Week 5 Data Modeling (Supervised Learning)

Overview

This week's topic will cover the Data Modeling phase (supervised learning methods) of a data mining process, as depicted in Figure 1.

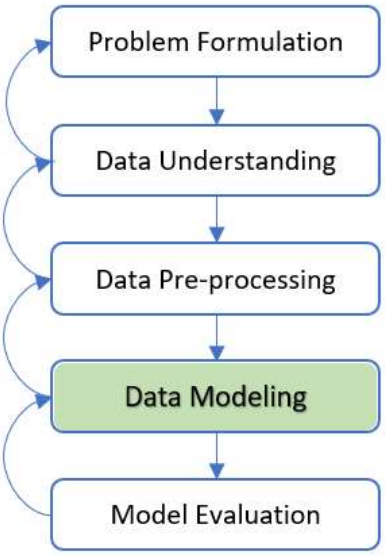


Figure 1 Data Modeling Phase

Data modeling methods may be categorised as either supervised or unsupervised learning methods for modeling data. We have studied the unsupervised learning methods in Week 4.

In supervised learning methods (a.k.a. predictive modeling), there is a particular prespecified target attribute. The learning algorithm is given many training data as examples to provide the values of the target attribute. The algorithm may then learn which values of the target variable are associated with which values of the input attributes as the predictor. For instance, our mini-case study of customer churn prespecified the Churn as the target attribute and other attributes such as payment transaction details and demographics as the predictor. In other words, the case study predicts if a customer will churn based on the input attributes (demographics and transactions) as the predictor.

This week, we will study the commonly used supervised learning methods, i.e., Decision Trees, Neural Networks, and Ensemble Learning. We will use Python to implement the supervised learning methods for the data modeling phase of data mining. We will implement Regressions as well; however, excluding the explanation of its concepts because students are expected to have learned it in Statistics-related subject(s).

5.1 Supervised Learning Methods

A model is the abstract representation of the data and its relationships in a given data set. Different data modeling algorithms represent a model differently. A few hundred data modeling algorithms are available today, derived from statistics and machine learning approaches. In addition, many viable market commercial and open source tools implement these algorithms.

As data mining practitioners, we need an algorithm overview, know how it works, and determine what parameters need to be configured based on our understanding of the problem requirements and data. In supervised learning, the algorithms need a known data set with target variable actual values to "learn" for training the model. Figure 2 shows the steps in the modeling phase of a supervised data modeling phase.

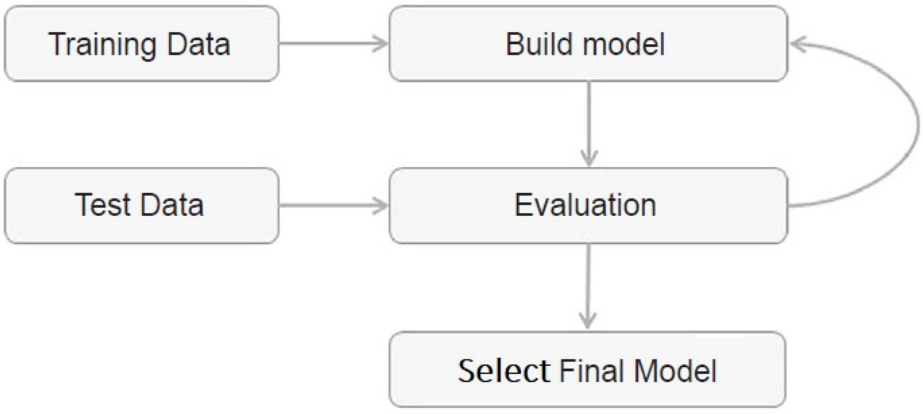


Figure 2 Steps in Data Modeling Phase with Supervised Learning Methods

Data Set Splitting for Training and Validating Models

To build a model, we need to use a previously prepared data set where we know all the attributes, including the target class attribute. This data set is called the training data set and is used to develop a model. We also need to check the validity of the developed model with another known data set called the test or validation data set. The overall available data set is split into training and test data sets for validity. There is no standard rule of thumb for the ratio of the data to go to training and the test data set, but there is at least a 50% or more for the training data set.

Types of Prediction in Supervised Learning Methods

The problems to solve and data availability determine what data modeling techniques to use. Further, the type of prediction performed by a supervised learning method also influences the selection of appropriate data modeling techniques. In general, there are two types of prediction: 1. Classification, 2. Estimation.

1. Classification

Classifiers (i.e., data modeling techniques for Classification) take a list of attributes and decide into which of many classes the target variable is exhibiting these attributes. Automatic target class recognition and future event prediction are examples of a classifier. Within classification, the commonly used algorithms are Decision trees, Neural networks, Bayesian models, k-NN, Logistics regression, and Ensemble models.

Our mini-case study of customer churn prediction is a classification problem. Therefore, we will use several commonly used classification techniques, i.e., Decision Trees Classifier of Decision Trees, Multi-Layer Perceptron Classifier of Neural networks, and Ensemble method, to model the data set to demonstrate the data modeling phase concept.

For the implementation in Python, the customer churn data set will be split into training and test sets. The training set will be used to develop the model, and the test set will be used to evaluate the model's validity.

2. Estimation

Estimators (i.e., data modeling techniques for Estimation) take a list of input attributes and assign a numeric value to the target label demonstrating these attributes.

To elaborate on how we can implement Estimators' supervised learning methods, we reuse the customer churn data set but convert the target variable data type to numeric to fulfill the characteristics of an Estimator in predicting the numeric value of a target variable. We will use several commonly used estimation techniques, i.e., Decision Tree Regressor of Decision Trees, Multi-layer Perceptron Regressor of Neural Networks, and Multilinear Regressions, to model the data set to demonstrate the data modeling phase concept.

Note: It is beyond the scope of this course to explain the detailed algorithms and concepts of all supervised learning methods. For this course, we apply several algorithms to demonstrate the concept of the data modeling phase by applying the commonly used techniques and learn how we can implement them in Python. Nevertheless, students will learn an overview of the applied techniques and the basics of how they work.

Note: It is beyond the scope of this course to explain the detailed algorithms and concepts of all supervised learning methods. For this course, we apply several algorithms to demonstrate the concept of the data modeling phase by applying the commonly used techniques and learn how we can implement them in Python. Nevertheless, students will learn an overview of the applied techniques and the basics of how they work.

Python provides many algorithms for both Classification and Estimation. Students can refer to the https://scikit-learn.org/ for available techniques and the documentation of their APIs.

Classification and Estimation are two types of data modeling approaches that can be used to develop models to predict future data trends. As described earlier, classification predicts categorical (discrete, unordered) target variables (a.k.a. labels), estimation models continuous-valued labels. For example, we can build a classification model to predict whether a potential customer will churn or not. In contrast, an estimation model predicts potential customers' expenses in monetary units on products given their demographics, such as income and occupation.

In the next sections, we will examine the basics of Classification and Estimation, including how they work and how to build models using the basic concepts.

5.2 Basics of Classification

A bank loan officer must analyze the data to learn which loan applicants are at risk or safe in paying back a loan. Likewise, a customer management executive needs data modeling to help predict whether a customer with a given profile will churn. Also, a medical scientist analyzes cancer-related data to predict one of three specific therapies a patient should obtain. Each of these examples is a Classification problem, where a model or classifier is developed to predict categorical labels. For example, "safe" or "risky" for the loan application data; "yes" or "no" for the customer management data; or "therapy X," "therapy Y", or "therapy Z" for the medical data. These labels are discrete values, where the ordering among values has no meaning. For example, the values 0 and 1 may represent churn and non-churn, implying that there is no order among these churn statuses.

How does Classification Work?

Data classification is a two-step process: firstly, training data to develop a model; and secondly, applying the model developed.

Firstly, a classifier is developed in the first step, representing a predetermined data attribute set. This step is the training (i.e., learning) phase, involving a classification algorithm that develops the classifier by "learning" from training data set records and their associated target labels. A record, R, is represented by an n-dimensional

attribute vector, $R = (r_1, r_2, \ldots, r_n)$, expressing n measurements made on the record from n data set attributes (A), respectively, $A_1, A_2, \ldots, A_n$.

Each record, R, is assumed to belong to a predefined class as determined by another data set attribute called the target label attribute. For example, in our mini-case study, the target label attribute is churn, with either 'yes' or 'no' values. The target label attribute is discrete-valued and unordered. It is categorical in that each value serves as a category (a.k.a. class). Each record making up the training set is referred to as a training record and selected from the data set. In the context of classification, data records are also referred to as data points, samples, tuples, examples, objects, or instances.

Training to Develop a Model

Because the target label of each training record is supplied, the training step is also known as supervised learning, indicating the learning of the classifier is "supervised" because it is trained to which label each training record belongs. Therefore, this first step of the classification process can also be viewed as learning a function, y = f (R), that can predict the associated target label y of a given record R. We wish to learn a function that separates the data labels in this representation.

Typically, this function is represented in classification rules such as decision trees or mathematical formulae. In our mini-case study example, the function is represented as classification rules that identify a customer's status as churn or non-churn, as

diagrammatically shown in Figure 4.

