Machine Learning

A Course Material for

CSC 572

J. D. Akinyemi ***PhD***

Contents

General Introduction and Course Objective

In the course of your programme, you have been exposed to several fields of Computing, including programming, databases, and data communication among others. In this course, you will yet be exposed to another fast-growing field of Computing. Machine Learning (ML) is a sub-field of Artificial Intelligence (AI) that enables computers to predict particular phenomena based on identified or learned patterns from data.

This course will NOT teach you:

- a particular programming language
- Large-scale ML solutions
- Other aspects of (AI) (e.g., Robotics, Search, Game Playing, etc.)

But it will teach you:

- Basic Machine Learning types
- How to choose the appropriate ML type for a task
- How to implement simple ML solutions

Course Curriculum Contents

CSC 572 (Machine Learning)

Definition and Types of Machine Learning, Data Preprocessing, Exploratory Data Analysis, Features Engineering, Supervised Learning, Unsupervised Learning, Neural Networks and Deep Learning, Model Evaluation, Ensemble Learning, ML Ethics and Bias, ML Libraries and Tools, Applications of ML, Laboratory exercises and Course Projects.

Semester 1, LH 30; PH 45; 3U; Status C

Study Session 1: Introduction to Machine Learning

Expected Duration: 1 week or 2 contact hours

Introduction

In this study session, you will be introduced to Machine Learning and its different types. The specific learning outcomes of this session are as follows:

Learning Outcomes

When you have studied this session, you should be able to explain:

- Definition of Machine Learning
- Types of Machine Learning Algorithms
- Real-world Applications of Machine Learning

1.1 Definition of Machine Learning

Artificial Intelligence (AI) deals with making machines that can think and act rationally or humanly. Recently, the most dominant field of AI has been Machine Learning (ML). The term Machine Learning was first used by Arthur Samuel of IBM in a 1959 paper in which he presented a checkers-playing algorithm (Samuel, 1959). Since then, ML has continued to be used as a term referring to the task of making computers perform a task by learning from data/examples.

One of the most popular definitions of ML include is given by Tom Mitchell as a task in which "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*" (Mitchell, 1997).

For this purpose of our learning, we will provide a fairly simple definition of Machine Learning as a task that involves exposing a computer program to several data patterns with the aim of teaching the program to recognize similar previously unseen data patterns. With respect to Tom Mitchell's definition, E stands for the data patterns, T stands for the task of recognition, and P stands for how accurately our ML program recognizes given data patterns.

In Machine Learning, the goal is to teach a computer program to recognize other data patterns without explicitly writing program instructions to do so. To achieve this, the computer program is fed with a good number of data patterns similar to the ones we want it to be able to recognize. The process of feeding the program with several data patterns to teach it to recognize similar patterns is often referred to as “training” while the process of feeding it with previously unseen data patterns is often called “testing”. From the above definitions, it can be observed that data is a major driver of Machine Learning. Therefore, the type of data we want our ML program to recognize is the type of data we will usually use to train the program.

In the rest of this text, we shall use the terms Machine Learning and learning interchangeably. Although an ML model can differ slightly from an ML algorithm, for the sake of simplicity, we shall use both terms interchangeably to refer to a Machine Learning program.

Tutorial Questions

1. In your own words, describe ML
2. What are the goals of ML

1.2 Types of Machine Learning

In this text, we shall discuss 3 basic types of ML. Supervised learning, Unsupervised learning, and Reinforcement learning. Figure 1.1 illustrates the classification of Machine Learning algorithms

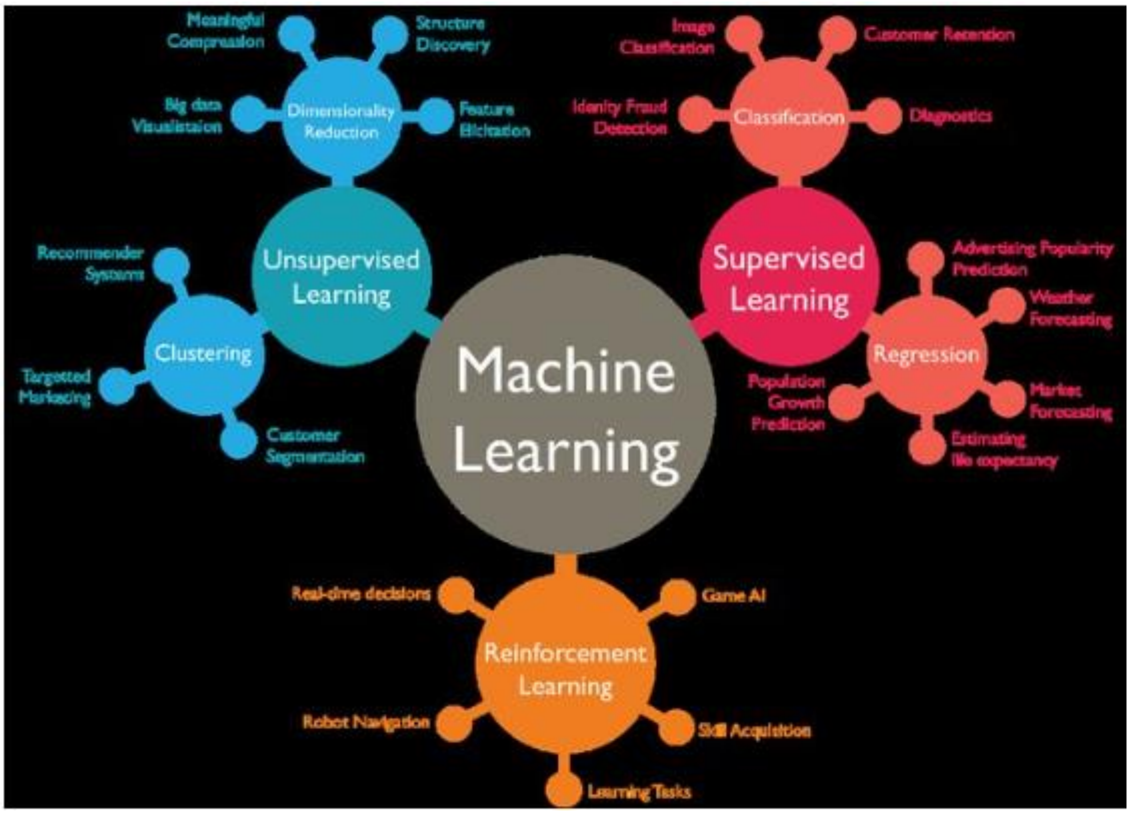

Figure 1.1: Classification of Machine Learning Algorithms (Source: (Brenninkmeijer, 2019))

1.2.1 Supervised learning

In Supervised learning, the data is labelled with the target/expected output. These target outputs are then used to check how well an ML algorithm is doing in predicting new data instances. Supervised learning is further divided into classification and regression. A learning task is said to be a classification task if the target outputs are discrete values while a regression task will have continuous-valued target output values.

An example of a classification task is classifying cancer tumours as malignant tumours or benign based on the target output labels associated with the cancer data. In this example, the target output values are exact and there can be no values in-between or outside of them. An example of a regression problem would be predicting the prices of houses given the data of houses labelled with their prices. In this case, price is a real number and so the target as well as the predicted outputs values cannot be expected to be exact e.g., 30,538,708.78 can be the price of a house. So, essentially, supervised learning tasks (whether classification or regression tasks) require the expected/target output values (also called labels) to be provided with the data.

1.2.2 Unsupervised learning

In contrast to supervised learning, unsupervised learning is characterized by the absence of target output labels in the dataset. Thus, supervised learning algorithms are expected to figure out the outputs of the data samples fed into it. What happens in unsupervised learning is a form of transformation of the data into some other form which gives insight into some interesting patterns in the data. For instance, in Clustering, the input dataset is transformed by creating clusters identified by their centres. Each cluster represents a section of the dataset which shares certain properties which are not obvious in the dataset but were discovered by the unsupervised learning algorithm. In the case of dimensionality reduction, the transformation would be to take the attributes/properties of each example in the dataset and return a subset which is believed to be optimal for a given task.

Examples of unsupervised learning tasks would involve clustering the sales data of an organization to find interesting, but previously unknown patterns in the data. For instance, there is the story of a retail store which from the analysis of their data discovered that men aged 30-40 years who purchased diapers in their store between 5 pm and 7 pm on Fridays almost always purchased beers as well (Watson, 2012). Is it not surprising to find that there could be a correlation between beers and diapers? But that is what Data Analytics, Data Science and Machine Learning can reveal. Another example would be trying to find the set of optimal features (also called attributes) needed to predict the yield of rice on a farm given several hundreds of features including climatic, agricultural, economic and biological factors among others. It may eventually turn out that only a handful of features, perhaps less than 20, are crucial to effectively perform the prediction. This is an example of dimensionality reduction.

1.2.3 Reinforcement learning

In reinforcement learning, we are trying to teach an AI agent to make the right decisions by penalizing its wrong decisions and rewarding its right decisions. Specifically, the AI agent is a machine which lives in an environment and can detect the state of that environment at every point in time. It can therefore take actions to move from one state of the environment into another and each action it takes attracts either a reward or a penalty depending on whether it is the kind of action it should take at that state or not. The goal of reinforcement learning, therefore, is to make the AI agent learn a policy, which is essentially a function that determines the optimal action to perform in each state of the environment. An optimal action is one which maximizes the agent's rewards.

A good example of reinforcement learning in action is to think of an automated room cleaner that is designed to be able to go around a room on its own and suck up dust and dirt in the room. In this case, the room is the environment and the states will be the location the agent is in per time and the situation of the location, whether it's clean or dirty. The agent may be able to take the action to suck up dirt or to move on from a location if there is no dirt. In this case, we would want the agent to be able to learn a policy that makes it suck up dirt at dirty locations and move on from clean locations.

1.2.4 Other Types of Learning

Apart from these three basic types, you may find some texts listing other ML types such as semi-supervised and self-supervised learning which are somewhat in between supervised and unsupervised learning. We have chosen not to consider them as basic ML types in this text because each of them can be derived from the basic types. We provide brief descriptions of each one in the next two paragraphs for your understanding. In the rest of the text, our discussions will be focused on the 3 basic types described above.

In semi-supervised learning, the dataset mostly contains examples that are not labelled with the target output values and a few examples which are labelled with target outputs. The goal of this kind of learning is to use the large unlabeled data to understand the distribution of the dataset to use it to improve prediction. It can be very useful in situations where the intended learning type should have been supervised, but only a few of the available data are labelled with the target output, especially when it is difficult or expensive to label the rest of the dataset.

Self-supervised learning is a type of learning in which target outputs are automatically generated from a completely unlabeled dataset (i.e., dataset without target labels), and in the second stage of this learning, the generated target outputs are used to train a learning algorithm in a supervised manner. So, we can say that self-supervised learning begins in an unsupervised manner (dealing with completely unlabeled data) but concludes its task in a supervised manner. It is therefore very useful for supervised learning tasks for which it is difficult to obtain large volumes of labelled data.

Tutorial Question

1. Can you think of other examples of supervised and unsupervised learning tasks?
2. Read more on Semi-supervised and self-supervised learning and give examples of their real-world applications.

1.3 Machine Learning in Practice

Up to this point, some of you might have wondered if there are real-world examples of the application of ML. This section discusses some real-world applications of ML, some of which are quite popular among many students and professionals.

1. **Weather Forecast**: With the availability of large historical weather data, complex Deep Learning (DL) models and of course, improved satellite technology, weather forecasts have become more accurate than they used to be. Many of you now have daily weather forecasts available on your smartphones, even though they're not always exact, most of the time, they are accurate even up to the timing of the forecasts.
2. **Natural Language Processing**: We live in a world where generative AI writing is gradually "stealing the writing show" and making it easier than ever to write pieces of text. This has been made possible by Large Language Models (LLM) which are trained on large corpus of texts with billions of parameters in complex learning models. This is just another example of AI, ML, and DL in action.
3. **Computer Vision**: Vision systems, including facial recognition, generative arts and object detection have also become very popular these days. These are made possible by large the availability of large datasets, complex learning models and high-performance computing infrastructure.
4. **Recommender Systems:** Have you ever tried to buy an item online and started getting recommendations of other similar or matching items to buy? That's a

Recommender System at Work. Some texts even discuss this as a type of ML, but actually, it is an example of the many applications of ML. Recommender systems are a class of Information retrieval systems that help to rank items based on users' rating of those items. It also keeps track of similarities in user preferences and uses this to recommend items to users. The first one described, which uses the rating of items to recommend them to a user is often referred to as content-based recommendation while the one which uses user preferences is known as collaborative. There are also hybrid approaches which combine both methods.

