# Introduction to Machine Learning

## Objectives

- Understand the core concept of Machine Learning and its role in automating tasks.
- Define and explore the main types of Machine Learning: supervised, unsupervised, and reinforcement learning.
- Explore and apply machine learning algorithms, including K-Nearest Neighbors, Decision Trees, Random Forests, and Logistic Regression, with a focus on regression and classification tasks.
- Learn the basics of neural networks and the evolution of NLP models.
- Utilize machine learning and deep learning libraries like Scikit-learn, PyTorch, and TensorFlow.

## Introduction

This lesson is an introduction to the main concepts and applications of artificial intelligence including machine learning, deep learning, and neural networks. With the recent increased popularity of large language models, there has also been a surge in artificial intelligence. We will first go over the core elements of machine learning, such as supervised learning, unsupervised learning, and reinforcement learning. We will then reiterate from the previous Python module how to set up an environment using Anaconda, and an alternate way to set up a Python environment using virtual environments, which are part of the Python standard library. We will then introduce the machine learning workflow and go over an example using the Iris dataset. During this we will cover some of the key metrics for evaluating a machine learning model. After that we will go over some common machine learning algorithms, focusing on regression and classification. After the machine learning section, we will introduce natural language processing. We will talk about the underlying concepts such as neurons. We will then expand on neural networks and discuss deep learning. We will

discuss the components of deep learning such as the associated layers and mathematical functions.

# 1. Introduction to Machine Learning

Machine learning is the subset of artificial intelligence related to the applications and development of statistical algorithms that builds models by learning from data and therefore can generalize to unseen data.

## 1.1 Core of Machine Learning

The core essence of machine learning (also known as ML) is the use of statistical algorithms which allow machines to "learn" from data and to then be able to make decisions and predictions without the need to be explicitly programmed for specific tasks. The key difference between machine learning and traditional programming is that in traditional programming the programmer defines specific rules, and in ML the programmer defines the algorithm and that allows the computer to discover the patterns and relationships in the data on its own, and this is how it generalizes to new and unseen data.

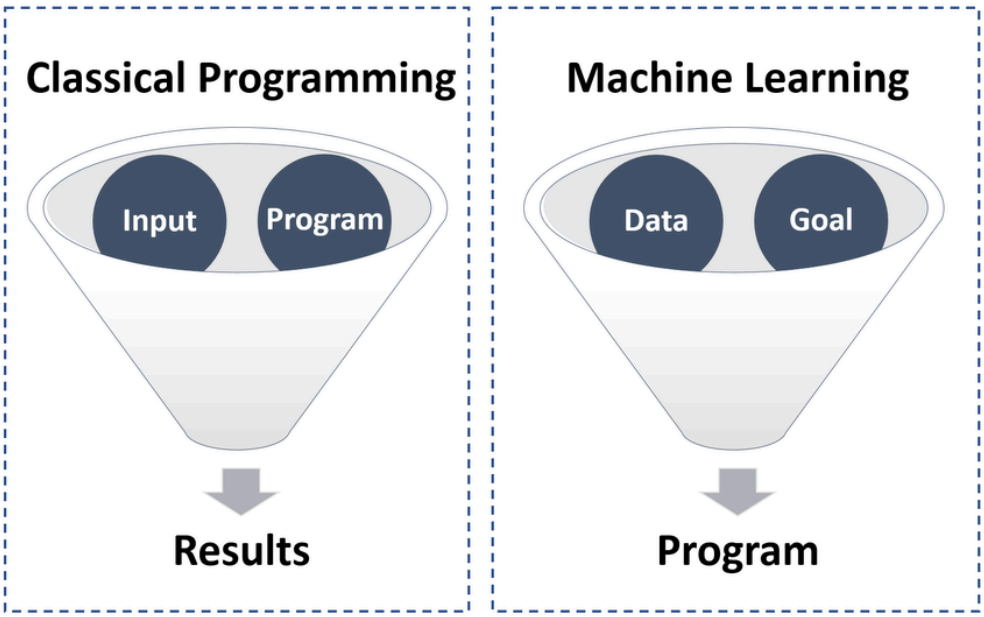



*Figure 1: The main conceptual difference between classical programming and machine learning. In classical programming the programmer defines rules, and in ML the algorithm learns the rules. Image from*

*https://www.researchgate.net/figure/The-difference-between-Classical-Programming-and-Machine-Learning-The-different-phases_fig3_370884019.*

> **Note:**
>
> The term machine learning was thought up in 1959 by an IBM employee named Arthur Samuel. He invented the earliest machine learning model in the 1950s which could calculate the probability of winning in checkers for both sides based on the configuration of the board.

Now we will cover the core components of ML.

### 1.2 Data

Data is the representation of some phenomena in the real world that you study and use to make predictions/draw conclusions. This is the raw information that the model learns patterns from. Data can take some of the following forms:

- **Numerical data:** This includes continuous data (e.g., age, temperature, income) and discrete data (e.g., number of items purchased, customer ID).
- **Categorical data:** This includes data that can be divided into categories (e.g., colour, car make, country).
- **Text data:** This includes natural language text, such as documents, emails, and social media posts.
- **Image data:** This includes digital images, which can be represented as arrays of pixels.
- **Audio data:** This includes sound recordings, which can be represented as waveforms or spectrograms.
- **Time series data:** This includes data that is measured over time, such as stock prices, sensor readings, or the brightness of a star.

### 1.3 Models

A model is the final learned system that is used for making predictions. One way to think of the difference between the algorithm and model is that for linear regression the algorithm is a general y = mx + b, and the model is after you have determined

values for m and b so it could be y = 5x + 3 where m = 5 and b = 3. In other words, the model is the trained system, typically in the form of a mathematical structure or function, that is created after the algorithm processes the data.

> **Note:**
>
> The large language model ChatGPT uses a more complex model called a **G**enerative **P**retrained **T**ransformer. We will go much more in depth with this later!

## 1.4 Supervised learning

Machine learning algorithms can typically be split into two categories: supervised and unsupervised. Supervised learning uses *labeled* data, where the end result (such as the category) is known. The end result is typically known as the "ground truth". The patterns in the labeled data are determined and from this you can apply it to new unseen data where the ground truth is unknown.

Supervised learning algorithms typically do one of two things: regression or classification.

### 1.4.1 Some supervised regression algorithms

Regression is the statistical process of estimating the relationship between a one of or a set of dependent variables (the "x-axis"), and the independent variable (the "y-axis"). Some regression algorithms are:

- **Linear Regression:** Models a linear relationship between the dependent and independent variables.
- **Logistic Regression:** Used for binary classification problems, predicting the probability of an event occurring.
- **Polynomial Regression**: Models a non-linear relationship by fitting a polynomial curve to the data.
- **Decision Trees:** Creates a tree-like structure to make decisions and determine the independent variable value.
- **Random Forests:** Uses an ensemble of decision trees to improve accuracy and reduce overfitting.

- **Support Vector Machines (SVMs):** Specifically, this would be Support Vector Regression, finds a function through the data (often a hyperplane) that fits the data points as closely as possible while maintaining some margin of tolerance.

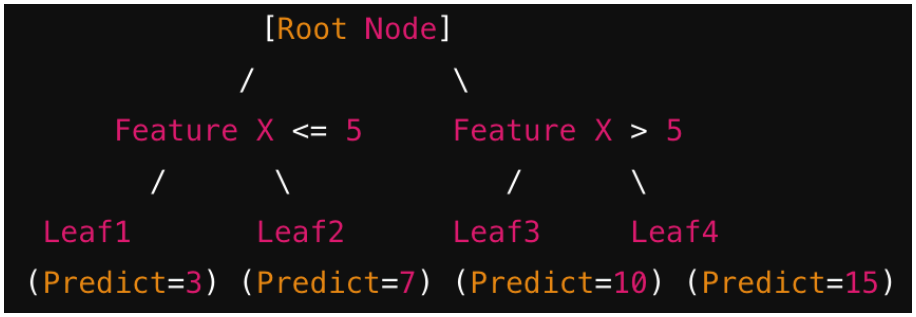



*Figure 2: A simplified illustration of the random forest regression algorithm.*

### 1.4.2 Some supervised classification algorithms

Classification is the process of assigning objects into pre-existing categories, classes, or labels. The properties of the data are the independent variables (for example for fruit this could be roundness, size, and colour), and the label is the dependent variable (for fruit this could be apple, banana, etc). Some classification algorithms are:

- **Decision Trees:** Create a tree-like structure to classify objects based on features.
- **Random Forests:** Combines multiple decision trees to improve accuracy and reduce overfitting, predicts a label unlike in regression where it predicts a continuous value.
- **Support Vector Machines (SVMs):** Specifically, would be Support Vector Classification, finds a hyperplane that separates data points into different classes.
- **K-Nearest Neighbors (KNN):** Classifies new data points based on the majority class of their nearest neighbors.

## 1.5 Unsupervised learning

Unsupervised learning uses *unlabeled* data, where the ground truth is unknown. The algorithm then learns hidden patterns, structures, or clustering in the unlabeled data. This is done by grouping data based on similarities in their features.

Most unsupervised learning algorithms fall into either clustering, or dimensionality reduction.

- **K-means clustering:** Partitions n observations into k clusters, where each observation is in the cluster with the closest mean, i.e. cluster centroid.
- **Hierarchical clustering:** Groups objects into clusters that follow a hierarchical structure.

### 1.5.1 Some unsupervised dimensionality reduction algorithms

Dimensionality reduction is a method where a new representation of a dataset is found which is in a lower number of dimensions than the original representation. The new representation should still describe the majority of the variance. This has advantages such as improving the computational efficiency for further analysis, putting the data in a form where it can be better visualized, reducing overfitting, and reducing noise.

- **Principal component analysis (PCA):** The most common unsupervised dimensionality reduction algorithm. Projects the data into a lower dimensional space while maintaining its variance. This is good for capturing linear relationships but does not work well for non-linear relationships within the data.