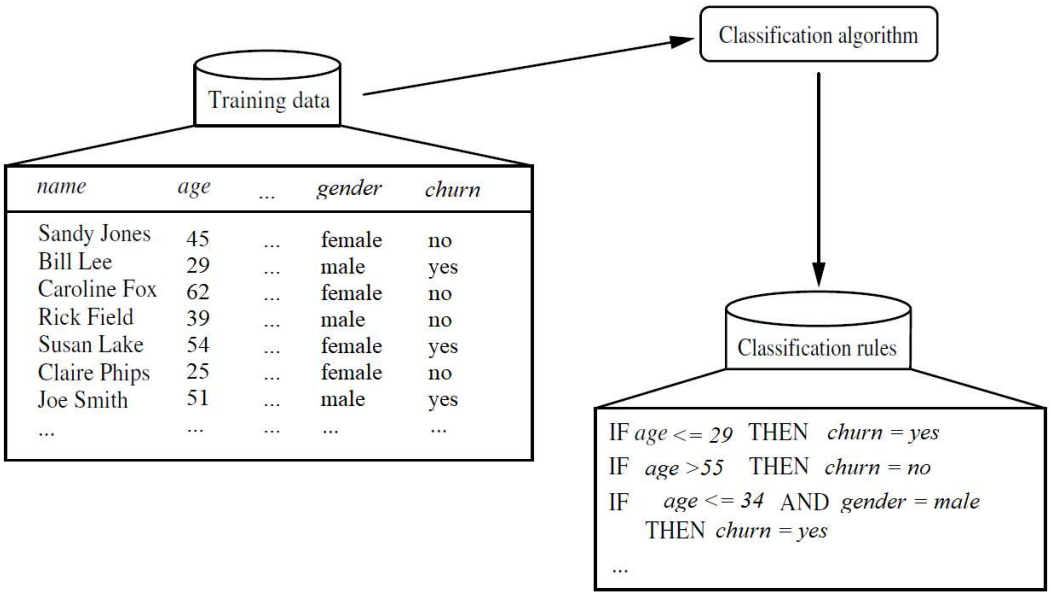


Figure 4 Training a Model in a Supervised Learning Process

Do take note that the data has been simplified for illustrative purposes. The rules can categorize future data records and give more profound insight into the data set contents.

Another way of generating classification IF-THEN rules is through a tree-like structure called a classification tree or a decision tree. Figure 5 shows a simplified decision tree corresponding to the customer churn classification data.

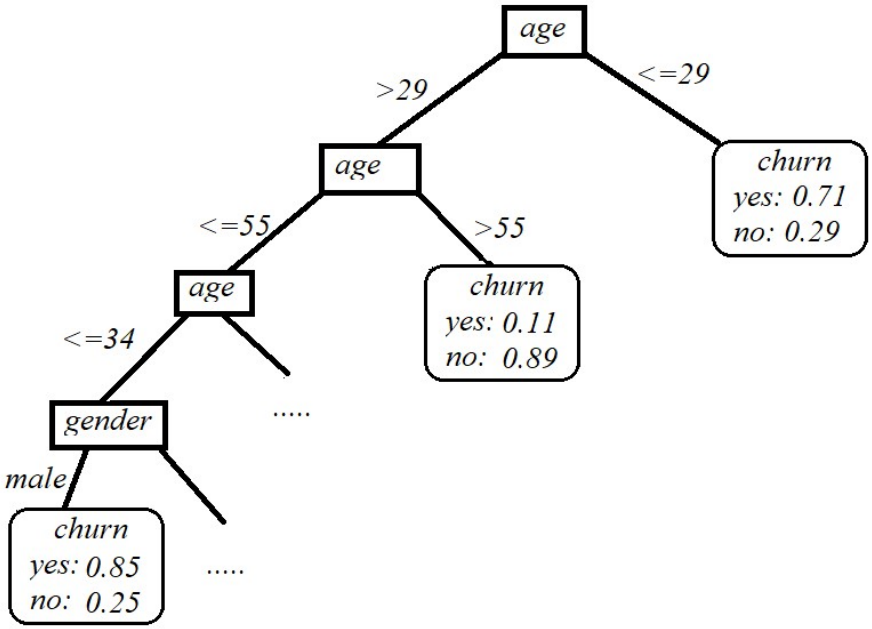


Figure 5 A Classification model in Tree Structure Representation

Applying the Model Developed

The second step uses the model for classification, as diagrammatically illustrated in Figure 6.

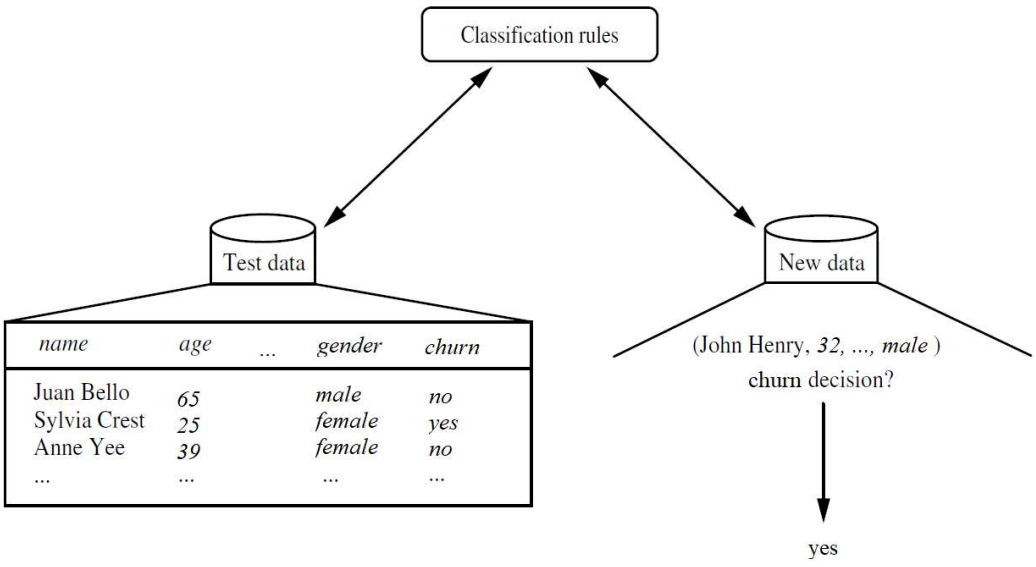


Figure 6 Applying the Model Developed to Classify Future Target Label

Firstly, the predictive accuracy of the classifier is assessed. If the training data set is used to measure the accuracy of the classifier, this assessment would likely be over-

optimistic because the classifier tends to overfit the data. It is because, during the learning process, it may contain some anomalies in the training data set that are not present in the overall data set. Therefore, a test set is utilized, made up of test records and associated target labels. These records are randomly selected from the overall data set. They are separated from the training records, meaning they are not used to develop the classifier.

The accuracy of a classifier on a given test data set is the percentage or rate of test set records correctly classified by the classifier. In addition, the associated target label of each test record is compared with the learned classifier's target prediction for that record.

Suppose the accuracy of a classifier is considered acceptable. In that case, the classifier can be used to classify future data records for which the target label is unknown. For example, the classification rules learned in Figure 4(a) from the training data set of the previous customer churns can be used to classify churn or non-churn for future customer profiles.

In Week 6, we will explore several methods to assess performance for model evaluation.

In the next few sections, we will learn the basics of several commonly used algorithms for Classification, such as Decision trees, Neural networks, and Ensemble learning methods. We will then use Python to implement the algorithms for solving our customer churn mini-case study's classification problem to demonstrate the data modeling concept.

There are hundreds of data modeling algorithms. We can only choose the commonly algorithms to demonstrate the concept of data modeling phase in this course.

We will also apply the Logistic Regression algorithm to implement the data modeling phase; however, the explanation of its basic concept is excluded in this topic because

students are expected to have learned the concepts in Statistics-related courses such as the Statistical Methods for Data Science course.

5.2.1 Basics of Decision Trees

Decision trees (a.k.a classification trees) are one of the most commonly used data modeling techniques. From a data mining practitioner's perspective, decision trees are easy to develop, and from a business user's viewpoint, the models developed are easy to interpret. As the name suggests, decision trees divide a data set into labels (or classes) belonging to the target variable(s).
Regardless of whether a decision tree has a binary (for example, 'yes' or 'no', 0 or 1) or multiple labels (for example, 'therapy X', 'therapy Y' or therapy Z'), decision trees are used when the target variable is categorical.

How it Works

A decision tree model takes a form of an inverted tree where an attribute is tested with a condition (for example, IF gender = female) in each node. At the end of the decision tree path is a leaf node where a prediction about the target variable is made based on conditions set forth by the decision path. Finally, the nodes split the data set into subsets. In a decision tree, the concept is to split the data set based on the homogeneity of data.

For example, assuming we have two variables, weight, income, and age, that predict if a person is probably to sign up for a gym membership. In our training data set, if we see that 90% of the people older than 35 signed up, we may split the overall data set into two partitions: one containing people older than 35 and the other having people under 35. The first partition is now '90% pure' from the label (or class) they belong to. First, however, we need a rigorous measure of impurity which meets specific criteria based on calculating a proportion of the data that belong to a label. These criteria are straightforward:

1. The impurity measure of a data set must be at a maximum when all possible labels are equally represented. In the membership example, in the initial overall data set,

if 50% of records belonged to the 'no' sign up and 50% of records belonged to the 'yes' sign up, then this non-partitioned initial overall data set would have a maximum (100%) impurity.

2. The measure of impurity of a data set must be zero when only there is only one label remains. For example, if we assemble a group of only those individuals who signed up (i.e., label='yes') for the membership, therefore only members labeled 'yes'), then this subset has a 100% purity (maximum) or 0% impurity (minimum).

The commonly used measures for purity/impurity are Entropy or the Gini index. The entropy refers to log(1/P) or –log P, where P is the probability of an event occurring. If the probability for all events is not identical, we need a weighted expression, and thus entropy, E, is adjusted as follows:

$$E = \sum p_l log_2(p_l)$$

Where *l* = 1, 2, 3, …, m denote the m labels of the target variable. The pl represents the proportion of records that belong to a label l. For our membership example, there are two labels: 'yes' for members and 'no' for non-members. If our data set contains records with 50% for each label, then the entropy of the data set will be:

$$E = -\left[(0.5log_2\ 0.5) + (0.5log_2\ 0.5)\right] = -\ log_2 0.5 = -\ (-1) = 1$$

However, if we split the data set into two sets of 50 records, each having all members (i.e., label = 'yes') and non-members (i.e., label = 'no', the entropy of either of these two partitioned sets is:

$$E = -\left[1log_2\ 1\right] = 0$$

Any other proportion of records within a data set will yield entropy values between 0 (minimum) and 1 (maximum). The Gini index (G) is similar to the entropy measure in its characteristics and its formula as follows:

$$E = \sum (1 - p_l{}^2)$$

The Gini value ranges between 0 and a maximum value of 0.5. Entropy or Gini formulations can be used to form partitions in the data.

To further elaborate on the concept, we take the classical well-known example of golf data set to apply the entropy notions for building a decision tree introduced by Quinlan [1]. The data set is shown in Table 1.

