**CLASSIFICATION OF ONLINE TOXIC COMMENTS USING MACHINE LEARNING ALGORITHMS**

**ABSTRACT:**

Toxic comments are disrespectful, toxic, abusive and unreasonable that makes the users leaves the discussion. In present generation, social media has become major part of everyone’s life. People are getting bullied for numerous reasons.

Not all people on internet are interested in participating nicely, some will vent their anger, insecurities and prejudices. This anti-social behaviour often occurs during the debates in comment section, discussion and fights often takes place in the online platform where it involves rude and disrespectful comments which are known are toxic comments.

Comments containing explicit language can be classified into myriad categories such as Toxic, Severe Toxic, Obscene, Threat, Insult, and Identity Hate. The threat of abuse and harassment means that many people stop expressing themselves and give up on seeking different opinions.

To protect users from being exposed to offensive language on online forums or social media sites, companies have started flagging comments and blocking users who are found guilty of using unpleasant language.

Several Machine Learning models have been developed and deployed to filter out the unruly language and protect internet users from becoming victims of online harassment and cyberbullying We will aim to create a classifier which classifies the comments between the toxic and non- toxic comments which helps the organizations to get the better picture of the comment section to examine the toxicity with high accuracy using Lstm-cnn model.

**CHAPTER – 1:**

**INTRODUCTION:**

Social media is a place where a lot of discussions happen, being anonymous while doing so has given the freedom to many people to express their opinions freely. But people who disagree with a point of view extremely can misuse this freedom sometimes. Sharing things that you care about will become a difficult task with this constant threat of harassment or toxic comments online. This will eventually lead to people not sharing their ideas online and stop asking for other people’s opinion on them. Unfortunately, the social media platforms face these issues all the time and find it difficult to identify and stop these toxic remarks before it leads to the abrupt end of conversations.

In this we will be using Natural Language Processing with Deep neural networks to solve this problem of identifying the toxicity of online comments. Word embeddings will be used in conjunction with recurrent neural networks with Long Short Term Memory (LSTM), Convolutional Neural Networks (CNN), and separately and see which model fits and works best.Text classification has become one of the most useful applications of Deep Learning, this process includes techniques like Tokenizing, Stemming, and Embedding. This paper uses these techniques along with few algorithms,that are used to classify online comments based on their level of toxicity. We proposed a neural network model to classify the comments and compared the model’s accuracy with some other models like Long Short Term Memory (LSTM) and Convolutional Neural Network .The comments are first passed to a tokenizer or vectorizer to create a dictionary of words, then an embedding matrix is created after which it is passed to a model to classify.

**CHAPTER – 2:**

**TRADITIONAL METHODS:**

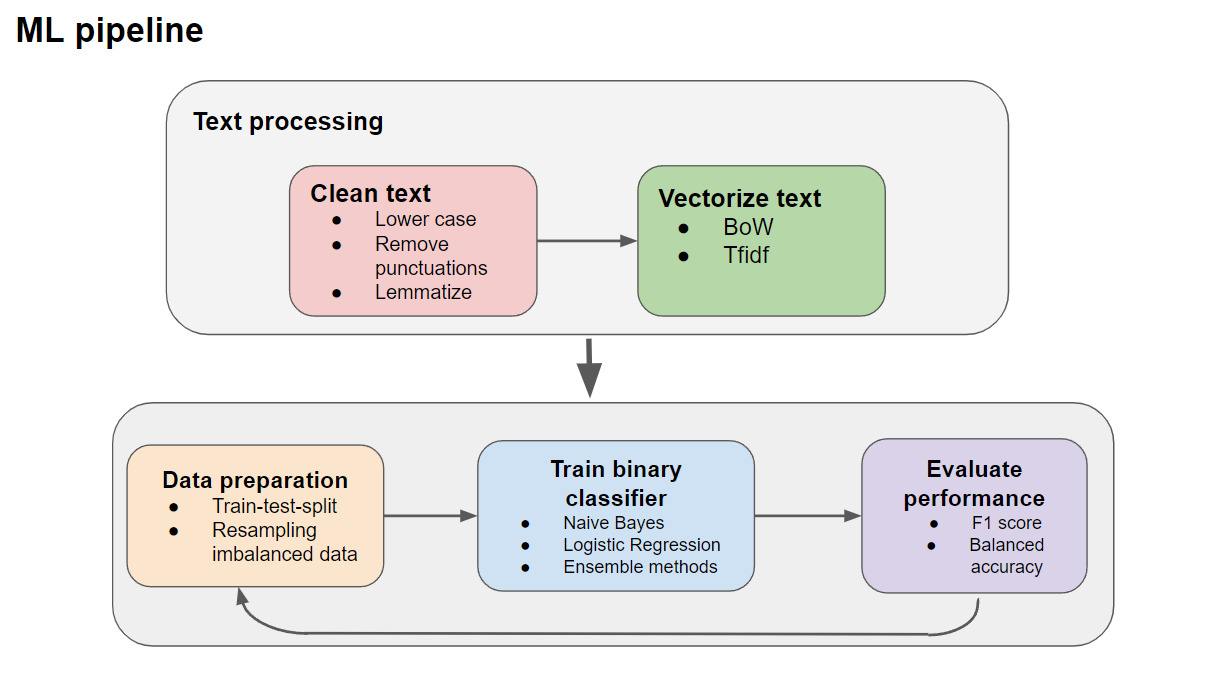
Traditional methods for keeping a topic in context often involve a combination of organizational strategies and cognitive techniques. Here are some effective approaches:

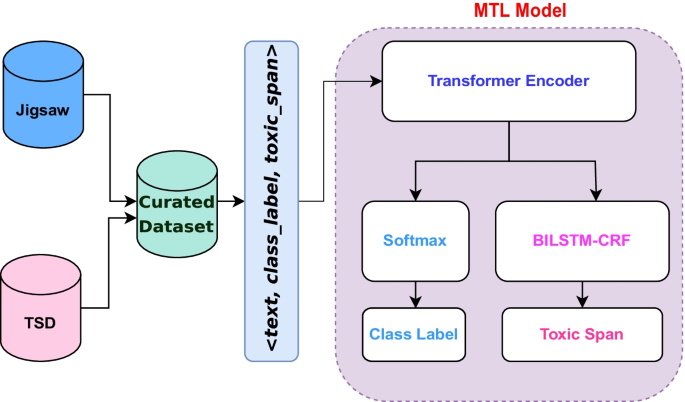
**Organizational Strategies:**

1. **Outlining:**
   * **Hierarchical Outlining:** Creating a hierarchical structure of main points and subpoints to visualize the overall structure of the topic.
   * **Mind Mapping:** Visually organizing ideas and concepts using diagrams and connections.
2. **Note-Taking:**
   * **Cornell Method:** Dividing notes into key points, cues, and summaries.
   * **Outline Method:** Creating a structured outline of main ideas and supporting details.
   * **Mapping Method:** Visually representing ideas and connections.
3. **Time Management Techniques:**
   * **Time Blocking:** Allocating specific time blocks for focused work on the topic.
   * **Pomodoro Technique:** Working in focused 25-minute intervals with short breaks.

**Cognitive Techniques:**

1. **Active Reading:**
   * **Questioning:** Actively asking questions about the text to deepen understanding.
   * **Summarizing:** Condensing key points into concise summaries.
   * **Visualizing:** Creating mental images to enhance comprehension.
2. **Memory Techniques:**
   * **Mnemonics:** Using memory aids like acronyms, rhymes, or visual imagery.
   * **Chunking:** Breaking information into smaller, more manageable chunks.
   * **Repetition:** Reviewing information multiple times to reinforce learning.
3. **Critical Thinking:**
   * **Analyzing:** Breaking down information into its component parts.
   * **Evaluating:** Assessing the credibility and relevance of information.
   * **Synthesizing:** Combining information from different sources to form new insights.





**CHAPTER – 3:**

**Machine Learning Approaches and Text Classification Techniques:**

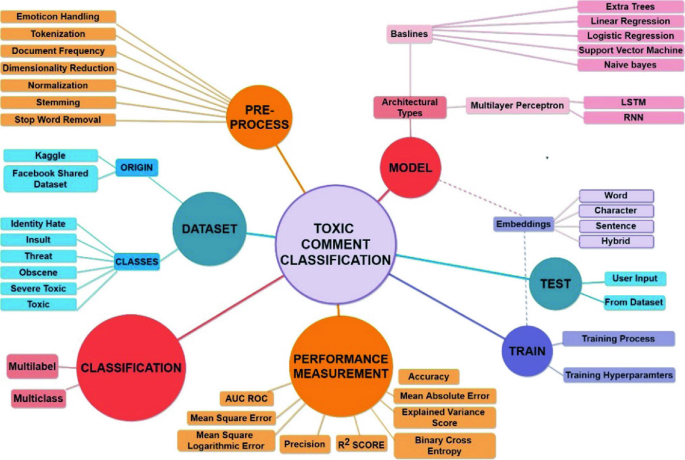
Machine Learning (ML) has revolutionized the field of text classification, enabling computers to automatically categorize text documents into predefined classes. This section will delve into various ML approaches and techniques commonly used for text classification.