Summary

In this introductory chapter, you have learned the meaning of ML. We also studied the different types of ML and discussed each one briefly. In subsequent chapters, we shall go into the details of some of these ML types, especially supervised and unsupervised learning

Now attempt to answer the following simple questions:

Self-Assessment Questions

1. How does ML differ from conventional programming?
2. Compare and contrast Supervised and Unsupervised learning
3. What is semi-supervised learning and how does it differ from self-supervised learning?
4. Give other examples of real-world applications of ML
5. ML has the capacity to replace human effort in the workplace. Discuss.

Study Session 2: Data Preprocessing

Expected Duration: 1 week or 2 contact hours.

Introduction

Data preprocessing involves all the processing activities carried out on data before the data is processed or analysed by a Machine Learning (ML) model. These activities are very important in determining the success or failure of subsequent analysis of an ML model on the data. In this study session, we will consider two broad categories of data preprocessing activities which include data cleaning and exploratory data analysis. We will walk through an example to give you a practical understanding of the application and impact of these operations.

Learning Outcomes

When you have studied this session, you should be able to explain:

- Data preprocessing.
- Exploratory Data Analysis (EDA) and its implications.
- Data cleaning and various activities that can be performed to achieve it.

2.1 Exploratory Data Analysis

Many times, datasets to use for machine learning can contain missing details, duplicate entries, incorrect/corrupted data, wrong data format etc. This is more common when datasets are created by combining data from varied multiple sources. If these anomalies are not corrected, they will pose a major setback to the performance of Machine Learning (ML) algorithms on such datasets. Hence, there is often the need to clean ML datasets. Due to the large volumes of data often used for ML, these anomalies may not be immediately apparent in the data without looking out for them. The process of exploring data to understand or discover these anomalies is referred to as Exploratory Data Analysis (EDA).

In this section, we will discuss several methods used for EDA and illustrate their usage with relevant examples. We will cover methods for manually checking through your dataset as well as automated methods for deriving statistical summaries and visualisations from your dataset.

1. **Manual EDA**: Although this is rigorous and not very popular, some errors can only be detected in this way. Take for instance, you have a collection of videos which are expected to conform to specific content requirements (e.g., no specific brand name should be mentioned in the video), although there are automated tools that may help you check some of the video content as well as metadata, relying on these tools alone can even lead to more errors as they may not give 100% accuracy. Although manual visual inspection of your dataset should be limited, especially when the dataset is quite large, you will need to determine what visual inspections will be required based on the nature of your dataset and the intended application domain.
2. **Automated EDA**: There are several statistical techniques and tools that can help give an overview or a summary of your dataset. We will attempt to cover some common ones but do bear in mind that what is covered here is not exhaustive and you will indeed come across other methods as you engage in your own research and development. We will cover descriptive statistical methods such as mean, median, mode, standard deviation, and variance as well as graphical methods such as histograms, bar charts, pie charts etc. You should note that some of the graphical methods apply to univariate analysis (when you have just one feature/attribute/field in the dataset), and others apply to multivariate analysis (when you have more than one feature/attribute/field in the dataset).
    a. **Univariate analysis**: Analysis involving only one variable/attribute/field.
        i. *Frequency distribution*: The simplest form of Statistic to obtain on a dataset is the frequency of its distribution. A distribution simply refers to the set of values [for a given feature/attribute] in a dataset. The frequencies could be obtained for each individual value or group of values.
        ii. *Measures of central tendency*: This includes the mean, median and mode. The mean is the average of the values in a distribution. The median is the middle value in a sorted distribution. If a

distribution contains two median values, a single value is obtained by computing the mean of both values. The mode gives the value with the highest frequency in a distribution.

iii. *Measures of Spread/Variability*: These statistical metrics measure the extent of spread or variations within a distribution. The range measures the difference between the two extreme (maximum and minimum) values in a distribution. The variance measures how far each value in a distribution is from the mean. The square of the variance is the standard deviation and thus exaggerates the variance of each value from the mean.

b. **Multivariate Analysis**: Analysis involving multiple variables/attributes/fields. In many ML datasets, you often want to study the relationships between variables in order to give insight into their contribution to predicting the target variable, especially. Multivariate analysis is often carried out using graphical representations of data such as histograms, line plots, scatter plots, run charts, box plots, etc.

Tutorial Questions

1. What can be the advantages and disadvantages of manual EDA?
2. What can be the advantages and disadvantages of automated EDA?

2.2 Data Cleaning

Data cleaning could involve several different operations including the fixing or removal of duplicates, missing values, incorrect data and outlier values among others. In this chapter, we will explore some of these methods in detail. We can measure the cleanliness of a dataset by a method called *VACUCU* which is simply an acronym for **V**alidity, **A**ccuracy, **C**ompleteness, **U**niformity, **C**onsistency and **U**niqueness (Pritha Bhandari, 2022).

Data must be **valid** by conforming to certain requirements as expected for certain data or information types. For instance, entries of 30th February in a date field or a negative number in an age field are invalid. Without appropriate data validation techniques on such fields at the point of entry, it is easy for this kind of invalid entry to sneak into data.

Data must also be **accurate** in terms of its content being as close to the true values as possible. While validity checks that entries conform to a certain format, accuracy confirms that entries are as true as possible. For instance, having created a survey to take people’s responses about how often they go to the cinema, you could provide options such as “every day”, “once a week”, “once a month”, “bi-monthly”, and “annually”. Due to the ambiguity of the option “bi-monthly”, this option will be selected by some people who go to the cinema twice a month as well as those who go once in two months. The answers received from the first group of people, therefore do not represent an accurate piece of information if “bi-monthly” means once in two months. These kinds of errors are difficult to detect when data has been collected.

The **completeness** of data is another important characteristic of clean data. For instance, if text fields in a dataset contain incomplete sentences, it will be difficult to make sense of such entries as their representations will be broken or inaccurate and thus negatively impacts the performance of any ML model built on such a dataset.

It is also very important that data contains **uniform** entries as non-uniform entries can cause significant disparities in their representations. For instance, recording heights without specifying the unit of measurement will produce values entered in different units (meters, feet, etc.) and the range of these values will be significantly different, thus creating additional problems with the validity of the entries.

Entries must also be **consistent** within examples in a dataset as well as across a dataset. For instance, having an example within a dataset whose age is 7 years and educational qualification is PhD is inconsistent and one of the fields (age or education) is very likely to be incorrect.

It is also very important that every observation in a dataset is **unique** and duplicate entries must be removed.

2.2.1 How to clean data.

The techniques to employ for data cleaning will vary depending on the nature of the data and its intended application. It is often easier to enforce automated checking during data collection to ensure that the resulting dataset is as clean as possible, but many times, this is not even possible depending on some data collection methods (e.g., data scraping). Many times, also, what is available to use is a secondary (an already existing) dataset which is even more difficult to clean. In this section, we will consider some data-cleaning techniques which can potentially improve the quality of our datasets.

1. **Data Validation**: Data validation is a technique you could use to enforce constraints on certain fields of data during the data collection process. It helps to minimize the amount of data cleaning you would have to do since it checks and enforces validation at the point of data collection. Examples of constraints that could be enforced include *data type constraints*, *value range constraints*, and *mandatory field constraints*. Data type constraints check that the value entered for a field is of a specified data type only, e.g., age can only be numeric, a name must be text, or a date must be a date type in a specified format such as 'dd/mm/yy'. Range constraints check that values entered for a particular field lie within a range of values. This is often enforced on numeric fields such as height, weight, age etc. For instance, when collecting data on house prices, it might be reasonable to specify that prices range from ₦100,000 to ₦ 1 billion, thus house prices outside this range of values will not be acceptable. Mandatory field constraints ensure that certain fields must be filled, for instance, a field indicating a confirmation that a user agrees to the terms and conditions of a particular service.
2. **Data Screening**: After data has been collected, it's usually good to have a backup copy (either offline or online) before you start cleaning so that you can have something to fall back on in case anything goes wrong at any point in the cleaning. It is important to screen your dataset for possible inconsistencies, missing data, duplicates and other discrepancies. This can be done by manually observing the data or using statistical methods. This is the same process of EDA which we covered in section 2.1 so this is not repeated here.
3. **Data Diagnosis**: It's important to correctly diagnose the "dirt" in data so as to remove or reduce them as much as possible. This is where the EDA techniques highlighted in section 2.1 come into play. The EDA techniques can help detect all kinds of anomalies in the dataset including duplicate entries, missing data and outliers among others. Let's examine some of these anomalies briefly:

a. *Duplicate entries*: Duplicate entries mean that your dataset contains observations/examples which are repeated. You should simply remove all repetitions and keep a unique copy of each example. For instance, if your data is tabular, with the examples in the rows and their attributes in the columns, you should remove all rows which are duplicated and keep only one unique copy of each row.

b. *Outliers*: Outliers represent extreme values (too high or too low) within the dataset. Outliers must be handled carefully and the decision to either remove or retain them must be scientifically justifiable. Sometimes, outliers could represent the true nature of the data while at other times they may be erroneous values. For instance, the number of hospital admissions at a particular time may be extremely high, due to a pandemic outbreak such as COVID, compared to all other times. In this case, such an outlier value should not be removed but retained in the data to represent such circumstantial spikes. In another case, you could have a negative value (or an unreasonably high value e.g., 10,500) for an attribute such as a person's age. This will clearly be an error and should be removed.

c. *Missing data*: When significant portions of data contain empty or null values, it can pose a lot of problems for ML algorithms to perform well on such data. Therefore, missing data must be properly handled. By missing data, we refer to examples containing missing values for certain attributes and not one for containing missing values for all attributes – that is simply to be removed from the dataset. Usually, we don't want to remove an entire example because a few attributes have missing values because that would significantly reduce the size of the dataset. Therefore, several efforts have been designed to attempt to estimate missing values. Some of these methods include using the mean/mode of the other non-missing values of this particular attribute, sometimes one could also leave the missing fields (if there are not many of them) the way it is as some ML algorithms can find their way around it. Another solution is to use data imputation in which you use the other attributes whose values are complete to train an ML model to predict the attribute with the missing values. In this case, those examples whose values for this particular attribute are non-missing are used as the training set and the ones with missing values are used as the test set for prediction. All these techniques must be used with caution so as not to risk creating biases in the data.

Summary

In this chapter, you have learned how to preprocess data and get it ready for an ML algorithm. This practice is very important because the success of any ML system depends on the data it is trained and validated on. Therefore, data preprocessing will, to a large extent, determine the success or failure of the ML models built. We have discussed how to clean data by validation, screening and diagnosis. We have also discussed EDA techniques for improving our knowledge of the data and thus making us better informed of the ML algorithms that might be appropriate for the learning task.

Now attempt to answer the following simple questions:

Self-Assessment Questions

3. Explain some EDA techniques you know?
4. What are some methods of dealing with missing data?
5. When will it be appropriate to remove or retain outlier values?
6. Describe the 6 characteristics that can be used to define the cleanliness of data.

Study Session 3: Feature Engineering

Introduction

In this study session, we will learn about Feature Engineering. We will learn about features and the different types, the common concepts in Feature Engineering as well as the process of feature engineering.

Learning Outcomes

At the end of this session, you should be able to:

- Describe Feature Engineering.
- Explain the basic concepts of Feature Engineering.
- Describe the techniques involved in Feature Engineering.
- Explain some common feature transformation techniques.

3.1 Introduction to Feature Engineering

Feature Engineering is a very important, yet less-studied concept in Machine Learning (ML). The overall performance of an ML model will largely depend on the features it is trained with. Feature Engineering is a process which involves transforming raw data into some representation which is believed to be usable and meaningful for an ML model. The goal of Feature Engineering is to represent the data in such a way that simplifies their processing and maximizes the performance of the ML model.

We can simply refer to a feature as an attribute of an observation in a dataset. For instance, in a dataset of customer records, one of the features could be the age of the customer. However, features can be much more complicated than this. Features can be numeric such as age, weight, height, house number etc., or categorical such as the colour of a car, the name of a person, or the natural gender of a person. A special type of categorical feature is the Boolean feature type which means that the feature can only have one of two values (true or false), an example of this is the natural gender which can either be male or female.