Table 1 Golf data set [adopted from [2]

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	FALSE	no
sunny	80	90	TRUE	no
overcast	83	78	FALSE	yes
rain	70	96	FALSE	yes
rain	68	80	FALSE	yes
rain	65	70	TRUE	no
overcast	64	65	TRUE	yes
sunny	72	95	FALSE	no
sunny	69	70	FALSE	yes
rain	75	80	FALSE	yes
sunny	75	70	TRUE	yes
overcast	72	90	TRUE	yes
overcast	81	75	FALSE	yes
rain	71	80	TRUE	no

Fundamentally, we need to answer two questions at each step of the tree-building process: where to split the data and when to stop splitting. Next, we will use the example golf data set to examine the considerations in answering these questions.

How Data Is Used to Build a Decision Tree

1. Where to split data?

The golf data set has fourteen records with four attributes: Temperature, Humidity, Wind, and Outlook. The target attribute represents whether or not to play cricket with two labels: 'yes' and 'no'. We will use the data set to understand how to build a decision tree.

The decision tree model-building process starts by partitioning the original data set on the four input attributes. Let us begin with the Outlook attribute. This attribute has three possible values: sunny, overcast, and rain. When it is overcast, there are four records where the result was target attribute Play = 'yes' for all four records, as presented in Table 2, so the ratio of records, in this case, is 100% or 1.0 in entropy value.

Table 2 Four records showing Outlook attribute containing overcast [2]

Row No.	Play	Outlook
3	yes	overcast
7	yes	overcast
12	yes	overcast
13	yes	overcast
4	yes	rain
5	yes	rain
6	no	rain
10	yes	rain
14	no	rain
1	no	sunny
2	no	sunny
8	no	sunny
9	yes	sunny
11	yes	sunny

Therefore if we split the data set here, the resultant four record partition will be 100% pure for Play = 'yes'. Mathematically, we can calculate the entropy using Eq. 4.1 for this partition is as follows:

$$E_{outlook=overcast} = -\left(\frac{0}{4}log_2\left(\frac{0}{4}\right) - \frac{4}{4}log_2\left(\frac{4}{4}\right)\right) = 0.0$$

Similarly, we can compute the entropy in the other two cases for Outlook:

$$E_{outlook=sunny} = -\left(\frac{2}{5}log_2\left(\frac{2}{5}\right) - \frac{3}{5}log_2\left(\frac{3}{5}\right)\right) = 0.971$$

$$E_{outlook=rain} = -\left(\frac{3}{5}log_2\left(\frac{3}{5}\right) - \frac{2}{5}log_2\left(\frac{2}{5}\right)\right) = 0.971$$

For the whole attribute, the total “information” (I) is calculated as the weighted sum of these component entropies, including considering the proportion for each possible value. There are four records of Outlook = overcast; therefore, the ratio or proportion for overcast is:

$$P_{outlook=overcast} = \frac{4}{14}$$

The other ratios for each (i.e., for Outlook = sunny and rain) are 5/14:

$$I_{outlook} = (P_{outlook=overcast} \times E_{outlook=overcast}) + (P_{outlook=sunny} \times E_{outlook=sunny}) + (P_{outlook=rain} \times E_{outlook=rain})$$

Thus, we derive the following:

$$I_{outlook} = \left(\frac{4}{14} \times 0\right) + \left(\frac{5}{14} \times 0.971\right) + \left(\frac{5}{14} \times 0.971\right) = 0.693$$

If we do not partition the data along with the three values for the Outlook attribute, the total information would be merely the weighted average of the respective entropies for the two labels with overall ratios of 5/14 (Play = 'no') and 9/14 (Play = 'yes'):

$$I_{outlook(no\ partition)} = -\left(\frac{5}{14}\right) log_2 \left(\frac{5}{14}\right) - \left(\frac{9}{14}\right) log_2 \left(\frac{9}{14}\right) = 0.940$$

By doing these splits, we decreased entropy and gained some information. This gain is known as information gain. For the Outlook scenario, it will be:

$$I_{outlook(no\ partition)} - I_{outlook} = 0.940 - 0.693 - 0.247$$

Using the same logic, we can now calculate and derive information gain values for the other three attributes, as follows: *Temperature = 0.029; Humidity = 0.102*; *Wind = 0.048*; and *Outlook = 0.247*.

For numeric variables, possible split points to examine are essentially averages of available values. For example, the first potential split point for the Humidity attribute could be average [65,70], which is 67.5; the next potential split point could be average [70,75], which is 72.5, and so on.

We use similar logic for the other numeric attribute, Temperature. The algorithm computes the information gain at each potential split point and chooses the one that maximizes it. Another way to tackle this would be to discretize or group the numerical ranges; for example, Temperature >=80 could be considered as 'Hot', between 70 to 79 as 'Mild', and less than 70 as 'Cool'.

Based on the information gain derived for all attributes, i.e., Temperature (0.029), Humidity (0.102), Wind (0.048), and Outlook (0.247), It is obvious that if we split the data set into three sets along with the three values of Outlook, we will experience the highest information gain. This split gives the first node of the decision tree, as shown in Figure 7.

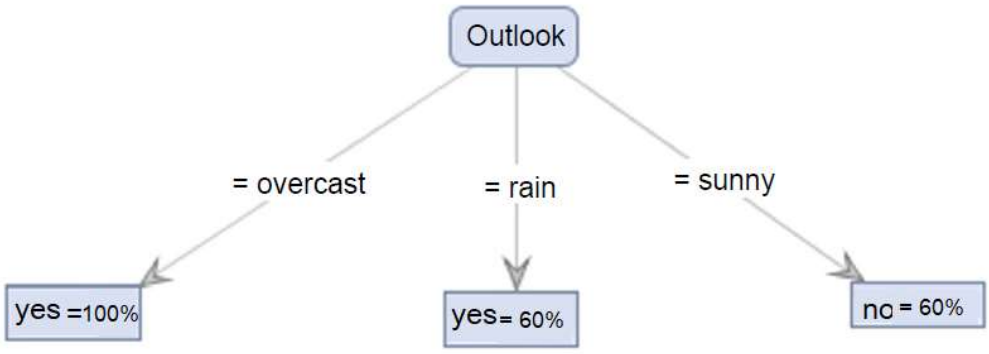


Figure 7 Outlook attribute split yields three branches

As noted earlier and in Table 2, the leaf node for the attribute Outlook = 'overcast' branch consists of four records. These four records belong to the label Play = 'yes', implying a maximum purity (5/5=100%). The other two branches contain a mix of labels. For example, the Outlook = 'rain' branch has three (out of five) 'yes' results (3/5 = 60%), and the Outlook = 'sunny' branch has three (out of five) 'no' results (3/5=60%).

Accordingly, not all the final partitions are 100% homogenous. This finding indicates that we could apply the same process for each subset until we get "purer" outcomes. So we revert to the first question: where to split the data? Fortunately, we already answered this when calculating the information gain for all attributes. Therefore, we use the other attributes that yielded the highest gains. Following the reasoning, we can split the Outlook = 'sunny' branch along with Humidity (which generated the second-highest information gain) and the Outlook = rain branch along with Wind (which yielded the third-highest gain). The fully grown decision tree is shown in Figure 8.

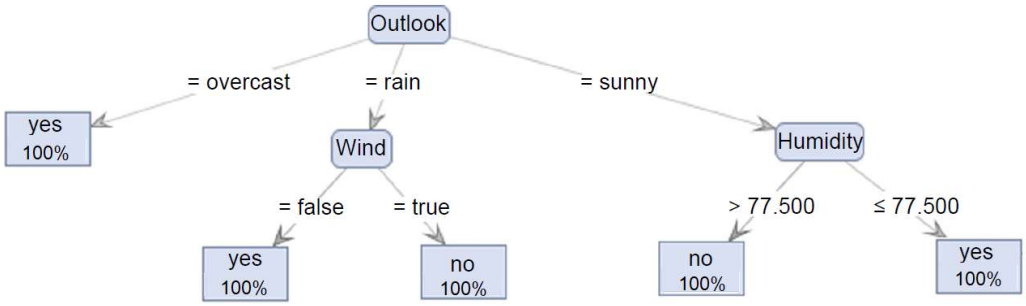


Figure 8 The fully grown decision tree.

2. Pruning a Decision Tree: When to Stop Splitting Data?

In real-world data, it is improbable that we will get leaf nodes that are 100% homogeneous, i.e., 100% purity, as we just witnessed for the golf data set. In this case, we will need to guide or train the algorithm on when to stop growing the tree. There are several situations where we can terminate the growing process:

- No attribute satisfies a minimum information gain threshold (as demonstrated in the example of deriving the information gain of all attributes in the golf data set).
- When the process has reached a maximum depth: as the tree grows more complex (i.e., larger), we likely run into an "overfitting" situation that makes the model cannot generalize well in prediction.
- There are less than a certain number of records in the current subtree: a mechanism of applying a minimum record numbers threshold to prevent overfitting.

What is Overfitting?

Overfitting happens when a model tries to remember or memorize the training data rather than generalizing the association between inputs and output attributes. Overfitting usually performs very well on the training data set but poorly on any new data previously unknown by the model. It likely provides a useless model for unseen data because the model cannot generalize well in prediction.

To prevent overfitting, we may need to limit tree growth or downsize it using a pruning process. All three stopping approaches based on the abovementioned situations form the pre-pruning of the decision tree because the pruning happens before or during the tree's growth.

Some methods will not restrict the number of tree branches, let the tree grow as deep as the data permits, and then prune those branches that do not effectively change the classification accuracy. This approach is known as post-pruning. Post-pruning may be a better option because we will not overlook any small but potentially significant associations between attribute values and labels if we let the tree reach its maximum depth. However, one disadvantage of post-pruning is

that it demands additional computations, potentially causing resource waste when we need to prune the tree again to adjust the classification performance.

In the next section, we will implement Classification using the decision tree algorithm in Python.

References:
[1] Quinlan, J. (1986). Induction of Decision Trees. Machine Learning, 81–106.
[2] Kotu, V. and Deshpande, B., 2014. Predictive analytics and data mining: concepts and practice with rapidminer. Morgan Kaufmann.

