THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

MODULE 2 UNIT 2
The machine learning pipeline

Table of contents

Learning outcomes:

LO3: Identify the role of the bias–variance trade-off and cross-validation as key parts of the machine learning pipeline.

LO4: Analyse the output of an appropriate machine learning model in R to inform decisions.

1. Introduction

In Unit 1, the machine learning model was defined and discussed with specific reference to the *k*-nearest neighbours (KNN) classifier and polynomial regression. The discussion also highlighted the problems that come with underfitting and overfitting. Unit 2 builds on this foundation by taking a closer look at the method of optimising a model to ensure accurate results and to eliminate underfitting and overfitting.

In this lesson, important concepts associated with fitting models are discussed, including the importance of the bias–variance trade-off, the test error, and cross-validation. From there, the machine learning pipeline is introduced as a method of optimising a model to generate the most effective output. As this unit builds on the concepts introduced in Unit 1, the same examples of the KNN classifier and polynomial regression will be used.

2. Concepts in the machine learning pipeline

In this section, some important concepts are explored as an introduction to the machine learning pipeline. Understanding the overall concepts of the bias–variance trade-off, test error, and cross-validation will facilitate a thorough understanding of the machine learning pipeline and how it is applied to specific examples.

2.1 Bias–variance trade-off

To understand the bias–variance trade-off, it is crucial to comprehend both bias and variance as two sources of error in a model (Fortmann-Roe, 2012). Bias refers to an error that occurs as a result of a model's simplistic assumptions when fitted onto a data set. If the bias is high, it means that the model is unable to effectively capture all the patterns in the input data. The inability to capture the patterns effectively leads to underfitting (Agarwal, 2018). The error caused by bias is the difference between the expected model prediction and the actual value that should be predicted. In simple terms, the error indicates how far off the model prediction is (Fortmann-Roe, 2012).

Variance refers to the extent to which the model is dependent on the training data (Koehrsen, 2018). It indicates the error that is generated by a complex model attempting to fit the data. If the variance is high, it means that the model attempts to encompass all the data points in the data set, which leads to overfitting (Agarwal, 2018). The error caused by the variance indicates how much the predictions for a given data point vary every time the same model is applied to the data set (Fortmann-Roe, 2012). The variance between different outputs that the model produces when applied to the same data set should not be significant (James et al., 2013:34).

Table 1: The interrelationship between model fit, bias, and variance. (Adapted from: Agarwal, 2018)

Model fit	Bias	Variance
Correct fit	Low	Low
Underfit	High	Low
Overfit	Low	High

Following from the information in Table 1, consider the following illustration of the contribution of bias and variance towards the total error of a model.