Numeric features are ordered and can be further classified into qualitative and ordinal features. Qualitative features are usually continuous-valued features which are ordered and also have linear scale. Examples of this include age, weight and height which can all be real-valued (integers are also real numbers). Apart from being ordered, qualitative features have a meaningful numerical scale or unit of measurement e.g., height can be measured in meters or inches, age can be measured in years, months, weeks etc. Ordinal features, however, do not have any meaningful unit of measurement, but they are also ordered, an example of this is house number. It's common sense that house number 1 will likely be close to the entrance of the street and that houses 6 and 7 will be close to each other. But we cannot infer that house number 20 will be bigger than house number 1, because the numbering is not a scale of measurement.

The above descriptions and classifications are mostly pertinent to structured data where data can be perfectly represented as a table, observations as rows and features as columns. With unstructured data such as images, speech or text, features are at higher levels. In such cases, a feature will then mean whatever meaningful unit of each instance is sufficient for maximizing an ML model's performance in that problem space. For instance, in an image, it could be a single pixel or group of pixels that define a specific visual attribute such as lines, edges, spots etc.

3.2 Important Concepts in Feature Engineering

We will now briefly describe some important concepts in Feature Engineering. To make the descriptions of these concepts clearer, we have defined a very simple dataset in Table 3.1 which we will use throughout this chapter. The dataset in Table 3.1 is that of housing prices in Nigeria and it includes attributes such as the size of the house in square foot, the number of rooms in the house, the location of the house and the price of the house in millions of naira. Given Table 3.1, a learning task might be to predict the price of houses given the other 4 features. In this case, the price will be referred to as the dependent or target variable while the remaining features will be referred to as the independent or predictor variables.

Can you attempt to state the category to which each of the attributes/features in Table 3.1 belong?

Table 3.1: Sample dataset of house prices in Nigeria

Size (sq. Ft.)	**No. of rooms**	**Town/City**	**State**	**Price (Million Naira)**

2400	3	Ilorin	Kwara	7
3200	3	Ibadan	Oyo	15
2400	2	Ikeja	Lagos	13
2100	2	Sokoto	Sokoto	1.5
3500	3	Port Harcourt	Rivers	17.3

1. **Feature Importance**: This is a measure of the importance of a particular feature(s) for a given learning task. It is usually done by assigning scores to features and then using these scores to give a sort of ranking to the features to indicate the degree of their importance to the learning task at hand. Feature importance can prove very useful for feature construction, extraction or selection. Several statistical methods such as correlation coefficient can be useful for computing feature importance. Some learning algorithms also perform feature importance internally, therefore allowing the most relevant features to be used for learning.
2. **Feature Extraction:** Sometimes, you may be provided with raw data with no well-formed features and you may have to construct or create new features suitable for the given learning task. This process often leads to the construction or creation of new features which are more structured and suitable for learning. This process is often carried out by features extraction or dimensionality reduction algorithms which transform data into more suitable formats for machine learning models.
3. **Feature Selection:** Many times, one can get a set of features that is too large or contain many features that are irrelevant to the problem at hand. In such cases, feature selection methods are employed to select the most relevant methods. A common example of a feature selection algorithm is the Principal Component Analysis (PCA) which transforms the original features from an n-dimensional space to an m-dimensional space, where $m < n$.

It is not uncommon to find feature engineering processes in which the above concepts overlap or are used interchangeably. For instance, some features engineering paradigms would perform feature extraction and selection at a go and sometimes using a single algorithm, while some others will perform feature importance and feature selection together. These are possible because all these concepts involve some form of transformations of the input features and some of these transformations can handshake in several ways.

3.3 Features Engineering Techniques

Feature Engineering often involves several steps, some of which are described in the following list:

1. **Data Understanding**: The very first step in feature engineering is to understand the data you have been given. This can include studying the structure of the data in terms of the types/categories of features as well as the relationship between features. It can also involve gaining some knowledge of the problem domain which may help handle missing data and outlier feature values. For instance, in the dataset in Table 3.1, it may be helpful to employ the knowledge and expertise of an estate agent, quantity surveyor or architect to gain more understanding of the data. For instance, one can easily observe that the price of the house on the second-to-the-last row is exceptionally low compared to others. This is an example of an outlier, especially when we consider the fact that other houses with similar sizes cost far more than that. In such cases, it may be helpful to use the domain knowledge to either validate or correct the data before going on to make features out of it.
2. **Data Preprocessing**: Many times, datasets contain noise in various forms including missing values and outliers. Such issues are dealt with in the data preprocessing stage. To make these noisy samples obvious, exploratory data analysis needs to be carried out. Thereafter, appropriate measures can be taken to clean them. For instance, a decision can be made about outlier values, either to validate them or replace them with an average of the feature values of other instances in the dataset. A decision also needs to be made about missing values, whether those instances are to be removed (in cases, where there are relatively few instances of missing data) or the missing values should be estimated using

methods such as data imputation. Data imputation is particularly useful when the number of instances whose values are available for a particular feature is significantly more than those whose values are missing. In data imputation, we use the available values of that feature to train a model to predict the missing ones.

3. **Feature Transformation**: This is where most of the feature engineering work takes place. This stage could involve all or some of feature extraction, feature construction, feature encoding, feature selection, feature normalization and feature standardization. For instance, in the dataset of Table 3.1, you will observe that there are two textual features, *city* and *state*. And these are categorical features which need to be encoded into numeric features before they can be used to build an ML model. Subsection 3.4 discusses some popular feature transformation techniques in detail.
4. **Evaluation and Re-engineering**: Feature engineering, like ML, is an iterative process. So, there is no hard-and-fast rule for obtaining the most optimal features, but to engineer a set of features, use it to train a model, evaluate the performance of the model and then go back to re-engineer the features to improve the performance of the model. This process continues until optimal performance is reached or the features are believed to be optimal enough based on provided benchmarks.

3.4 Feature Transformation Techniques

Feature transformation is at the heart of Feature Engineering and there are many techniques which have been employed over the year and even more are being developed from time to time. The particular feature transformation technique to employ depends on the problem domain and the feature values. In this subsection, brief descriptions of some common feature transformation techniques are provided.

1. ***One-hot encoding***: This is a method of encoding categorical features into vectors of binary values. These vectors then become a useful representation of the feature for an ML algorithm to be trained on. The vector is created such that the presence of a feature is represented as the binary value 1 and its absence is represented as 0. For instance, in the dataset in Table 3.1, the feature “state” can be encoded as a 37-dimensional vector (36 states + FCT) where each vector only

contains a 1 at the position of the feature value and contains a 0 for all other entries. So, for instance, the one-hot-encoded vector of Zamfara would be a 37-dimensional vector which has a 1 at the index 36 and 0 at all other indexes.

2. ***Label-encoding***: One-hot encoding can be very high-dimensional and therefore memory-inefficient when you have categorical features which have a high number of distinct values. For instance, the city/town feature in Table 3.1 can have as many distinct values as the total number of cities/towns in all 36 states + the FCT. There are several memory-efficient techniques to one-hot encoding in these cases. One such simple alternative is label-encoding which simply assigns a unique numerical value to each unique feature value. So, the label-encoded value of Zamfara would just be a single number, 36, for instance. While this is memory-efficient, it erroneously introduces an ordinal relationship, which does not exist between the states. Label encoding is therefore suitable for categorical features which do exhibit an ordinal relationship e.g., the number of rooms as in Table 3.1.
3. ***Binning***: This is a method used to discretize continuous numerical variables by categorizing the values into bins. A good example of this in Table 3.1 is the price feature which can range from 0 to several millions. Instead of having to cater for all these continuous values, many of which may be absent, it may be more appropriate to bin the prices into age groups such as less than 1 million, 1-5 million, 6-10 million, etc.
4. ***Word embedding***: This is a more advanced technique for creating a suitable and efficient numeric representation for categorical features. Specifically, word embeddings are often used in text processing or Natural Language Processing (NLP) to create a numerical vector to represent words or sentences. It can include or rely on earlier processes such as *tokenization* in which the entire text is broken down into words and *stemming* in which words are trimmed to their root/base words. The eventual word embedding may be based on an algorithm such as Term-Frequency Inverse Document Frequency (TF-IDF) which creates a vector for the text based on the frequency of individual words in a document as well as the scarcity of the word across the entire dataset. More commonly these days, word embeddings are based on ML models which have been pre-trained to produce a vector of representation from word tokens. Examples of this include GloVe, Word2Vec, FastText and Transformer embeddings.

5. ***Bag of words***: This is a simple collection of the words of a large body of text into a representation referred to as a "*Bag of Words*" (BoW). BoW essentially represents the presence or absence of a word in a piece of text without any regard for grammar or word order.
6. ***Scaling, Normalization and Standardization***: This is a very important process in feature engineering, especially in the presence of numeric features spanning a high range of values. For instance, in Table 3.1, the size of the houses can be seen to be in the range of thousands; actually, these values could be several times larger than the values supplied in the table. Large values tend to prove difficult for ML algorithms to handle as they slow down their convergence or even make it difficult to converge to a global optimum. Hence, there is a need to put feature values within a very narrow range so that the minimum and maximum values are not too wide apart. In the case of the room size feature, we can start by scaling the values by only representing the values in their units. So, 2100 becomes 2.1, 3400 becomes 3.4, etc. This scales down the values, but they are still large because a 12000sq-ft house will be represented as 12. We can further scale down these values by normalizing them. Normalizing values will squeeze all the values between 0 and 1 so that the maximum value of the feature is represented as 1 and the minimum is represented as 0 and all other values are appropriately represented somewhere in between 0 and 1. This effectively narrows the range of values and speeds up the convergence of ML algorithms. To make this even better, we can standardize the values by obtaining their standardized z-scores which centralizes each value around the mean and scales it by the standard deviation of the entire values of the feature. The choice of either normalization or standardization or both depends on the problem domain and the type of features.
7. ***Applied mathematical functions***: There are also feature transformation techniques that are based on specialized mathematical functions. These types of techniques have proven to be useful for highly unstructured data such as audio, images and videos. They employ Mathematical concepts in basic or linear algebra, differential equations, integration, probability distributions etc. to project high-dimensional unstructured data into a relatively low-dimensional space. Examples of these types of techniques include Principal Component

Analysis (PCA), Histogram of Gradients (HoG), Gaussian Mixture Model (GMM), Gabor Filters, etc.

Tutorial Questions

1. Consider the dataset in Table 3.1. What will be the best type of transformation to apply to the features “city” and “state”?
2. Justify your answer to 1 above.
3. Can you think of and name 5 other feature transformation methods or algorithms that you know?

Summary

In this session, we have considered Feature Engineering, which we defined as the transformation of raw data into meaningful representations suitable for learning. We explained common concepts of Feature Engineering and described some basic techniques including data understanding, data preprocessing, feature transformation and continuous re-engineering. We also provided basic descriptions of some common feature transformation methods. Having gained a good understanding of features and how they are engineered, we will proceed, from the next session onwards, to begin discussing the different types of ML one after the other.

Self-Assessment Questions

1. What is Feature Engineering?
2. Differentiate between feature extraction and feature selection.
3. Describe any three methods used to achieve feature transformation in feature engineering.

Study Session 4: Supervised Learning

Introduction

One of the classes of Machine Learning (ML) that we identified in the first lesson is Supervised Learning. Supervised learning is a class of ML that deals with labelled data, i.e., data labelled with target responses. Supervised learning is of two types, classification tasks and regression tasks. This classification of supervised learning tasks is based on the nature of the values in the target response variables. In classification, the response variable contains discrete values while the response variable contains continuous values in regression.

The rest of this lesson will discuss each type of supervised learning in detail and discuss some popular types of algorithms in each class of supervised learning.

Learning Outcomes

At the end of this lesson, you should be able to:

1. Describe supervised learning.
2. Explain classification and regression.
3. Describe some common classification and regression algorithms.

4.1 The General Model of Supervised Learning

Supervised learning algorithms explore the relationship between the data and the provided target responses to find a function that can model this relationship as accurately as possible with the goal of being able to predict responses on previously unseen, but similar data items. So, given a dataset X with target responses Y, the goal of supervised learning is to predict $\hat{Y}$ as accurately as possible. To achieve this, supervised learning algorithms while being trained on a dataset use the target responses to evaluate their performance on the dataset and then adjust accordingly to improve in the next training iteration. During the evaluation, the algorithm checks the difference between Y (the actual response) and $\hat{Y}$ (the predicted response) to arrive at a loss value which it then tries to minimize in every subsequent training iteration.

To understand the mathematical model of supervised learning algorithms, here is a commonly used mathematical equation:

$$\hat{Y} = \boldsymbol{w}x + \boldsymbol{b} \tag{4.1}$$

where $\boldsymbol{w}$ and $\boldsymbol{b}$ are weight and bias variables respectively and *x* represents the vector of variables/features in an observation within a dataset.