5.2.2 Decision Trees Implementation

To implement the decision trees algorithm, we use the mini-case study customer churn dataset that we have previously pre-processed and saved in the ChurnFinal.csv file. We use the following Python codes to import necessary libraries:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

df = pd.read_csv('ChurnFinal.csv')
df_inputs = pd.get_dummies(df[['Gender', 'Age', 'PostalCode', 'Cash', 'CreditCard',
            'Cheque','SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
df_label = df['Churn']
```

Based on our problem and data understanding for the customer churn case study (refer to Week 2 content), the solution requires an approach to classify if a customer is going to churn. Thus, we indicate the target attribute as Churn in the data set; we use Gender, Age, PostalCode, Cash, CreditCard, Cheque, SinceLastTrx, SqrtTotal, SqrtMax, and SqrtMin attributes as inputs to predict the churn by assigning them to user-defined variables df_label and df_inputs respectively for later use.

We use the get_dummies() function to convert categorical to indicator attributes(s) to avoid incompatibility of data modeling methods that cannot handle categorical

attributes, for example, Regression and Neural Network algorithms (we will apply these techniques in later sections).

In the above codes, we split 80% of the data set for training the model and 20% (i.e., test_size=0.2) as a test set to assess the model. The random state (random_state) is a seed to the random number generator to ensure numbers are generated in the same order.

Next, we spit the original data set to train and test the model, using the Python train_test_split () function with the following example codes:

```
X_train, X_test, Y_train, Y_test = train_test_split(df_inputs, df_label,
            stratify=df_label, test_size=0.2, random_state=1)
```

In the above codes, we split 80% of the data set for training the model and 20% (i.e., test_size=0.2) as a test set to assess the model. The random state (random_state) is a seed to the random number generator to ensure numbers are generated in the same order.

For a detailed explanation of the train_test_split() API parameters, refer to the official website https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?highlight=train_test_split#sklearn.model_selection.train_test_split.

After splitting the data set into train and test sets, we define the decision tree as the data modeling technique using DecisionTreeClassifier(), and train the decision tree using fit() function the train data set (X_train and Y_train) in function:

```
min_sample = int(len(df) * 0.1)
dtree = DecisionTreeClassifier(criterion = 'entropy', splitter="best", max_depth=7,
            min_samples_leaf=5, min_samples_split=min_sample, random_state=42)
dtree.fit(X_train, Y_train)
```

To assess how well the decision tree model developed based on the criteria above, we use the test data (X_test) for the model to predict the churn outcomes using the predict() function.

```
#Predict the response for test dataset
y_predict = dtree.predict(X_test)
print("Model Accuracy     : ", metrics.accuracy_score(Y_test, y_predict))
```

To assess the how well the classification outcomes, we can derive the model accuracy with the accuracy_score() function by comparing the predicted (i.e., y_predict) and actual (i.e., y_test) churn outcomes.

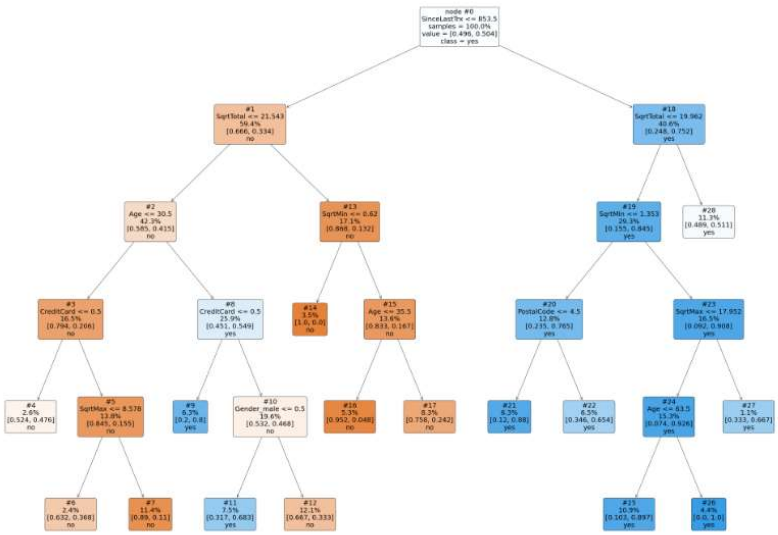
After running all the codes together given above, we obtain a model accuracy of 0.78 printed on the console terminal as Model Accuracy : 0.78. This result indicates that the model has an accuracy of 78%, implying that 78% of the time, it correctly classifies churn outcomes ('yes' or 'no').

Besides accuracy, we will examine more measures of model performance in Week 6.

To view the generated decision tree structure, we can plot the tree using the following Python codes:

```
#plot tree
target = list(df['Churn'].unique())
feature_names = list(df_inputs.columns)
from sklearn import tree
from matplotlib import pyplot as plt
import os
plt.clf()
strFile = "plot_dtree.png"
if os.path.isfile(strFile):
   os.remove(strFile)
fig = plt.figure(figsize=(50,37))
_ = tree.plot_tree(dtree, feature_names=feature_names, class_names=target, filled=True,
        label='root', node_ids=True, proportion=True,rounded=True, impurity=False)#,
fontsize=20)
fig.savefig(strFile)
```

The above codes plot the tree and save it in the plot_dtree.png file. After running the codes, the generated model tree structure view is as follows:



To interpret the nodes, take the example of node #26. The predicted churn is 'yes', with 100% purity, i.e., 1.0 for 'yes' and 0.0 for 'no'.

For a detailed explanation of the plot_tree() API parameters, refer to the official website https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html?highlight=plot_tree#sklearn.tree.plot_tree.

5.2.3 Basics of Artificial Neural Networks

The artificial neural network (ANN) method approaches prediction problems by creating a mathematical explanation that closely imitates the biological function of a neuron. Consider the simple linear mathematical model:

$$\mathrm{Y} = 1.5 + 2\mathrm{X}_1 + 3\mathrm{X}_2 + 4\mathrm{X}_3$$

Y is the calculated target attribute as output, and X_1, X_2 and X_3 are input attributes. The number 1.5 is the intercept, and 2, 3, and 4 are the coefficients. A coefficient is a scaling factor as a linear transformation that adds a constant and multiplies its responding input attribute value, such as the X_1, X_2 and X_3 input attribute values. This model representation is somehow comparable to Regression methods. Figure 9 (4.38) presents this simple model in a topological form.

In the topology shown in Figure 9, X_1 is the input attribute value passes through a node, symbolized by a circle. Then the X_1 value is multiplied by its weight, which is 2, as remarked in the connector. Correspondingly, all other input attributes (X_2 and X_3) pass through a node and scaling transformation. The last node is a different case with no input variable; it merely has the intercept. Finally, all the connectors' values are calculated and summarized in an output node that generates the predicted output Y. The topology also represents an elementary artificial neural network (ANN) of a simple linear model, as shown in Eq. 1 in this example.

The ANNs shape more complex nonlinear data relationships by learning through adaptive weights adjustments between the nodes. The ANN is a mathematical model motivated by the biological nervous system; therefore, we can observe some biological terms used in its concept.

In ANN terminology, nodes are named units. The input layer is the first layer of nodes nearest to the input. Likewise, The last layer of nodes is named the output layer. The output layer executes an activation function with the summarization and transfer function that scales the output into the expected range of the output value.

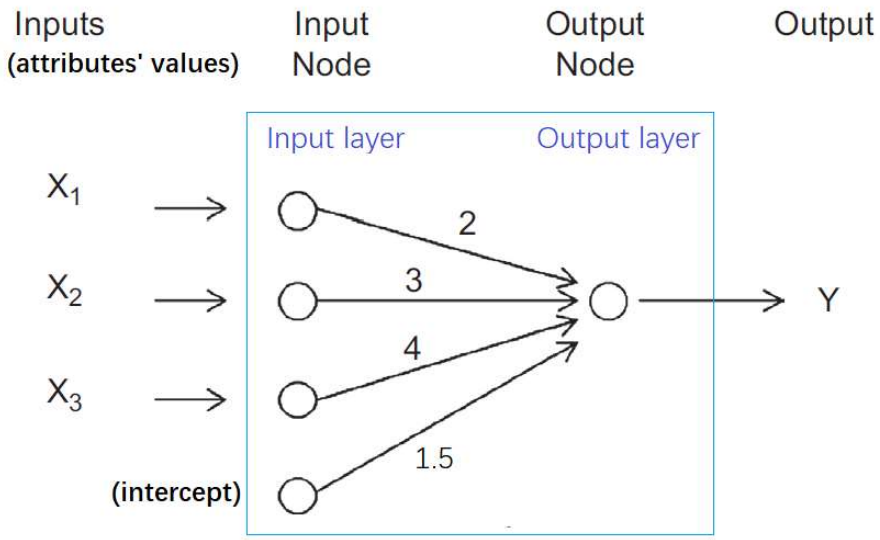


Figure 9 An example of an ANN structure with a two-layer topology

As shown in Figure 9, this two-layer topology with one input and one output layer is called a perceptron, the most simplistic structure of an artificial neural network. A

perceptron is a feedforward neural network where the input moves in one direction, and there are no loops in the topology.

An ANN k is usually employed to model nonlinear relationships between input and output attributes. This modeling is realized by having more than one layer in the topology, apart from the hidden layer consisting of input and output layers, connecting input from previous layers and applying an activation function. The output is eventually computed by a complex combination of input values, as shown in Figure 10.

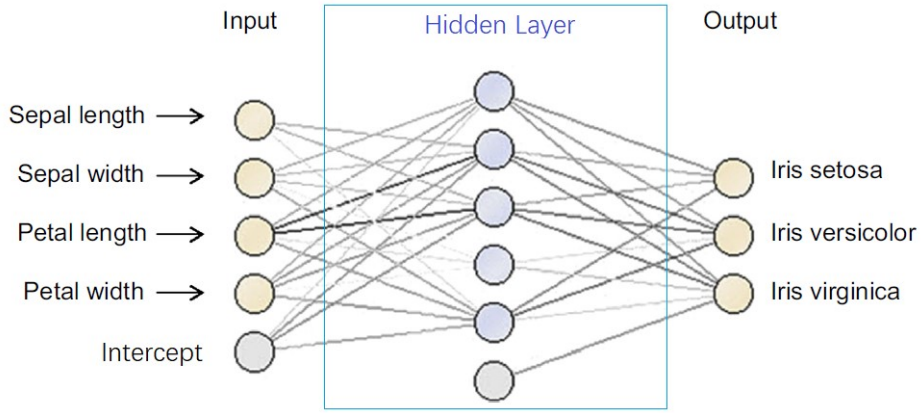


Figure 10 The Iris data set example ANN topology.

To practically explain how the ANN works, let us consider the Iris data set's simple example. Recall that this data set has four input attributes: sepal length, sepal width, petal length, and petal width. The target attribute has three labels: Iris setosa, Iris versicolor, and Iris virginica. Using the ANN method, the Iris data set produces a three-layer structure with three output nodes, one for each label predicting species as a categorical data type for Iris. Note that users can specify the number of layers during the implementation.

A predicted label is determined based on the maximum value of the output label. The topology in Figure 10 is a feedforward ANN with one hidden layer. Depending on the problem to solve, we can utilize a multiple hidden layers topology with looping where the output of one layer will be the input for preceding layers. However, it is

time-consuming to build an approximating model because defining the topology settings is challenging in neural network modeling.

The output node's activation function is a transformation function, usually consisting of summarization and a transfer function. The transfer functions range from a sigmoid to a bell curve, linear, or logistic. Because of the transformation function and the presence of multiple hidden layers, we can use the ANN to closely approximate nearly any mathematical continuous data association between input and output attributes. However, searching for an optimal solution is relatively time-consuming with multiple user settings, such as the topology, transfer function, and the number of hidden layers.

How an ANN Works

An ANN learns the data relationship between input attributes and the output labels via a back propagation technique. Provided a network topology and activation function, the critical training task is to identify the weights of each link between the nodes. The process is relatively intuitive and mimics the signal transmission in biological neurons.

The ANN model utilizes every training record to calculate the error of the predicted output compared to the actual output of the target attribute value. The model then uses the error to adjust the weights to minimize the error for the next training record and repeats this step until the error drops within the acceptable range. Following are the core stages in creating an ANN from a training data set.

- Stage 1: Deciding the Topology and Activation Function For an instant, assume a data set with three numeric input attributes (X_1, X_2, X_3) and one numeric output (Y). We use a topology with two layers and a simplified activation function to model the data relationships, as presented in Figure 11. We do not use any transfer function for this example.

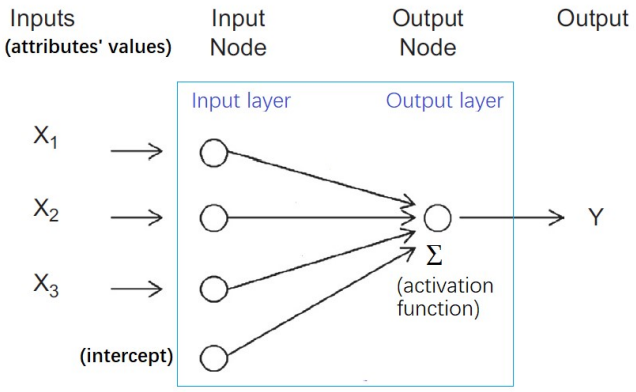


Figure 11 An ANN topology with two layers and a simplified activation function.

- Stage 2: Initiating Weight Assume the initial weights for the four links are 1.5, 2, 3, and 4. Usually, the initial weights are assigned randomly by data mining tools according to the random seed number default in the systems. We take an example model and a test record with all the inputs as number 1 and the known (i.e., actual value) output as 15. Consequently, $X_1 = X_2 = X_3 = 1$ and output Y = 15. Figure 12 shows the initiation of the first training record.

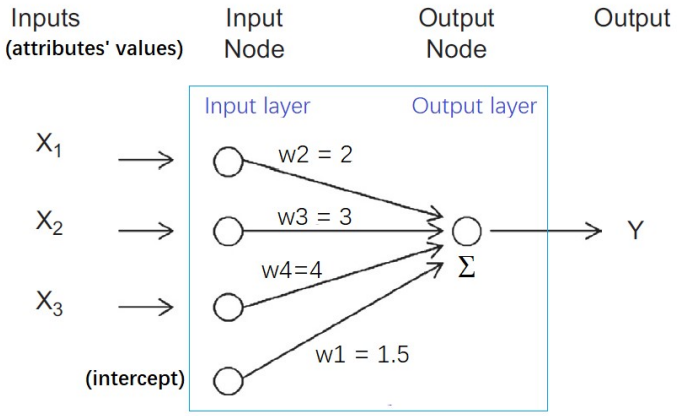


Figure 12 Initiating weights for the first training record.

- Stage 3: Calculating Error We can compute the predicted output of the record from Figure 12. This calculation is a straightforward feedforward process of passing through the input attributes' values and computing the predicted target output. The predicted output Y of the present model is 1.5 + 1 * 2 + 1 * 3 + 1 * 4 = 10.5. The difference between the actual output from the training record and the predicted output is the error: e = Y − y For the example of the training record, the error is 15 – 10.5 = 4.5.

- Stage 4: Adjusting Weight Weight adjustment is the most crucial part of learning in an ANN. The error computed in the previous stage is passed backward from the output node to all other nodes in a reverse direction. The weight of each link is adjusted from its previous value by a fraction of the error. The fraction λ applied to the error is called the learning rate, with a value ranging from 0 to 1. The new weight of the link (w) is the sum of the previous weight (w') and the product of the learning rate and ratio of the error (λ * e). That is, w = w ' + λ * e. The selection of λ is challenging in an ANN implementation. Therefore, some model processes start with λ close to 1 and decrease the λ value while training each cycle. With this approach, any outlier records likely causing errors later in the training cycle will not degrade the quality of the model. Figure 13 demonstrates the topology with its error propagation. The current weight of the first link is w2 = 2. Assume the learning rate is 0.05. The new weight will be w2 = 2 + 0.05 * 4.5/3 = 2.075. The error is divided by the number 3 because the error is back-propagated to three links from the output node. Likewise, the weight of each link will be adjusted. A new error will be calculated for the next training record in the next cycle. This cycle continues until the iterative process executes all the training records. The same training record can be run repeatedly in the subsequent cycles until the error rate is lower than a threshold.

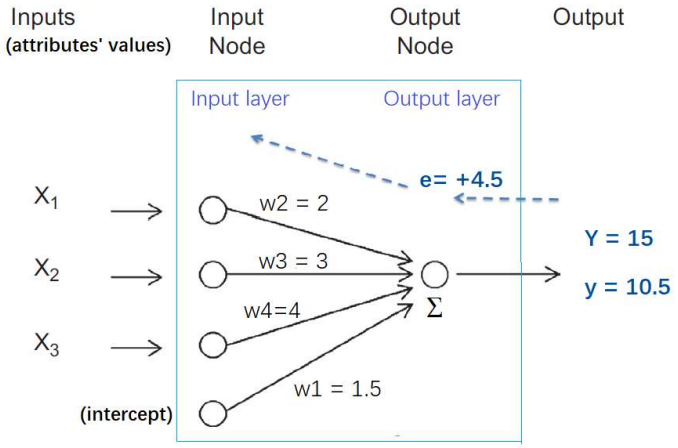


Figure 13 An ANN topology with error back propagation.

In real-world cases, there will be multiple hidden layers and multiple output links, one for each categorical label. However, an ANN model works well with numeric inputs and outputs because it represents a mathematical model. Suppose the input

contains a categorical attribute, then a preprocessing phase is required to convert the categorical attribute into multiple numeric attributes, i.e., a dummy variable creation method for each attribute value. Therefore in the case of nominal attributes, this preprocessing phase expands the number of input links for a neural network and accordingly demands more computing resources. Thus, an ANN is more suitable for numeric attributes, although it can also support classification problems with categorical target attributes.

5.2.4 Neural Networks Implementation

To implement the neural networks algorithm, we use the mini-case study customer churn dataset that we have previously pre-processed and saved in the ChurnFinal.csv file. We use the following Python codes to import necessary libraries:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd

#Loading Dataset
data = pd.read_csv('ChurnFinal.csv')

df_inputs = pd.get_dummies(data[['Gender', 'Age', 'PostalCode', 'Cash', 'CreditCard',
        'Cheque', 'SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
df_label = data['Churn']
```

Based on our problem and data understanding for the customer churn case study (refer to Week 2 content), the solution requires an approach to classify if a customer is going to churn. Thus, we indicate the target attribute as Churn in the data set. We use Gender, Age, PostalCode, Cash, CreditCard, Cheque, SinceLastTrx, SqrtTotal, SqrtMax, and SqrtMin attributes as inputs to predict the churn by assigning them to user-defined variables df_label and df_inputs respectively for later use.

Next, we spit the original data set to train and test the model, using the Python train_test_split () function with the following example codes:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(df_inputs, df_label,
                    stratify=df_label, test_size=0.2, random_state=1)
```

In the above codes, we split 80% of the data set for training the model and 20% (i.e., test_size=0.2) as a test set to assess the model. The random state (random_state) is a seed to the random number generator to ensure numbers are generated in the same order.

For a detailed explanation of the train_test_split() API parameters, refer to the official website https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html?highlight=train_test_split#sklearn.model_selection.train_test_split.

After splitting the data set into train and test sets, we define the neural network algorithm as the data modeling technique using MLPClassifier(), and train the decision tree using fit() function the train data set (X_train and Y_train) in function:

```
# feature scaling
from sklearn.preprocessing import StandardScaler

# standardizes the data values into a standard format
scaler = StandardScaler()
scaler.fit(X_train)                    #fit only on training data
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)      # apply same transformation to test data

from sklearn.neural_network import MLPClassifier

# define the neural network algorithm
nn = MLPClassifier(solver='lbfgs', alpha=1e-05 , activation = 'identity',
random_state=1,
             hidden_layer_sizes=(200,6), learning_rate = 'adaptive',
             learning_rate_init=0.0001, max_iter=500)

# train the decision tree
nn.fit(X_train, Y_train)
```

In the above code, StandardScaler() function standardizes the data values into a standard format. StandardScaler standardizes an attribute by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation. Further, we use fit_transform() along with the assigned object to transform and standardize the data. StandardScaler() further removes outliers (if they still exist in the data set after we pre-processed the data) since it involves the estimation of the empirical mean and standard deviation of each attribute.

For a detailed explanation of the MLPClassifier() API parameters, refer to the official website https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

To assess how well the decision tree model developed based on the criteria above, we use the test data (X_test) for the model to predict the churn outcomes using the predict() function.