## 1.6 Reinforcement learning

Reinforcement learning (RL) is a subset of machine learning where the models "learn" to make decisions by interacting with the data and receiving positive or negative feedback based on each action. The models focus on making decisions that maximize the total rewards of a situation. This is different from supervised learning because there are no predefined ground truths, RL learns through experience. Below are some key elements of RL:

- **Agent:** The model which can be called the "learner" or "decision maker".
- **Environment:** The data, which is what the agent interacts with.
- **State:** The situation in which the agent is in.
- **Action:** The moves that the agent can make.
- **Reward:** The feedback that the agent receives from the environment based on the action.

Some applications of reinforcement learning can include:

- **Playing games:** RL has been used to program computers to play games like chess, poker, and pong.
- **Robotics:** RL has been applied to robots learning to perform tasks such as picking up objects, navigating environments, and assembling parts.
- **Finance:** RL can be used to optimize trading strategies by learning from changes to the market, to find a balance between profit and risk.
- **Natural language processing:** RL can be used to improve responses in conversational AI (which we will learn much more about later) by learning from user feedback. You have probably had ChatGPT ask you if its response was useful.

RL works by learning the optimal strategies through trial and error. This is done by the agent going through actions in the environment, getting either a reward or penalty, and then adjusting the actions to maximize the sum of the rewards. The learning process involves the following concepts:

- **Policy:** Strategy by agent to determine next action using the current state.
- **Reward function:** Function that provides the feedback signal based on the current state and action.
- **Value function:** Function that calculates expected cumulative reward from a state.
- **Environment model:** Representation of environment that predicts future states and rewards for strategy planning.

### 1.6.1 Some reinforcement learning algorithms

Below are some of the common RL algorithms:

- **Q-learning:** Finds the optimal actions for a finite Markov decision process (MDP). An MDP can also be called a stochastic control problem and is a model for sequential decision making for uncertain outcomes.
- **SARSA:** Full name is state-action-reward-state-action, is another algorithm for MDP's.

## 1.7 Hands-on, setting up the Python environment

Now we will set up the Python environment. There are different options for this, the two we will cover are Anaconda and Python standard library virtual environments usually called "venv". Note that we also introduced environments in chapter 7 of the Introduction to Python asynchronous lesson.

### 1.7.1 Advantages and disadvantages of Anaconda:

Learn more about Anaconda here:

https://www.anaconda.com/

Anaconda is typically easier to set up, the analogy I would use is that when it comes to Python environments, Anaconda is like driving automatic and venv is like driving manual. The pros of Anaconda are that it has easier package management with the package manager conda, which can install both Python and non-Python dependencies. This makes it very easy to use the data science packages like numpy, scipy, pandas, tensorflow, or PyTorch, because they are included in the Anaconda installation. There is also a better compatibility for cross-platform, if you need to work on projects across Windows, Mac, and Linux. Overall, the Anaconda advantages are in its user friendliness and wide compatibility.

The disadvantages of Anaconda are that with its user friendliness there is a trade-off in how lightweight it is. Installing Anaconda takes more disc space due to the pre-installed data science packages. Also with relying on the conda package manager, there is a possibility of problems with version compatibility.

### 1.7.2 Advantages and disadvantages of venv:

Read the documentation on virtual environments here:

https://docs.python.org/3/library/venv.html

Virtual environment, or venv, is a tool that is included with the Python standard library and is set up using the command line. It is very customizable and lightweight as it is not a large installation like Anaconda. It is great for smaller projects where you want an isolated environment. If you were to run code on a cluster you would use a venv. The disadvantages are that it isn't cross compatible like Anaconda, you could run into problems doing a project across Windows, Mac, and Linux, as it relies heavily on your computer's OS.

### 1.7.3 Anaconda setup:

The Anaconda environment is where you can run machine learning scripts and notebooks using Python. First, open your terminal (also known as command prompt) like in the Intro to Python material. On Windows, you can open a terminal by looking up “Anaconda Prompt” in the search bar of windows and then clicking on it. In Mac OS, you can open a terminal by launching the terminal. On GNU/Linux, the command line can be accessed by several applications like xterm, Gnome Terminal or Konsole.

With your command prompt now open, input the following instructions. (When there is a dollar sign followed by code, that means it is a command for the command prompt).

First, create a directory for your project and then navigate to it.

**Code:**

```
# creating and moving to directory
$ mkdir ml_project
$ cd ml_project
```

Next, use conda create to create your environment. In this document I will call it "ml_dns" but you can give it any name you would like.

**Code:**

```
# create conda environment
$ conda create --name ml_dns
```

Now that your conda environment has been created, we activate it with conda activate so we can install the required package.

**Code:**

```
# activate the environment
$ conda activate ml_dns
```

Now that the environment is the active environment, we will use conda install to install the necessary packages.

**Code:**

```
# install various libraries to your environment
$ conda install scikit-learn numpy scipy pandas matplotlib
```

Here is an explanation of each of the packages just installed:

- **NumPy:** A library for numerical computations in Python, stands for Numerical Python.
- **SciPy:** A library for scientific and technical computing, built on top of NumPy.
- **Pandas:** A library for data manipulation and analysis, providing data structures like DataFrames and Series, along with tools for data cleaning, transformation, and analysis.
- **matplotlib:** A plotting library for creating visualizations in Python.
- **scikit-learn:** A machine learning library providing tools for data preprocessing, modeling, and evaluation.

After the packages finish downloading, we can verify the installation using conda list to list the downloaded packages.

**Code:**

```
# list the installed packages
$ conda list
```

Your conda environment is now set up to do some machine learning! If you are done for now you can use conda deactivate to deactivate the environment.

**Code:**

```
# deactivate the environment. always do this when done
$ conda deactivate
```

With scikit-learn (and its dependencies) all installed you are now ready to implement the machine learning algorithms discussed in this chapter.

### 1.7.3 venv setup:

First, create a directory for your project and then navigate to it.

**Code:**

```
# creating and moving to directory
$ mkdir ml_project
$ cd ml_project
```

Next, create the virtual environment.

**Code:**

```
# creating venv
$ python3 -m venv ml_dns
```

This will create the virtual environment called ml_dns in the ml_project directory.

Next, activate the virtual environment so that we can install the required packages.

**Code:**

```
# activate venv
$ source ml_dns/bin/activate
```

You will now have (ml_dns) appear at the start of the command prompt line. This tells you that the venv is now active. We will now install the required packages for our project. The first step towards this is upgrading pip. Pip is the Python package manager. Pip is to virtual environments what conda is to Anaconda environments.

**Code:**

```
# upgrade pip
$ (ml_dns) pip install --upgrade pip
```

With pip upgraded, we can now install the required packages.

**Code:**

```
# install various libraries
$ (ml_dns) pip install scikit-learn numpy scipy pandas matplotlib
```

The packages will now install. Verify they are installed with pip list.

**Code:**

```
# list installed packages
$ (ml_dns) pip list
```

This will output the installed packages. One last step we will do is saving the project's dependencies to a text file, this will help if we want to share our project.

**Code:**

```
# save list of libraries to text file
$ (ml_dns) pip freeze > requirements.txt
```

Finally, lets deactivate the venv.

**Code:**

```
# deactivate venv
$ (ml_dns) deactivate
```

**Note on naming your venv:**

In this example we named the venv "ml_dns", to be descriptive. However it is common to always call your venv ".venv". You will know what the environment is from the requirements.txt, or you could also make a text file with a note explaining the venv. Putting the dot at the start makes it a hidden folder, so it will be more in the "backend". Using the same name for all of your venvs will allow you to create custom bash commands in your bashrc.

Important note on venv and Github:

**Do not commit your venv to your Git repo and push it to GitHub.** A venv contains many files and takes a large amount of disk space. Make sure that you tell Git to ignore the venv in your .gitignore file.

****Committing your venv will break your GitHub repo.****

# 2. Introduction to Implementing Machine Learning

In this chapter we will go more in depth into some of the core machine learning algorithms (supervised and unsupervised), and the overall machine learning workflow.

## 2.1 The machine learning workflow

We will now cover the overall machine learning workflow as a step-by-step guide. This includes loading the data, preprocessing the data, initializing and training the model, and then evaluating with accuracy metrics.

For this example, we will be using the Iris dataset. This dataset contains lengths and widths of Iris flowers. The columns are called Sepal Length, Sepal Width, Petal Length and Petal Width. The final categories are Setosa, Versicolour, and Virginica. Since the results are distinct categories and not continuous values, this is a **classification** problem. The goal is to use the lengths and widths of data where the category is unknown to predict the category.

The overall workflow involves the following steps:

- **Import libraries:** Import the required libraries and modules needed for your analysis.
- **Load dataset:** Load the data to be analyzed. Split the data into inputs (also known as features, typically represented by the variable X), and outputs (for supervised learning, also known as labels, typically represented by the variable y).
- **Split data:** Split the data into two subsets. A larger one for training (typically around 80%) and a smaller one for testing (typically around 20%).
- **Define and train model:** Initialize the model of choice (for example regression, classification, etc) and then fit the model to the training data.
- **Make predictions:** Using the trained model make predictions on the test data.
- **Evaluate model:** Statistics such as accuracy, precision, and recall are calculated to quantify the model performance.
- **Improve model:** Use hyperparameter tuning to improve the models performance on predicting the training data.

Later we will build on this and add more steps for practical application.

### 2.2 First ML analysis on the Iris dataset

We will now walk through a quick analysis where we will run a classification algorithm on the Iris dataset. We will create this in a .py script file, so once it is created, we will activate the conda library to run the code.

The first step is importing the necessary libraries.

**Code:**

```
# import ML libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.datasets import load_iris
```

Next, we load the Iris dataset which is available in scikit-learn.

**Code:**

```
# load iris dataset
iris = load_iris()
```

Next, we split the data into our inputs (also known as features, represented by X), and our outputs (also known as targets, represented by y).

**Code:**

```
# create feature and target arrays
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
```

Next, we split the data into two subsets: training and testing.

**Code:**

```
# split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
shuffle = True, random_state=42)
```

**Note:**

The reason why we split the data into training and testing is so that we have two separate datasets for creating the model, and then evaluating it. Remember one of the core parts of machine learning is that the models generalize to new, unseen data.