In the above equation, $\hat{Y}$ is the predicted response and to predict this response, we must solve the linear equation. The equation may vary from one algorithm to another, but the concept is very much the same. While *x* represents the values of the features of the data provided in the dataset, $\boldsymbol{w}$ and $\boldsymbol{b}$ are unknown values that must be found. Thus, we need to find the correct values of $\boldsymbol{w}$ and $\boldsymbol{b}$ that give the value of $\hat{Y}$ which is closest to Y (i.e., minimizes the loss). But how can the best values of $\boldsymbol{w}$ and $\boldsymbol{b}$ be found? By a process called "training", during which the learning algorithm is exposed to several examples from their dataset as well as their corresponding target labels which allow the algorithm to evaluate its performance by comparing its predicted responses to the target responses. Supervised learning algorithms are also characterized by what is referred to as a loss function which is used to determine how good its predictions are. The simplest way to explain this is just to take the loss as the difference between the target and predicted values, although it can often be more complicated than this, their difference is still the general and most basic idea.

4.2 Classification

In classification, the responses with which data is labelled are expected to contain discrete values. By discrete values, we mean categorical values, where every example in the dataset must belong to one of the categories and not in between. A good example of this would be trying to classify the picture of an animal as either that of a dog or a cat, trying to classify a brain cell as cancerous or non-cancerous, or trying to classify a patient as hypotensive, normal, or hypertensive. The last example was included to indicate that the categories do not always have to be two. They can be more than two, in fact, they can be 1000 different classes, as long as each is distinct from the other and there is no way for any example to belong in between any two of them. Examples of classification algorithms include logistic regression, decision trees, support vector machines, K-nearest neighbours, neural networks and many more.

We will briefly look at the workings of three common types of classification algorithms.

1. **Logistic regression:** This is the simplest and most common type of classification algorithm. Even though its name indicates regression, it is indeed a classification algorithm. It derives its name from statistical regression because its mathematical model is similar to that of linear regression. Logistic regression uses the standard logistic function (also referred to as the sigmoid function) to squeeze the output of a linear function between 0 and 1 according to the equation in (4.2) which gives the graph in figure 4.1.

$$\hat{Y} = \frac{\mathbf{1}}{\mathbf{1}+\boldsymbol{e}^{-(\boldsymbol{wx}+\boldsymbol{b})}} \tag{4.2}$$

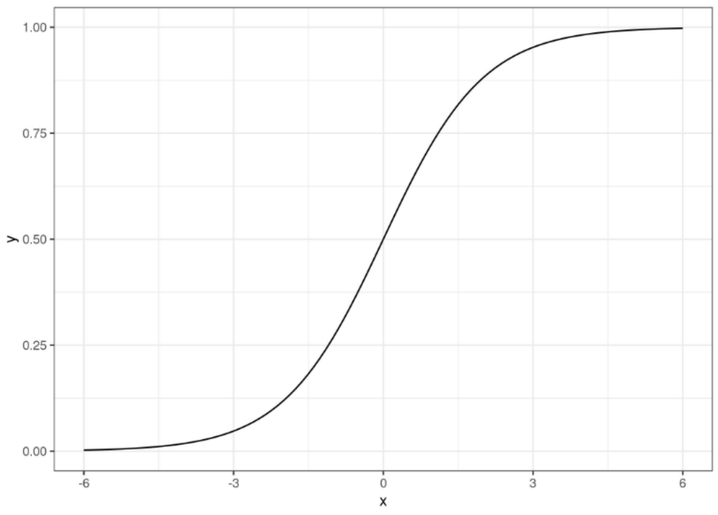

Figure 4.1: Plot of a sigmoid function

From the graph in Figure 4.1, the sigmoid function tends to categorize values close to 0 as 0 and those close to 1 as 1. Thus, logistic regression seeks to maximise the probability that a particular example belongs to a given class, a concept often referred to as maximum likelihood estimation.

2. **Decision Tree:** Decision Tree is another class of powerful and popular supervised learning algorithms. Decision Trees can be used for both classification and regression. As the name implies, it is based on a tree consisting of nodes and branches. To make a prediction, an example is compared with the nodes of the tree starting from the root and branching to appropriate internal nodes based on the result of the comparison of the value of

the example with the value in each tree node. When the branching eventually leads to a leaf node, then a final class is arrived at which is predicted as the class of the input example. This is only a simple illustration that applies to a fairly simple case of classification. Decision trees can be more complex than this, especially when used for regression.

3. **Support Vector Machine (SVM):** SVMs are a powerful class of ML algorithms which makes decisions by finding an optimal separation hyperplane between examples. It aims to use the feature/attributes of the examples to separate them into their corresponding classes and gives a margin around the separation hyperplane to minimize misclassification errors. The examples falling into these margins are referred to as support vectors. The goal of SVM is to find a separating hyperplane that maximises these margins, thereby reducing misclassification costs. Figure 4.2 illustrates the SVM in operation as explained above. Note that Figure 4.2 illustrates a 2-class linear classification problem, but SVM is also very suitable for non-linear problems with the help of the "kernel trick". While SVM was originally designed for binary (2-class) classification, it has since been extended to solve multi-class classification as well as regression problems.

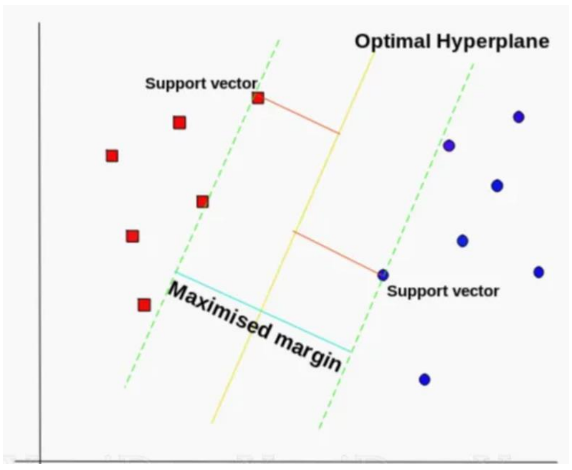

Figure 4.2: Support Vector Machine Separating Hyperplane (Source: (Rushikesh Pupale, 2018)*)*

Tutorial Question

Compare and contrast Logistic Regression, Decision Trees and SVMs. Some areas of comparison may include *discriminatory power*, *mathematical model*, *ability to solve linear/non-linear problems*, *usability for classification as well as regression*.

4.3 Regression

Unlike classification, in regression, the target labels are continuous values (i.e., real values). This means there are no discrete categories into which data points can be classified, rather, a real-valued label has to be estimated for them. Common examples of regression problems include housing price prediction which we discussed in lesson 3, predicting attributes such as age, height, and weight or climatic factors such as temperature and humidity. Examples of regression algorithms include linear regression, random forest, gradient boosting, etc.

1. Linear Regression: This is, perhaps, the simplest regression algorithm. Linear regression learns features which have a linear (or direct) relationship with the target variables. Unlike SVM which tries to maximise the distance between it separating hyperplane and the input sample points, linear regression chooses a hyperplane that is as close as possible to the sample points. This is because the hyperplane in SVM is a decision boundary, while that in linear regression is a line of "best fit". This is illustrated in Figure 4.3 using a 1-dimensional feature vector. Figure 4.3 shows us that *x* (the independent variable) can be used to derive new values of *y* (the dependent variable) once a regression line has been "best fit" to the existing points of x.

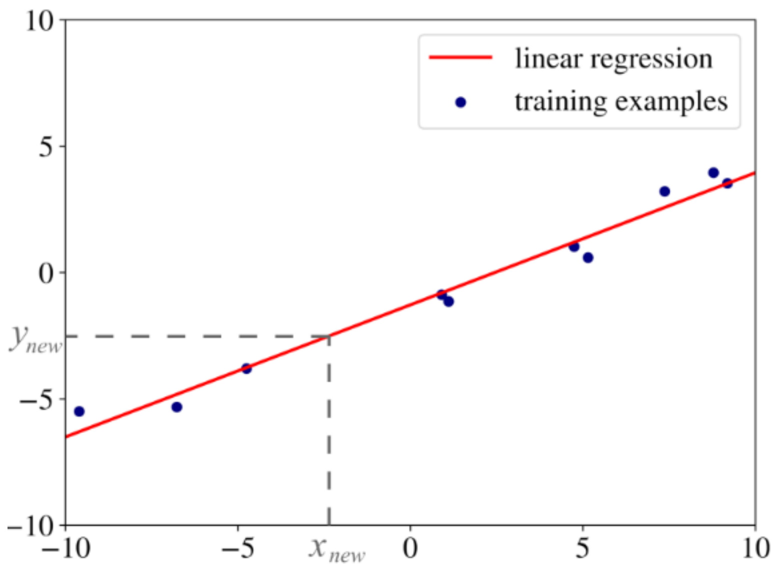

Figure 4.3: Graph of a linear regression line (Source: (Burkov, 2019)*)*

2. **Random Forest:** Random Forest is an ML algorithm built as an ensemble of decision trees. It can be used for classification by taking a majority vote on the decisions of all the decision trees. It can also be used for regression by taking an average of the decisions of the decision trees.

Tutorial Question

Attempt a discussion of other ML algorithms which have been mentioned (but not discussed) in this lesson.

Summary

In this lesson, you have learnt about supervised learning algorithms – a class of learning algorithms that requires target response values to make predicted responses on new input examples. We examined the two main types of supervised learning algorithms namely classification and regression algorithms. We have also given brief descriptions of some of these learning algorithms. In the next lesson, we will be examining the unsupervised learning algorithms which do not requires target response values to make "so-called" predictions.

Self-Assessment Questions

1. Briefly explain supervised learning algorithms.
2. Differentiate between classification and regression.
3. Describe any two supervised learning algorithms that can perform classification as well as regression.

Study Session 5: Unsupervised Learning

Introduction

In this lesson, we shall consider unsupervised learning methods. Unlike supervised learning, unsupervised learning does not require the target responses to be provided with the dataset (it does not require labelled data). Unsupervised learning algorithms are good at finding hidden patterns and structures in data without prior knowledge of the expected responses. Unsupervised learning allows us to gain valuable insights into data which could be very useful in market analysis and anomaly detection to mention a few. Examples of unsupervised learning algorithms include clustering, dimensionality reduction and association algorithms, each of which will be briefly discussed in this chapter.

Learning Outcomes

At the end of this lesson, you should understand and be able to explain:

- Unsupervised learning algorithms
- Clustering algorithms
- Dimensionality reduction algorithms

5.1 Clustering

This is perhaps the most common type of unsupervised learning. In clustering, the aim is to find similarities in data and, using these similarities, group data into clusters containing related or similar data items. This is often used in Biomedical engineering or bioinformatics for gene sequencing by analyzing several gene codes and grouping them together based on identified similarities. It can also be useful in big data analytics tasks such as market segmentation in which a clustering algorithm groups customers, sales or orders based on hidden similarities which are discovered by the algorithm. Other applications of clustering include image segmentation, anomaly detection and pattern recognition.

Perhaps the most popular clustering algorithm is the K-means clustering, which partitions the data into *k* subsets each subset being clustered around a chosen centroid which is the mean data item to which all other data items in the cluster are closest. K-means is a computationally efficient algorithm that scales large datasets, this is, perhaps a major reason for its popularity, along with its simple and intuitive nature. K-means is popularly used in customer segmentation, image compression and document clustering.

Another example of a clustering algorithm is Hierarchical clustering. In Hierarchical clustering, a tree-like hierarchy of clusters is created by recursively merging or splitting clusters based on the similarity between data points. This results in a dendrogram (a tree-like graphical plot) that illustrates the relationships between clusters at different levels of granularity. Hierarchical clustering is particularly useful when the number of clusters is not known beforehand, as it allows the user to choose the desired number of clusters based on the dendrogram's structure. It can be used in biological taxonomy to classify species based on shared genetic traits, creating a tree-like representation of their evolutionary relationships.

Tutorial Question

Present a comparison of any two clustering algorithms apart from the ones discussed in the text. Points to consider in making comparisons can include their modes of operation, their mathematical model, and their advantages and disadvantages.

5.2 Dimensionality Reduction

Another class of supervised learning we will consider is Dimensionality reduction. Dimensionality reduction is a powerful concept used to simplify complex data and gain insights from it. Imagine having a large dataset with many features or attributes. Sometimes, dealing with such high-dimensional data can be overwhelming and computationally expensive. Dimensionality reduction comes to the rescue by transforming this data into a lower-dimensional space, thereby reducing the number of features while preserving its essential patterns and relationships. It helps us focus on the most critical aspects of the data, making it easier to visualize, analyze, and extract meaningful information from the data.

