Name: CHAN King Yeung
SID: 1155119394
STAT4001 Assignment 3

Chapter 6: Nonlinear methods

The chapter spotlighted on several extension of regression models to fit nonlinear data.

Given a true/false question, the Pearson correlation between X and Y is zero implies independence. The answer is definitely false if they are not multivariate normal variables. Linear regression, as the name suggested, is a robust algorithm to explore the linearity of variables. However, it can be a disaster if we use the method to model nonlinear data. Fortunately, there are several methods to investigate the curvilinear relation among variables. We are going to introduce some common approaches to deal with nonlinear data.

To model a nonlinear structure, we want a model which has a smooth curvature to describe the data pattern. Increasing the order of polynomial in regression model does achieve the goal, namely polynomial regression. The objective function of polynomial regression is defined as $\beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d$. It has the same objective as a linear regression, which is trying to minimise the residual sum of squares. In other words, there is no different to find the coefficient estimates as linear regression. For nonlinear association, the value of the response variable varies at different datapoint. In the situation, the coefficients are not an interest, but the estimated value for the response variable. In addition, the model uses a hierarchical design to determine the most appropriate order. The hierarchical structure guarantees the model can be maximised or minimised at any point. Usually, we build a model with a sufficiently large degree and perform model selection to drop higher orders until the model reaches the optimal one. Of course, cross-validation is an alternative to pick up the size of degree as well. Furthermore, polynomial regression is very sensitive to data because it requires a high degree in order to produce flexible fits. Because of this reason, the end tail of the curve usually erratic and worse for extrapolation. In short, polynomial regression easily overfits the train data because of the high order of degree.

Instead of using polynomial regression, we can use step functions to model the nonlinear structure. The step function splits data into several intervals and takes the average of each interval as a function. Intuitively, it is not worth to mention when we compare to polynomial regression. Yet, if we put exploratory variables into the model, the situation changes as we are creating multiple distinct linear regression models at different intervals. It is also known as a piecewise linear regression model. Although it seems pretty good to use multiple models to explain different variation at different intervals, the model is not continuous. Regression lines are disjoined at knots, a point where separates intervals. We fail to find the maxima or the minima since the model is not differentiable. Replacing the step functions with $(x - c_k)_+$ would be an alternative to overcome the problem, it is a linear spline model with K knots. Mathematically, the objective function is $\beta_0 + \beta_1 x + \beta_2 (x - c_1)_+ + \cdots + \beta_{K+1} (x - c_K)_+$. Similar to polynomial regression, we can increase the order of polynomial in the model to make the curvature smoother and differentiable. We always define with three degrees of order, which is known as cubic spline with K knots. Put it into another way, we can imagine cubic spline is a model which consists of $K + 1$ identical but continuous polynomial regression models in distinct regions. The model complexity is almost the same as a polynomial regression with degrees of $K + 3$. Unlike polynomial regression, we can increase the model flexibility without increasing the number of degrees but adding more knots. However, the tail behaviour is similar to polynomial regression which is unstable for prediction. If we introduce some tweaks in the cubic spline, we can obtain a natural cubic spline which extrapolates linearly beyond

the boundary knots at $x_0 < \min(x)$ and $x_0 > \max(x)$. As such, natural cubic spline has less degree of freedom than the original one, and less likely to vary dramatically. In other words, natural cubic spline avoids the issue of erratic behaviour at the end tail of the curve and improves the performance in extrapolation.

So far, we have not discussed the placement of knots. Out of instinct, we need to determine the number of knots and the corresponding locations. The more the knots, the rapid the objective function changes. Yet, we need substantial support from a scatter plot to inspect the density of data points. All the procedures are done manually with the support of cross-validation. We have to find an optimal package which achieves the minimal cross-validation error. It is exhausting and tedious to find the optimal one. Luckily, with a desired amount of degrees, smoothing splines can automatically pick the best location for knots by giving a roughness penalty to the objective function. The models that we have mentioned have the same objective, which is minimising the residual sum of squares, that is $\arg\min(RSS)$. Alternatively, smoothing splines penalise the objective function to control its wiggliness, which is formulated as $\arg\min\left[RSS + \lambda \int f''(t)^2 dt\right]$. The variability is controlled by a tuning parameter $\lambda \geq 0$, and $\hat{f}(\cdot)$ will have higher variability with lower bias when λ is small. Smoothing spline does not require much manual definition as the effective degrees of freedom is also calculable, which is $df_\lambda = \sum S_{\lambda_i}$. The optimal degrees of freedom can be found by leave-one-out cross-validation. In conclusion, smoothing spline is very user-friendly, which puts all the works to a computer instead.