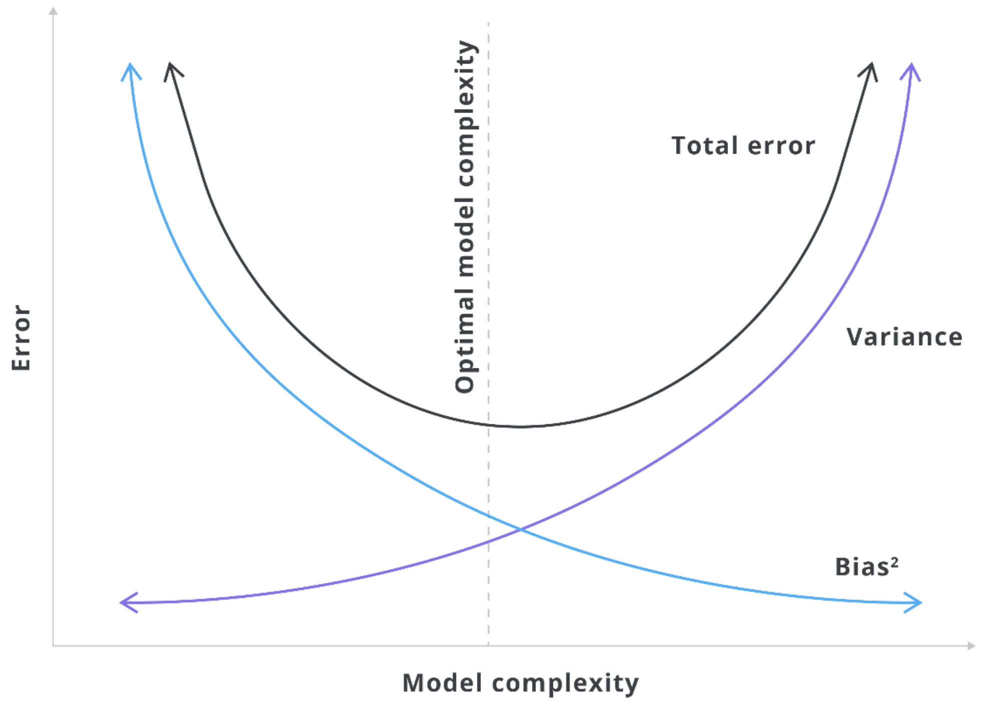

Figure 1: Contribution of bias and variance towards the total error. (Adapted from: Fortmann-Roe, 2012)

From Figure 1, it is clear that the model should not be too complex (leading to overfitting) or too simple (leading to underfitting), as these extremes impact the error rate. To minimise the expected test error, it is important to choose a model that achieves low variance and low bias (James et al., 2013:34).

Explore further:

Module 1 discussed the different types of data and highlighted the importance of data for machine learning. Refer to the visual introduction to machine learning from Module 1. Building on this and using the same data set, explore this interactive example of model tuning and the bias–variance trade-off.

2.2 Test error

The concepts of training error and test error were introduced in Unit 1. When considering the training error in isolation, it does not account for the bias–variance trade-off. To account for this, a data set is split into two parts: the training data and the testing data. Once the correct type of model is chosen, it is applied to the training data set.

However, maximising the accuracy or minimising the training error could lead to overfitting. To solve this, after the model has been trained using the training data set, it is then fitted onto the testing data set to generate predictions. The testing error metric is then applied to these predictions to determine the model's performance on unseen data. If the test accuracy or error is close to the training accuracy or error, it indicates that the optimal trade-off between bias and variance has been found (Gupta, 2017).

2.3 Cross-validation

Validation in machine learning refers to the "process of deciding whether the numerical results quantifying hypothesised relationships between variables, are acceptable as descriptions of the data" (Sanjay.M, 2018). Validation is one of the most important techniques to ensure the accuracy and stability of a machine learning model (Drakos, 2018). Therefore, validation means that the model's performance can be tested on unseen data. This is done by keeping a sample of the data separate to be used to validate the model, which is also known as the validation data set (Drakos, 2018).

Cross-validation a useful technique when assessing the effectiveness of a model to avoid overfitting (Gupta, 2017). In cross-validation, the training data is split into a number of subsets, or folds. A portion of the training data set is held aside, and the model is fitted on the remaining training data. The metrics are generated on the portion of the training data that was held aside – the validation set (Koehrsen, 2018). The model is fitted onto this validation set after the training data set; the model's error can then be analysed because it is fitted onto unseen data for the first time. This enables the analyst to select the optimal tuning parameter that minimises the testing error or maximises the testing accuracy.

A tuning parameter is used to control the model's behaviour, such as the polynomial degree in polynomial regression, the value of *k* in KNN methods, and the number of variables included in calculating the predictions. The tuning parameter cannot directly be learnt from the model, but should be provided by cross-validation. Different values of the tuning parameter usually yield different models.

Consider the following cross-validation methods that can be applied in different situations:

- **Holdout method:** This simple method involves keeping a part of the training data set separate to test the model after it has been trained; the separate data set becomes the testing data set, which is the unseen data used to determine the accuracy of the fitted model. This method sometimes involves high variance due to the uncertainty relating to the data included in the validation data set since it is unknown which data points will end up in the different sets.

- ***k*-fold cross-validation:** This form of cross-validation repeats the holdout method a number of times. First, the data is divided into *k* portions of roughly equal size. Then, with each iteration, a different portion of the training data is allocated for performance testing. This method overcomes the shortfall of the holdout method by generating multiple random samples. The evaluation of the model fit is therefore not dependent on the characteristics of only one sample. By using different portions of the training data, the essential patterns and trends of the data remain intact, ensuring the accuracy of the model. Determining *k* depends on the particular application and other practical considerations. Popular choices of the value of *k* include *k* = 5 and *k* = 10.