To illustrate this, consider the housing price prediction dataset introduced in session 3 and imagine that the dataset has about 1000 features including features such as the name, phone number, email address, and occupation of the house owner among other features. Apparently, these kinds of personal details of the house owner do not necessarily contribute to the value of the house and may very well be removed to reduce the number of features in the dataset while retaining those features like the size, location and style of the house which are more likely to contribute to its value.

A popular example of a dimensionality reduction algorithm is the Principal Component Analysis (PCA). The primary purpose of PCA is to transform high-dimensional data into a low-dimensional space while retaining the original variability of the data. PCA achieves this by identifying the principal components, which are new orthogonal axes representing the most significant directions of variance in the data. The first principal component captures the most substantial variation, the second captures the second most significant variation, and so on. By projecting the data onto these principal components, PCA reduces the data's complexity while preserving its essential patterns and relationships, making it easier to visualize and interpret the data effectively.

5.3 Association

Association algorithms are an interesting example of unsupervised learning, and they have been used for years in the field of Data Mining. Association algorithms aim to discover interesting relationships, patterns, or associations between variables within large datasets without the need for labelled output. One popular association algorithm is the Apriori algorithm, which identifies frequent item sets in transactional data, thereby helping to understand which items are often purchased together. For instance, in a retail store, the Apriori algorithm can reveal that customers who buy milk and bread are also likely to purchase eggs. By uncovering such associations, businesses can optimize product placement, recommend complementary items, and gain valuable insights into customer behaviour, all without any predefined labels or supervision in the learning process.

Tutorial Question

Write out 5 interesting applications of unsupervised learning and for each application, try to identify which of the three classes of unsupervised learning algorithms have been used.

Summary

In this lesson, you have learned about unsupervised learning – a type of learning that does not require labelled data. We have examined three major types of unsupervised learning, namely clustering, dimensionality reduction and association. Unsupervised learning algorithms generally help to extract meaningful hidden patterns and insights from data which can be very useful for decision-making. Applications of unsupervised learning include market analysis, customer segmentation and features extraction and selection.

Self-Assessment Questions

1. Explain unsupervised learning.
2. Compare and contrast Clustering and Association.
3. A fast-growing enterprise has provided you with data of their customers' patronage. They want to draw insight from this data to see how they can improve their services. Which class of unsupervised learning algorithms will be suitable for each of the following tasks
    a. Identify the services/products that customers like.
    b. Identify the set of services/products that are often requested together.
    c. Identify the different customer groups in order to provide targeted adverts to them.

Study Session 6: Neural Networks and Deep Learning

Introduction

Artificial Neural Networks (or simply Neural Networks) is one of the most popular Machine Learning algorithms around. Neural networks have been around for a long time but have recently gained attention because of the success of deep neural networks. In this lesson, you will learn about the basics of Neural networks and deep learning. We will also consider some interesting applications of these networks.

Learning Outcomes

At the end of this lesson, you should:

1. Understand the basics of neural networks.
2. Understand the basics of deep learning.

6.1 Neural Networks Basics

Neural networks represent a revolutionary paradigm in the field of artificial intelligence and machine learning, drawing inspiration from the intricate structure and functionality of the human brain. These computational models have demonstrated remarkable capabilities in solving a wide range of complex tasks, from image and speech recognition to natural language processing and game playing. This lecture introduces the fundamental concepts behind neural networks, their architecture, and the underlying principles that empower them to learn and generalize from data.

6.1.1 Biological inspiration

The biological inspiration behind neural networks stems from the intricate workings of the human brain, the most sophisticated known information processing system. The fundamental building blocks of artificial neural networks, known as neurons, draw parallels with their biological counterparts. Figure 6.1 shows the structure of a biological neuron. Just as neurons in the brain transmit and process signals, artificial neurons mimic this behavior by receiving inputs, applying activation functions, and producing outputs. Just like biological neurons are connected to other neurons by the axon, artificial neurons also establish a connection with other neurons and pass values through the entire network using these connections.

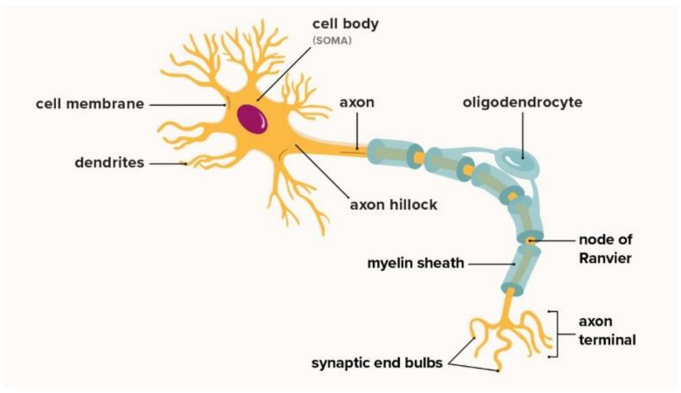

Figure 6.1: Structure of a biological neuron

This emulation of biological neural connections allows artificial neural networks to capture complex patterns and relationships in data, enabling them to excel in tasks such as pattern recognition, language processing, and decision-making. While artificial neural networks are far simpler than the brain's neural networks, the connection to biology has paved the way for breakthroughs in machine learning, yielding systems capable of remarkable feats previously thought to be within the realm of human cognition.

The next subsection presents a step-by-step explanation of the neural network architecture.

6.1.2 Neural networks architecture

Let's look at the architecture of a simple neural network also called a feedforward network or a Multilayer Perceptron (MLP) network. Consider the simple MLP network in Figure 6.2 containing an input layer, a hidden layer and an output layer. We will now look in details at each layer.

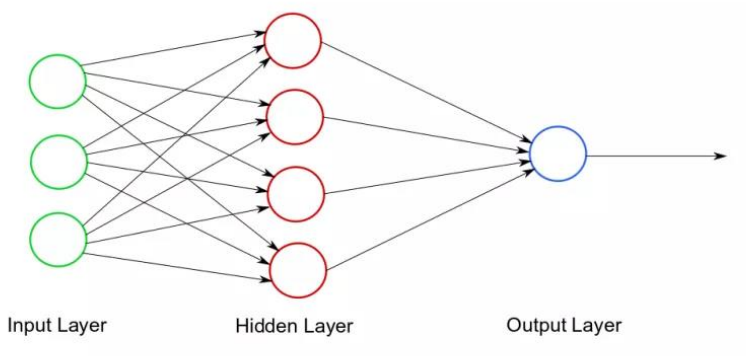

Figure 6.2: A simple neural network architecture

1. **Input Layer**: The neural network starts with an input layer. This layer consists of neurons that represent the features or attributes of the input data. Each neuron corresponds to a specific feature, and the values fed into these neurons are the input data points. In the network of Figure 6.2, the input layer contains 2 neurons meaning there are two specific features of the data that are fed into the network.
2. **Weighted Sum and Activation**: Each neuron in the input layer is connected to neurons in the next layer, called the hidden layer. These connections are associated with weights. The input values are multiplied by their corresponding weights, and the weighted sum of these products is calculated for each neuron in the hidden layer. An additional bias term might also be added to the weighted sum. This weighted sum is then passed through an activation function. Common activation functions include the sigmoid function, ReLU (Rectified Linear Unit), and tanh.
3. **Hidden Layers**: A neural network can have one or more hidden layers between the input and output layers. Each hidden layer consists of neurons that process the outputs of the previous layer. The process of calculating the weighted sum, applying the activation function, and passing the result to the next layer is repeated for each hidden layer. This process is often referred to as forward propagation and it proceeds throughout the network to produce a prediction at the output layer.
4. **Output Layer**: The final hidden layer is followed by the output layer. This layer produces the network's predictions or outputs. The number of neurons in the output layer depends on the task at hand. For example, in a binary classification task, there might be one neuron with a sigmoid activation function that outputs a probability value. In a multi-class classification task, the number of neurons in the output layer corresponds to the number of classes, often with a softmax activation function to produce class probabilities.
5. **Loss Function**: Once the network generates predictions, these predictions are compared to the actual target values from the training data using a loss function. The loss function quantifies the error between the predicted and actual values.
6. **Backpropagation and Optimization**: The goal of the network is to minimize the loss function, effectively reducing the prediction error. This is done through a process called backpropagation. The gradients of the loss with respect to the

weights and biases are calculated using the chain rule of calculus. These gradients indicate how much each weight and bias needs to be adjusted to minimize the loss. Optimization algorithms like gradient descent are used to update the weights and biases iteratively, nudging them in the direction that reduces the loss.

7. **Training and Learning**: The neural network is trained by repeatedly feeding training data through the network, calculating predictions, calculating the loss, and updating the weights and biases using backpropagation and optimization. This process allows the network to learn patterns and relationships in the training data.

By adjusting the architecture, activation functions, and other parameters, neural networks can be adapted to a wide range of tasks, making them a versatile tool in machine learning and artificial intelligence.

Tutorial Question

Mention some of the parameters of a neural network that can be adjusted to obtain better results.

6.2 Deep Learning

In a neural network architecture with a single hidden layer, only one type of transformation can be applied to the input data for making predictions. With more hidden layers, more transformations can be applied and more complex patterns in the data can be learnt and use to make better predictions. This is the idea behind deep learning.

6.1.2 Types of Deep Learning Networks

There are several types of deep learning networks, but we will attempt to cover the most popular types. Most of the existing networks are form of one of these types. We also mention some notable examples of each type.

1. **Deep Belief Networks (DBNs)**: DBNs are a type of generative neural network architecture that consists of multiple layers of stochastic, latent variables

(hidden layers) along with visible layers. They are composed of an unsupervised, generative pre-training phase followed by a supervised fine-tuning phase. During pre-training, each layer is trained as a restricted Boltzmann machine (RBM), a type of energy-based probabilistic model. The learning process involves adjusting the weights to minimize the difference between the model's generated data and the input data. Once pre-training is complete, the network can be fine-tuned using backpropagation and labeled data for supervised tasks. DBNs have been used in various applications, including collaborative filtering, anomaly detection, and feature learning. DBNs were popular in the early 2010s, but their prominence has diminished in recent years due to the rise of other deep learning architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

2. **Convolutional Neural Networks (CNNs)**: CNNs are primarily used for image recognition and computer vision tasks. They leverage a hierarchical structure of convolutional layers to automatically learn and extract features from images. They excel at handling spatial relationships in images, making them ideal for tasks like object detection, image classification, and facial recognition. Some popular CNN architectures include AlexNet, the VGG family of architectures, the ResNet family of architectures, MobileNet, etc.
3. **Recurrent Neural Networks (RNNs)**: RNNs are designed to work with sequential data, such as time series or natural language text. They have loops that allow information to persist across different time steps, enabling them to model sequences and capture temporal dependencies. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks are specialized variations of RNNs that mitigate the vanishing gradient problem and are particularly effective for tasks like language modeling, speech recognition, and sentiment analysis.
4. **Generative Adversarial Networks (GANs)**: GANs consist of two neural networks: a generator and a discriminator. They work in tandem to generate synthetic data that closely resembles real data. GANs are widely used in image synthesis, style transfer, and generating realistic images, audio, and text. They have also been employed in creating deepfake content (i.e., artificial images or videos that appear real).

5. **Autoencoders**: Autoencoders are used for dimensionality reduction and feature learning. They consist of an encoder network that compresses input data into a lower-dimensional representation (encoding), and a decoder network that reconstructs the original data from the encoding. Autoencoders are applied in tasks like image denoising, anomaly detection, and unsupervised feature learning.
6. **Attention Networks**: Attention mechanisms have become a crucial component in various deep learning architectures, such as transformers. They enable models to focus on specific parts of input data, enhancing their ability to handle long sequences and complex relationships. Transformers power models like BERT, GPT, and T5, which excel in language understanding, generation, and translation.

These are just a few examples of the diverse types of deep learning networks. Each network type has its strengths and weaknesses, making them suitable for different tasks and domains. The field of deep learning is constantly evolving, with new network architectures and improvements being developed to tackle a wide range of challenges. So, while this list seems to present a wide category of networks, it is not intended to be exhaustive.

6.1.2 Applications of Deep Learning

The ability of deep network architectures to learn many complex patterns from huge volumes of data has made them widely applicable in many domains some of which are discussed as follows.

