1 Learning Outcomes

- Describe how a model's response for a linear model differ from a non-linear model
- Compare and contrast model's response for different models
- Describe what is meant by a decision boundary
- Based on the geometric model's response, indicate whether or not a model is overfit

2 Limitations of linear models

Linear models are easy to understand and implement models and have the advantages of interpretability. However, linear models can have significant limitations. In regression, the true relationship between the response and features might be non-linear, so that modeling it with a hyperplane might not be the best approximation. In classification, classes might not be linearly separable and we might need non-linear decision boundaries that separate the classes in the feature space.

Linear models model a straight-line relationship, which might not be flexible enough to model more complex relationship. This is why non-linear models are said to be more complex or flexible than linear models, and can therefore have better predictive power.

To find a non-linear mapping between feature vector and the corresponding label, we can use non-linear models such as K-nearest neighbors and decision tree or extend linear models by adding polynomial features. We're going to look at graphical examples of each of these approaches, compare between them and see how a model's response changes with its complexity.

3 Extending linear models: polynomial features

Adding polynomial features is a very simple way that directly extends linear models to accommodate non-linear relationships. A new set of features can be created by raising each feature x_i to the power of $2, 3, \cdots, d$, where d is the maximal exponent[1]:

$$x_i^2, x_i^3, \cdots, x_i^d$$

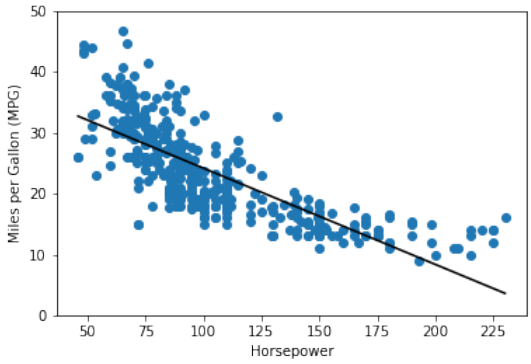
In this way, we're not only combining a weighted sum of each feature in a linear model but also of their polynomial versions:

$$\sum_{i=1}^{m}(w_i x_i) + w_0 \xrightarrow{\text{extending linear models}} \sum_{i=1}^{m}\left(w_i x_i + w_{i2}x_i^2 + w_{i3}x_i^3 + \cdots w_{id}x_i^d\right) + w_0$$

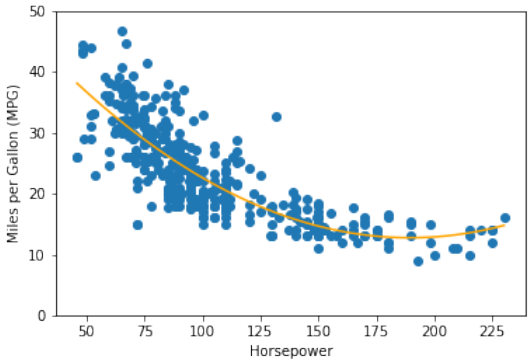
Let's look at some examples.

Regression

[1] "Generally speaking, it is unusual to use d greater than 3 or 4 because for large values of d, the polynomial curve can become overly flexible and can take on some very strange shapes"- Statistical Learning, page 290



(a) Modeling the relationship of mpg vs horsepower with a straight line might not be the best approach.



(b) The linear model can be extended to model the relationship of mpg vs horsepower with a quadratic curve.

Figure 1: Linear fit vs non-linear fit

Figure 1a shows the mpg (gas mileage in miles per gallon) versus horsepower for a number of cars (data source). Let's say we're interested in predicting the mpg from the horsepower of a car. The black line is the line that results from training the data with a linear regression model. We can see that the relationship between mpg and horsepower is in fact a curved relationship, so that fitting the data with a straight line might not be the best approach. Can we do better? A simple approach for creating a non-linear mapping from horsepower to mpg is to include a transformed version of the feature horsepower. For example, we can raise the feature horsepower to the power of 2 and treat it as an additional feature. In this way instead of fitting the data with the line:

$$\text{mpg} = w_1 \times \text{horsepower} + w_0$$

we're fitting the data with the following quadratic function:

$$\text{mpg} = w_1 \times \text{horsepower} + w_2 \times \text{horsepower}^2 + w_0$$