- **Stratified *k*-fold cross-validation:** This method is a slight variation of the *k*-fold cross-validation method used for data sets with an imbalance in response variables. In this method, the partitions of the data are selected so that each contains a similar percentage of samples per target class to the complete data set. In the case of binary classification, this means that each partition contains roughly the same proportions of the two types of class labels.

- **Leave-*p*-out cross-validation:** Leave-*p*-out cross-validation involves using *p* observations as the validation set and the remaining observations as the training set. If an *n* number of data points exists in the original data set, then *n* − *p* samples are used to train the model. The *p* samples that were removed are used as the validation set to test the accuracy of the model. This procedure is then repeated multiple times, where a different set of *p* observations is chosen for every iteration. Five-fold cross-validation is a special case of leave-*p*-out cross-validation by choosing $p = [{}^{n}/_{5}]$.

(Gupta, 2017)

Regardless of the type of cross-validation used, it is a useful way to evaluate the effectiveness of the model to generate an output from unseen data (Koehrsen, 2018). If, for example, a model must be trained to distinguish two types of output from a data set, and the data points are represented by red and blue dots, the data set will be split into different folds and sections to train and test the model. This example of *k*-fold cross-validation is shown in Figure 2.

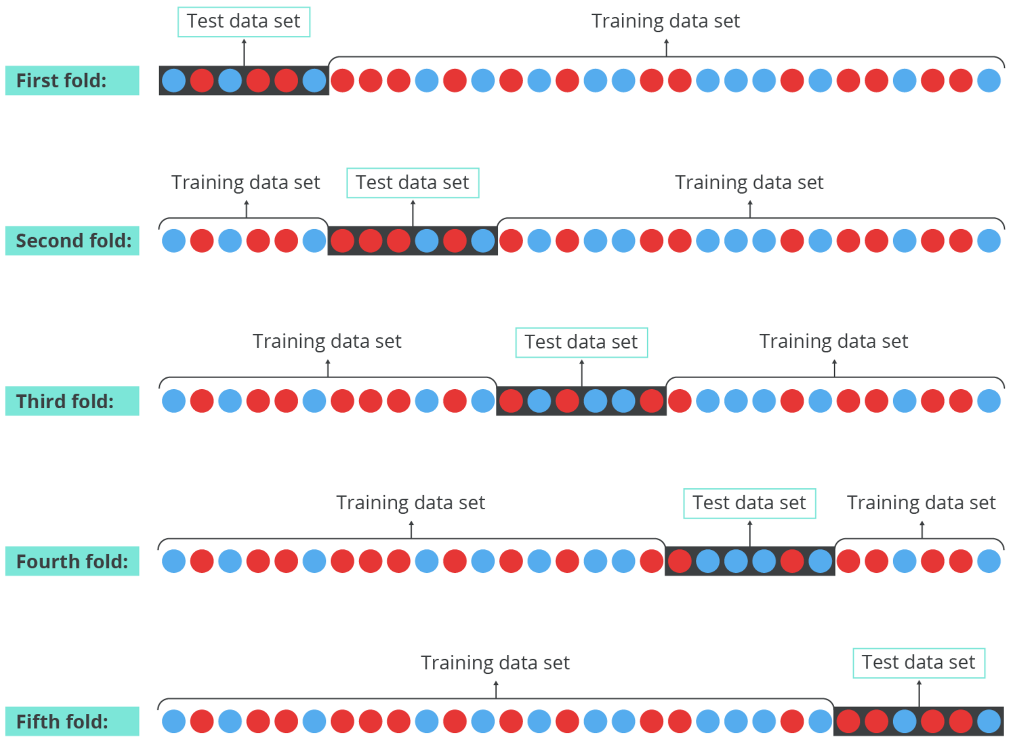

Figure 2: Five-fold cross-validation.

The model is validated, as by doing this, it is verified on multiple subsets of data, which improves its accuracy (Srinivasan, 2018). Despite the importance of ensuring the accuracy of a model, this process also has the following limitations:

- **Unknown data:** When the model is trained and tested on specific data, it might yield insightful and accurate outputs. However, it is not a certainty that the model will always be accurate when fitted onto different data sets. Note that this problem is not unique to cross-validation, but applies to all machine learning models.