1. <u>Image Recognition and Classification</u>: Deep learning has revolutionized computer vision by enabling systems to identify and categorize objects within images and videos. Applications range from self-driving cars to medical image analysis.
2. <u>Natural Language Processing (NLP)</u>: Deep learning techniques power language models that can understand, generate, and translate human language. Chatbots, sentiment analysis, language translation, and text summarization are some examples.
3. <u>Speech Recognition and Generation</u>: Deep learning is pivotal in developing accurate speech recognition systems like virtual assistants (e.g., Siri, Google

Assistant) and speech-to-text technologies. It's also used for generating human-like speech in applications like audiobooks and voice assistants.

4. Autonomous Systems: Deep learning is integral to creating self-driving cars and drones, enabling them to perceive their environment, make decisions, and navigate safely.
5. Healthcare: Deep learning plays a significant role in medical image analysis, including the detection of diseases from medical images like X-rays, MRIs, and CT scans. It's also used in drug discovery, personalized medicine, and predictive analytics.

Tutorial Question

Discuss any other areas of application of deep learning apart from the ones discussed above.

Summary

In this lesson, we have discussed neural networks and their applications. We have learnt the fundamentals of simple neural networks and extended the knowledge to deep networks. Not that neural networks are used primarily for supervised learning (classification and regression), but more recently, they're also being applied to self-supervised learning. Neural networks have become very powerful due to the availability of huge volumes of data and more powerful hardware.

Self-Assessment Questions

1. What differentiates neural networks from other learning algorithms?
2. What factors are responsible for the recent successes of neural networks?
3. What are the processes used for passing values back and forth within a neural network?
4. Compare and contrast a biological and an artificial neuron.
5. What do you understand by the vanishing/exploding gradient problem in neural networks?

Study Session 7: Model Evaluation

Introduction

Having learnt how to build machine learning models and use them for prediction, it is important to know how to evaluate them. As a matter of fact, we cannot build successful models without an appropriate evaluation model being put in place during the model training. The machine learning development process is indeed an iterative what that involves building models, evaluating them and using the evaluation feedback to improve the model in the next iteration. This iteration stops when the Machine Learning (ML) engineer decides that the model built is good enough, based on pre-defined metrics. In this chapter, we will learn about the various components involved in evaluating ML models and how they contribute to improve the overall model.

Learning Outcomes

At the end of this lesson, you should:

1. Understand model evaluation and its importance.
2. Know the performance metrics for supervised and unsupervised learning.
3. Understand the different validation protocols available.
4. Understand overfitting and underfitting.

7.1 The Importance of Model Evaluation

Model evaluation is a critical step in the machine learning pipeline that involves assessing the performance and generalization ability of a trained model. It answers the fundamental question: how well does our model perform on unseen data? While the process of training a machine learning model involves optimizing it to fit the training data, the true test of a model's usefulness lies in its performance on new, unseen data.

Some fundamental points in model evaluation are explained here:

Generalization: The ultimate goal of machine learning is to create models that can generalize well to unseen data. This ensures that the model can make accurate predictions on real-world instances beyond the training dataset.

Avoiding Overfitting: Model evaluation helps identify whether a model has overfit the training data. Overfitting occurs when a model learns the noise in the training data rather than the underlying patterns. Evaluation helps ensure that the model has captured the relevant patterns and is not making overly specific predictions.

Decision Making: Machine learning models are often used to make important decisions in various domains, such as healthcare, finance, and autonomous driving. Proper evaluation ensures that these decisions are based on reliable predictions.

Model evaluation is not just a one-time task; it is, indeed, an iterative process that goes hand in hand with model improvement. Here are some points of interest between model evaluation and model improvement.

Feedback Loop: The insights gained from evaluating a model's performance can guide the iterative improvement of the model. If the evaluation reveals shortcomings, it's an opportunity to refine the model, adjust hyperparameters, and consider feature engineering.

Validation Set: During model development, a validation set is often used for intermediate evaluation. This helps in tuning hyperparameters and comparing different model versions before finally testing on a separate test set.

Hyperparameter Tuning: Model evaluation plays a crucial role in hyperparameter tuning. By experimenting with different hyperparameter settings and evaluating their impact on performance, you can fine-tune a model for optimal results.

Monitoring in Production: Even after deploying a model, evaluation does not stop. Models in production need continuous monitoring and re-evaluation to ensure their performance remains consistent over time.

Learning from Mistakes: Model evaluation can reveal where a model is making errors. Understanding these errors can lead to insights about the data, model assumptions, or areas of improvement.

Having understood the importance of model evaluation and its connection to model improvement, we will go on to discuss some important metrics which may be used to evaluate the performance of machine learning models.

7.2 Performance Metrics

Performance metrics are quantitative measures used to assess the quality of predictions made by machine learning models. These metrics provide insights into how well the model is performing and help in comparing different models or configurations. Different types of machine learning tasks, such as classification and regression, require specific metrics to evaluate their performance accurately. We will look into some common types of metrics for classification and regression tasks.

7.2.1 Classification evaluation metrics

Here are some of the most popular performance metrics for classification tasks.

1. **Accuracy**: Accuracy is one of the most basic metrics and represents the ratio of correctly predicted instances to the total instances in the dataset. Sometimes, this ration can be multiplied with 100 to express percentage accuracy. Accuracy can be misleading in imbalanced datasets, where one class dominates the other.
2. **Precision, Recall, and F1-Score**: **Precision** is the ratio of true positive predictions to the total predicted positives. It measures how many of the predicted positive instances are actually correct. **Recall** (Sensitivity) is the ratio of true positive predictions to the total actual positives. It measures how many of the actual positive instances were predicted correctly. **F1-Score** is the harmonic mean of precision and recall, providing a balance between the two metrics. These metrics are crucial when the cost of false positives or false negatives is significant, as they focus on different aspects of the model's performance.
3. **Receiver Operating Characteristic (ROC) Curve**: The ROC curve illustrates the trade-off between the true positive rate (sensitivity) and the false positive rate as the discrimination threshold varies. It helps in selecting an appropriate threshold based on the desired balance between true positives and false positives.
4. **Area Under the Curve (AUC)**: AUC represents the area under the ROC curve and provides a single scalar value that summarizes the performance of a

classifier across various threshold settings. A higher AUC indicates better overall classification performance.

5. **Confusion Matrix**: A confusion matrix is a table that visualizes the performance of a classification algorithm. It shows the count of true positives, true negatives, false positives, and false negatives. From the confusion matrix, metrics like precision and recall can be calculated.

We will now consider a simple example to illustrate the use of some of these metrics. We will evaluate the performance of an ML model that classifies emails as either spam or non-spam emails.

Suppose Table 7.1 is the confusion matrix of the spam classification model.

Table7.1: Confusion matric for a spam classification model

		True class	
		Spam	Non-Spam
Predicted class	Spam	**40 (TP)**	**3 (FP)**
	Non-Spam	**10 (FN)**	**120 (TN)**

We will now calculate some evaluation metrics based on these values:

Precision: TP / (TP + FP) = 40 / 43 = 0.9302

Recall: TP / (TP + FN) = 40 /50 = 0.8

Accuracy: (TP + TN) / (TP +TN +FP +FN) = 160 / 173 = 0.9249

F1-Score: 2 * Precision * Recall / (Precision + Recall) = 2 * 0.9302 * 0.8 / (0.9302 + 0.8) = 1.4 /1.7302 = 0.8902

From the example above, you can clearly see that the accuracy score overestimate the model's performance simply because the number of negative classes is thrice the number of positive classes, but the F1-score balances this out.

7.2.2 Regression evaluation metrics

1. **Mean Absolute Error (MAE)**: MAE calculates the average absolute difference between the predicted and actual values. It gives an idea of how far off the predictions are on average.
   $\sum(|y - \hat{y}|)/N$,
   where y is the actual value, $\hat{y}$ is the predicted value and N is total number of instances.
2. **Mean Squared Error (MSE)**: MSE calculates the average squared difference between predicted and actual values. It emphasizes larger errors more than MAE.
   $\sum(y - \hat{y})^2/N$,
3. **Root Mean Squared Error (RMSE)**: RMSE is the square root of the MSE. It has the same unit as the target variable, making it more interpretable.
   $\sqrt{\sum(y - \hat{y})^2/N}$,
4. **Coefficient of Determination (R^2)**: R^2 measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides an indication of how well the predictions fit the actual data. A value closer to 1 indicates a better fit.

These metrics offer a comprehensive view of a model's performance in both classification and regression tasks. Understanding the nuances of each metric helps data scientists and machine learning practitioners choose the appropriate evaluation criteria for their specific problem.

Tutorial Question

Plot the ROC and determine the AUC of the model whose confusion matrix is given in Table 7.1. You are free to choose assumed threshold values.

7.2.3 Unsupervised Learning

In an unsupervised setting, where you are dealing with tasks like clustering and dimensionality reduction, the performance metrics focus on different aspects of the data's structure and the model's effectiveness in capturing it. Unsupervised learning evaluation is often more qualitative and context-dependent compared to supervised settings. These metrics help practitioners assess the effectiveness of clustering algorithms and dimensionality reduction techniques, providing guidance on the suitability of these methods for the given data and problem.

Here are a few performance metrics commonly used in unsupervised learning:

1. **Silhouette Score**: This is often used to evaluate the performance of a clustering algorithm. The silhouette score measures how similar an object is to its own cluster compared to other clusters. It ranges from -1 to 1. Higher values indicate well-defined clusters, where instances are closer to their own cluster and farther from neighboring clusters.
2. **Explained Variance Ratio**: This is often used for evaluating a dimensionality reduction model such as Principal Component Analysis (PCA). In PCA, each principal component captures a portion of the total variance in the data. The explained variance ratio of a component is the proportion of the dataset's total variance it explains.
3. **Manifold Learning Visualization**: Visualization techniques like t-SNE (t-Distributed Stochastic Neighbor Embedding) and UMAP (Uniform Manifold Approximation and Projection) project high-dimensional data into lower dimensions while preserving the local structure. The quality of visualization can provide insights into the effectiveness of the dimensionality reduction technique.

7.3 Cross-Validation

Cross-validation is a technique used to assess the performance of a machine learning model by splitting the available data into multiple subsets for training and testing. This approach helps provide a more robust estimate of a model's performance than a single train-test split. It's particularly useful for addressing issues like overfitting and obtaining a better understanding of how the model generalizes to new, unseen data. Cross-validation is more suited to small to medium-sized datasets as it can be impractical for very large datasets.

We will now explore some common methods of performing cross-validation.

A. **K-Fold Cross-Validation**:

i. Process: The dataset is divided into *k* subsets or "folds" of approximately equal size. The model is trained *k* times, with each fold serving as the validation set once and the remaining folds as the training set. Performance metrics are then averaged across all iterations to assess the model's generalization ability.
ii. Benefits: It provides a more reliable estimate of a model's performance by using different subsets of data for both training and validation. It reduces the impact of randomness in the train-test split.

B. **Stratified Cross-Validation**:
i. Purpose: This is especially useful for classification tasks with imbalanced class distributions. It ensures that each fold maintains the same class distribution as the original dataset.
ii. Process: Stratification ensures that each fold has a representative mix of classes, preventing one fold from having significantly fewer instances of a particular class.

C. **Advantages and Limitations**:

Advantages

a. It provides a more accurate estimate of a model's performance by using multiple validation sets.
b. It helps in selecting optimal hyperparameters, as the model is evaluated on different data subsets.
c. It reduces the risk of overfitting by validating on different data than the training set.

Limitations

a. It can be computationally expensive, especially with large datasets or complex models.
b. It does not entirely eliminate the effects of data distribution variations.

D. **Hyperparameter Tuning and Cross-Validation**:
a. Grid Search and Cross-Validation: Cross-validation is often used in combination with grid search for hyperparameter tuning. Different

hyperparameter combinations are evaluated using cross-validation, and the combination with the best average performance is selected.

b. Nested Cross-Validation: To avoid overfitting the hyperparameters to the validation set, a nested cross-validation is used. The outer loop performs model evaluation, while the inner loop performs hyperparameter tuning using cross-validation.

Cross-validation is a crucial technique for model assessment that aids in obtaining a more accurate and unbiased estimate of a model's performance. It helps mitigate the impact of randomness, prevents overfitting, and guides hyperparameter tuning for better model generalization, but it can be computationally expensive.

7.4 Overfitting and Underfitting

Overfitting occurs when a model becomes too complex and fits the training data's noise and fluctuations rather than the underlying patterns. As a result, the model's performance is excellent on the training data but poor on unseen data. Overfitting can be thought of as the model "memorizing" the training data. Underfitting happens when a model is too simple to capture the true relationships within the data. It fails to capture the patterns and trends, resulting in poor performance both on the training data and new data. Underfitting can be thought of as the model oversimplifying the problem.