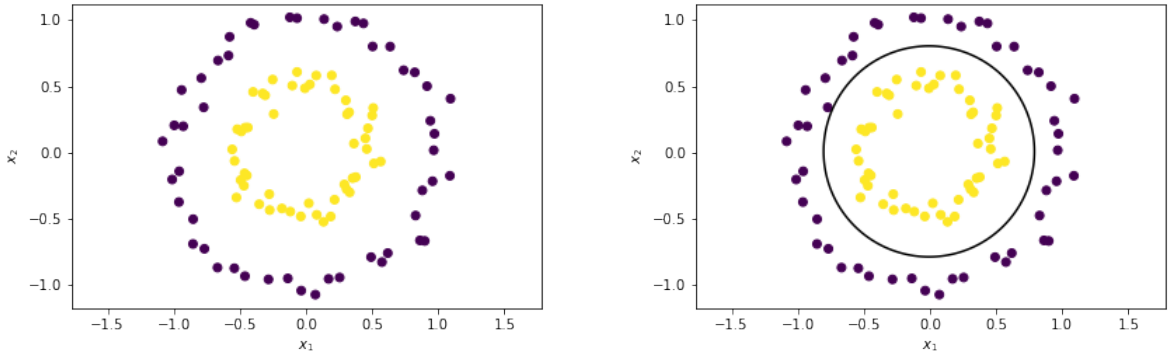
In figure 1b, the orange curve shows the resulting quadratic fit to the data, which suggests that incorporating the quadratic term results in a better data fit. Note that by treating horsepower^2 as a second feature ($x_2 = \text{horsepower}^2$), the quadratic model can be seen as a multiple linear regression model with two features $x_1 = \text{horsepower}$ and $x_2 = \text{horsepower}^2$. So we can use standard linear regression techniques in order to find the parameters w_1, w_2 and w_0.

Classification

Figure 2a shows an example of 2-dimensional simulated data where a feature vector could either belong to class 0 or class 1. We can see that two classes can't be separated with a line, however a circular decision boundary can perfectly separate the two classes. To do so, we can incorporate in the model the squares of the two features, i.e., x_1^2 and x_2^2. And instead of finding the line with equation: $w_1 x_1^2 + w_2 x_2^2 + w_0 = 0$ that divides the 2-dimensional feature space, we can find the circle with equation:

$$w_1 x_1 + w_2 x_2 + w_3 (x_1^2 + x_2^2) + w_0 = 0$$

that can separate the two classes (the two quadratic terms are assigned the same weight, because we're looking for a circle not an ellipse equation).



(a) (b)

Figure 2: This is an example where the data cannot be separated with a line. However, the linear model can be extended to separate the two classes with a circular decision boundary.

Note that by treating the sum of the quadratic terms as an additional third features, i.e., $x_3 = x_1^2 + x_2^2$, the model can be seen as a linear classification model with 3 features. So we can use any linear classification model (logistic regression, linear SVM) to find the weights $w_1, w_2, w_3 and w_0$. Figure 2b shows the resulting circular decision boundary that resulted from training the 3-dimensional data with logistic regression. The idea here is that by adding this additional feature, we're transforming the data from low to high dimensional space (from 2D to 3D) where it is easy for a linear model to separate the data. Figure 3 shows how in the 3-dimensional feature space the data is linearly separable.

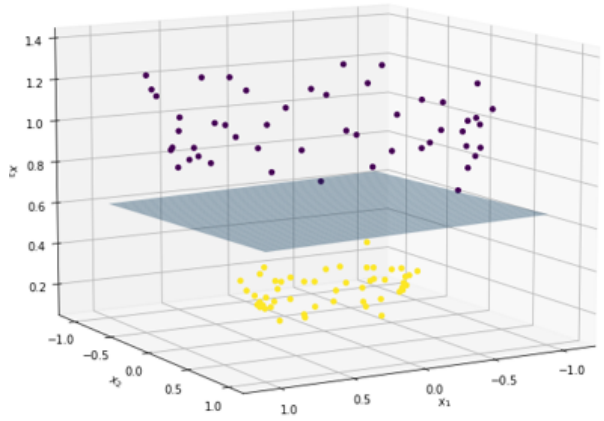


Figure 3: By going from 2D to 3D feature space, the data becomes linearly separable.