- **Changes in data:** The model is usually trained and tested on data that was collected over a certain period of time. It is not realistic to expect the same model to make accurate predictions over a long period of time, as the circumstances may change, which impacts the relationship between variables.

- **Unrepresented samples:** The training and testing errors may be due to an imbalanced sample, which is unrepresentative of a real-world scenario. As a result, the model will be inaccurate when fitted onto a real data set.

(Srinivasan, 2018)

The process of cross-validation can be repeated multiple times. By choosing different random subsets of data for each iteration of validation, it reduces the chance of problems associated with cross-validation occurring.

Having gained a better understanding of the important concepts covered in this section, explore how these concepts are applied in the machine learning pipeline to optimise the functionality of a model. Before moving on to the next section, test your understanding of these important concepts by answering the following questions.

At this point in the lesson, you have the opportunity to engage with a practice quiz to test your understanding of the content. Access this lesson on the Online Campus to engage with this quiz.

3. The machine learning pipeline

The machine learning pipeline offers a three-step guide to train a model to solve a specific problem. Before delving deeper into the different steps, engage with Figure 3, which provides a guide to apply the machine learning pipeline to any model.

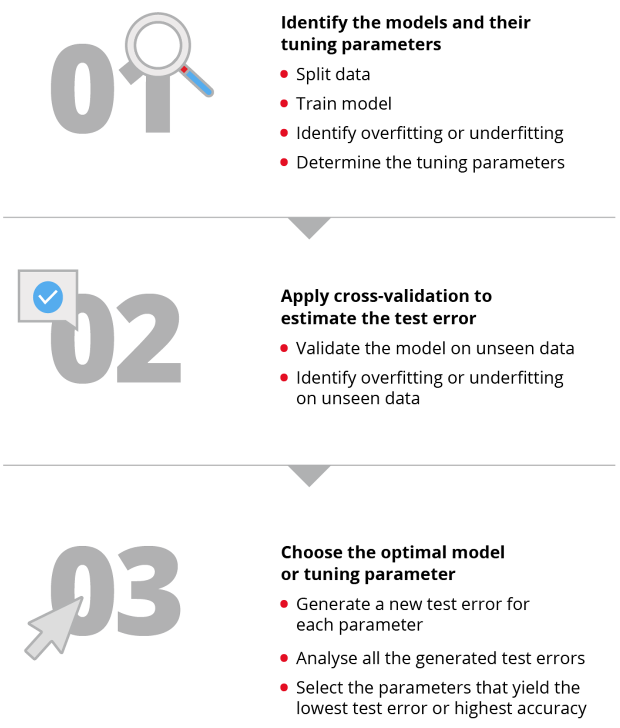
01
Identify the models and their tuning parameters
• Split data
• Train model
• Identify overfitting or underfitting
• Determine the tuning parameters
02
Apply cross-validation to estimate the test error
• Validate the model on unseen data
• Identify overfitting or underfitting on unseen data
03
Choose the optimal model or tuning parameter
• Generate a new test error for each parameter
• Analyse all the generated test errors
• Select the parameters that yield the lowest test error or highest accuracy

Figure 3: The steps of the machine learning pipeline.

In the next three sections, each step within the machine learning pipeline is discussed.

Step 1: Identify the models and their tuning parameters

By now, it should be clear that data sets are split into the training set and the testing set. The training set is used to train the model, and the testing set is used to test the model's accuracy. The training set is therefore used to fit a model and to tune it to yield the most accurate outputs when fitted onto unseen data (Elite Data Science, n.d.).

After choosing the parameters, the relevant model is applied to the training data set and then the testing data set. Training the model takes up the bulk of the machine learning process. During this step, the model is improved incrementally by applying it onto different data sets and assessing its accuracy (Yufeng G, 2017).

Therefore, when it appears, during testing, that the model is overfitted or underfitted, the parameters must be tuned. It is essential to distinguish between model parameters and hyperparameters to comprehend the tuning process. Model parameters are learned directly from the training data set, whereas hyperparameters are not. Hyperparameter values are set before the training process begins, and are therefore decided before the model is fitted (Elite Data Science, n.d.). The hyperparameters are thus tuned to ensure optimal model performance (Bhattacharjee, 2017).