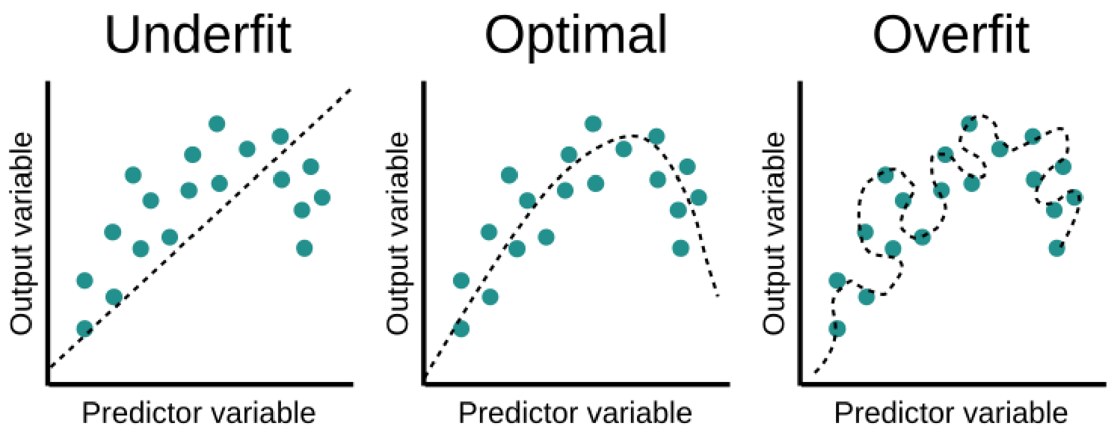

Figure 7.1: Illustration of underfit, optimal and overfit models

Figure 7.1 shows three plots indicating underfitting, optimal fitting and overfitting. In the underfit model, you will observe that model cannot properly fit to the data because it models a linear relationship while the data, in fact, exhibits a non-linear relationship with the output variable. In the overfit model, the model is too complex and fits to every data point (which may include outliers), thus, it performs poorly on new/unseen data. The middle plot shows an optimal fit which captures the non-linearity of the data rather than every sample point.

The Bias-Variance Trade-off

The bias-variance trade-off refers to the balance between model complexity and the ability to generalize to new data. High bias occurs with overly simplistic models, leading to underfitting as the model cannot capture the underlying patterns in the data. High variance occurs with overly complex models, leading to overfitting. The model captures noise along with the patterns. As shown in figure 7.1, the ideal model complexity lies in the middle of this trade-off, achieving a balance between bias and variance.

Techniques to Address Overfitting and Underfitting

<u>Overfitting</u>

a. Use simpler models or reduce the number of features to decrease model complexity.
b. Apply regularization techniques that penalize overly complex models.
c. Increase the amount of training data to help the model generalize better.

<u>Underfitting</u>

a. Use more complex models or techniques that can capture intricate relationships in the data.
b. Add more relevant features to provide the model with more information.
c. Ensure the model has enough capacity to learn the underlying patterns.

Learning Curves

A very useful technique for detecting overfitting or underfitting in ML models is a learning curve. Learning curves show the model's performance on both training and validation/test data as the amount of training data increases. These curves can help identify whether a model is overfitting or underfitting as shown in Figure 7.2.

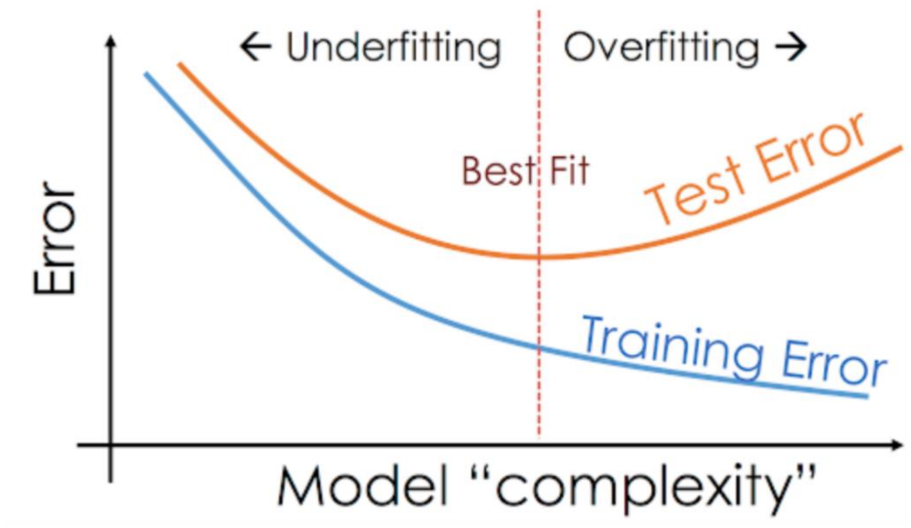

Figure 7.2: Underfitting, optimal fitting and overfitting in a learning curve

As observed in the figure, the optimal fit is obtained when the difference between the training and test errors is minimal as the model complexity increases.

- If both training and validation errors are high, it indicates underfitting (as seen on the left side of Figure 7.2).
- If training error is low and validation error is high, it suggests overfitting (as seen on the right side of Figure 7.2).
- Convergence of training and validation error at low levels indicates a well-fitted model (as seen in the middle of Figure 7.2).

Understanding overfitting and underfitting helps in model selection, hyperparameter tuning, and improving model performance. It is crucial to strike a balance between model complexity and the model's ability to generalize to new data.

Summary

In this chapter, you have learnt about model evaluation and must have realized how important this is. Model evaluation must be carried out appropriately, otherwise, we risk overestimating or underestimating the performance of the model. For this reason, we have discussed performance metrics and the specific machine learning situations to which they are best suited. We have also discussed cross-validation and its importance for appropriate model selection. Finally, we discussed overfitting and underfitting and learned how to identify and correct them.

Self-Assessment Questions

1. What is the key difference between overfitting and underfitting in the context of machine learning models? How does each affect a model's performance on training and unseen data?
2. Describe the purpose and process of K-Fold Cross-Validation. How does it address the limitations of a single train-test split in evaluating a model's performance?
3. In a classification problem, why might accuracy be a misleading performance metric, especially in the presence of imbalanced class distributions? Provide an example scenario to illustrate this.
4. What is the significance of model evaluation in the context of machine learning? How does continuous evaluation contribute to the overall improvement of machine learning models?
5. If a model's training error is significantly lower than its validation error, what does this suggest about the model's performance? What steps can be taken to address this situation?

Study Session 8: Ethics and Bias in Machine Learning

Introduction

How would you feel if a face recognition system flags you as a criminal because of your gender or ethnicity? This is an example of bias in Machine Learning (ML) algorithms which is often introduced by the data used to train the models. Therefore, care must be taken in modelling ML solutions to problems to avoid introducing biases and unfairness into the algorithm. In this study session, we will discuss the different types of bias that can be found in Machine Learning and how to mitigate them.

Learning Outcomes

At the end of this lesson, you should

1. Understand bias and fairness in ML
2. Understand the different types of bias in ML
3. Understand key concepts for mitigating bias in ML

8.1 The Meaning and Impact of Bias in Machine Learning

Bias in machine learning refers to the presence of systematic and unfair inaccuracies in the predictions and decisions made by algorithms due to various factors, including skewed data, flawed algorithms, and human biases. Biased models can lead to discriminatory outcomes, reinforcing existing social biases, and perpetuating inequalities. Moreover, biased algorithms can erode public trust in AI systems, hindering their adoption and acceptance. Biases can be in terms of gender, ethnicity, social status, age, etc. An example of a gender-biased ML system can be seen when an ML-powered hiring system favours one gender over the other, perhaps because it was trained with data that suggests that one gender is more suited to a job than the other. Another example will be when an ML-powered law enforcement system unjustly flags people of certain ethnicities as criminals.

Fairness is the principle of treating all individuals and groups fairly, without discrimination or favoritism. Ensuring fairness is crucial to preventing unjust outcomes and maintaining ethical AI systems. Addressing bias is essential for creating ethical and effective machine learning models. Failure to mitigate bias can lead to serious consequences, including legal and ethical challenges, as well as negative societal impact. Achieving fairness and reducing bias supports the development of equitable AI systems that contribute positively to society by avoiding harmful biases. By understanding the concept of bias in machine learning and recognizing its impact, we can delve deeper into the various facets of bias and explore strategies to address and mitigate bias effectively.

8.2 Types of Bias

1. **Data bias**, also known as sampling bias, occurs when the training data used to build a machine learning model is not representative of the real-world population it aims to serve. For example, if a medical diagnosis model is trained primarily on data from a specific demographic group (e.g., a particular age group), it might perform poorly on other groups. Data bias can arise due to various reasons, such as uneven data collection, historical inequalities, or sampling methods that inadvertently exclude certain groups.
2. **Algorithmic bias** refers to biases that emerge from the design or structure of the machine learning algorithms themselves. For instance, if an algorithm gives preferential treatment to certain groups in its decision-making process, it may lead to unequal treatment. Bias can stem from the training data, the algorithm's architecture, or the features used in the model. Insufficiently diverse training data or inappropriate features can amplify algorithmic bias. Algorithmic bias can manifest in different forms, including:
    a. *Group Bias*: When the algorithm favors one group over others, leading to disparities in outcomes.
    b. *Cognitive Bias*: Reflecting human cognitive biases, the algorithm might generalize patterns from biased data, perpetuating those biases in predictions.
3. **Human Bias** can inadvertently be introduced during the data collection, preprocessing, and model development stages. Biased human judgments and interpretations can propagate through the system. Humans involved in labeling data or developing models can introduce their own biases, consciously or unconsciously. This could involve subjective decisions about what data to include, how to label it, or what features to consider.

Understanding these types of biases helps in recognizing where biases can originate within a machine learning process, facilitating the development of strategies to detect, prevent, and mitigate bias effectively.

8.3 Mitigating Bias in Machine Learning Models

We will now proceed to discuss some steps for mitigating bias in ML models. This list is not exhaustive and there are many other methods that could be employed depending on the type of bias suspected and the nature of the problem at hand.

Data Preprocessing

Data preprocessing is an extremely important step for mitigating bias in ML models. As discussed in study session 2, data preprocessing, is significantly impactful on the quality of an ML model. Some data preprocessing steps that may be employed to fight bias include:

- *Data augmentation and re-sampling*: Augmenting the dataset with synthetic samples or re-sampling techniques can balance the representation of different groups, reducing data bias.
- *Removing biased features*: Identifying and excluding features that contribute to bias can help mitigate its impact on the model's predictions.

Algorithmic Techniques

Apart from data preprocessing, techniques can also be used to tune learning algorithms to reduce their sensitivity to biases. Here are some of such techniques:

- Fairness-aware algorithms explicitly incorporate fairness considerations into their optimization process. They aim to provide equitable outcomes for different groups while making predictions.
- Regularization and re-weighting techniques like adding fairness-related regularization terms or adjusting sample weights during training can also encourage an ML model to pay more attention to underrepresented groups.

Transparency and Explainability

Advances in Deep Learning and their versatility in sensitive fields such as healthcare have necessitated the explainability of such models. Although, this can be difficult, it is required to build trust in systems. Developing models that offer transparency in their decision-making process can help identify the factors influencing bias. This enables stakeholders to understand why certain predictions are made. Tools that provide insights into how certain features contribute to biased predictions can also aid in diagnosing and addressing bias.

In conclusion, by implementing these strategies to address bias, machine learning practitioners and developers can work towards building fairer and more ethical AI systems that deliver accurate predictions while minimizing discriminatory outcomes.

Summary

In this study session, we have discussed the topic of bias and fairness in Machine Learning. Note that there are also legal frameworks which enforce some of these requirements, but their study is beyond the scope of this course. As an ML practitioner, you must always strive for fairness and transparency in your ML models. This can be very difficult in many cases, but we must keep taking steps towards it because it is the right thing to do.

Self-Assessment Questions

1. What do you understand by data bias in machine learning, and what are its potential consequences? Provide an example to illustrate your answer.
2. Explain the difference between algorithmic bias and human bias in machine learning. Provide examples of each type of bias and how they can impact model outcomes.
3. How can transparency and interpretability of machine learning models help in addressing bias? Provide examples of how model interpretability can be beneficial for identifying and mitigating bias.

4. Describe two techniques for addressing bias in machine learning algorithms. Provide a brief explanation of how each technique works and how it contributes to creating fairer models.