The data shown in the previous examples is a simple data made of one or two features. Going beyond that number of features makes data visualization hard, and we might have no idea what value for d to choose (how many polynomial terms to add). In this case, we can try different values

of d and choose the value that provides the best performance. This is done as part of the model selection phase and we will discuss in future notes how to correctly perform model selection. We will also learn that with SVM we can apply the kernel trick that allows us to transform the data into an infinite-dimensional space.

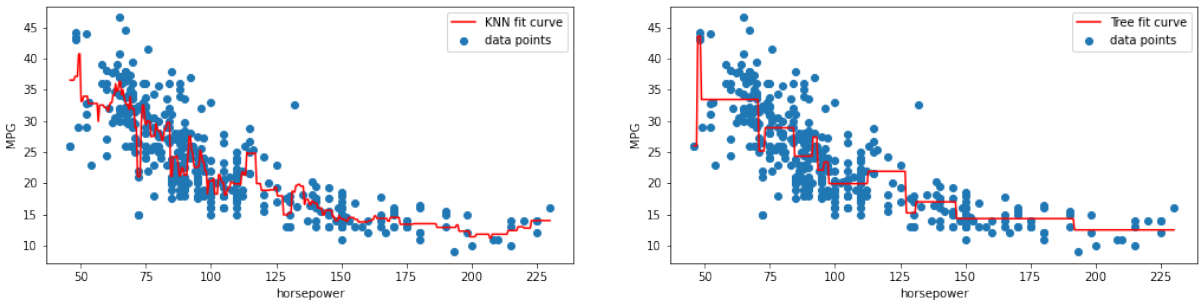
Adding polynomial features to a linear model is not the only way to make linear models accommodate for non-linear relationship. Advanced methods like spline regression allows fitting piecewise non-linear curve to the training data. The interested reader can check chapter 7 of *An introduction to statistical learning* that discusses in details these other approaches.

4 Non-linear models

Now that we've talked about linear models and how to extend them, let's look at some examples of non-linear models, namely K-nearest neighbors (KNN) and decision trees. Both can be used with regression or classification. We'll focus in theses notes at showing graphical examples of their model's response and we will discuss these models with more details in later notes.

4.1 Non-linear regression models

Let's go back to the regression problem of predicting mpg from horsepower. Figures 4a and 4b show the fitted curves that result from training the data with KNN and regression decision tree respectively. Note how these models find a fitted curve that is more flexible than a straight line. The KNN fitted curve is so flexible that it is hard to be modeled with a mathematical equation of fixed form. The fitted curve of a regression tree is made up of a series of horizontal lines of different levels (can be thought as bins).



(a) KNN (b) Decision Tree

Figure 4: Non linear model's response.

These models can become overly flexible that they can over-fit the data: an over-fit model is a model that performs very well on the training data but does not generalize well to unseen data. Let's look at an example where KNN can be an overfit. KNN model's flexibility changes with changing its k. What is k? A KNN model makes its prediction for a given feature vector based on the k nearest feature vectors of the given feature vector, where k could be $1, 2, \cdots$, or n, where

n is the number of training data samples. Figure 5 shows how the flexibility or complexity of KNN changes with changing k: in particular, we see that if we decrease k, the model's response becomes less smooth, we say that the model's complexity or flexibility increases as we decrease k. In particular, for $k = 1$, we notice how the model's response tries to replicate the exact responses observed in the training data, which resulted in a very complex response that might not generalize well to unseen data (example of overfitting).