In the Unit 1 lesson, two practical examples of machine learning models were explained: KNN and polynomial regression. For the KNN classifier, the model's hyperparameter is the number of nearest neighbours, or *k*, that is chosen for the model. Polynomial regression's hyperparameter is the order of the polynomial.

Once the data is understood in its entirety, the most suitable test parameters should be chosen. In the next step of the machine learning pipeline, the model is estimated and tested with these parameters.

Step 2: Apply cross-validation to estimate the test error

When the model has been trained, the analyst should determine how well it performs on unseen data. It is not certain whether the model will yield the desired accuracy and variance when fitted onto new data. Therefore, some form of quality assurance must be done to guarantee the model's accuracy (Sanjay.M, 2018). As discussed previously, it is important to remain mindful of overfitting the model on data sets and the high variance it creates. To overcome this challenge, the model must be validated.

By applying one of the different cross-validation methods, the performance of the model can be tested on unseen data. This is done by keeping a sample of the data separate from the training and testing sets to test how the model performs on new data (Drakos, 2018).

After the model has been validated, the optimal model or tuning parameter can be chosen.

Step 3: Choose the optimal model or tuning parameter

After cross-validation has been applied, repeat Step 1. Thereafter, new parameters are chosen, and a new test error is generated for each parameter. This process is repeated multiple times to generate multiple test errors. All of the generated test errors are then analysed, and the model parameters that produced the lowest test error (or highest accuracy) should be selected and fitted onto the input data.

Selecting the optimal model can be done by considering certain model aspects and asking the following questions:

- **Performance:** Which model shows the best performance when tested?
- **Robustness:** How well does the model perform across different metrics?
- **Consistency:** Does this model have one of the best cross-validation scores when fitted onto the training data set?
- **Solution:** Does the model solve the original problem it was supposed to?

(Elite Data Science, n.d.)

Now that you understand the different steps in the machine learning pipeline, Section 4 elaborates on how these steps are applied to the KNN classifier and polynomial regression.

4. Applying the machine learning pipeline

This section explores the application of the machine learning pipeline to two practical examples: the KNN classifier and polynomial regression.

4.1 KNN

In Unit 1, a KNN model was applied to the MNIST data set. However, the data set was not split into training and testing sets before fitting the model; as a result, the test error was not calculated. In this example, the KNN algorithm is applied with specific reference to the machine learning pipeline as a tool to optimise the model's performance. The same data set of handwritten digits is used for this example.

Step 1: Identify the models and their tuning parameters

The first step is to choose a machine learning model. This is best done by looking at the type of data the model will be fitted on. Due to the way that a handwritten digit is classified into different classes between 0 and 9, the KNN model is a suitable option. It is a good idea to choose a broad array of tuning parameters to develop an understanding of the data and the best fit of the model. To do this, the tuning parameters for this example will be k = 1 to k = 25 in increments of 2.

Step 2: Apply cross-validation to estimate the test error

Various methods of cross-validation can be applied. In this example, leave-one-out cross-validation is used. Note that this method is part of the leave-p-out cross validation method. The model is estimated using the tuning parameters described in Step 1 and the derived accuracies are listed in Table 2.

Table 2: The *k*-values and associated accuracies.

Value of *k*	Accuracy
1	0.9438
3	0.9458
5	0.9384
7	0.9337
9	0.9297
11	0.9331
13	0.9297
15	0.9264
17	0.9230
19	0.9230
21	0.9197
23	0.9183
25	0.9177

The accuracies depicted in Table 2 are illustrated in Figure 4.