```
#Predict the response for test dataset
y_predict = nn.predict(X_test)

from sklearn import metrics
print("Model Accuracy     : ", round(metrics.accuracy_score(Y_test, y_predict),4))
```

To assess the how well the classification outcomes, we can derive the model accuracy with the accuracy_score() function by comparing the predicted (i.e., y_predict) and actual (i.e., Y_test) churn outcomes.

After running all the codes together given above, we obtain a model accuracy of 0.7023 printed on the console terminal as Moddel Accuracy : 0.7023. This result indicates that the model has an accuracy of 70.23%, implying that 70.23% of the time it correctly classifies churn outcomes ('yes' or 'no').

Besides accuracy, we will examine more measures of model performance in Week 6.

5.2.5 Logistic Regression Implementation

Regression trees are comparable in function to classification trees. They may also be used for numeric estimation problems when the target variable is numeric or continuous: for example, predicting the price of a product based on several input variables.

The commonly used regression approach for classification problems is the Logistic Regression. Remember that the predictors or input variables may be categorical or numeric in either case. It is the target variable that determines the type of prediction required.

We will not examine the concept of Regression again in this course because students are expected to have learned Regression algorithms concepts in Statistics-related courses such as the Statistical Methods for Data Science course.

To demonstrate how we can use Python to build a classifier model using the Regression approach with the LogisticRegression() function, the following codes show the example of the implementation for the customer churn data set.

```
#Importing necessary Libraries
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd

#Loading Dataset
data = pd.read_csv('ChurnFinal.csv')

#Generating Matrix of Features
df_inputs = pd.get_dummies(data[['Gender', 'Age', 'PostalCode', 'Cash',
        'CreditCard', 'Cheque', 'SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
df_label = data['Churn']

#Splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(df_inputs, df_label,
                stratify=df_label, test_size=0.3, random_state=7)

# scaling to remove potential outliers
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression
lg = LogisticRegression(solver='liblinear', random_state=7, max_iter=300)
lg.fit(X_train, Y_train)

# obtain the model intercept and coefficient of each input attribute
print(f"intercept: {np.round(lg.intercept_,4)}")
fieldList = np.array(list(df_inputs)).reshape(-1,1)
coeffs = np.reshape(np.round(lg.coef_,2),(-1,1))
coeffs=np.concatenate((fieldList,coeffs),axis=1)
print(pd.DataFrame(coeffs,columns=['Attribute','Coefficient']))

# fit the model to compare predicted and actual target output values
y_predict = lg.predict(X_test)

# assess the model permance
```

```
from sklearn import metrics
print("Model Accuracy     : ", round(metrics.accuracy_score(Y_test, y_predict),4))
```

Observe the model accuracy of 0.7626 printed on the console terminal as Moddel Accuracy : 0.7625. This result indicates that the model has an accuracy of 76.25%, implying that 76.25% of the time it correctly classifies churn outcomes ('yes' or 'no').

We also observe the output of intercept: [0.03] with the Coefficient (in 2 decimal points) of each input attribute as follows: Age = 0.59, PostalCode = -0.01, Cash = 0.05, CreditCard = -0.40, Cheque = -0.14, SinceLastTrx = 0.69, SqrtTotal = -0.25, SqrtMax = -0.19, SqrtMin = 0.12, and Gender_male = -0.29. Thus, the generated Logistic Regression model is mathematically represented as follows:
Churn = 0.03 + 0.59(Age) - 0.01(PostalCode) + 0.05(Cash) - 0.40(CreditCard) - 0.14(Cheque)+ 0.69(SinceLastTrx) - 0.25(SqrtTotal) - 0.19(SqrtMax) + 0.12(SqrtMin) - 0.29(Gender_male)

For a detailed explanation of the LogisticRegression() API parameters, refer to the official website, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logisticregression#sklearn.linear_model.LogisticRegression

5.2.6 Basics of Ensemble Learning

Ensemble modeling is a process in which multiple diverse models are developed to predict an outcome using different modeling algorithms or training data sets. The ensemble model then aggregates each base model's prediction and results in the final prediction for the unknown data.

The reason for using ensemble models is to decrease the generalization error of the prediction. In machine learning and statistical learning, generalization error indicates how accurately an algorithm can predict outcome values for previously unseen data. The lower the generalisation error, the better a model performs in prediction.

Therefore, as long as the base models are diverse and independent, the model's prediction error reduces when the ensemble approach is employed.

The ensemble approach aims at the 'wisdom' of crowds in performing a prediction. Even though the ensemble model has multiple base models within the model, it works as a single model.

In the next section, we will use the Python Multi-layer Perceptron classifier to demonstrate the implementation of neural networks.

Different Types of Ensemble Learning Methods

Although there are several Ensemble learning methods, the following briefly gives an overview of the most-used techniques.

1. Bagging-based Ensemble Learning

    Bagging-based learning is a sampling technique in which we select "n" records from a data set of "n" records. However, the choice is entirely random, i.e., each record can be selected from the original data set, so each record is equally probable to be selected in each iteration of the process.

    After forming the samples, separate models are trained with the bagged samples. The samples are sourced from the training set in practical experiments, and the sub-models are assessed using the testing set. Finally, the final output prediction is combined across the sub-model predictions. Figure 15 demonstrates how a bagging-based ensemble model is built.

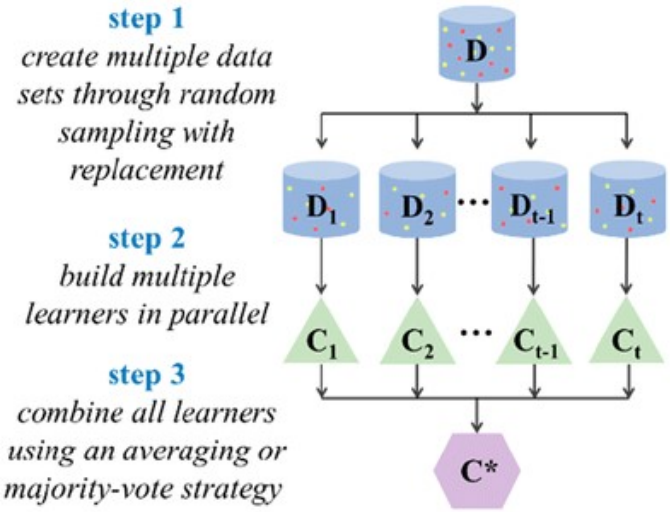


Figure 15 Example of a bagging-based learning model (adopted from [1])

2. Boosting-based Ensemble Learning

Boosting is a form of sequential learning technique. The algorithm works by training a model with the entire training set, and subsequent models are constructed by fitting the error values of the initial model. With this approach, Boosting-based learning tries to give higher weight to those records that the previous model poorly estimated.

After generating the sequence of the models, the model's predictions are weighted based on their accuracy scores, and the outcomes are combined to create a final computation. The XGBoost (Extreme Gradient Boosting), GBM (Gradient Boosting Machine), and AdaBoost (Adaptive Boosting) are the most commonly used algorithms used in Boosting technique. Figure 16 diagrammatically elaborates on the building process of a bagging-based ensemble model.

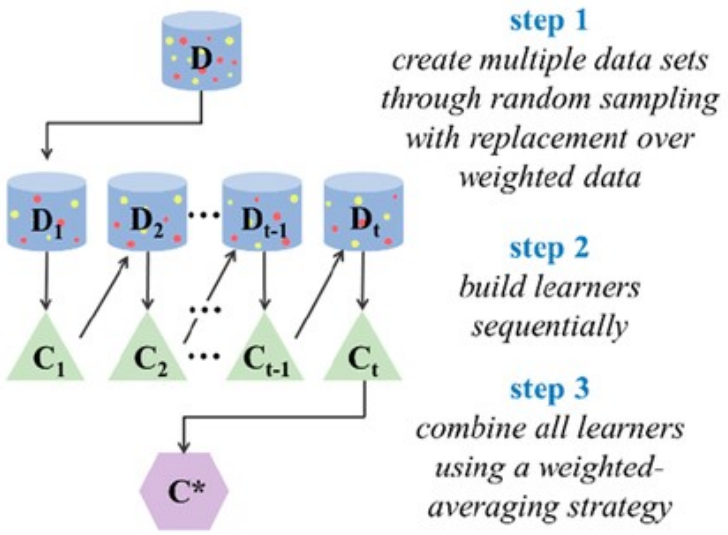


Figure 16 Example of a boosting-based learning model (adopted from [1])

3. Voting-based Ensemble Learning

The voting-based learning method starts with constructing two or more different models using the same data set separately. Then a Voting based Ensemble model can be used to enfold the constructed models (i.e., sub-models) and combine the models' predictions.

After creating the voting-based learning model, the ensemble model can be applied to predict new data (i.e., test data or future unknown data). Finally, weights are assigned to the predictions made by the sub-models. Figure 17 shows how a voting-based ensemble model is created.

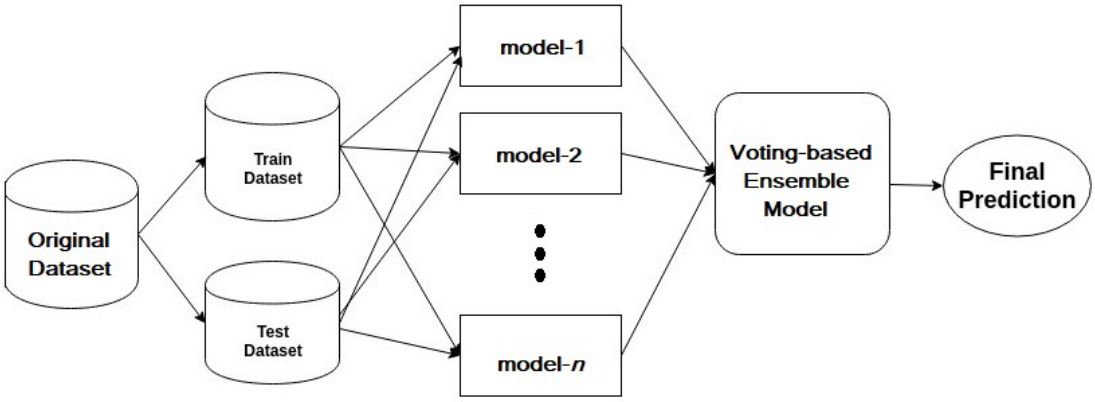


Figure 17 Example of a voting-based learning model.

NOTE: It is beyond the scope of this course for a detailed explanation of the Ensemble learning methods. However, for optional reading, students can refer to the official Python API website https://scikit-learn.org/stable/modules/ensemble.html?highlight=xgboost# for further details of implementation.

References:

[1] Yang, X., Wang, Y., Byrne, R., Schneider, G. and Yang, S., 2019. Concepts of artificial intelligence for computer-assisted drug discovery. Chemical reviews, 119(18), pp.10520-10594.

5.2.7 Ensemble Learning Implementation

Ensemble modeling is a process in which multiple diverse models are developed to predict an outcome using different modeling algorithms or training data sets. The ensemble model then aggregates each base model's prediction and results in the final prediction for the unknown data.

A brief overview of different Ensemble modeling is given in Section 5.2.6. In the following, we will use the different ensemble learning methods to solve the customer churn classification problem using Python for the implementation.

For all the available Ensemble methods that can be implemented in Python and their APIs, students can refer to https://scikit-learn.org/stable/modules/ensemble.html?highlight=xgboost#

The following Python codes show the implementation of Bagging-based, Boosting-based, and Voting-based Ensemble models. The comments embedded in the codes give explanations to guide the rationale of the programming logic.