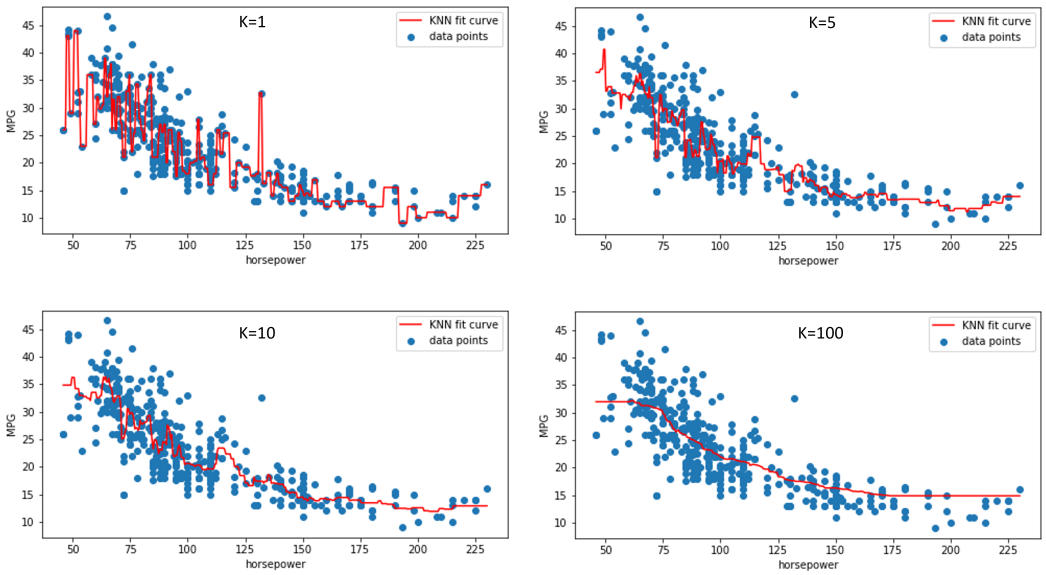


Figure 5: Complexity of KNN model changes with changing k, the number of nearest neighbors.

Figure 6 shows how the flexibility of the response curve of a regression decision tree changes if we restrict its depth (number of nodes on the longest path from the root to a leaf). From figure 6, how does the flexibility of a decision tree changes with its depth? Which figure do you think that the model is overfit?

We can also visualize the response of the regression models of KNN and decision tree for when we have two features, as shown in figure 7. Note how these models find a fitted surface that is more flexible than a plane.

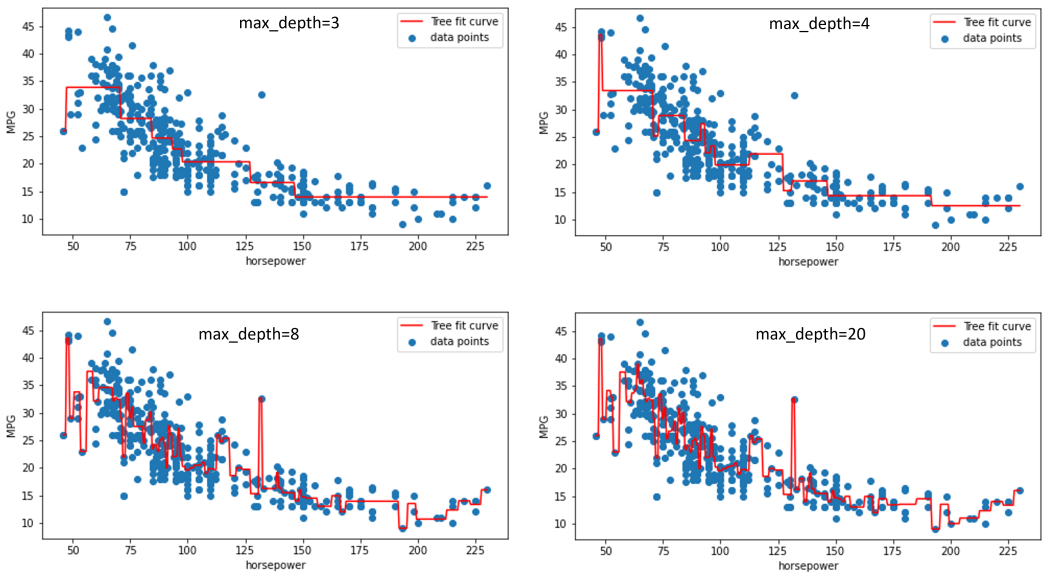


Figure 6: Complexity of the tree model changes with changing its depth.