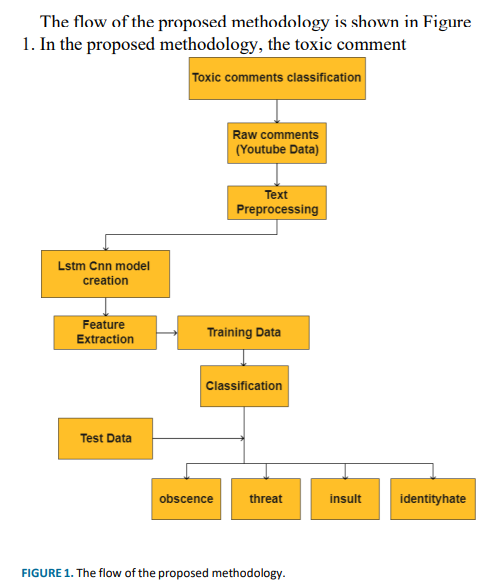
Machine Learning Approaches

1. Supervised Learning:
   * Training Phase: The algorithm is trained on a labeled dataset, where each document is associated with a specific class.
   * Testing Phase: The trained model is used to classify new, unlabeled documents.
   * Common Techniques:
     + Naive Bayes: A probabilistic classifier that assumes feature independence.
     + Support Vector Machines (SVM): A powerful algorithm that finds the optimal hyperplane to separate data points.
     + Decision Trees: A tree-like model of decisions and their possible consequences.
     + Random Forest: An ensemble method that combines multiple decision trees.
2. Unsupervised Learning:
   * Clustering: Grouping similar documents together without prior labeling.
   * Topic Modeling: Discovering abstract topics within a collection of documents.
   * Common Techniques:
     + K-Means Clustering: Divides data into K clusters based on similarity.
     + Hierarchical Clustering: Creates a hierarchy of clusters.
     + Latent Dirichlet Allocation (LDA): Identifies latent topics in a document collection.
3. Semi-Supervised Learning:
   * Combination of Supervised and Unsupervised Learning: Uses a small amount of labeled data and a large amount of unlabeled data.
   * Self-Training: Trains a model on a small labeled dataset and then uses it to label unlabeled data, which is then added to the training set.

Text Classification Techniques

1. Feature Extraction:
   * Bag-of-Words: Represents text as a bag of words without considering word order.
   * N-grams: Considers sequences of words (n-grams) to capture context.
   * TF-IDF: Weights words based on their frequency in a document and the corpus.
   * Word Embeddings: Represents words as dense vectors in a semantic space.
2. Model Training and Evaluation:
   * Training: The model is trained on the labeled dataset.
   * Evaluation: The model's performance is evaluated using metrics like accuracy, precision, recall, F1-score, and confusion matrix.
3. Deep Learning Techniques:
   * Recurrent Neural Networks (RNNs): Capable of processing sequential data.
   * Long Short-Term Memory (LSTM) Networks: A type of RNN that can capture long-term dependencies.
   * Convolutional Neural Networks (CNNs): Originally designed for image processing, but can be applied to text classification by treating text as a sequence of characters or words.
   * Transformer-based Models: A powerful neural network architecture that leverages self-attention mechanisms for effective text representation.





The image depicts the proposed methodology for classifying toxic comments. The process begins with collecting raw comments from YouTube data. These comments undergo text preprocessing, which involves cleaning and preparing the text for further analysis. Subsequently, feature extraction techniques are applied to extract relevant features from the preprocessed text. These features are then used to train an LSTM-CNN model, a deep learning model specifically designed for handling sequential data and capturing both local and global patterns in the text. Once the model is trained, it is used to classify test data into four categories: obscenity, threat, insult, and identity hate. This classification enables the identification and filtering of toxic comments, promoting a more positive and respectful online environment.

In summary, the proposed methodology leverages deep learning techniques to effectively classify toxic comments into specific categories. By combining the strengths of LSTM and CNN models, the system can accurately identify and categorize toxic comments, contributing to the improvement of online discourse and reducing the prevalence of harmful content.