```
#import necessary libraries
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np

#Loading Dataset and specify inputs/output
data = pd.read_csv('ChurnFinal.csv')
X = pd.get_dummies(data[['Gender', 'Age', 'PostalCode', 'Cash', 'CreditCard',
        'Cheque', 'SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
```

```
        Y = data['Churn']

        #Splitting dataset into training and testing dataset
        from sklearn.model_selection import train_test_split
        X_train,X_test,Y_train,Y_test = train_test_split(X, Y, stratify=Y,
                test_size=0.2, random_state=1)

        # feature scaling
        from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()
        scaler.fit(X_train)
        X_train = scaler.transform(X_train)
        X_test = scaler.transform(X_test)       #

        # Bagged-based Ensenble using Decision Trees for classification
        from sklearn import model_selection
        from sklearn.ensemble import BaggingClassifier
        from sklearn.tree import DecisionTreeClassifier

        cart = DecisionTreeClassifier()
        num_trees = 100

        # create the sub models using multiple Decision Trees, e.g. 100 trees
        es_Bag = BaggingClassifier(estimator=cart, n_estimators=num_trees,
random_state=7)

        # Boosting-based Ensemble using AdaBoost Classification wiht DEcision Trees
        from sklearn.ensemble import AdaBoostClassifier
        num_trees = 100
        # create the sub models using multiple Decision Trees, e.g. 100 trees
        es_Boost = AdaBoostClassifier(n_estimators=num_trees, random_state=7)

        # Voting-based Ensemble for Classification using variouus methods
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.svm import SVC
        from sklearn.neural_network import MLPClassifier
        from sklearn.ensemble import VotingClassifier

        # create the sub models using LogisticRegression, DecisionTree, and
SupportVector
        estimators = []
        model1 = LogisticRegression(random_state=7)  # Logistic Regression Classifier
        estimators.append(('logistic', model1))
        model2 = DecisionTreeClassifier(random_state=7) # Decision Tree Classifier
        estimators.append(('cart', model2))
        model3 = SVC(random_state=7) #Support Vector Classifier
        estimators.append(('svc', model3))

        # create the ensemble model
        es_Vote = VotingClassifier(estimators)
```

```
# Model performance
from sklearn import metrics
# use for-loop to print the performance of each ensemble model
for model in (es_Bag, es_Boost, es_Vote):
        model.fit(X_train, Y_train)
        y_predict = model.predict(X_test)
        print(model.__class__.__name__, ' Ensemble accuracy: ',
                metrics.accuracy_score(Y_test, y_predict))
```

After running the above codes, we can observe the following results printed:

```
BaggingClassifier Ensemble accuracy: 0.74
AdaBoostClassifier Ensemble accuracy: 0.735
VotingClassifier Ensemble accuracy: 0.73
```

These results indicate that with the parameter settings defined in the codes, Bagging-based Ensemble performs slightly better with an accuracy of 0.74 compared to Boosting-based and Voting-based learning methods using the customer churn data set.

5.3 Estimation Techniques using Python

Estimators (i.e., data modeling techniques for Estimation) take a list of input attributes and assign a numeric value to the target class demonstrating these attributes. Suppose the marketing manager wants to estimate the likelihood that a given customer will churn based on his/her demographic and transaction profiles. This data analysis task is an example of Estimation with numeric prediction, where the model created predicts a continuous-valued or ordered value instead of a categorical label.

Regression is a statistical method and is the most commonly used for Estimation problems. Therefore, sometimes the Estimation is also called Regression prediction.

Classification and Estimation are the two major types of prediction problems. Both Classification and Estimation have a similar predictive modeling process, as Figure 2 of Section 5.1. However, for Estimation, the terminology of "target label attribute" is no longer applicable because the target attribute for which values are being predicted is a continuous value type (or ordered) instead of categorical (discrete and unordered).

In our mini-case study of customer churn example, instead of predicting if a given customer is going to churn or not, it would be possible for the customer relationship department manager to understand the likelihood of churn in terms of ratio. With this possibility, the data mining problem to solve becomes Estimation instead of Classification. We would replace the categorical target attribute, churn, with the continuous-valued between 0 and 1 as the estimated attribute value and develop a predictive model for the case.

Note that prediction can also be considered as a function:

$$Y = f(x)$$

where X is the input (e.g., a record describing a customer profile), and the output Y is a continuous or ordered value (such as the estimated churn ratio representing the likelihood); We expect to learn a function that models the association between X and Y.

Estimation and Classification also differ in the methods used to develop their respective models. Similar to Classification, the training set used to develop an estimation model should not be used to assess its accuracy, but an independent test data set should be used. The performance of an Estimation model is assessed by calculating an error based on the difference between the predicted/estimated value and the actual known value of Y for each of the test records, X.

To demonstrate how we can implement Estimators' supervised learning methods, we reuse the customer churn data set but convert the target variable data type to numeric to fulfill the characteristics of an Estimator in predicting the numeric value of a target variable.

In the following sections, we will use several Python-supported algorithms for the building Estimation model to solve the customer churn problem. We will use the commonly used Estimation techniques, i.e., Decision Tree Regressor, Multi-layer

Perceptron Regressor of Neural Networks, and Multilinear Regressions, to model the customer churn data set for demonstrating the data modeling phase concept.
For a detailed explanation of the Python-supported algorithms and their APIs for Estimation prediction, refer to the official website https://scikit-learn.org/stable/.

5.3.1 Decision Tree Regression

In this section, we will implement Estimation to estimate the likelihood that a given customer will churn based on his/her demographic and transaction profiles.

For the purpose of demonstrating the concept of Estimation modeling, and reusing the same churn dataset, we need to transform the initially categorical target attribute type values ('yes' or 'no') with numeric values (1 or 0), though not semantically precise but somehow the 1 and 0 present a churn estimate value.

The following Python codes show the example implementation of the data modeling phase to solve the Estimation problem using the Pyhton-supported Decision Tree Regressor algorithm. The comments embedded in the codes give explanations to guide the rationale of the programming logic.

```
# import necessary libraries
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import DecisionTreeRegressor

# specify dataset source, train and test sets
df = pd.read_csv('ChurnFinal.csv')

# Convert all the attributes' data type to numeric for Estimation
df.loc[df['Churn'] == 'yes', 'Churn'] = 1
df.loc[df['Churn'] == 'no', 'Churn'] = 0
df['Churn'] = pd.to_numeric(df['Churn'], errors='coerce').astype('float')

# specify inputs and label
df_inputs = pd.get_dummies(df[['Gender', 'Age', 'PostalCode', 'Cash',
'CreditCard','Cheque','SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
df_label = df['Churn']
```

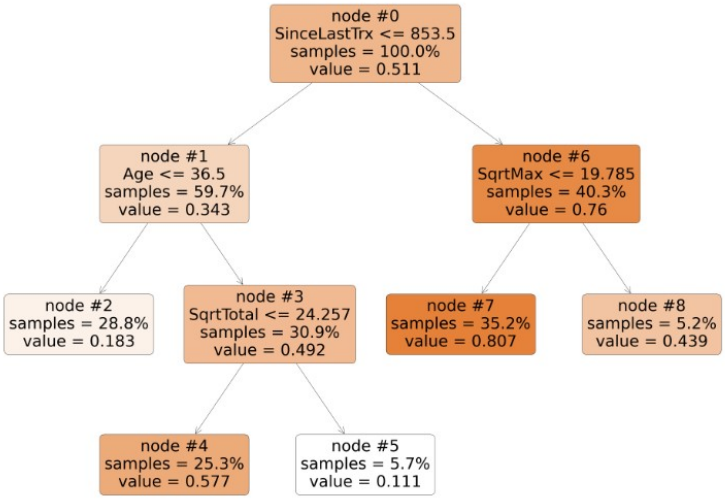
```
#The random state is a ransom seed number generator to ensure same order numbers.
#Using splitter=best, the model takes the feature with the highest importance
min_sample = int(len(df) * 0.1)     # example of assigning the minimmum sample size

# create a decision tree regressor object
dtree_rg = DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=15,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.005, #min_impurity_split=None,
                      min_samples_leaf=5, min_samples_split=min_sample,
                      min_weight_fraction_leaf=0.0, #presort='deprecated',
                      random_state=7, splitter='best')