Study Session 9: Introduction to ML Tools and Libraries

Introduction

In this last study session, we will be studying useful libraries and tools for solving Machine Learning (ML) problems. There are several programming languages which could be used for tackling ML. However, Python has become widely used for many reasons, including its simplicity and a vast collection of useful libraries. Therefore, in this session, our focus will be on Python libraries which can help us solve many of the Machine Learning tasks discussed in this manual.

Learning Outcomes

At the end of this lesson, you should

1. Know the basic libraries available for ML in Python and their functionalities.
2. Know the Python libraries useful for data visualization.
3. Know how to use the Python ML libraries to solve simple ML tasks.

9.1 Basic ML Libraries in Python

There are several popular Python libraries for machine learning that are widely used by researchers, data scientists, and developers to build and train machine learning models. Here are some of the most well-known ones:

1. **Scikit-learn**: Scikit-learn is one of the most widely used libraries for classical machine learning algorithms. It provides a simple and consistent interface for various machine learning tasks like classification, regression, clustering, dimensionality reduction, and more. It's known for its ease of use and strong documentation.
2. **TensorFlow**: Developed by Google, TensorFlow is an open-source machine learning framework that supports both deep learning and traditional machine learning. It offers a flexible architecture for building neural networks and other machine learning models, along with tools for deploying models in various environments.
3. **PyTorch**: PyTorch is another powerful deep learning framework that has gained a lot of popularity. It's known for its dynamic computation graph, making it

easier to debug and experiment with models. PyTorch also provides great support for GPU acceleration and research-oriented development.

4. **Keras**: Originally a separate library, Keras has become an integral part of TensorFlow. It provides a high-level API for building and training neural networks. Keras is designed to be user-friendly, making it a good choice for beginners and those who want to quickly prototype deep learning models.
5. **XGBoost**: XGBoost (eXtreme Gradient Boosting) is an efficient and highly optimized gradient boosting library. It's particularly popular for structured/tabular data and has won numerous Kaggle competitions due to its strong performance.
6. **LightGBM**: LightGBM is another gradient boosting framework that's known for its speed and efficiency. It's designed to handle large datasets and provides better performance for certain use cases compared to other boosting libraries.
7. **CatBoost**: CatBoost is a gradient boosting library developed by Yandex that's designed to handle categorical features well without requiring extensive preprocessing. It automatically handles categorical data, which can save time and effort during feature engineering.
8. **Pandas**: While not a machine learning library itself, Pandas is an essential data manipulation and analysis library. It provides powerful data structures and functions to help clean, transform, and preprocess data before feeding it into machine learning models.
9. **NLTK (Natural Language Toolkit)**: NLTK is a library for working with human language data (text). It provides tools and resources for tasks like tokenization, stemming, part-of-speech tagging, and more, making it a valuable resource for natural language processing (NLP) projects.
10. **spaCy**: Similar to NLTK, spaCy is another NLP library that's known for its speed and efficiency. It's often used for more advanced NLP tasks like named entity recognition, dependency parsing, and text classification.

These libraries form the foundation for many machine learning and deep learning projects, and they each have their own strengths and use cases. Depending on your specific needs and the nature of your project, you might find certain libraries more suitable than others.

9.2 Data Visualisation Tools

There are several Python libraries that are widely used for data visualization, enabling you to create various types of plots, charts, and graphs to effectively communicate insights from your data. Here are some popular ones:

1. **Matplotlib**: Matplotlib is one of the most widely used and versatile libraries for creating static, interactive, and animated visualizations in Python. It offers a wide range of plot types and customization options, making it suitable for creating a variety of charts and graphs.
2. **Seaborn**: Seaborn is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical visualizations. It's particularly well-suited for tasks involving statistical analysis and exploratory data visualization.
3. **Plotly**: Plotly is a library that allows you to create interactive and web-ready visualizations. It supports a variety of plot types and can generate interactive graphs that you can embed in web applications or notebooks.
4. **Bokeh**: Bokeh is another interactive visualization library that's designed for creating interactive, web-ready plots. It provides a high-level interface for creating complex visualizations with interactive elements, such as tooltips, zooming, and panning.
5. **Altair**: Altair is a declarative statistical visualization library that's focused on simplicity and ease of use. It allows you to create visualizations by defining the plot components using a concise and intuitive syntax.
6. **Pandas Plotting**: Pandas, the data manipulation library, also offers built-in plotting capabilities. You can create basic plots directly from Pandas DataFrames using methods like "`.plot()`."
7. **Holoviews**: Holoviews is a high-level library that allows you to create interactive visualizations with minimal code. It's designed to work seamlessly with other plotting libraries like Matplotlib, Bokeh, and Plotly.
8. **Ggplot**: Ggplot is a Python port of the popular R package ggplot2. It follows the grammar of graphics concept, making it easier to create complex and well-designed visualizations.

9. **WordCloud**: WordCloud is a library specifically designed for creating word clouds, which are visual representations of word frequency in a text corpus.
10. **NetworkX**: If you're dealing with network data and graphs, NetworkX is a library that allows you to create, analyze, and visualize complex networks and graphs.

These libraries offer a range of options for creating static and interactive visualizations, and the choice depends on your specific requirements, the complexity of your data, and the type of visualization you want to create.

9.3 ML Case Studies

As a final discussion in this manual, we will walk though some Machine Learning examples with code examples in Python. Note that care has been taken to ensure the supplied codes are correct, but you may still need to carry out a few fixes to get them to work depending on your programming environment and library versions. We will consider examples of classification and regression tasks.

9.3.1 Classification

Let's consider a case study of a machine learning classification task where we'll use the famous Iris dataset to classify iris flowers into different species based on their sepal and petal measurements. The iris dataset contains 50 samples of 3 flower classes: Iris Setisa, Iris Virginica and Iris Versicolor. Each sample is represented with 4 features: sepal length and width, petal length and width. We'll go through each step of the process, from loading the data to evaluating the model.

Step 1: Import Libraries: Import libraries needed to perform the entire ML task

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

Step 2: Load and Explore the Data: Load the data into a Pandas dataframe and visualise the first few rows. This helps you gain an idea of what the data looks like. Not that this simple dataset is pre-loaded into the sklearn library for ease of use, but this will not always be the case with real-world tasks.

```
# Load the Iris dataset
from sklearn.datasets import load_iris
iris = load_iris()

# Create a Pandas DataFrame from the dataset
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
columns=iris['feature_names'] + ['species'])

# Display the first few rows of the DataFrame
print(iris_df.head())
```

Step 3: Preprocess the Data: Don't forget how important data preprocessing is. Here, we basically split the data into feature (independent) variables and target (dependent) variables. We further split the data into training and test sets and standardize the features. The training set would be used to train the model and the test set would be used to test the model's performance. This is a very small dataset and we're just using this to illustrate how things work. In the real world, your datasets will be much larger and more complex, and you'll need to have the conventional three data splits (training, validation and test sets) in order to create a good model.

```
# Splitting the data into features (X) and target (y)
X = iris_df.drop('species', axis=1)
y = iris_df['species']

# Split the data into training and testing sets
X_train,    X_test,    y_train,    y_test    =    train_test_split(X,    y,
test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 4: Build and Train the Model: In this case, we have chosen the KNN classifier as our ML algorithm. As you can see, we also had to make a choice of the number of neighbours in the KNN algorithm. This is just one of the many hyperparameters you may have to set values for when using a model. Any other classifier could work equally well here. You should try out other classifiers and compare their performance. Go through the Scikit-learn (sklearn) library documentation to see how to use other classifiers and don't forget to import them appropriately.

```
# Initialize the K-Nearest Neighbors classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)

# Train the classifier on the scaled training data
knn_classifier.fit(X_train_scaled, y_train)
```

Step 5: Make Predictions: Having trained your model on the dataset, you should make predictions on the test set.

```
# Predict the labels for the test set
y_pred = knn_classifier.predict(X_test_scaled)
```

Step 6: Evaluate the Model: Now evaluate the performance of your model on the test set by checking for simple metrics like accuracy. You can also generate a classification report which gives you the precision, recall and F1-score. As you can see, you don't have to manually calculate these values, they're already implemented in the sklearn library, you only need to know how to call them appropriately.

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
class_report = classification_report(y_test, y_pred,
target_names=iris['target_names'])
print("Classification Report:\n", class_report)
```

In this case study, we loaded the Iris dataset, preprocessed the data by splitting it into training and testing sets, standardized the features, built a K-Nearest Neighbors classifier, made predictions, and evaluated the model's performance using accuracy and a classification report. Keep in mind that this is a basic example. Depending on the complexity of your classification task, you might need to experiment with different algorithms, hyperparameters, and feature engineering techniques to achieve the best results.

9.3.2 Regression

Now, let's consider a case study of a regression task where we'll use the Boston Housing dataset to predict the median value of owner-occupied homes in Boston (a city in the United States). As before, we'll go through each step of the process, from loading the data to evaluating the model.

Step 1: Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Step 2: Load and Explore the Data:

```
# Load the Boston Housing dataset
from sklearn.datasets import load_boston
boston = load_boston()



# Create a Pandas DataFrame from the dataset
boston_df = pd.DataFrame(data=boston['data'],
columns=boston['feature_names'])
boston_df['MEDV'] = boston['target']



# Display the first few rows of the DataFrame
print(boston_df.head())
```

Step 3: Preprocess the Data

```
# Splitting the data into features (X) and target (y)
X = boston_df.drop('MEDV', axis=1)
y = boston_df['MEDV']



# Split the data into training and testing sets
X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,
test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 4: Build and Train the Model

```
# Initialize the Linear Regression model
linear_reg = LinearRegression()
```

```
# Train the model on the scaled training data
linear_reg.fit(X_train_scaled, y_train)
```

Step 5: Make Predictions

```
# Predict the target values for the test set
y_pred = linear_reg.predict(X_test_scaled)
```

Step 6: Evaluate the Model: Note that the evaluation metrics used here are different from the ones used for the classification task.

```
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)


# Calculate R-squared
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

In this case study, we loaded the Boston Housing dataset, preprocessed the data by splitting it into training and testing sets, standardized the features, built a Linear Regression model, made predictions, and evaluated the model's performance using Mean Squared Error (MSE) and R^2. Remember that regression tasks can involve more complex models and feature engineering, and you might need to experiment with different algorithms and techniques to achieve the best results for your specific regression problem.

Summary

In this section, we have briefly described some Python libraries popularly used for Machine Learning tasks. These libraries make it easier to quickly develop models so that you can quickly build a working model and then focus on improving. We have also provided two practical case studies of solving classification and regression tasks. This code can be used as a template to stat you up on similar ML tasks while you work to fit it to the peculiarities of the task. You should always aim to get a working model first and then start improving several aspects of the code to get an improved model.

Self-Assessment

1. Which Python library is widely used for classical machine learning algorithms like classification, regression, and clustering?
2. Which deep learning framework provides a dynamic computation graph and is known for its research-oriented development?
3. Which Python library is designed to create interactive and web-ready visualizations, allowing you to embed graphs in web applications?
4. In the context of the Iris dataset, what does the term "species" refer to?
5. Which Python library is known for its grammar of graphics concept and allows you to create complex and well-designed visualizations?

References

Brenninkmeijer, B. (2019). *On the Generation and Evaluation of Tabular Data using GANs* (Issue October 2019). Radboud University.

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*. https://doi.org/10.1080/15228053.2020.1766224

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. https://doi.org/10.1007/978-3-540-75488-6_2

Pritha Bhandari. (2022, October). *Data Cleaning | A Guide with Examples & Steps. Scribbr.* Scibbr. https://www.scribbr.co.uk/research-methods/data-cleaning/

Rushikesh Pupale. (2018). *Support Vector Machines (SVM) - An Overview*. Towards Data Science. https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989#:~:text=SVM%20or%20Support%20Vector%20Machine,separates%20the%20data%20into%20classes.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 535–554. https://doi.org/10.1147/rd.441.0206

Watson, N. (2012). *Diapers, Beer, and Data Science in Retail*. Contemporary Analytics (CAN). https://canworksmart.com/diapers-beer-retail-predictive-analytics/#:~:text=The legend says that a,have beer in their carts.

Online Resources

Harshil Patel. (2021, August 30). *What is Feature Engineering — Importance, Tools and Techniques for Machine Learning*. Towards Data Science. https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10

Jason Brownlee. (2020, August 15). *Discover Feature Engineering, How to Engineer Features and How to Get Good at It*. Data Preparation. https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

Natasha Sharma. (2023, April 12). *K-Means Clustering Explained*. Neptune.AI. https://neptune.ai/blog/k-means-clustering#:~:text=%E2%80%9CK%2Dmeans%20clustering%20is%20a,prototype%20of%20the%20cluster.%E2%80%9D%20%E2%80%93