Furthermore, there are different ways to model nonlinearity. Local regression plots a curve instead of a line to explain the variation of variables, which is more flexible. It considers the weighted least squares of the K nearest data points to a focal point. The closer the point is to the focal point, the higher weight on determining the curvature. It repeats to find all the focal point until all points reach the objective, which is $\arg\min[RSS + K(x_0, \boldsymbol{x})]$. Besides, we can use a generalised additive model to fit a nonlinear structure. As the name suggested, it consists of several models by adding them into a single function, that is $\beta_0 + f_1(x_1) + \cdots + f_p(x_p)$ where $f_i(\cdot)$ is arbitrary function modelling nonlinearity. A remind for the generalised additive model is it does not accept any interaction term, which overlaps the space of exploratory variables.

All in all, the nonlinear structure often happens in real-life problems, and we have to deal with it precisely. The models we have discussed are some common model to achieve the ultra goal. Increasing the order of polynomial can make the curvature of a regression function smoother to capture the nonlinear variation. Besides, splitting the space of variables is a way to explore more fluctuate densities. More or less there are some advantages and disadvantages, but those drawbacks can be overcome by other models.

Chapter 7: Tree-based methods

The chapter focused on the evolution of decision tree with the regression problem and classification problem.

In Artificial Intelligence, a decision tree is one of the simplest and robust learning algorithms to extract the pattern from examples for prediction. It is a popular algorithm because of its user-friendly property which everyone can easily to interpret the results. For statistical learning, tree-based methods break the predictor space into small fragments and extract the data pattern. If the target output is a quantitative form, we consider the tree as a regression tree. Otherwise, it is known as a classification tree.

The question 'How to split the predictor space' becomes the core part of the tree development. We need to consider the information provided from each split. For a regression tree, we consider $RSS = \sum(y_i - \bar{y})^2$ of a split which can minimise the error from the train data. Similarly, for a classification tree, we often utilise information gain[1] to evaluate the importance of each predictor. The tree-building process continues until all the predictors are used, or no train data left after a split. However, it may raise an overfitting problem which will lead to poor performance in predicting unseen data. Such a fitted model is not desirable for the low prediction accuracy. Hence, we consider a general principle in inductive learning which is Ockham's Razor. Ockham's Razor principle is a concept which describes the fact that simple thing always does better than the complex one. In other words, we often prune a tree to make it simpler to generalise the model for increasing the prediction accuracy. More formally, we tolerate more noise in the fitted model and push up its variance as well, such that we can make a bias-variance trade-off to improve the test error. To prune a tree, we can use a turning parameter α to control the model complexity. The final goal is to minimise the error from the train data, hence, we have $\arg\min\left(\sum RSS_j + \alpha|T|\right)$ where the optimal value of α can be found by using cross-validation.

In statistic, the population parameters are always unknown, and we need to estimate them through statistical inference. If we use a single sample to infer the population, the variance may be very large, and the predicted value may not be favourable. Hence, we use to collect a large sample size to make inference via statistic. Bagging also applies a similar idea to reduce the variance of prediction by constructing some number of trees simultaneously. To grow different trees, it employs bootstrap to resample data from the same set of data. Then, it takes the average of predicted outputs to estimate the target result for the regression problem, that is $\hat{f}(x) = \frac{1}{B}\sum \hat{f}_i(x)$ where B is the number of built trees, and $\hat{f}_i(\cdot)$ is the estimated function for the i^{th} tree. On the other hand, for the classification problem, bagging takes advantage of a majority vote to find the proper label for observation instead of using the mean function. Mathematically, we can describe the idea of binary majority vote as a sigmoid function with $t = 0.5$, such that $\hat{f}(x) = \text{sigmoid}_{0.5}\left[\hat{f}_i(x)\right]$. However, some trees may be identical since the bootstrap may produce the same set of data to build a tree. When trees are highly associated with others, averaging the outcomes may not valuable to reduce the variance. Simply put, it wastes time to build so many trees if trees are highly correlated. The problem motivates us to the random forest, which is an extension of bagging. Random forest eliminates the number of predictors into $p/3$ for a regression tree whereas $\sqrt{p}$ for a classification tree. A tree grows with a random selection from the reduced number of predictors, and build trees simultaneously like building a forest with dissimilar trees. The elimination of predictor also acts as a

[1] Information Gain defined as $initial\ entropy - (weight\ average) \times entropy(A)$ where A is a predictor.

way to prune a tree. Recall the Ockham's Razor principle, a simpler model usually consistent with the most likely hypothesis. With fewer available predictors in trees development, the complexity of built trees become lower, which improves the performance to generalise the data pattern. Although random forest does not guarantee the improvement in prediction, it performs better than bagging in general. In the development of forest, moreover, we can have a better understanding of which predictor is more likely to contribute to the forest. For a predictor which has the most participation in declining the RSS or information gain will be considered as the most important predictor.