Because of this, we use the training dataset to fit the model, allowing it to learn the patterns and relationships in the features. During this process the testing dataset is hidden from the model. Once the model is trained, we make predictions on the testing dataset, where we do know the "ground truth" or outputs to the specific inputs, but the model does not know those values.

We then compare the predicted outputs with the real outputs and calculate performance metrics by comparing these two vectors (we will cover these performance metrics more in depth).

This process also helps us to determine if the model is underfitting (failing to capture the underlying patterns in the data) or overfitting (learning the training data too well and unable to generalize to new unseen data).

A typical split is to use 70% to 80% for training and therefore 20% to 30% for testing.

Now, we define our model. We will be using the random forest classifier and set it to use 50 estimators.

**Code:**

```
# initialize model
model = RandomForestClassifier(n_estimators = 50,
random_state=42)
```

With our model defined, we now train our model. In scikit-learn this is very simple, all we need to do is model.fit.

**Code:**

```
# train model
model.fit(X_train, y_train)
```

Now, we make predictions on the *testing* data.

**Code:**

```
# make predictions
y_pred = model.predict(X_test)
```

We have now used our trained model to make predictions on the testing data. We would now compare the predictions from the model to the ground truth values by calculating various performance metrics.

First, let's calculate the accuracy. Accuracy measures the overall correctness of a classification model. It's the ratio of correct predictions to the total number of predictions.

