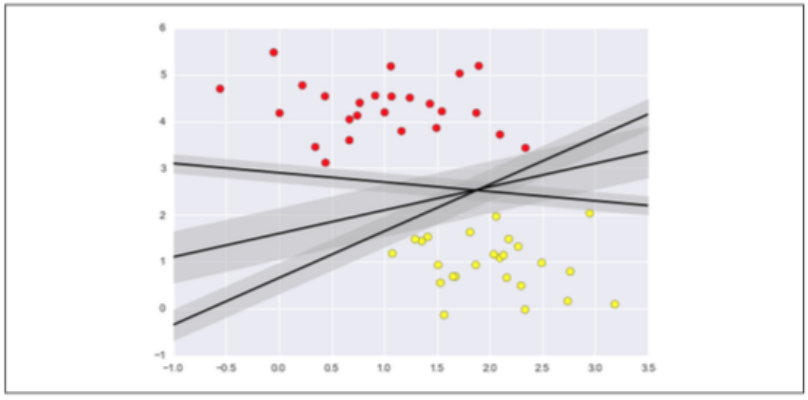

Figure 7: Lines with margins. Source- VanDerPlas 2017

In support vector machines, the line that maximizes this margin

is the one we will choose as the optimal model. Support vector machines are an example of such a maximum margin estimator.

SVM parameters: C for softening the margins

In the previous examples, the two classes were fairly distinct making it easy for the algorithm to find the perfect boundary between the two classes. In real life this is going to be rarely the case. We are more likely to have intertwined data with some overlaps. As you can see in figure 8.

To deal with that situation we need to allow for some points to fall within the margins as long as this allows for a better fit. The softness of the margin is handled with the parameter C. The higher C is, the more stringent are the margins. Looking at figure 9, on the left hand side we have a fitted line with C set to 10 while on the right hand side we fitted a line with C set to 0.1.

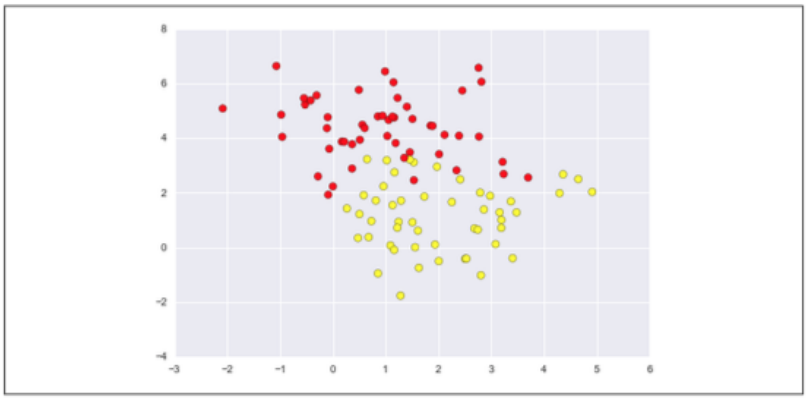

Figure 8: Complex data. Source- VanDerPlas 2017

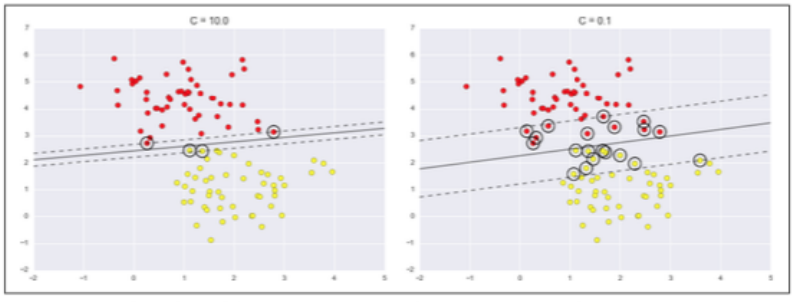

Figure 9: The impact of the parameter C on the boundary line. Source- VanDerPlas 2017

Kernelized Vector Machine

When kernels are added to SVM we get quite powerful non-linear machine learning algorithm. This is illustrated by the following example.