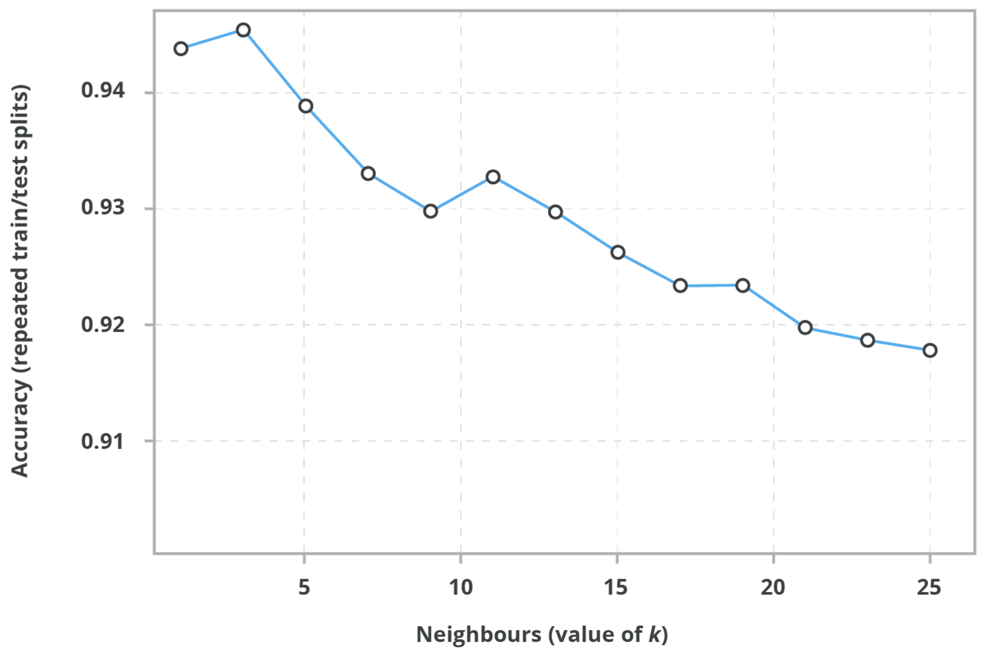

Figure 4: Model accuracy for each value of *k*.

By plotting the accuracies on a graph, it is easier to identify the value of *k* that yields the highest accuracy rate.

Step 3: Choose the optimal model or tuning parameter

After the accuracies for each *k*-value has been calculated, the value of *k* that yields the highest accuracy rate should be chosen as the model parameter. Based on the accuracies depicted in Table 2 and Figure 4, the accuracy is highest when *k* = 3.

Pause and reflect:

Recall the KNN example in the Unit 1 lesson where the accuracy of the model was also at its highest when *k* = 3.

After choosing the value of *k* and identifying the optimal tuning parameter, the model can be fitted onto unseen data to calculate predictions.

4.2 Polynomial regression

In Unit 1, the polynomial regression was fitted onto an artificial data set, which will also be used to show the effect of the machine learning pipeline. As with the KNN example, the data set was not split into training and testing sets; therefore, the test error was not calculated. As such, the model improved as the order of the polynomial increased, but this process could have caused overfitting. Therefore, the next section uses the same example to demonstrate how the machine learning pipeline is applied when fitting a polynomial regression line onto a data set to ensure optimal performance.

Step 1: Identify the models and their tuning parameters

The easiest method of determining the best model for a given data set is to visualise the data as a scatterplot. The data set used for this example is visualised in Figure 5.

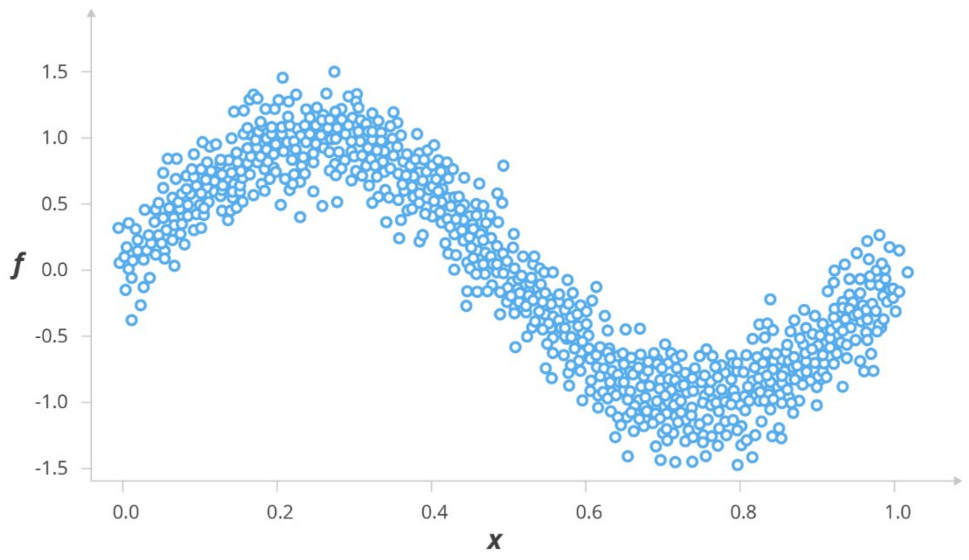

Figure 5: Scatterplot of the polynomial regression data set.