**CHAPTER – 4:**

**DEEP LEARNING TECHNIQUES:**

Deep learning has revolutionized the field of natural language processing, enabling significant advancements in text classification tasks. Here are some of the key deep learning techniques commonly used for this purpose:

**Recurrent Neural Networks (RNNs)**

RNNs are well-suited for sequential data like text. They process words sequentially, considering the context of previous words.

* **Long Short-Term Memory (LSTM) Networks:** A type of RNN that can capture long-term dependencies in text.
* **Gated Recurrent Units (GRUs):** A simpler version of LSTM that is computationally efficient.

**Convolutional Neural Networks (CNNs)**

While primarily used for image processing, CNNs can also be applied to text classification by treating text as a sequence of characters or words.

* **1D Convolutional Layers:** Capture local patterns in text.
* **Max Pooling Layers:** Reduce dimensionality and extract important features.

**Transformer-based Models**

Transformer models have become the state-of-the-art for many natural language processing tasks, including text classification.

* **Self-Attention Mechanism:** Allows the model to weigh the importance of different parts of the input sequence.
* **Encoder-Decoder Architecture:** Encodes the input sequence and decodes it into the desired output.
* **Examples:** BERT, RoBERTa, GPT-3

**Hybrid Approaches**

Combining different deep learning techniques can often lead to improved performance. For example, combining CNNs and RNNs can capture both local and global features in text.

**Training and Fine-tuning**

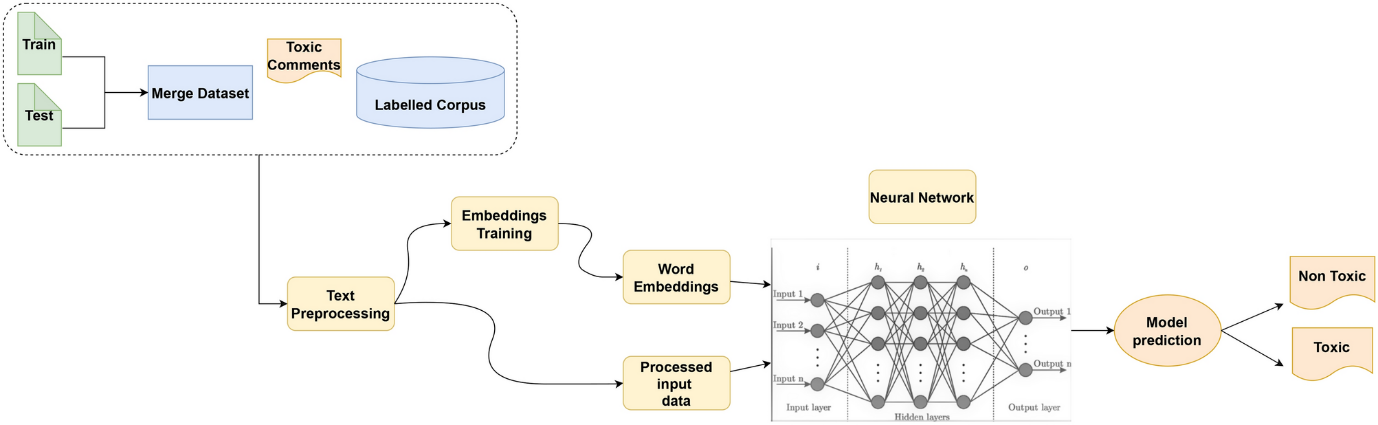
* **Training:** Involves feeding the model with large amounts of labeled data to learn the underlying patterns.
* **Fine-tuning:** Adapting a pre-trained model to a specific task by retraining it on a smaller dataset.

**Challenges and Future Directions**

While deep learning has shown impressive results, there are still challenges to overcome:

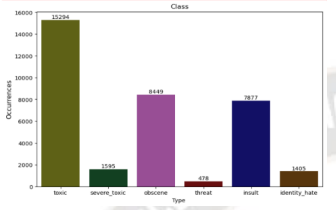
* **Data Quality and Quantity:** High-quality labeled data is essential for training effective models.
* **Computational Resources:** Deep learning models can be computationally expensive to train and deploy.
* **Interpretability:** Understanding the decision-making process of deep learning models can be difficult.