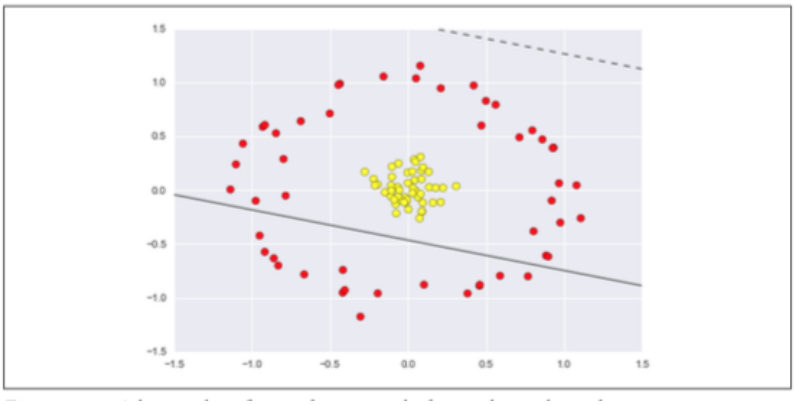

Figure 10: Non-linear boundaries are needed. . Source- VanDerPlas 2017

Looking at figure 10, it is clear that a straight line decision boundary will perform poorly. In this case we can consider projecting the data into a higher dimensional space defined by some polynomial and Gaussian function, making it possible to fit a plane (linear) into a non-linear relationship. Kernelized Vector Machine does exactly by using the trick[9] that is not computationally heavy to transform the data from figure 10 into a higher dimension as in figure**??**.

Once the data is transformed SVM will fit a plane which in the two dimensional space will look as in figure 11

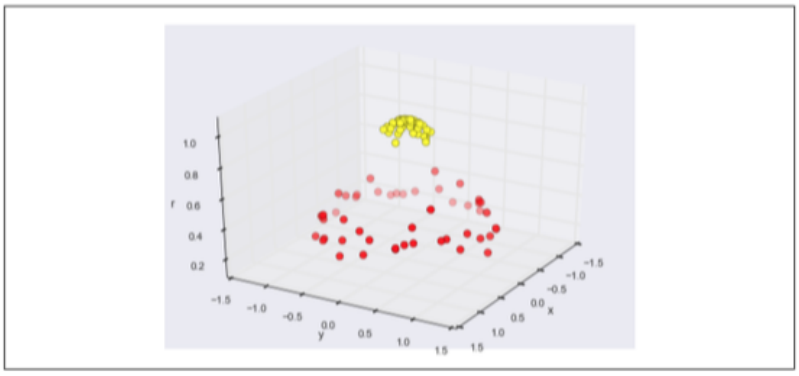

Figure 11: The Kernel trick adds a dimension to the data making possible to find a plane to seperate and classify the day. Source- VanDerPlas 2017

[9] The kernel trick: Adding dimensions to our data to deal with non-linearity is quite powerful. However we don't know which features to add and adding many features might make computation very expensive. This is where the kernel trick comes in. The kernel function works by directly computing the distance (more precisely, the scalar products) of the data points for the expanded feature representation, without ever actually computing the expansion.

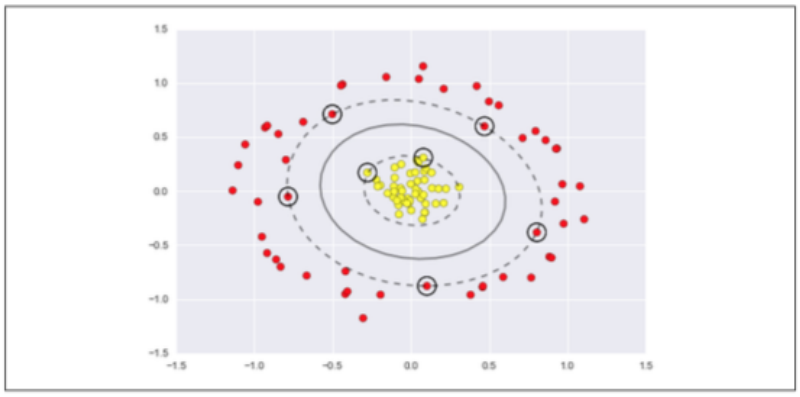

Figure 12: Thanks to the kernel function we can fit a non-linear decision boundary into our data. Source- VanDerPlas 2017

Decision Trees

Decision trees are very intuitive ways to label a data point into class. It is like playing the 20 questions game where the player needs to guess the name of the animal with the least questions possibles. (see figure 13) The Binary splitting makes this method highly efficient: In an optimally designed tree each question should cut the number of options by half. To optimize it one need to know when question need to be asked when. Let's consider the data in figure 14. A decision tree built on this data will split at each level, the gradient in two along some criterion. Each time the gradient is split we add one level of depth. In figure 15 we see how the classification evolve as we go deeper.