# # split the data set to train (ratio of 0.8) and test sets (ratio of 0.2)
X_train, X_test, y_train, y_test = train_test_split(df_inputs, df_label, test_size=0.2,
random_state=7)

# fit the decision tree regressor
dtree_rg.fit(X_train, y_train)

#Predict the response for test dataset
y_predict = dtree_rg.predict(X_test)

# Calculating Mean squared error
print('Mean Squared Error (MSE): ', round(metrics.mean_squared_error(y_test,
y_predict),3))
```

After running all the codes together given above, we obtain the model performance in Mean Squared Error (MSE) of 0.201 printed on the console terminal as Mean Squared Error (MSE): 0.201. This error indicates that the model has a variance of 0.201 in predicting the estimated churn likelihood compared to the actual value. Besides MSE, we will examine more measures of model performance in Week 6.

To view the generated decision tree structure, we can plot the tree using the following Python codes:

```
# plot tree
from sklearn import tree
from matplotlib import pyplot as plt
import os
plt.clf()
target = list(df['Churn'].unique())
feature_names = list(df_inputs.columns)
strFile = "plot_dtree.png"
if os.path.isfile(strFile):
   os.remove(strFile)
fig = plt.figure(figsize=(50,37))
_ = tree.plot_tree(dtree_rg, feature_names=feature_names, class_names=target,
filled=True,
        label='all', node_ids=True, proportion=True,rounded=True, impurity=False)
fig.savefig(strFile)
```

The above codes plot the tree and save it in the plot_dtree.png file. After running the codes, the generated model tree structure view is as follows:



Observe that instead of giving each label ('yes' or 'no') ratio in a node as in Decision Tree for solving Classification problem, Estimation using DecisionTreeRegressor() function presents the likelihood of churn. For example, in node #7, the value of 0.807 indicates the likelihood of churn is 0.807, which is near the value of 1, implying a given customer who satisfies the rule of node#7 is more likely to churn than no churn.

For a detailed explanation of the DecisionTreeRegressor() API parameters, refer to the official website, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html?highlight=decisiontree%20regressor#sklearn.tree.DecisionTreeRegressor.

5.3.2 Neural Networks

In this section, we will implement Estimation to estimate the likelihood that a given customer will churn based on his/her demographic and transaction profiles.

For the purpose of demonstrating the concept of Estimation modeling, and reusing the same churn dataset, we need to transform the initially categorical target attribute type values ('yes' or 'no') with numeric values (1 or 0), though not entirely precise but somehow the 1 and 0 present a churn estimate value.

The following Python codes show the example implementation of the data modeling phase to solve the Estimation problem using the Pyhton-supported Neural Networks' Multi-layer Perceptron Regressor (i.e., MLPRegressor() function) algorithm. The comments embedded in the codes give explanations to guide the rationale of the programming logic.

```
# import necessary libraries
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neural_network import MLPRegressor

# access dataset from source
df = pd.read_csv('ChurnFinal.csv')

# convert data types to numeric/float for Estimation modeling
df.loc[df['Churn'] == 'yes', 'Churn'] = 1
df.loc[df['Churn'] == 'no', 'Churn'] = 0
df['Churn'] = pd.to_numeric(df['Churn'], errors='coerce').astype('float')

# specify inputs and label
df_inputs = pd.get_dummies(df[['Gender', 'Age', 'PostalCode', 'Cash', 'CreditCard',
            'Cheque', 'SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
df_label = df['Churn']

#The random state is a random seed number generator to ensure same order number.
#Splitting dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(df_inputs, df_label,
                test_size=0.2, random_state=1)

# feature scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# create a MLPRegressor object
mlp = MLPRegressor(
    hidden_layer_sizes=(100,),  activation='relu', solver='adam', alpha=1e-05,
    batch_size='auto', learning_rate='constant', learning_rate_init=0.0001,
    power_t=0.5, max_iter=1000, shuffle=True, random_state=0, tol=0.0001,
    verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
    early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
    epsilon=1e-08)
```

```
# train the model using train data set
mlp.fit(X_train, Y_train)

#apply the created model using test data set
y_predict = mlp.predict(X_test)

# assess the model performance using mean squared error
print('Mean Squared Error(MSE): ',
            round(metrics.mean_squared_error(Y_test, y_predict),3))
```

After running all the codes together given above, we obtain the model performance in Mean Squared Error (MSE) of 0.209 printed on the console terminal as Mean Squared Error (MSE): 0.209. This error indicates that the model has a variance of 0.209 in predicting the estimated churn likelihood compared to the actual value. NOTE: Besides MSE, we will examine more measures of model performance in Week 6.

For a detailed explanation of the MLPRegressor() API parameters, refer to the official website, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor

5.3.3 Multiple Linear Regression

Regressions are the most common method for numeric estimation problems when the target variable is numeric or continuous. In this section, we will implement Estimation to predict the estimate that a given customer will churn based on his/her demographic and transaction profiles.

For the purpose of demonstrating the concept of Estimation modeling, and reusing the same churn dataset, we need to transform the initially categorical target attribute type values ('yes' or 'no') with numeric values (1 or 0), though not entirely precise but somehow the 1 and 0 present a churn estimate value.

The following Python codes show the example implementation of the data modeling phase to solve the Estimation problem using a Linear Regression (i.e., LinearRegression() function) algorithm. The comments embedded in the codes give explanations to guide the rationale of the programming logic.

import warnings

```
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np
from sklearn.linear_model import LinearRegression
from scipy import stats
import statsmodels.api as sm

# access the dataset from source
df = pd.read_csv('ChurnFinal.csv')

# assign categorical value to numeric values for Estimation modeling
df.loc[df['Churn'] == 'yes', 'Churn'] = 1
df.loc[df['Churn'] == 'no', 'Churn'] = 0
df['Churn'] = pd.to_numeric(df['Churn'], errors='coerce').astype('float')

# specify inputs and label
df_inputs = pd.get_dummies(df[['Gender', 'Age', 'PostalCode', 'Cash', 'CreditCard',
        'Cheque', 'SinceLastTrx', 'SqrtTotal', 'SqrtMax', 'SqrtMin']])
df_label = df['Churn']

# split dataset to train and test sets, proportion of 70% and 30% respectively
#The random state is a random seed to ensure same order number.
X_train, X_test, Y_train, Y_test = train_test_split(df_inputs, df_label,
            test_size=0.3, random_state=7)

# create a LinearGression object
model = LinearRegression()

# train the model using train data set
model.fit(X_train, Y_train)
model.fit(X_train, Y_train)

# obtain the model intercept and coefficient of each input attribute
print(f"intercept: {np.round(model.intercept_,2)}")
fieldList = np.array(list(df_inputs)).reshape(-1,1)
coeffs = np.reshape(np.round(model.coef_,2),(-1,1))
coeffs=np.concatenate((fieldList,coeffs),axis=1)

coeffs_inputs = pd.DataFrame(coeffs,columns=['Attribute','Coefficient'])

#convert the coefficient data type from object to float for later numrical comparison
```

```
coeffs_inputs['Coefficient'] = coeffs_inputs['Coefficient'].astype(float, errors =
'raise')

# to select only those coefficients have magnitude more than 0.01 (1% impact on target)
ci1 = coeffs_inputs[coeffs_inputs['Coefficient'] >= 0.01]
ci2 = coeffs_inputs[coeffs_inputs['Coefficient'] <= -0.01]
final_ci = pd.concat([ci1, ci2])
print(final_ci)

# assess the model using f-test p-value, r-squared and MSE measures
y_pred = model.predict(X_test)
from sklearn import metrics
est = sm.OLS(Y_train, sm.add_constant(X_train))
if est.fit().f_pvalue > 0.001:
        print('F-test p-value: ', round(est.fit().f_pvalue,3))
else:
        print('F-test p-value: <0.001')
print('R-sq score: ', round(metrics.r2_score(Y_test, y_pred),2))
print('Mean Squared Error(MSE) : ', round(metrics.mean_squared_error(Y_test,
y_pred),4))
```

Observe that in this example, we filter to select only those input attributes with a coefficient magnitude of 0.01 (i.e., 1% impact on the target attribute). This is one of the model selection methods we can choose to perform.

In Week 6, we will examine more approaches in model performance evaluation and comparison for selection.

After running all the codes together given above, we obtain the model performance in Mean Squared Error (MSE) of 0.1819 printed on the console terminal as Mean Squared Error (MSE): 0.1819. This error indicates that the model has a variance of 0.1819 in predicting the estimated churn likelihood compared to the actual value.

Besides MSE, we will examine more measures of model performance in Week 6.

We also observe the output of intercept: [0.31] with the Coefficient (in 2 decimal points) of each input attribute as follows: Age = 0.01, Cash = 0.01, Gender_female = 0.05, PostalCode = -0.01, CreditCard = -0.22, Cheque = -0.09, SqrtTotal = -0.01, and Gender_male = -0.05. Thus, the generated Linear Regression model is mathematically represented as follows:

Churn = 0.31 + 0.01(Age) + 0.01(Cash) + 0.05(Gender_female) - 0.01(PostalCode) - 0.22(CreditCard) - 0.09(Cheque) - 0.01(SqrtTotal) - 0.05(Gender_male)

Further, the R-sq score: 0.27 result indicates that the model explains 27% of the target churn data's variability (i.e. characteristics).

The F-test statistics test the overall significance of the regression model. Assuming 0.001 is the acceptance threshold for F-test significant level. Therefore, the F-test p-value of <0.001implies the model is significant, suggesting that the model provides a better fit to the data than a model that contains no input attributes or random guessing.

For a detailed explanation of the LinearRegression() API parameters, refer to the official website,https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linearregression#sklearn.linear_model.LinearRegression

5.4 Discussion Forum Activity

Time: 40 minutes

Purpose: This activity aims to use the dataset(s) chosen in Week 2 and apply any three supervised learning Classification and Estimation methods for data modeling.

Task: Use the dataset(s) that you have chosen in Week 2 Activity 1, and perform the following tasks:

- use any three supervised learning Classification methods for data modeling.
- use any three supervised learning Estimation methods for data modeling.