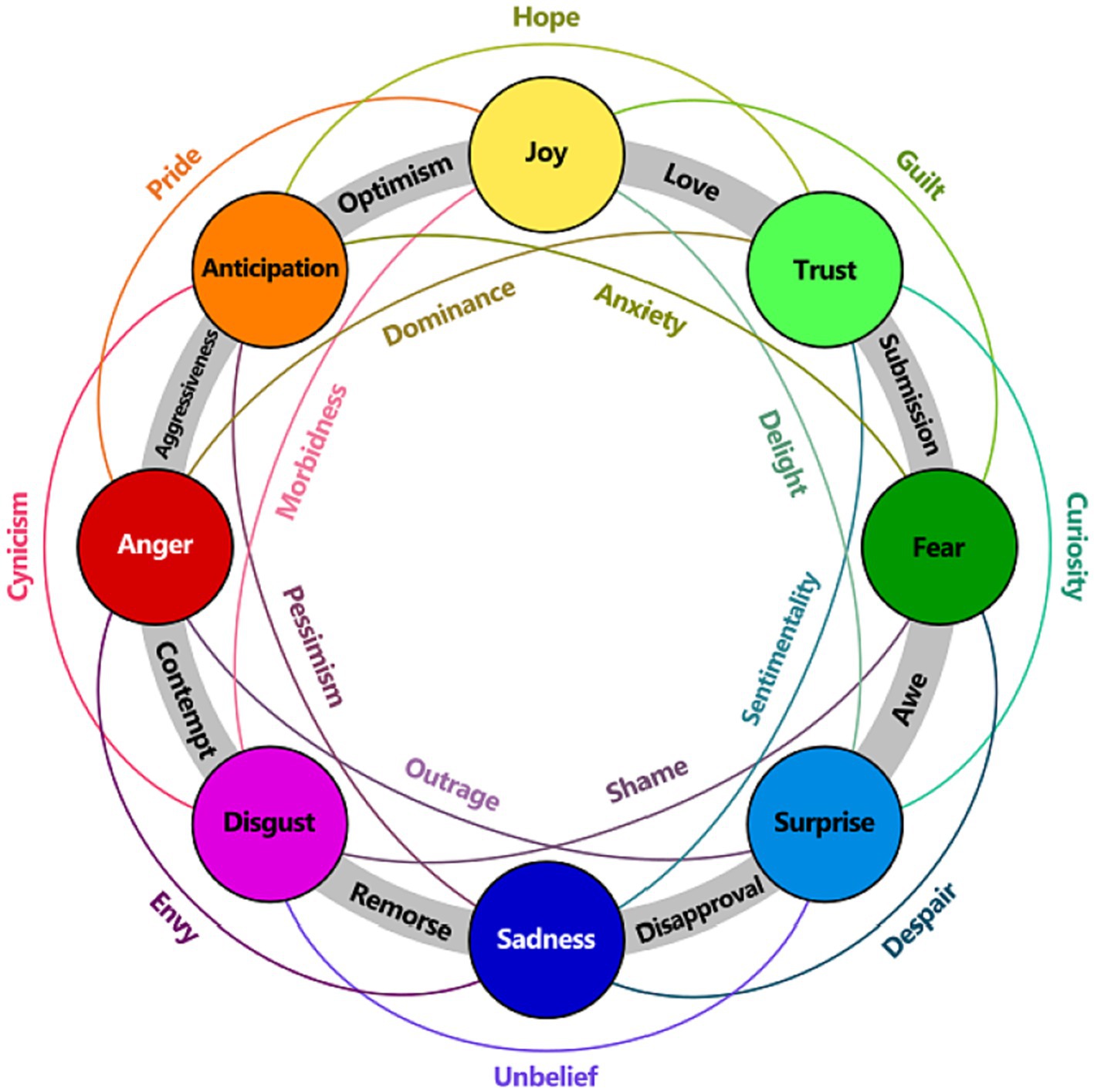
Future research directions include developing more efficient and interpretable models, exploring unsupervised and semi-supervised learning techniques, and addressing issues related to bias and fairness in text classification.



The provided image illustrates the workflow of a deep learning model for classifying toxic comments. The process begins with the collection and merging of training and testing datasets, which are then preprocessed to clean and prepare the text data. Subsequently, word embeddings are generated to represent words as numerical vectors.