**Code:**

```
# calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

$$\text{accuracy} = \frac{\text{\# of correct predictions}}{\text{\# of total predictions}}$$

We will now discuss some other performance metrics for classification problems.

Precision is the proportion of true positives (correctly predicted positives) over the total predicted positives. It is a measure of how many of the predicted positive instances were actually positive. So if your model predicted 100 instances as positive, and out of those, 90 were actually positive, the precision would be 0.9 or 90%.

Precision is especially important when the cost of false positives is high. For instance, in medical diagnoses, a false positive might lead to unnecessary treatments or surgeries, which can be harmful and costly.

It helps in finding the right balance between true positives and false positives. A high precision means that a positive prediction is likely to be correct, which is essential in scenarios where false positives are costly.

For example, in a spam filter, high precision ensures that only actual spam emails are flagged, minimizing the chances of legitimate emails being incorrectly marked as spam.

$$\text{precision} = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of false positives}}$$

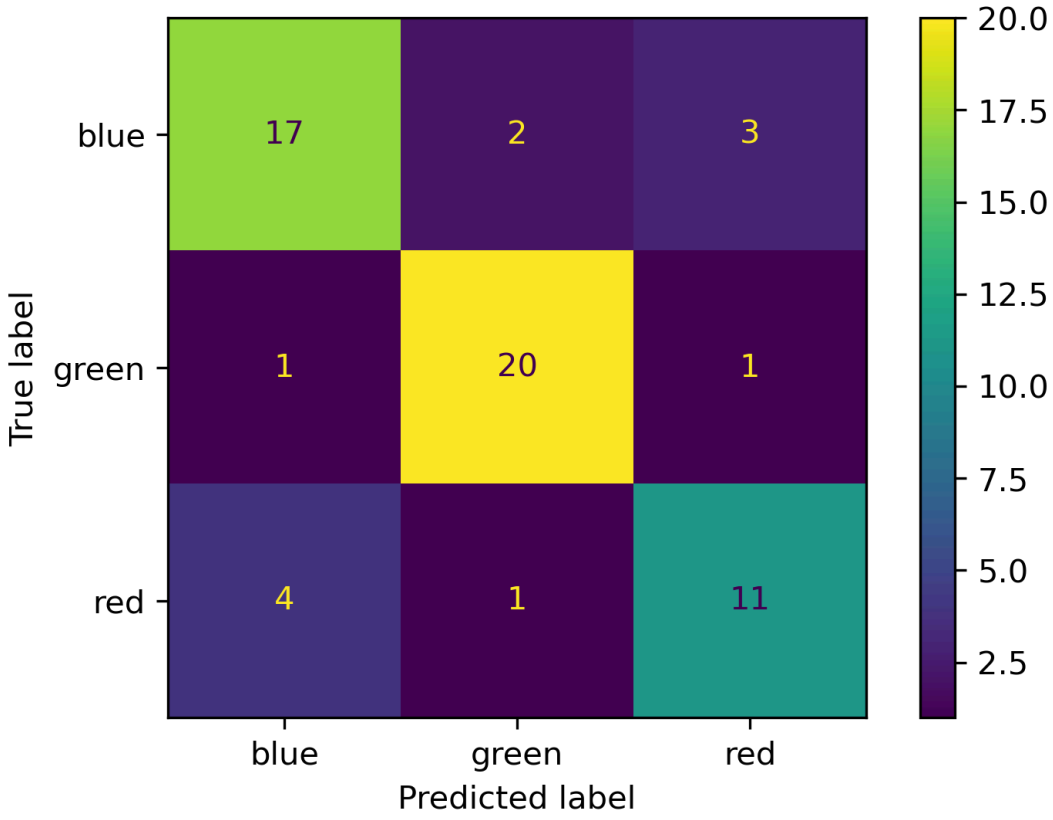
Recall measures the proportion of true positives over the total actual positives. It is a measure of how many of the actual positive instances were correctly predicted. If there were actually 100 positive instances, and your model correctly identified 71 of them, the recall would be 0.71 or 71%.

$$\text{recall} = \frac{\#\text{ of true positives}}{\#\text{ of true positives} + \#\text{ of false negatives}}$$

A confusion matrix is a table that visualizes the performance of a classification model. It shows the counts of true positives, true negatives, false positives, and false negatives.

- **True Positives (TP):** These are the cases where the model predicted the positive class correctly, so the actual result was positive, and the predicted result was positive. For example, if a medical test correctly identifies a disease in a patient who has it, it's a true positive.
- **True Negatives (TN):** These are the cases where the model predicted the negative class correctly, and the actual result was negative. For instance, if a security system correctly identifies a non-threat object to be a non-threat, it's a true negative.
- **False Positives (FP):** These are the cases where the model predicted positive, but the actual result was negative. In the medical context, it would mean the test wrongly indicates a disease when the person is healthy.
- **False Negatives (FN):** These are the cases where the model predicted negative, but the actual result was positive. For example, if a security system fails to detect an actual threat to be harmless that is a false negative.

In the confusion matrix, you want the diagonal elements to have the greatest numbers. This is a good tool to see if there is some systematic error that could be apparent if it is skewed towards certain off-diagonal elements.




F1 score is another metric that is calculated from precision and recall. It is useful when both false positives and false negatives could have major consequences. Imagine you had a classification model to predict if a patient has cancer. A false positive is very costly because then you would put the patient through intense treatment like chemotherapy which would cause harm to someone that is healthy. A false negative is also very costly as someone with cancer would not receive treatment in time.

F1 score is calculated using the harmonic mean of precision and recall.

**Note:**

The American health insurance company UnitedHealth was criticized in November 2023 for using an AI model with a 90% error rate to deny health care coverage. This involved the use of a computer algorithm called nH predict, and resulted in the class action lawsuit *Estate of Gene B. Lokken et al. v. UnitedHealth Group, Inc. et al.*

The exact accuracy metric used to determine the error rate hasn't been publicly stated, but it is, in my opinion, most likely some function of accuracy and precision.

**Sources:**
https://arstechnica.com/health/2023/11/ai-with-90-error-rate-forces-elderly-out-of-rehab-nursing-homes-suit-claims/

https://en.wikipedia.org/wiki/NH_Predict

https://litigationtracker.law.georgetown.edu/litigation/estate-of-gene-b-lokken-the-et-al-v-unitedhealth-group-inc-et-al/

## 3. Common Machine Learning Algorithms

We will now go into more detail on some of the most important machine learning algorithms from both supervised and unsupervised learning.

The scikit-learn machine learning map is very useful for choosing an algorithm based on your data! It is available here:

https://scikit-learn.org/stable/machine_learning_map.html

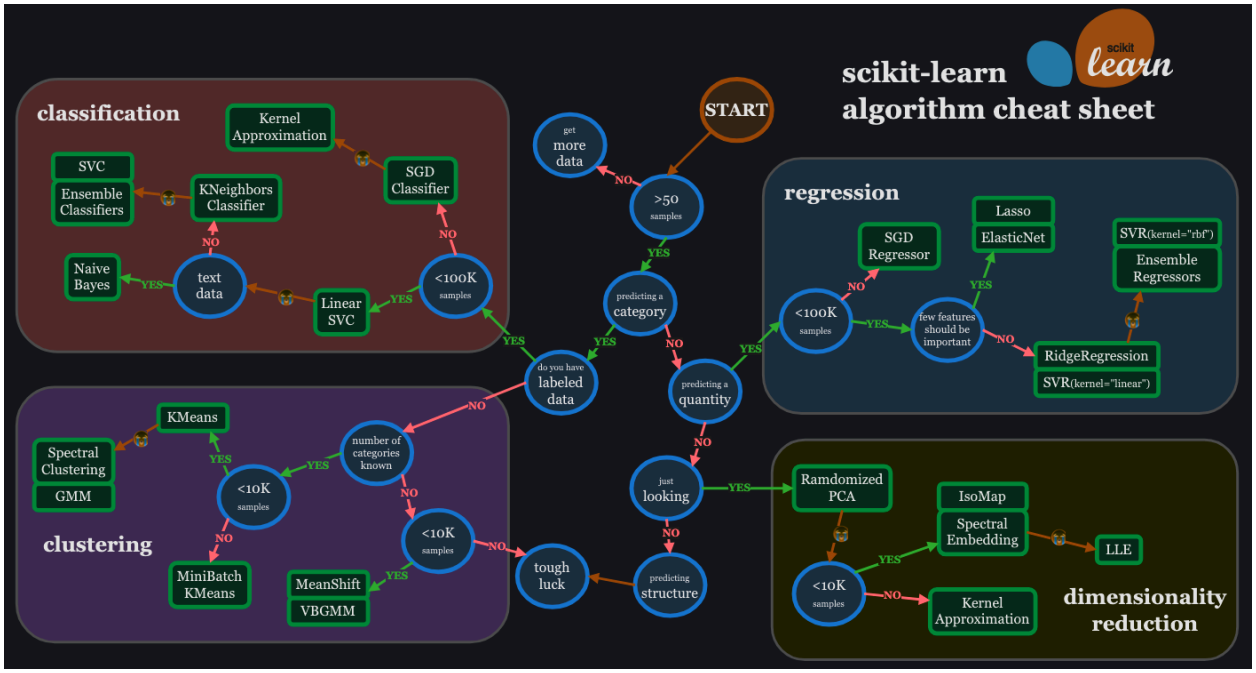



*Figure 3: The scikit-learn machine learning algorithm map. Available at the URL above.*

> **Note:**
>
> In the sections below, there is code for you to follow along and see how to implement these algorithms. It is in the hands on machine learning Jupyter notebook for chapter 3.

## 3.1 Regression

Regression is a statistical technique that is used for relating a continuous output value to one of or a set of input values. In machine learning, it is used for predicting a continuous numerical outcome based on the input features. The overall goal of regression is to determine the relation between a dependent variable and a set of independent variables. This is done by fitting a function to the data. There are different forms that this function can take, which we will cover. Regression is commonly used when we want to understand how changes in the input features influence the target variable or when we need to predict values like sales forecasts, house prices, or stock prices. It is especially useful when we have a dataset with continuous outcomes and

we want to predict new values, identify trends, or quantify the strength of relationships between variables.

### 3.1.1 Linear regression

Linear regression is a technique that models the relationship between a dependent variable and one or more independent variables by fitting a straight line (or flat plane in higher dimensions) through the data points. This line, known as the regression line, minimizes the sum of squared differences between the actual data points and the predictions. Linear regression is best suited for situations where there is a linear relationship between variables, such as predicting house prices based on square footage or estimating sales from advertising spend. It is often used as a baseline model due to its simplicity and interpretability, making it easy to understand how changes in input features affect the target variable.

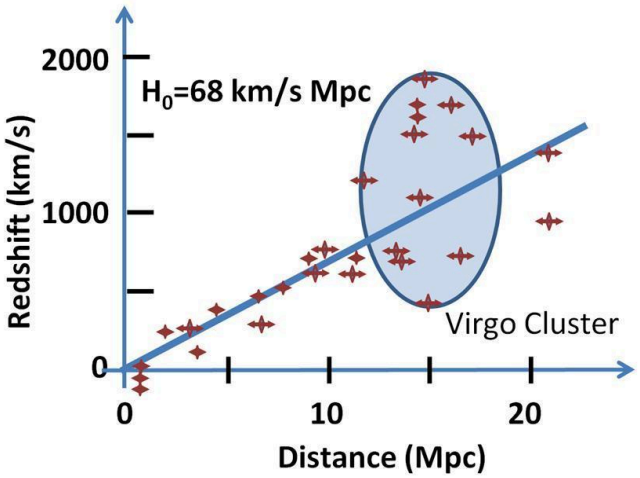



*Figure 4.1: A classic case of linear regression, the relation between redshift speed and distance of galaxies. This is called Hubble's law and was one of the key discoveries that was used to discover the accelerating expansion of the Universe. Image from Keel, W. C. (2007). The Road to Galaxy Formation (2nd ed.). Springer. pp. 7–8. ISBN 978-3-540-72534-3.*

Linear regression could be used for predicting house prices based on features like square footage, number of bedrooms, and how long ago the house was built.

**Example usage:**

Predicting house prices based on features like square footage, number of bedrooms, and how long ago the house was built.

### 3.1.2 Logistic regression

Logistic regression is a type of regression used when the target variable is binary or categorical, such as predicting whether a customer will buy or not buy a product. Instead of fitting a straight line, logistic regression fits an S-shaped curve (the logistic function) that predicts the probability of a certain class or event occurring. The output is bounded between 0 and 1, making it suitable for predicting a probability for situations such as spam detection or disease diagnosis. These probabilities can be extended to classification based on if the result is greater than or less than 0.5.