Due to the spread of the data points in the scatterplot, a polynomial regression would be suitable. Fitting a linear regression line onto this data set will lead to underfitting, which gives rise to inaccurate predictions. After identifying the most suitable model, the tuning parameters are chosen. Ideally, a polynomial regression should not be fitted with an order greater than 10, since this could result in overfitting.

Step 2: Apply cross-validation to estimate the test error

As before, cross-validation is then used across the different tuning parameters to optimise the model. For this example, repeated *k*-fold cross-validation is used to calculate the test error for each order of the polynomials. The cross-validation will be 10 folds and will be repeated 5 times. The tuning parameters are then applied to choose which order of the polynomial will be tested in each model. After applying cross-validation to the data set, the values of the estimated root mean square error (RMSE) of the orders between 1 and 15 are shown in Table 3.

Table 3: The estimated value of RMSE calculated per order of polynomials.

Order of polynomial	RMSE
1	0.4919266
2	0.4926768
3	0.2205515
4	0.2208139
5	0.2071710
6	0.2072827
7	0.2073782
8	0.2077113
9	0.2078872
10	0.2082576
11	0.2080279
12	0.2082606
13	0.2082510
14	0.2085488
15	0.2087478

To make it easier to identify the lowest value of RMSE, the data contained in Table 3 can be visualised as shown in Figure 6.

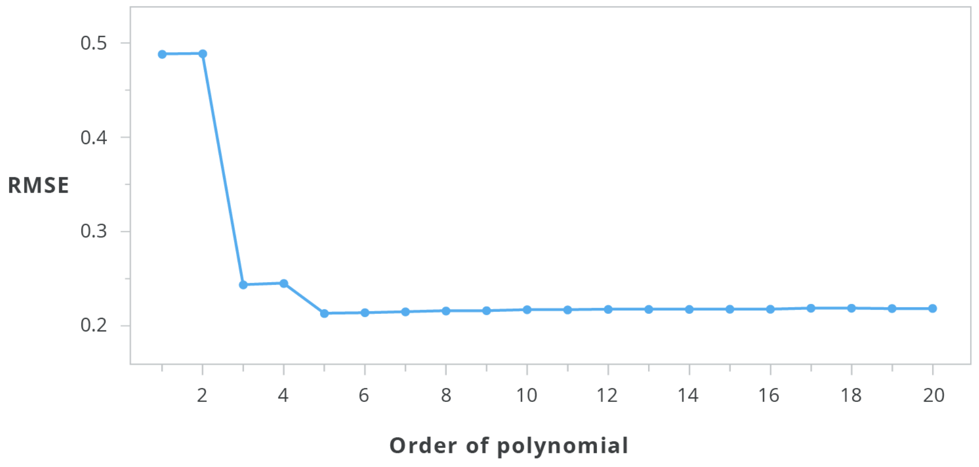

Figure 6: The value of RMSE in relation to the order of the polynomials.

After applying cross-validation and determining the test error, the optimal order of the polynomial can be chosen.

Step 3: Choose the optimal model or tuning parameter

Now that the RMSE has been estimated for each order, the lowest RMSE can be identified. In this instance, the lowest one is for a polynomial of order 5. Using the model estimated with a polynomial of order 5 on the test data yields the results shown in Figure 7.