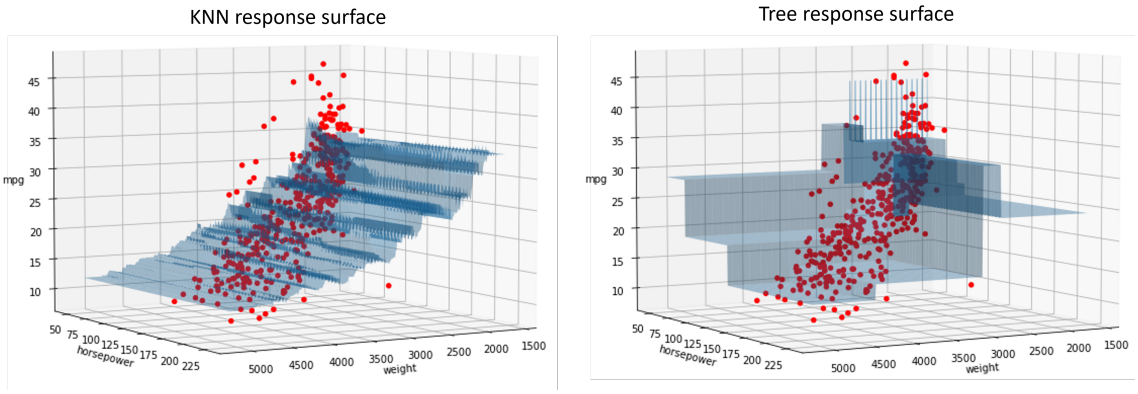


Figure 7: Non-linear surface responses

4.2 Non-linear classification models

Any classification model divides the feature space into separate regions where each region belongs to one class. When we want to predict the class of a new unlabeled data point, we check in which region it belongs and then we assign it to the corresponding class. With linear models, a hyperplane is what divides these regions. With non-linear models, a more flexible mathematical object is what separates the classes in the feature space. This separating object is what is called decision boundary, a hypersurface that defines the boundary of each classification region.

Let's look at some graphical examples that show the form of the decision boundary of each of the following classification models: KNN and decision trees. In these examples, we use a subset of the wine dataset (source). This data represents the results of a chemical analysis of some wines, and

consists of 13 measurements taken for different constituents (ash, alcalinity of ash, proanthocyanins, magnesium, ...) found in three types of wine (0, 1, 2).

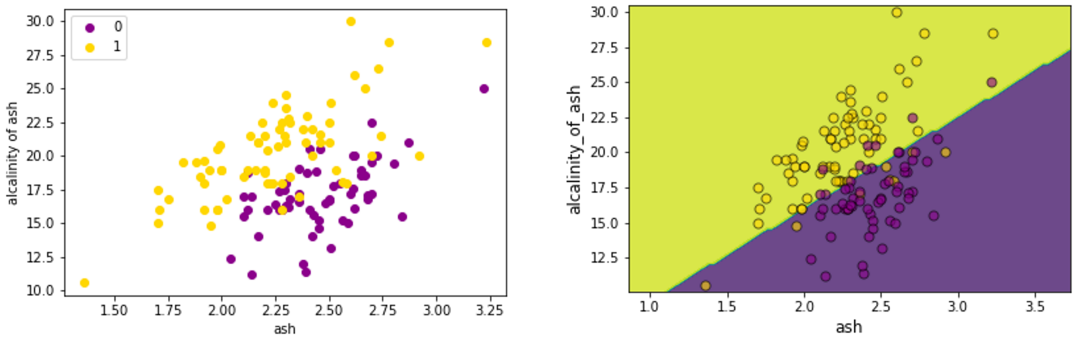
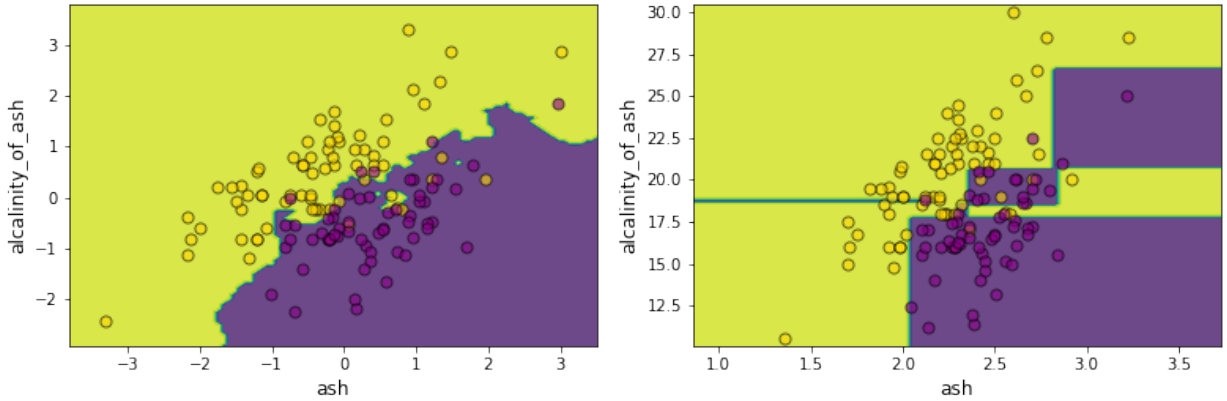


Figure 8: Decision boundary of linear model (logistic regression).