Instead of growing trees at the same time, we can build them one by one and introduce feedback to the next building tree. The method is called boosting which ensembles a set of weak learners to create a strong learner. We name a tree as a weak learner because the tree is controlled by a shrinkage parameter λ. In general, λ also controls the learning rate of the model development. The smaller the λ, the slower the learning speed. The learning rate is ranging from 0 to 1. Usually, we define λ as 0.01 or 0.001 since we want the forest learns the data pattern slowly. In boosting, for the regression problem, we propose to build B trees sequentially but not simultaneously. In addition, boosting use the same set of predictor values instead of using bootstrapped data. What is more even important, the response variable (or residuals r_i) is updated for every tree. Unlike the trees that we have mentioned before, trees in boosting estimate the residuals instead of the values of the response variable. Therefore, we cannot interpret the result from trees directly. The final prediction of boosting is defined as $\hat{f}(x) = \hat{f}_1(x) + \sum \lambda \hat{f}_i(x)$ which sum over all the residuals with the learning rate. Boosting starts with a single leaf which represents the initial guess of the weight for all observed values, that is $\hat{f}_1(x)$. To begin with, we need to specify the number of leaf nodes which controls the complexity of each tree. After building the first tree, the residuals tell the difference between the actual value and the predicted value, which is known as the feedback from the previous model. The next building tree will make use of the feedback to update the target outcomes and residuals with the given learning rate λ, that are $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_i(x)$ and $r_i \leftarrow r_i - \lambda \hat{f}(x)$. The process continues until it reaches B trees or adding additional trees does not significantly reduce the size of residuals. The size of B depends on the value of λ cause the amount learnt varies by λ. If B is too large, it may cause overfitting as well. Thus, we can use cross-validation to pick the optimal value of B while using boosting. In the random forest, trees are grown in a random manner to generalise data behaviour. In boosting, however, trees are built based on the feedback introduced in the previous trees which makes the learning process slower but a higher accuracy than the random forest.

Of course, boosting can extend to classification problem, which is called AdaBoost. Unlike other tree models always build a full-sized tree, AdaBoost only considers binary split trees. That is every classification tree in AdaBoost, there is only a root node with two leaf nodes. The ability to predict the class of observation is not significant for a tree with 1 split. As AdaBoost is a subset of boosting, we can understand AdaBoost ensembles a set of weak trees to build up a robust forest. Similarly, AdaBoost produces tree sequentially and provides feedback to the next building trees. Initially, each data are equally weighted with $1/N$. The weight of each data is updated for the next building tree. If a tree misclassifies a data, AdaBoost will tell the feedback to the next tree by increasing the weight of the corresponding data. A data with larger weight will give a signal to the tree take account to it. Besides, data with huge weight will be the focus to classify them. The normalised weighted error is defined as $\varepsilon_m = \frac{\sum w_n^{(m)} 1_{y_m(x) \neq t_n}}{\sum w_n^{(m)}}$ whereas the update rule for the $m + 1^{th}$ tree is formulated as $w_n^{(m+1)} = w_n^{(m)} \exp\left(\alpha_m 1_{y_m(x) \neq t_n}\right)$ where $\alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$ is also the weight of m^{th} tree in the final prediction. Since AdaBoost grow a binary tree consequently, the learning process is slower than other tree-based methods we have introduced before. However, it can learn some area in the predictor space that is hard to learn in other models.

In summary, the adaptability and interpretable make tree-based methods being so popular. Although a single tree may not perform well in predicting unseen data, there are some approaches to improve the prediction accuracy by constructing more trees as a forest. Those methods generalise the data pattern by Ockham's Razor principle or ensemble simpler trees to build a powerful forest. While feeding trees with feedback, boosting improves trees consequently and results in an accurate prediction.

Chapter 9: Unsupervised learning

The chapter concentrated on cluster analysis with several unsupervised learning models.

In supervised learning, we have a correct (or approximately correct) value of a function for some particular inputs, say $\big(x, f(x)\big)$. Learning algorithms can receive feedback from the train data and improve its performance. However, we may not always have such an ideal dataset. In unsupervised learning, we only use the underlying features to make a prediction. The most common approach is cluster analysis which tries to classify the dataset to facilitate the analysis, such as visualisation and exploratory data analysis.