**Example usage:**

Predicting whether a customer will buy a product based on features like age, income, and past purchase history.

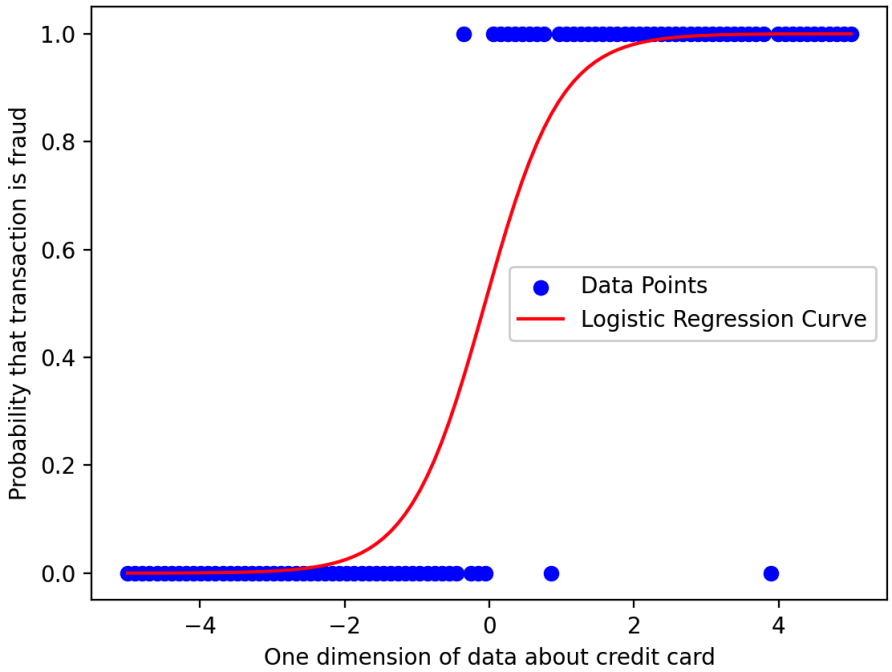



*Figure 4.2: An example of the logistic regression curve, to determine if a credit card transaction is fraudulent.*

### 3.1.3 Polynomial regression

Polynomial regression extends linear regression by fitting a polynomial equation to the data, which allows for modeling non-linear relationships between the dependent and independent variables. Instead of a straight line, it fits a curve that can bend to capture more complex patterns in the data. It's particularly useful when the relationship between variables is not linear, but a curved line fits better, such as predicting growth rates or modeling trends like temperature changes over time. The model includes higher-degree terms of the input variables (like x^2, x^3), making it more flexible but also more prone to overfitting if not properly tuned.

**Example usage:**

Modeling the growth rate of a bacteria population over time when the growth is non-linear.

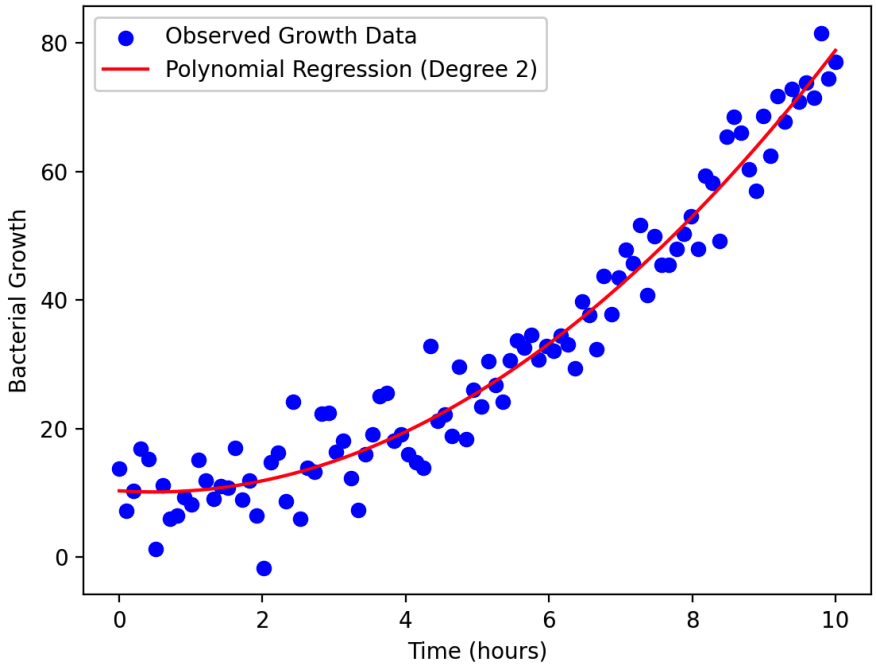



*Figure 4.3: An example of the polynomial regression curve, to quantify the growth of bacteria over a short period of time.*

### 3.1.4 Decision trees

Decision trees are non-linear regression models that work by splitting the data into subsets based on the values of the input features. Each node in the tree represents a decision rule based on a feature, and each leaf node represents a predicted outcome. The tree structure allows for easy interpretation and visualization of the decision-making process. Decision trees are useful for capturing complex, non-linear relationships and interactions between features, making them suitable for both regression and classification tasks. However, they are prone to overfitting, especially if the tree becomes too deep.

**Example usage:**

Decision trees for regression could be used for predicting the sales revenue for a retail store based on seasonal trends, holidays, and promotional campaigns.

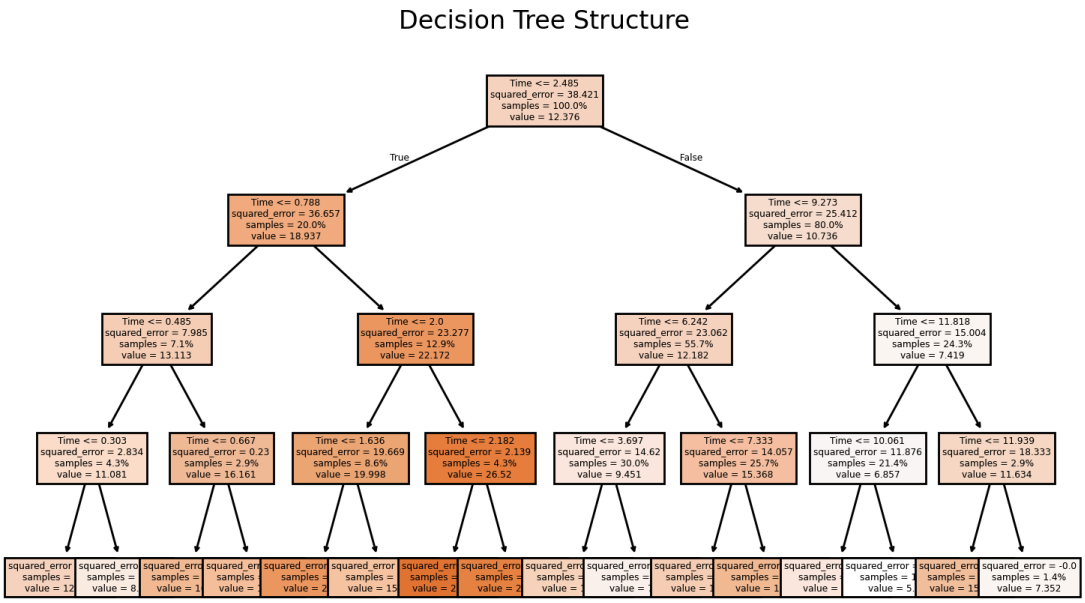



*Figure 4.4: A schematic of the decision tree structure to predict seasonal sales trends.*

*Figure 4.5: The result of the decision tree regression to predict sales during seasonal trends.*

### 3.1.5 Random forests

Random forests are an ensemble learning method that builds multiple decision trees and combines their predictions to produce a more accurate and robust output. In regression tasks, random forests aggregate the predictions of individual trees (typically by averaging) to provide a final prediction. By creating multiple trees based on different subsets of data and features, random forests help to reduce the overfitting problem that individual decision trees can suffer from, and they improve the model's generalization to new data. They are widely used for tasks where complex relationships between variables exist, such as predicting stock prices, weather patterns, or sales forecasts.

**Example usage:**

Random forests for regression could be used for estimating air quality index (AQI) based on weather conditions, traffic levels, and pollutant concentrations.

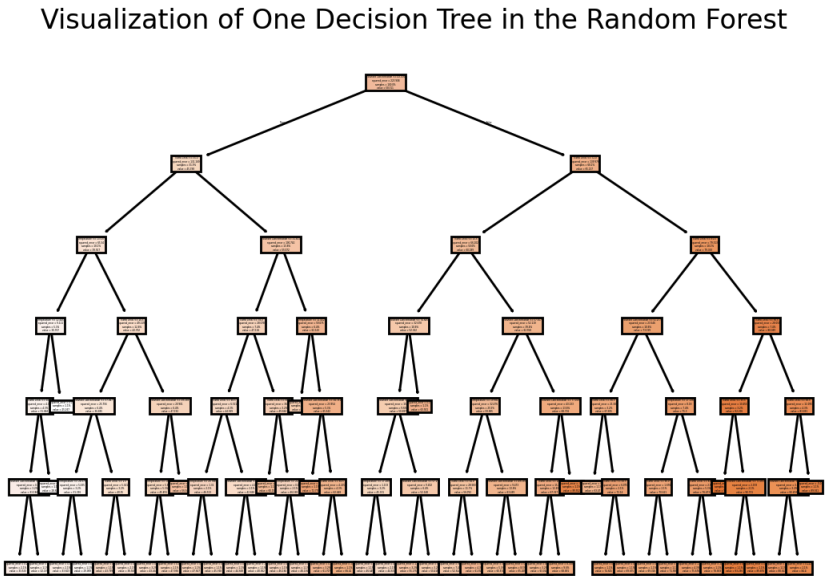



*Figure 4.7: A schematic of one decision tree in a random forest. I am aware that the text is way too small to read, I put this here to illustrate the depth that decision trees can go into, which is why they are great for quantifying more complex non-linear trends between many dimensions of data.*

### 3.1.6 Support vector regression (SVR)

Support Vector Machines (SVM) for regression, known as SVR (Support Vector Regression), aim to find a hyperplane that best fits the data while maintaining a margin of tolerance around it. Instead of minimizing the error directly, SVR attempts to find a balance between the complexity of the model and the error tolerance by creating some margin around the hyperplane where deviations are ignored. SVMs are particularly effective when the data is high-dimensional or when there is a non-linear relationship that can be transformed using kernel functions. They are useful in applications where precision is crucial, such as time series forecasting, financial modeling, or predicting biological responses.

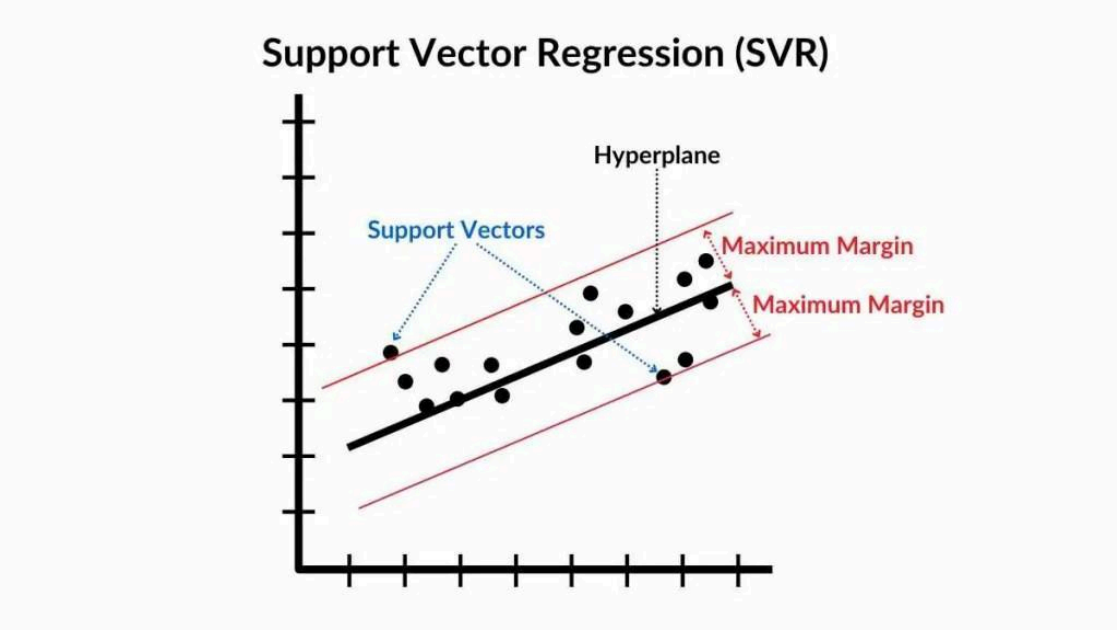



*Figure 4.8: An illustration of support vector regression including the support vectors and the margins. Image from https://spotintelligence.com/2024/05/08/support-vector-regression-svr/.*

**Example usage:**

Predicting the future price of a stock based on historical price movements and technical indicators.

## 3.2 Classification

Classification is a supervised learning technique that assigns a category (also known as the label) to the input variables. It involves training a model on a dataset where each data point is associated with a known label, allowing the model to learn the patterns that separate different classes. Once trained, the model can predict the label of new, unseen data points. Classification is particularly useful when the outcome we want to predict falls into *discrete categories* rather than continuous values. For example, it can be used to determine if an email is spam or not spam, classify types of diseases based on medical symptoms, or classifying customers into groups based on their behavior (such as a frequent buyer or an occasional buyer). It is ideal for scenarios where there

are clear categories or classes to distinguish between, and where accurate prediction of these labels is crucial for decision-making.

### 3.2.1 Decision trees

Decision trees are a versatile classification technique that works by splitting the data into subsets based on the values of input features, making a series of decisions at each node. Each internal node represents a feature-based decision, each branch represents an outcome of that decision, and each leaf node represents a class label. The tree is built by choosing the features that best reduce uncertainty at each step. Decision trees are easy to interpret and visualize, making them useful for understanding how classification decisions are made. However, they are prone to overfitting, especially if they become too deep, making them sensitive to variations in training data.

**Example usage:**

Decision trees for classification could be used for loan applicants as low-risk or high-risk based on features like credit score, annual income, and employment status.

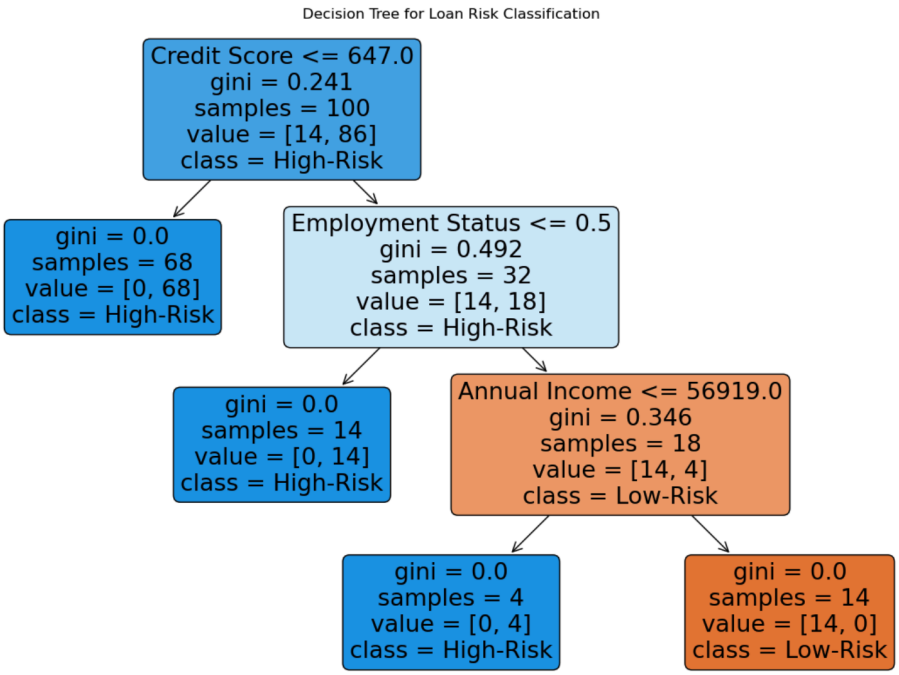



*Figure 4.9: A schematic of the decision tree structure to predict loan applicant risk.*

### 3.2.2 Random forests

Random forests are an ensemble method that uses multiple decision trees to improve the accuracy and robustness of classification. Each tree is built on a random subset of the data and a random subset of features, and the final class prediction is determined by voting among the individual trees' predictions. This helps to reduce the overfitting problem that individual decision trees can face and provides better generalization to new data. Random forests are particularly effective when there are complex interactions between features or when a dataset is noisy. They are widely used in applications like medical diagnosis, image classification, and fraud detection, often offering high accuracy.

**Example usage:**

Random forests for classification could be used for detecting fraudulent transactions in a banking system based on features like transaction amount, location, and time of day.

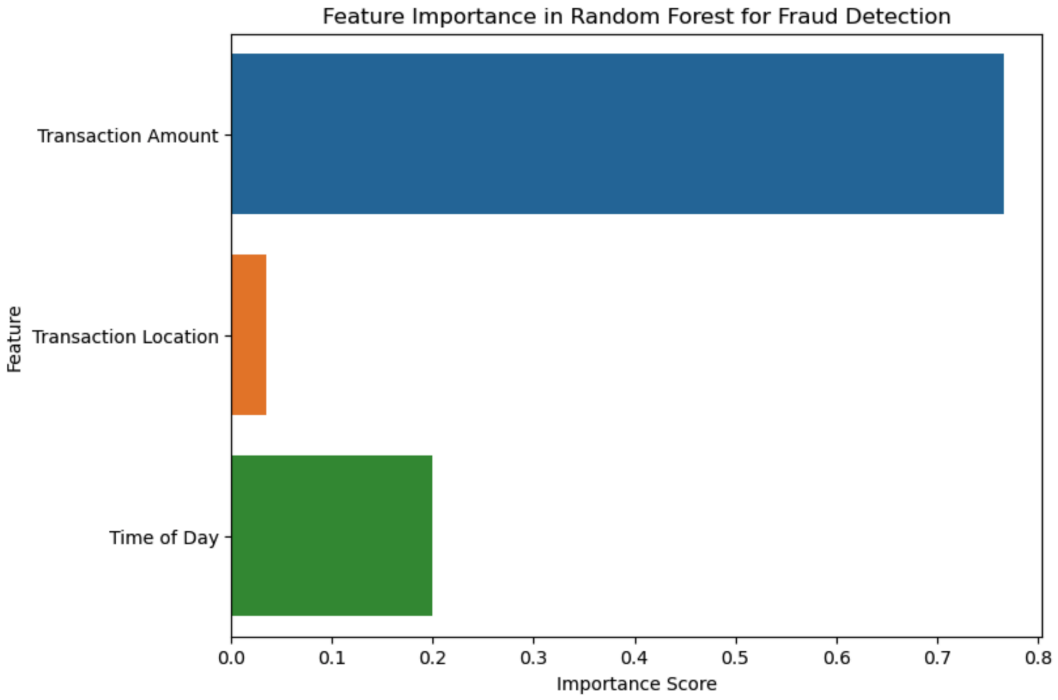


*Figure 4.10: A plot of the feature importance of the variables used to determine if a transaction is fraudulent. We can see that the amount spent is the best indicator.*