Let's assume we're interested in predicting the types 0 or 1 of wine from the two features: ash and alcalinity of ash as shown in figure 8. Figures 9a and 9b show the decision boundaries that result from training the data with KNN and classification decision tree respectively. Note that the decision boundaries of both models are more complex than the linear one. Note again that for KNN, the decision boundary cannot be represented with a mathematical equation of fixed form, and that a decision tree divides the feature space into rectangular regions.



(a) KNN (b) Decision Tree

Figure 9: Non linear decision boundaries.

Now this decision boundary can be also prone to overfitting. Figure 10 shows how the decision boundary of a KNN model changes with k (the number of nearest neighbors): the decision boundary becomes less smooth and more curved/complex as we decrease k, which means that, as we saw in the regression case, KNN model's complexity increases as we decrease k. In particular, for $k = 1$, we notice how the model's decision boundary tries to classify each point in the training dataset correctly, which resulted in a very complex decision boundary that might not generalize well to

unseen data (example of overfitting).

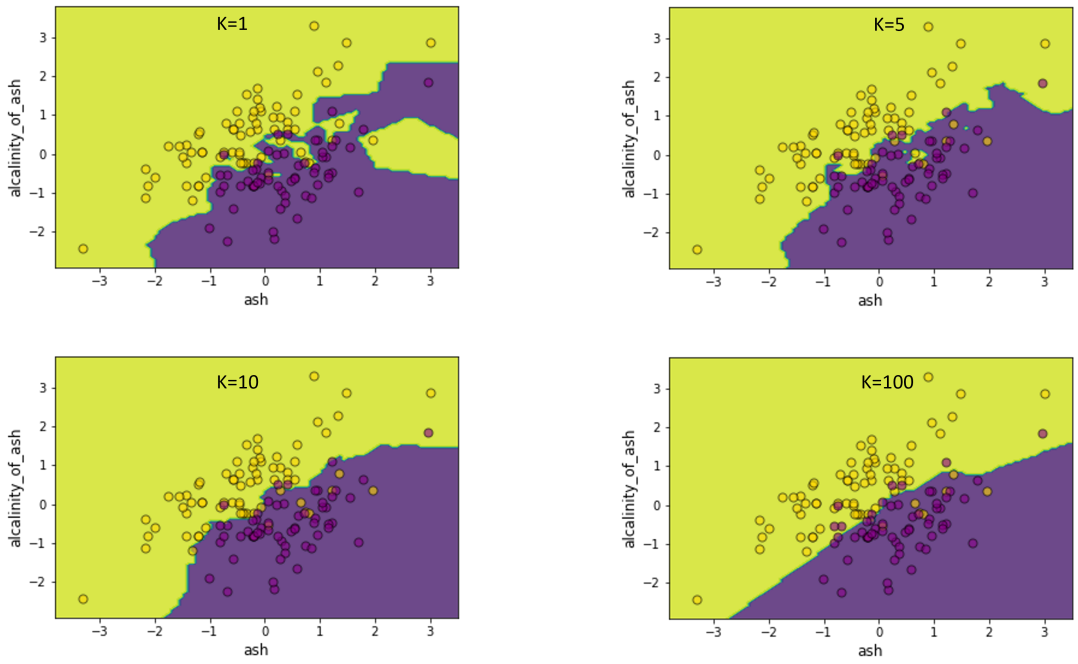


Figure 10: Complexity of KNN's decision boundary changes with changing k, the number of nearest neighbors.

Figure 11 shows how the decision boundary of a tree changes with its depth. Again, how does the flexibility of a decision tree changes with its depth?