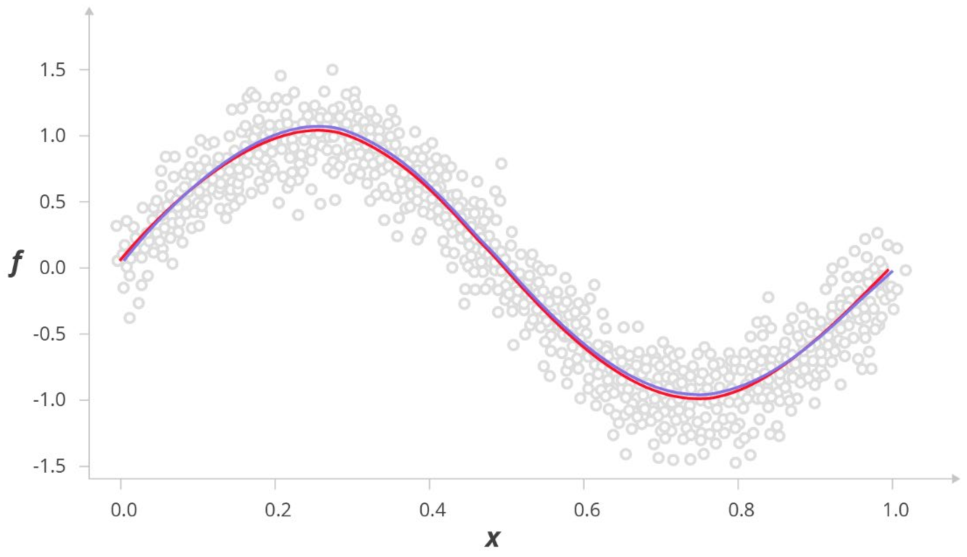

Figure 7: The model fit on the fifth order of the polynomial.

In Figure 7, the purple line shows the true function of the test set, while the red line shows the predicted values generated by testing the model. The predicted values (red line) almost entirely capture the true function (purple line). It is also clear that the polynomial regression line accurately represents the true values of this particular data set.

5. Practical examples of KNN and polynomial regression in R

After engaging with the applications of the machine learning pipeline to optimise the KNN and polynomial regression models, watch Video 1, which illustrates how this process is executed in R.

Video 1: Apply KNN and polynomial regression in R. (Access this lesson on the Online Campus to engage with this video.)

Note:

It is advisable to normalise variables prior to running K-NN if the range of the values the variables take are widely different.

At this point in the lesson, you have the opportunity to engage with a practice quiz to test your understanding of the content. Access this lesson on the Online Campus to engage with this quiz.

6. Conclusion

This lesson highlighted the basic concepts of modelling, fitting a model, bias errors, variance errors, and the machine learning pipeline. The first step is to identify the models and their tuning parameters. Next, cross-validation is applied to estimate the test error, and finally, the optimal model or tuning parameter is chosen. After this, the chosen model can be used to calculate predictions.

Navigate to the IDE activity in the next component to execute the steps of the machine learning pipeline on a new data set.

7. Bibliography

Agarwal, A. 2018. *Polynomial regression*. Available: https://towardsdatascience.com/polynomial-regression-bbe8b9d97491 [2019, November 22].

Bhattacharjee, J. 2017. *Some key machine learning definitions*. Available: https://medium.com/technology-nineleaps/some-key-machine-learning-definitions-b524eb6cb48 [2019, November 21].

Celisse, A. 2014. Optimal cross-validation in density estimation with the L^2-loss. *The Annals of Statistics*. 42(5):1879-1910.

Drakos, G. 2018. *Cross-validation*. Available: https://towardsdatascience.com/cross-validation-70289113a072 [2019, November 28].

Elite Data Science. n.d. *Model training*. Available: https://elitedatascience.com/model-training [2019, November 27].

Fortmann-Roe, S. 2012. *Understanding the bias-variance tradeoff*. Available: http://scott.fortmann-roe.com/docs/BiasVariance.html [2019, November 26].

Gupta, P. 2017. *Cross-validation in machine learning*. Available: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f [2019, November 26].

James, G., Witten, D., Hastie, T. & Tibshirani, R. 2013. *Introduction to statistical learning: with applications in R*. New York: Springer-Verlag.

Koehrsen, W. 2018. *Overfitting vs. underfitting: a complete example*. Available: https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765 [2019, November 22].

Sanjay.M. 2018. *Why and how to cross validate a model?* Available: https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f [2019, November 28].

Schönleber, D. 2018. *A “short” introduction to model selection*. Available: https://towardsdatascience.com/a-short-introduction-to-model-selection-bb1bb9c73376 [2019, November 25].

Srinivasan. 2018. *The importance of cross validation in machine learning* [Blog, 23 November]. Available: https://www.digitalvidya.com/blog/cross-validation-in-machine-learning/ [2019, November 28].

Yufeng G. 2017. *The 7 steps of machine learning*. Available: https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e [2019, November 28].