### 3.2.3 Support vector classifiers (SVC)

Support Vector Machines (SVM) are a powerful classification technique that seeks to find a hyperplane that best separates different classes in the feature space while maximizing the margin between the nearest data points of each class, known as support vectors. SVMs are effective in high-dimensional spaces and can handle both linearly separable and non-linearly separable data using kernel functions to map the data into a higher-dimensional space. This flexibility makes them suitable for tasks like text classification, image recognition, and bioinformatics. SVMs often perform well in complex classification problems, but they can be computationally intensive and may require careful tuning of parameters like the regularization parameter (C) and the kernel choice.

**Example usage:**

Classifying images of handwritten digits (e.g., 0-9) from the MNIST dataset.

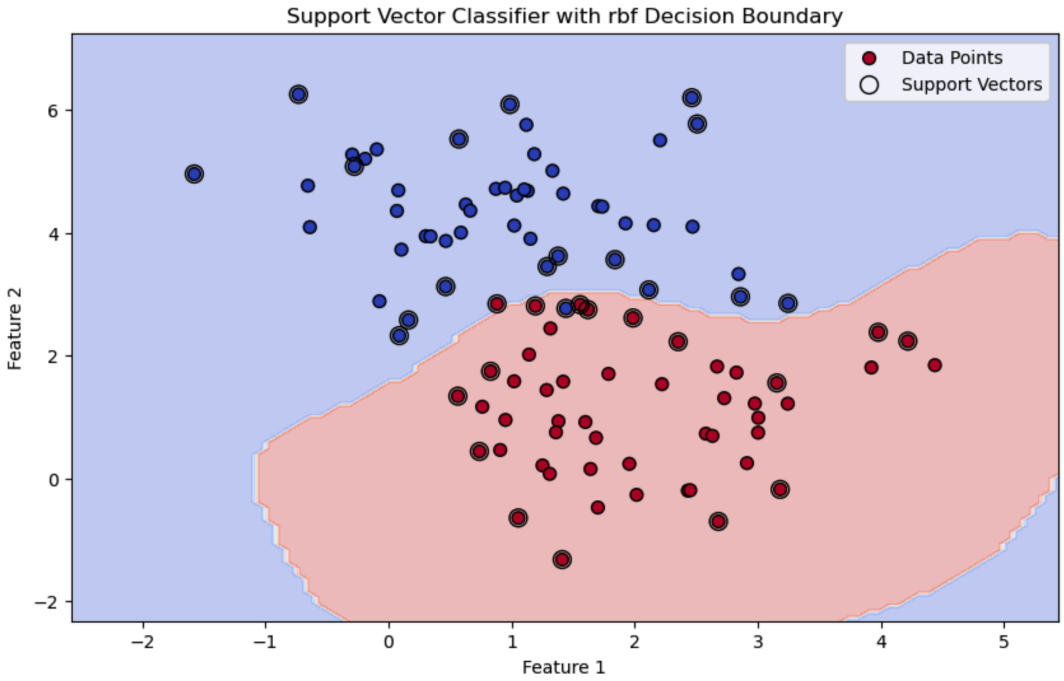



*Figure 4.11: A plot of a support vector classifier on an arbitrary 2D dataset, separating the red points from the blue points. An rbf kernel was used here.*

### 3.2.4 K-nearest neighbors (KNN)

K-Nearest Neighbors (KNN) is an instance-based learning method that classifies a new data point based on the majority vote of its k nearest neighbors in the feature space. It measures the distance (typically using Euclidean distance) between the new data point and the existing points in the training set, and assigns the class that is most common among the nearest neighbors. KNN is simple to implement and can work well for datasets with a clear separation between classes. However, it can be computationally expensive, especially for large datasets, as it requires storing all training data and computing distances for each new prediction.

**Example usage:**

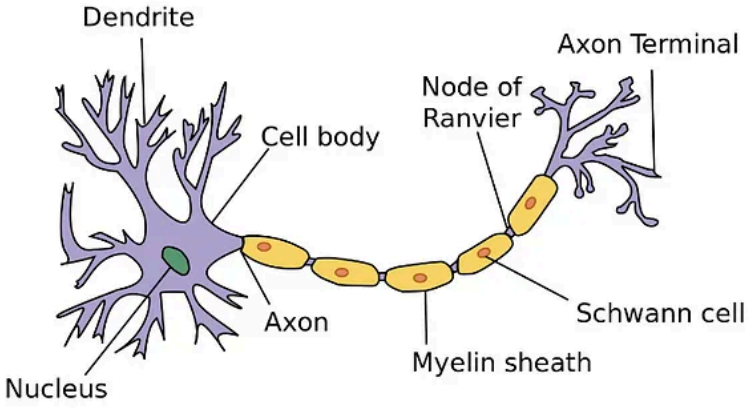
Recommending movies to a user based on the preferences of their nearest neighbors (users with similar tastes).

# 4. Introduction to Natural Language Processing

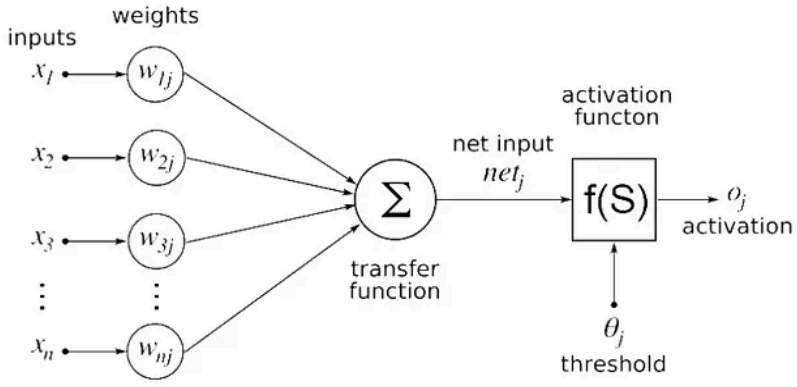
Neural networks and deep learning build on top of machine learning. Research for them began in the 1960s and the first breakthrough paper was published in 1965 by the mathematician Alexey Ivakhnenko in Ukraine. This paper outlined a method to train arbitrarily deep neural networks.

## 4.1 Neurons, the basis of natural language processing

Neural networks are models which use artificial neurons to mimic the brain's functionality. In the images below we show a comparison between a biological neuron and an artificial neural network.




Human neuron




Artificial neuron

First, consider the biological neuron. Information enters the neuron through dendrites, which receive electrical signals, or impulses, from other neurons. These impulses travel through the cell body, down the axon, and finally reach the axon terminals, where they are transmitted to other neurons.

Now, the artificial neural network operates similarly in concept but in a mathematical and computational context. Instead of dendrites, we have inputs (denoted as x^1,x^2,...,x^n), which could be features in our data. Each input is connected to the next layer of neurons by a weight (w_1,w_2,...,w_n), which adjusts the strength of the signal.
The network sums the weighted inputs, and then an activation function (denoted here as f) is applied to produce the output. This process, which includes summing the inputs and applying an activation function, helps the network learn patterns from data.
Thus, neural networks are inspired by the way biological neurons work but translate these impulses into mathematical functions to solve complex problems like image recognition, language processing, and predictive modeling.

To build on that idea of the artificial neuron, there is the Multilayer Perceptron (MLP) which is an extended version of the same concept, which forms a complete network. Each layer in the MLP contains multiple neurons that work together to process inputs and make predictions.

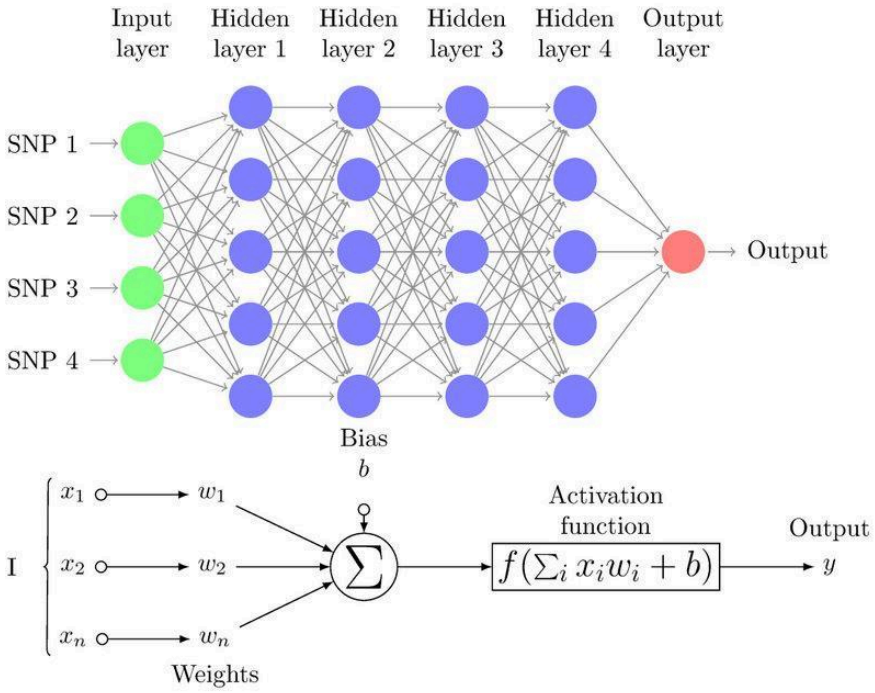



*Figure 6: Diagram of neural network.*

In the MLP, inputs pass through multiple hidden layers of interconnected neurons, each applying weights and activation functions, as described earlier. This layered structure allows the MLP to capture more complex patterns in the data compared to a single-layer network, making it powerful for tasks like classification and regression.

Essentially, MLP is a practical application of the neural network concept, taking it from a simple, single-neuron system to a more complex, multilayered system for more accurate predictions.

We have 4 Key Concepts of Neural Networks to remember:

**Types of Layers**:

- **Input layer**: Receives raw data, such as image pixels or customer features.
- **Hidden layers**: Perform non-linear transformations on data to extract features and patterns. There may be one or more hidden layers.
- **Output layer**: Provides the final result, such as a classification (cat vs. dog) or a prediction (house price).

**Neurons and how they work**:

- Each neuron receives inputs, weights them, sums them, and applies an activation function.
- The activation function introduces non-linearity, enabling the MLP to learn complex relationships.

**Connections and weights**:

- Neurons are connected through weighted connections.
- Weights determine the importance of each connection and are adjusted during learning.

**Learning**:

- The MLP learns from a training dataset.
- The backpropagation algorithm adjusts the connection weights to minimize the error between the model's predictions and actual values.

## 5. Introduction to Deep Learning

There was a period through the 1990s and early 2000s where powerful AI was still thought of by many as an unrealistic pipe dream. Neural networks were being used for some tasks, but the results were somewhat lacking. There were two major factors that came together which pushed us into the era of deep learning that we are in today.

The first was the increased availability of data. For a neural network to learn to understand a task, it needs to be exposed to a lot of different data. For a neural network to understand what a cat is, it needs to be exposed to a lot of pictures of cats, as well as a lot of pictures of things that are not cats. The era of the internet has supplied us with lots of pictures of cats and other data for networks to learn on.

The other major was the increased availability of processing power. If we're effectively training an artificial brain, let's think about our own for a second. How many gigabytes of information do we observe in each second? What's the "frame rate" of life? What is the resolution of the human eye?

While modern day AI is impressive, we humans process significantly more data in a second than an AI. One way that machines can try to catch up to us humans is with higher computing power.

> **Note:**
>
> For our purposes we use the Python library PyTorch for deep learning. Some PyTorch deep learning examples to get you started are available in the Jupyter Notebook here:
>
> https://colab.research.google.com/drive/1FP6isKvejdiSA17GfqjBBdZhzYjiSQis?authuser=1

## 5.1 What is deep learning?

What is deep learning and why is it important?

Let's first discuss the traditional paradigm for programming. In the traditional programming paradigm, the programmer explicitly defines the rules and logic for the program to follow, to achieve some outcome. This involves writing a series of instructions to describe how the input data is processed into the output data. For example, in a program where the area of a circle is calculated the input would be the radius r and the programmer would define the process or instructions as square r and then multiply by pi, and output this result as the area of the circle. This method is great for when the rules are precisely known such as when doing mathematics or processing data.

In contrast to this, deep learning flips this traditional paradigm of defining the rules. Deep learning involves a data-driven approach where the rules (or patterns) are learned from the data, rather than being explicitly programmed. Instead of defining the rules, the programmer provides the model architecture (for example a neural network) along with a large amount of training data that contains both inputs and the correct outputs for those inputs. The model is then "trained" on the training data and from this it learns the underlying patterns by adjusting the internal parameters in the defined model architecture. This is done through an iterative process where the model learns