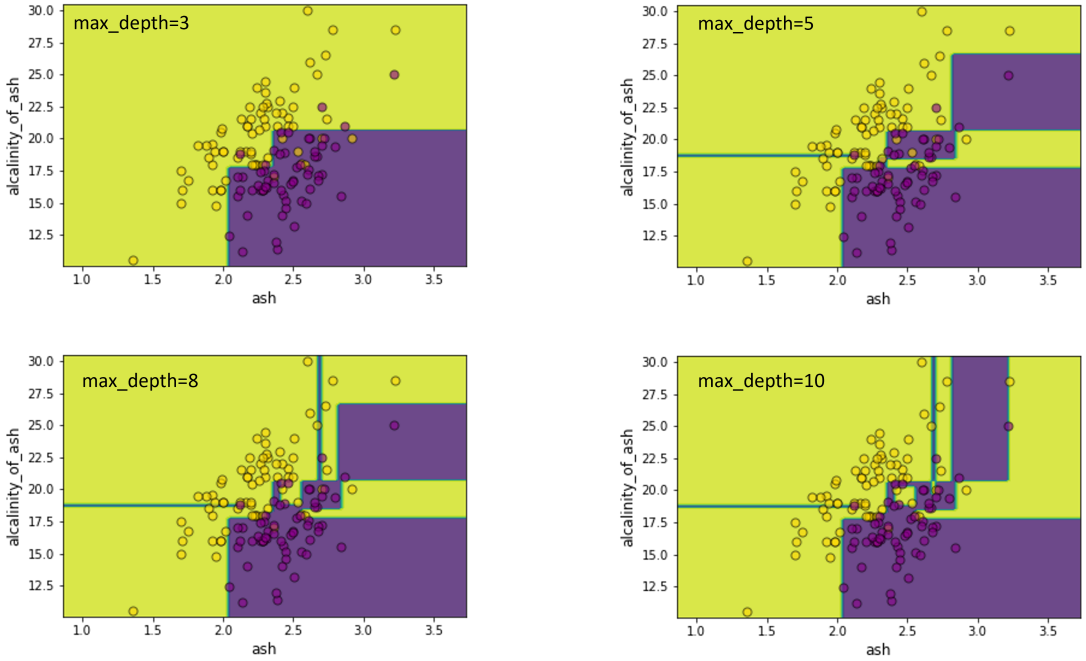


Figure 11: Complexity of a tree's decision boundary changes with changing its depth.

5 Feature scaling or processing and model's response

Increasing model's complexity by going from linear model to non-linear model might help enhance the predictive performance. However, pre-processing the data to make sure the training algorithm is fed with clean data is also important. This is because, if the data is not clean or informative enough, no matter how we change the complexity of the model, the performance would always be bad and might not change.

Clean data ensures that we have no missing entries and reasonable values for each feature. While the topic of data cleaning is not the main focus of this course, we will discuss the importance of feature scaling for each model, and why the performance of some machine learning models is sensitive to feature scaling. We will only motivate feature scaling here and we will discuss it later for each model.

Feature Scaling

Let's take again another subset of the wine dataset. As shown in figure 12, we assume we're interested in predicting the wine type 0 or 1, from the two features: magnesium and proanthocyanins. These two features have different range of values, and we're interested in checking if that would affect the decision boundary of KNN or decision tree. To do so, for each model, we used it to train on unscaled and scaled versions of the features, where feature scaling transform the features so that they belong to the same range of values. Figure 13 shows the decision boundary of KNN before and after scaling the features. We directly notice that the decision boundary changed

its form after the features were scaled. On the other hand, if we look at figure 14, we saw that the form of the decision boundary of a decision tree did not change with feature scaling.