Learning Summary

1. We learned about four families of algorithms for classification tasks: KNN, Bayesian, Vector Machines and Decision Trees.
2. We learned about the strength and weakness of each method.
3. We saw how the tuning parameter in each method plays a role.
4. We saw how trade-off works between complexity and ease of interpretation.

References

Trevor Hastie Gareth James, Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 8th edition, 2017. ISBN 978-1-4614-7138-7.

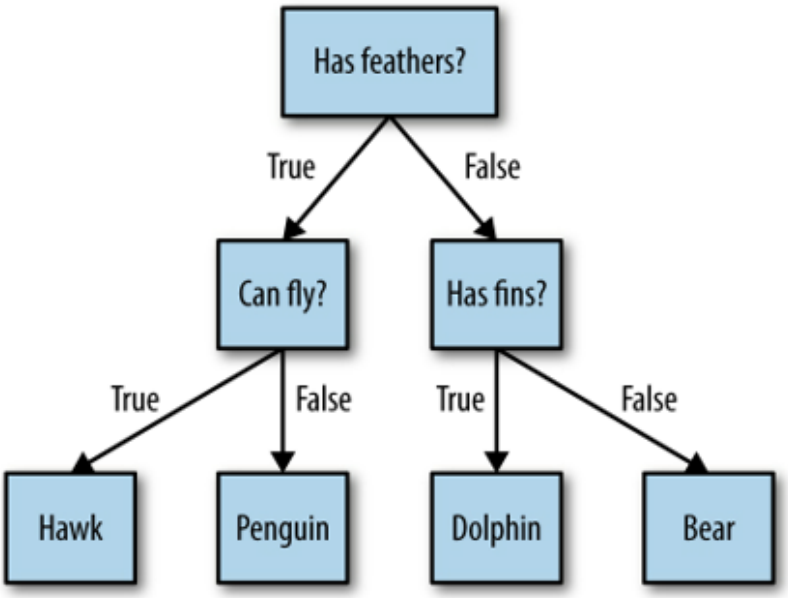

Figure 13: The Animal guessing game.
Source- Muller 2017

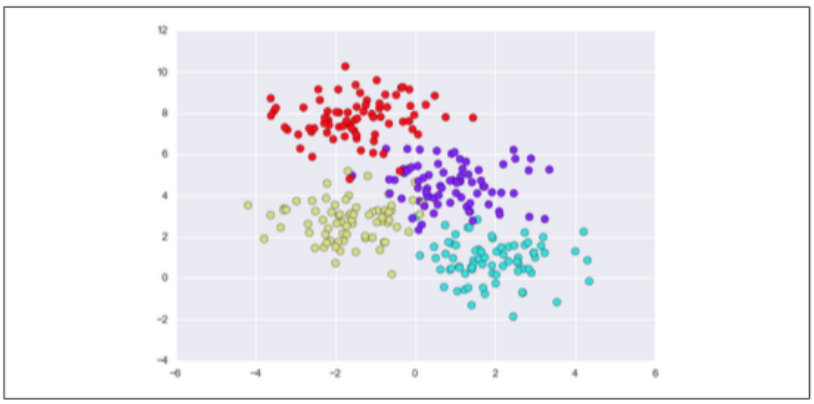

Figure 14: The Animal guessing game.
Source- Muller 2017

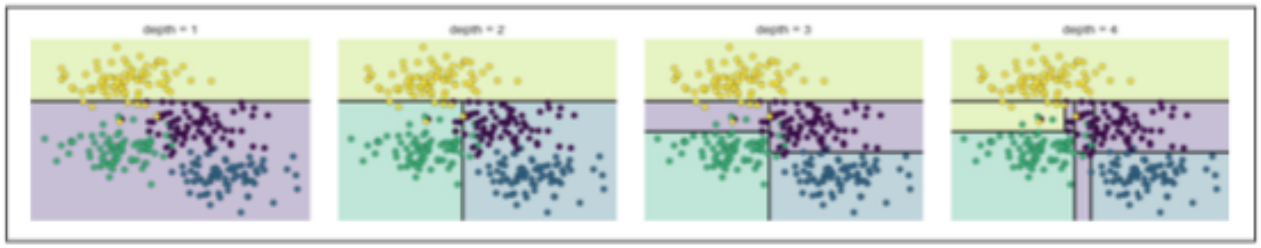

Figure 15: Visualizing the classification as we go deeper.

Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5.

Jake VandePlas. *Pytho Data Science Handbook*. O'Reilly, second edition, 2017. ISBN 978-1-449-36941-5.