the ideal parameters which minimize the difference between the predictions and the actual outputs.

The key difference is that in the traditional approach, rules are defined explicitly, while in the deep learning approach, the rules are implicitly learned from the data. This makes deep learning powerful for handling complex problems where the relationships between inputs and outputs are difficult to define mathematically, like natural language processing or image classification.

## 5.2 The deep learning process

Normally to build a non-deep-learning classifier or regressor, you come up with a set of rules, and program those rules. Then when you want to classify things, you give it a piece of data, and the rules are used to select a category.

With deep learning, we first need a list of variables: inputs and their corresponding outputs. We don't need to know the relationship between the inputs and the outputs, but the more accurately we can observe their values, the better. We will call this list of examples the dataset.

We then train a model by showing it the dataset and the correct outputs. The model keeps taking guesses and finds out if it was right or not. It slowly learns to correctly categorize over the course of training.

The fundamental difference is that instead of humans needing to identify and program the rules, a deep learning algorithm will learn them on its own. This is a fundamental shift in the way we can build software systems.

Deep learning is not always the right choice. If the decisions that need to be made to produce a result are very clear, it's likely that it should just be programmed in the traditional way. However, if the rules are nuanced or complex, and humans would have a hard time describing them, let alone programming them, deep learning is a great choice.

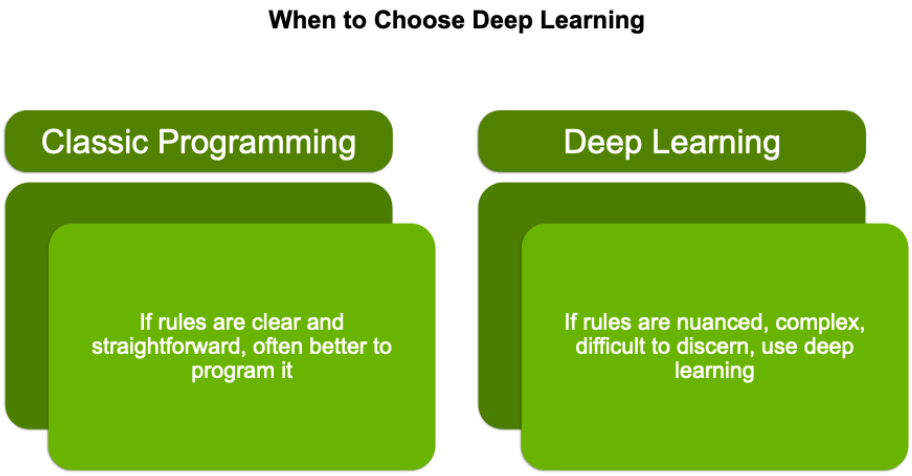



*Figure 7: When to use classical programming vs deep learning.*

One thing that separates Deep Learning from traditional machine learning is the depth and complexity of the networks that are used. For especially complicated tasks such as natural language understanding (used for translations, text generation) networks can have billions of parameters.

The deep in deep learning refers to the many layers in a model that learn the important rules necessary for completing a task. These are often called hidden layers.

## 5.3 The deep learning components

Now we will elaborate on the components that make up a deep learning model. We have discussed these briefly in the Natural Language Processing chapter, but we will go into more detail here.

### 5.3.1 Input layer

The input layer is the first layer of a deep learning model. It is the entry point of the data. Each node (can also be known as a neuron) in this layer represents a feature of the input data. For example, if you are analyzing house prices and the data consists of square footage, number of bedrooms, and number of bathrooms, there would be 3 nodes in the input layer. If you were analyzing a square image that is 20 by 40 pixels, there would be 800 nodes in the input layer. This layer does not perform any calculations, it just feeds the data into the model and into the next layer.

### 5.3.2 Hidden layers

The layers between the very first layer of the model (the input layer) and the very last layer of the model (the output layer) are called hidden layers. They perform the intermediate computations which cause the model to learn from data. Each hidden layer contains neurons, where each neuron does the following:

- Computes weighted sum of the inputs to that layer
- Applies an activation function to the weighted sum
- Sends the result of the weighted sum multiplied by the activation function to the next layer

### 5.3.3 Activation function

After each hidden layer's weighted sum is calculated, an activation function is applied. Let's imagine for a second that there is no activation function, only the weighted sums. No matter how many layers of weighted sums we pass the data through, adding up a series of linear functions will always result in a linear function. One of the strengths of deep learning is determining non-linear patterns and relations within the data. This is made possible by the activation layer. In other words, the activation layer introduces non-linearity to the model. This non-linearity allows the model to learn more complex patterns. Common activation functions include ReLU (Rectified Linear Unit), which outputs zero for negative values and the input itself for positive values, sigmoid for probabilities (output between 0 and 1), and tanh (hyperbolic tangent) for outputs between -1 and 1. The choice of activation function can significantly affect the performance and convergence of the model.

### 5.3.4 Output layer

The output layer produces the final result of the model. The predictions are based on the features that are learned in the hidden layers. For classification, this might include a softmax function to output probabilities for each class. For regression, it could be a linear function to predict continuous values. The number of nodes in the output layer represent the number of output features.

### 5.3.5 Loss function

The loss function determines how well the predictions from the model match with the actual values. In other words, it quantifies the difference between the predicted output (y hat) and the true target (y) for each input in the training set into a single value representing the error of the model. The goal is to minimize this loss.

There are different loss functions, these can include:

- **Mean squared error (MSE):** Used for regression, calculates the averaged squared difference between the predicted and actual values. Large errors are penalized heavily.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

- **Cross-entropy loss (also known as log loss):** Commonly used for classification, it measures the difference between the predicted probability distribution and true distribution. Predictions that are far from the true label are heavily penalized.

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

The choice of loss function will depend on the task at hand and how you would like to measure the performance of the model. This choice is very important as it directly will impact how the model adjusts during training.

### 5.3.6 Backpropagation

Backpropagation is a key element of neural networks and is the process that is used to update the model's weights based on the calculated loss. It involves the following steps:

1. **Forward Pass:** The input data is passed through the neural network (i.e. the input layer -> hidden layers -> output layer), and the initial predicted output is calculated.

2. **Compute loss:** Using the loss function, the loss is calculated quantifying how accurate the model was.

3. **Backward pass:** The gradient (also known as the slope) of the loss with respect to each weight is calculated. This quantifies how much each weight in the model contributed to the overall loss.

4. **Gradient computation:** For each weight in the model, backpropagation computes a partial derivative of the loss with respect to that weight. The gradient tells us how the loss changes with small changes in the weight.

5. **Gradient update:** The optimizer uses these gradients to adjust the weights, and then moves them in a direction which reduces the loss.

### 5.3.7 Optimizer

The optimizer uses the gradients that were calculated by the backpropagation to adjust the model's weights such that the loss is minimized. It determines how to change the weights. In other words, it fine-tunes the parameters in the neural network during model training. Here a balance must be found between convergence speed and accuracy.

**Gradient descent**

How exactly does the model "learn", i.e. fine tune its parameters? This is done through an operation called gradient descent. Gradient descent finds the set of parameters that minimizes the loss function.

To start off with a simple example, imagine there is only one parameter (or weight) in the model. You can make predictions on the training data with different values of that parameter and calculate the loss function for each value.