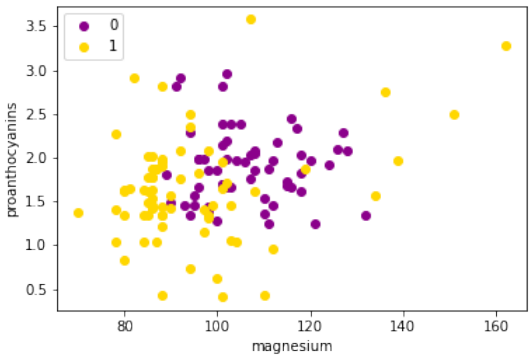
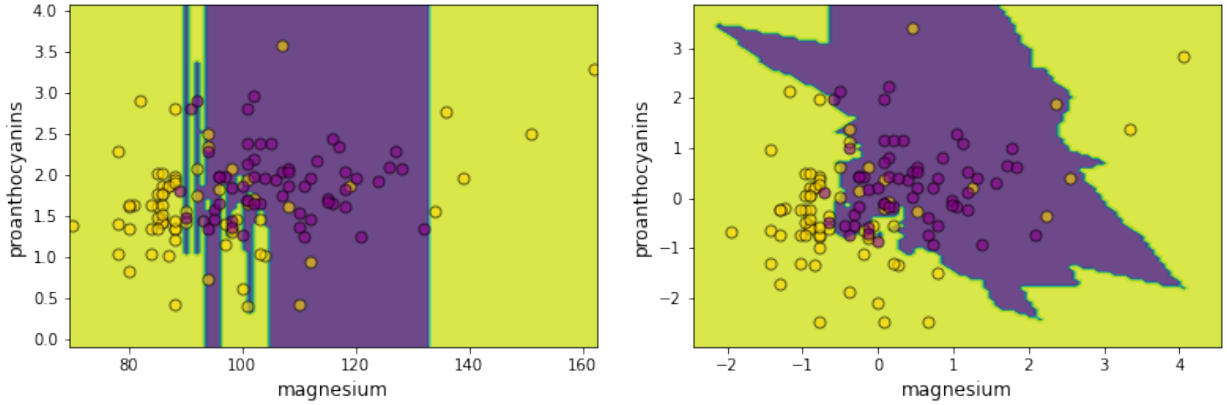


Figure 12: The two features have different range of values.

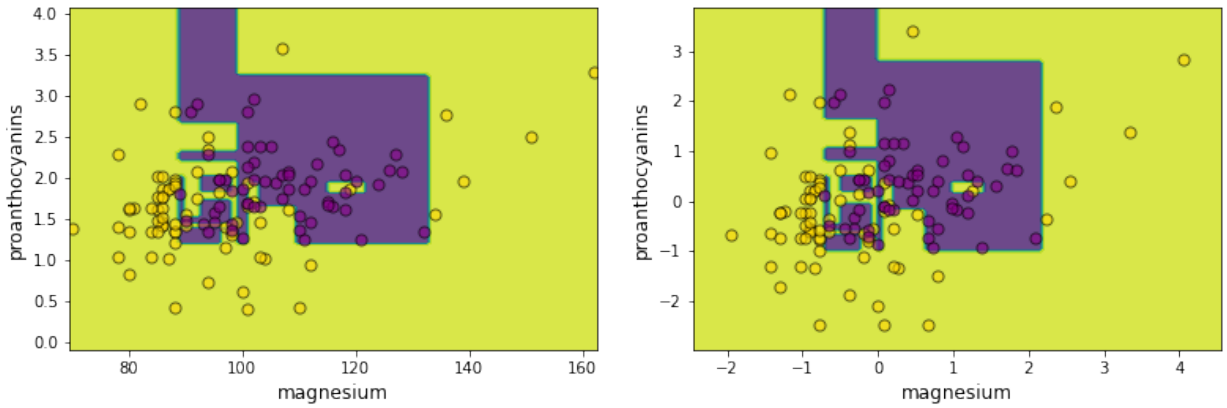


(a) KNN - No Scaling

(b) KNN - Scaling

Figure 13: Knn is sensitive to feature scaling.

We will learn later, why KNN is sensitive to feature scaling and its accuracy might be affected if the features are not scaled, and why decision trees are not sensitive to feature scaling. We will also discuss feature scaling for linear models as well.



(a) Tree - No Scaling

(b) Tree - Scaling

Figure 14: Decision tree is not sensitive to feature scaling.

6 Final remarks

Here we summarize the key differences between linear models and non-linear models.

Linear models are easy to understand and interpret. When used for prediction, they are computationally fast. Since they summarize data with a finite set of weights, storing trained linear models only requires storing the weights. On the other hand, non-linear models are more complex model that can model more flexible relationships. However, they can be more prone to over-fitting, not all non-linear models are interpretable, and some non-linear models might be computationally expensive when trained or used for prediction.

In these notes, we motivated using graphical examples the difference between linear and non-linear models, simple and complex/flexible models and the concept of overfitting. In the next notes, we will discuss more in details these concepts, how we can compare between the performances of these models using evaluation metrics and how to choose the best model using cross-validation. We will also discuss in more details the bias and variance of a model and how they are related to overfitting and underfitting.