To form groups with similar data, the k-means method is one of the most popular learning algorithms to measure the squared Euclidean distances. As the name suggested, the k-means algorithm divides data into k groups. The value of k can be variable, and we will discuss it later. The learning algorithm is very similar to Lloyd's algorithm. Both of them are trying to find a point in each group which can minimise the distance. Mathematically, we can formulate the idea into $\arg\min \sum \|x_i - m_{Z_i}\|_2^2$ where m_{Z_i} is the centroid (or centre) of Z_i cluster. The algorithm, initially, starts with k random centroids in the Euclidean space. It assigns the groups for all x_i to the nearest centroid computed by the Euclidean distance. Since our goal is to minimise the distance, we can take the partial derivative to find the value of m_{Z_i}, and the resulting m_{Z_i} is defined as $\frac{1}{n_{Z_i}} \sum_{i \in Z_i} x_i$. Then, the k-means method updates the value of centroid of Z_i by the average of x_i belonging to that corresponding group. It continues until the convergence of centroids (the value of centroids do not change substantially). The algorithm is a coordinate descent procedure which means that the objective function must decrease in every iteration. It guarantees the convergence of the solution. Yet, it is also a drawback of the k-means method, which will only converge to local optimal. Since the algorithm focus on minimising the objective function, it never knows the best clustering under a single situation. To overcome the problem, thus, it is common to perform the k-means method a couple of times to compare the total variation in each cluster. Moreover, the k-means algorithm is very sensitive to outliers. As we have discussed a second before, the algorithm updates the centroids based on the value of x_i. In other words, the centroids can be dragged by outliers if outliers are far away from the data centre. Because the algorithm considers the distances form all x_i, outliers must get their cluster instead of being ignored.

The distance measurement in the k-means method is fixed in using squared Euclidean distance. The use of other measurement motivates us to the other method call k-medoids clustering. The k-medoids algorithm is an analogy to the k-means method. Both of them are trying to minimise the distance between x_i and the centroids, but k-medoids algorithm accepts arbitrary distance measurement instead of squared Euclidean distance. The k-medoids algorithm could be more powerful when facing noisy dataset compare to the k-means method. Some distance measurement generalises the pairwise dissimilarities which are more favourable than squared Euclidean distance when the dataset consists of outliers. Again, the name of algorithms already gives us some hints about their properties. The k-means method use means to compute the centroids, whereas the k-medoids method utilises medoids to measure the distance of clusters. Medoids are the data points from the data set, that is the k-medoids algorithm treats k data points as centroids for each cluster. When the size of the observation grows, k-medoids clustering can keep as much information as possible. It seems the k-medoids algorithm is much more robust than the k-means method, however, the computing process can be a disaster. As the algorithm computes all the pairwise

distance from x_i, the time complexity is $O(n^2)$[2], where n is the number of observations in the dataset. The computation time is far away from the k-means method.

For many cases, we do not know the number of clusters in advance. Sometimes the value of k may be subjective to the topic under discussion. Yet, there are so many methods to finds the size of clusters, such as the scree plot. In the lecture, we went through the idea of gap statistic which is comparing k-means objective for observed data O_K to the objective for null data O_{Kb}. We usually use the Monte Carlo method to simulate a set of data based on the range from the original data. We call the generated dataset as null data since its distribution should not have any obvious structure. The reason we name the gap statistic is that it is the difference (or gap) between the average of O_{Kb} and O_K, hence, it is more understandable why the definition of gap statistic is defined as $Gap(K) = \frac{1}{B}\sum \log(O_{Kb}) - \log(O_K)$. If there is an underlying structure in the observed data, we expect the objective function for k-means will be smaller than that of null data. Thus, we need to maximise the gap statistic to select the value of k. As we mentioned the Ockham's Razor principle in the last chapter summary, we often want a simpler model to achieve our goal. The value of k can be considered as a parameter which controls the model complexity. Therefore, we often use the smallest value of k in $Gap(K) + sd(U_K) \geq Gap(K+1)$ in practice. After we obtain the favourable k, the learning algorithms in the previous paragraph is implementable to investigate the unknown knowledge in the data.

So far we have discussed some methods to group data into several clusters. However, there are still many ways to classify data in more concisely, that is hierarchical clustering. Recall the method of k-means and k-medoids, both of them require the value of k to determine the number of clusters. When we have no idea on the number of clusters being used, trying different values of k will result in different groupings. Hierarchical clustering employs nested clustering method to resolve the uncertain issue. The algorithm starts with n clusters where n is the size of the dataset. It repeatedly mapping the nearest data into one cluster. The "nearest" refers to the distance measurement. The process continues until all the data points are in a single cluster. We can use dendrogram to display the association between all the nearest data point. Unlike the k-means method or the k-medoids method, the hierarchical method only tells us which 2 data points (or clusters) are the most similar.

In conclusion, the learning algorithms we have introduced are some distance-based methods to classify data. They are very formidable to process the dataset. However, distance-based methods are restricted to the use of categorical variables and the scale of predictors. Moreover, the algorithms may converge to a constant when the number of dimensions increases. The performance under such circumstance will not be impressive. Having said that, clustering is one of the most common approaches in unsupervised learning algorithms. The power of algorithms can be even greater than the supervised one.

---

[2] Big-oh of n^2