> **Note:**
>
> The rate of change of a function is called a gradient (sometimes this will be called the derivative or the tangent). A practical example of a gradient is economic inflation, which is the change in the price of goods and services. A commonly referenced metric is the rate of inflation which is a second derivative (in other words, the gradient of the gradient).
>
> **Fun fact about derivatives:**

In 1972, President Richard Nixon stated that *the rate of increase of inflation was decreasing*. Journalist Hugo Rossi noted that this was "the first time a sitting president used the third derivative to advance his case for re-election." This is a third derivative because inflation itself is a derivative, and he referenced the rate of inflation (derivative 2) was decreasing (derivative 3).

**Back to optimizers**

While gradient descent is the most common form of optimizer, there are also other options.

Some common optimizers include:

- Stochastic gradient descent (SGD)
- Adam
- RMSprop

| **Optimizer** | **Advantage** | **Disadvantage** |
|---|---|---|
| Stochastic gradient descent (SGD) | Fast and efficient, good for large datasets | Convergence to optimal parameters might be noisy |
| Momentum | Converges quickly | Might not find most optimal set of parameters |
| Adagrad | Adapts the learning rate, good for sparse data | Learning rate decays over time |
| RMSprop | Prevents the rapid learning rate decay | Requires more fine-tuning |
| Adam | Converges quickly, adapts the learning rate | Uses lots of memory and may not find the global minimum |
| AdamW | Generalizes to more situations better | More intensive to compute |

Common choices are SGD, Adam, and AdamW.

**Learning rate**

An important parameter in optimization is the learning rate. This can be thought of as the step size that is taken in each step of the gradient descent. With a learning rate that is too small, you would have accurate results, but the model will be computationally expensive. With a learning rate that is too large, you will have less accurate results, but the optimizer will converge more quickly. It is important to strike a balance between these two extremes.

It is good practice to start with a small learning rate and adjust. This number would usually look something like 0.001, or 0.01, for example. You should pick this based on your chosen optimizer.

### 5.3.8 Training over many epochs

One cycle of going through the steps outlined above is called one epoch. The advantage of deep learning is training over many epochs, to find the most optimal parameters to make future predictions on data with unknown results.

It is very important to choose an appropriate number of epochs. With too few the model could be underfit, and with too many the model could be overfit. It is a good idea to track the loss over epochs, this is usually graphed. You want enough epochs such that the loss converges to a value.

### 5.3.9 Hyperparameter tuning

After going through the steps of training a neural network through many epochs, there is one final high level step to complete. This is hyperparameter tuning which can be thought of as tuning the dials that control the learning process itself. This can include:

- **Learning rate:** Adjusting the step size of the optimizers to reach a good trade off between computational price and efficiency
- **Number of epochs:** Adjusting the number of iterations the model goes through to reach a good trade off between underfitting, overfitting, and time to compute
- **Optimizer:** Adjusting the optimizer method (Adam, stochastic gradient descent, etc) to be best suited to your data and model architecture

- **Model parameters:** Adjusting parameters within the model such as the number of layers, neurons per layer, or activation functions (such as ReLU). This will affect how the model learns patterns within the data.

**Methods for hyperparameter tuning**

There are a few methods for hyperparameter tuning, varying in complexity, how computationally expensive they are, how accurate they are, and how long they will take to compute.

The simplest is **manual search**, where the researcher selects and tests values for each hyperparameter based on their prior knowledge and intuition. This is clearly easy to implement, but would take longer and is much less precise.

A common one is **grid search**, where the researcher defines a range for each hyperparameter that is to be fine-tuned. Every possible combination is then systematically tested. This results in the most precise hyperparameter values (within the defined sets), but is the most computationally expensive. Keep in mind that each additional hyperparameter adds a new dimension to the hyperparameter grid, drastically increasing the computations required, so for a large number of hyperparameters this can be inefficient.

Another common method is **random search**, where the researcher defines a range for each parameter just like in grid search, but instead of testing every possible combination, combinations are randomly sampled. This is less computationally expensive than grid search, with a trade off that you will not get the most optimal set of hyperparameters but rather an approximation.

A method that combines the advantages of grid and random search is **Bayesian optimization search**. This is like a random search, but instead of sampling hyperparameter combinations all with the same probability, you have an idea of the probability distributions of the most optimal hyperparameters. This can search a larger hyperparameter space (in both dimension and range), to efficiently find a good approximation of the optimal set of hyperparameters.

## Key Points

- Machine learning flips the paradigm of classical programming
- There are many different data types that can be used for ML analysis
- Supervised machine learning uses labelled data

- Unsupervised machine learning uses unlabelled data
- Reinforcement machine learning uses unlabelled data and a reward mechanism
- Python environments can be set up with Anaconda or venv
- Accuracy metrics are used to evaluate ML models
- Supervised machine learning is typically for regression or classification
- Natural language processing is based off neurons in the brain
- Deep learning is a powerful subset of AI
- The deep learning process also flips the traditional paradigm in traditional programming of defining rules, DL learns the rules

# References and Further Resources:


- Data in machine learning - Geeks for Geeks
- What is a Machine Learning Model - Databricks
- 8 Machine Learning Models Explained in 20 Minutes - DataCamp
- What is supervised learning? - IBM
- Choosing the right estimator [the scikit-learn ML map] - Scikit-learn
- Supervised machine learning - Geeks for Geeks
- What is unsupervised learning? - IBM
- 6 Dimensionality reduction algorithms with Python - Machine Learning Mastery
- Reinforcement Learning: What it is, Algorithms, Types, and Examples - Turing
- Anaconda - Anaconda website
- venv - Creation of Virtual Environments [venv official documentation] - Virtual environment documentation

- Python Virtual Environments: A Primer - Real Python
- Iris dataset - Scikit-learn Iris
- Metrics to Evaluate Your Machine Learning Algorithm - Towards Data Science
- Regression: Definition, Analysis, Calculation, and Example - Investopedia
- 4 Types of Classification Tasks in Machine Learning - Machine Learning Mastery
- What is NLP (Natural Language Processing) - IBM Natural Language Processing
- What is deep learning? - IBM Deep Learning
- Introduction to Deep Learning - Geeks for Geeks deep learning
- Everything you need to know about “Activation Functions” in Deep learning models - Towards Data Science - Activation Functions
- Loss Functions in Machine Learning Explained - Data Camp
- A Comprehensive Guide on Optimizers in Deep Learning - Analytics Vidhya
- What is gradient descent? - IBM
- Learning Rate in Neural Network - Geeks for Geeks
- Tuning the Hyperparameters and Layers of Neural Network Deep Learning - Analytics Vidhya