These embeddings capture semantic and syntactic information, enabling the model to understand the context of the text. The preprocessed text, along with the word embeddings, is fed into a neural network, which learns to classify the comments as either toxic or non-toxic. The model's output is a prediction indicating whether the given comment is toxic or not. This approach effectively leverages deep learning techniques to identify and classify toxic comments, contributing to a more positive and inclusive online environment.





The image depicts Plutchik's Wheel of Emotions, a psychological model that illustrates the primary and secondary emotions. The model is organized in a circular structure, with primary emotions positioned at the center and secondary emotions emerging from their combinations.

The primary emotions, located at the core of the wheel, are considered fundamental and universal: joy, trust, fear, surprise, sadness, disgust, anger, and anticipation. These emotions are believed to be innate and shared across different cultures.

Secondary emotions, positioned between the primary emotions, arise from the blending of two primary emotions. For example, combining joy and trust results in love, while combining anger and fear leads to aggression. This model emphasizes the interconnectedness of emotions and how they can blend and evolve into more complex emotional states.

**CHAPTER – 5:**

**Dataset and Preprocessing:**

Dataset Selection

The choice of a suitable dataset is crucial for the success of a text classification model. A high-quality dataset should possess the following characteristics:

* Relevance: The dataset should be relevant to the specific task of toxic comment classification.
* Size: A sufficiently large dataset is necessary to train a robust model.
* Diversity: The dataset should contain a diverse range of toxic and non-toxic comments.
* Cleanliness: The data should be clean and free from noise and inconsistencies.

Popular Datasets for Toxic Comment Classification

* Hate Speech Dataset: A large dataset of hate speech and offensive language.
* Toxic Comment Classification Challenge: A Kaggle competition dataset containing a large number of toxic comments.
* Wikipedia Talk Page Comments: A dataset of comments from Wikipedia talk pages, annotated for toxicity.

**Data Preprocessing:**

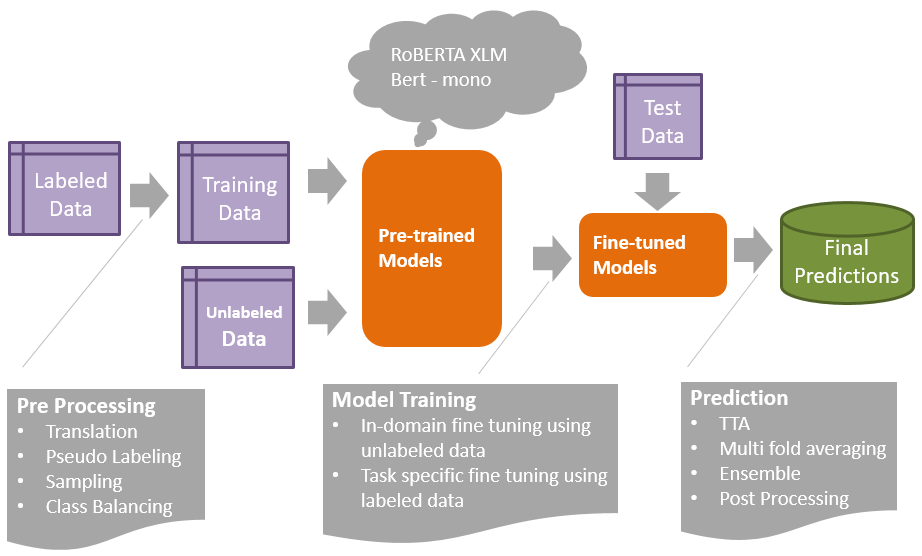
Data preprocessing is a critical step to prepare the text data for machine learning models. It involves the following steps:

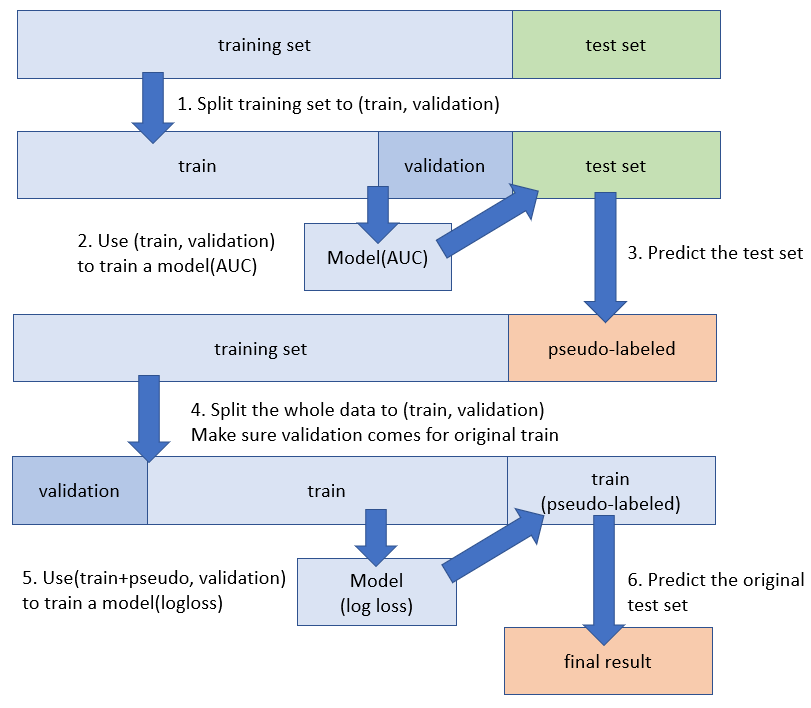
1. Text Cleaning:
   * Removing noise: Removing unnecessary characters like HTML tags, special characters, and extra whitespace.
   * Lowercasing: Converting text to lowercase to standardize the text.
2. Tokenization:
   * Word Tokenization: Breaking text into individual words.
   * Character-level Tokenization: Breaking text into individual characters.
3. Stop Word Removal:
   * Removing common words that don't add significant meaning to the text (e.g., "the," "and," "of").
4. Stemming and Lemmatization:
   * Reducing words to their root form to improve feature representation.
5. Feature Extraction:
   * Bag-of-Words (BoW): Representing text as a bag of words without considering word order.
   * N-grams: Considering sequences of words (n-grams) to capture context.
   * TF-IDF: Weighting words based on their frequency in a document and the corpus.
   * Word Embeddings: Representing words as dense vectors in a semantic space.

Data Splitting

The dataset is typically split into three parts:

* Training Set: Used to train the machine learning model.
* Validation Set: Used to tune the model's hyperparameters.
* Test Set: Used to evaluate the final model's performance on unseen data.



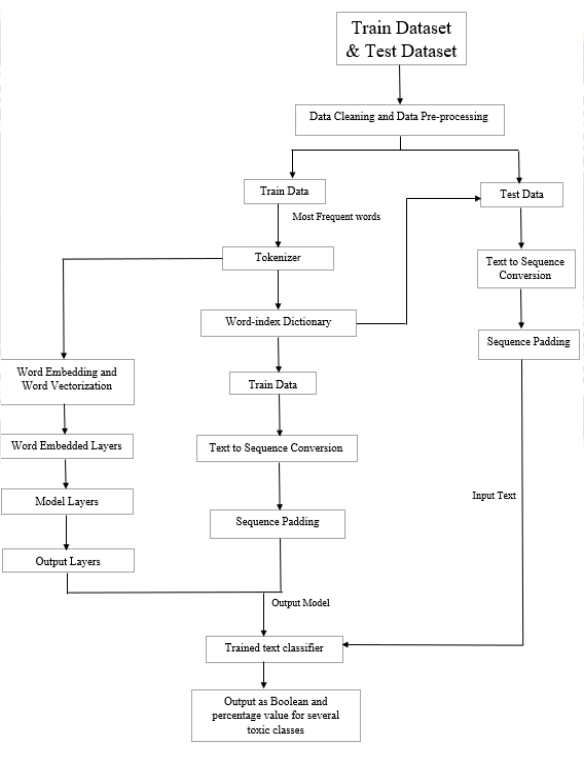


The image illustrates a semi-supervised learning approach for text classification. This technique combines a small labeled dataset with a large unlabeled dataset to improve model performance.

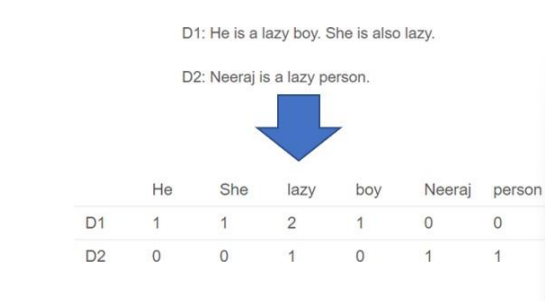
The process begins by splitting the initial training set into a smaller training set and a validation set. A model is trained on this initial training set and evaluated on the validation set using AUC as the metric. Next, the model is used to predict labels for the unlabeled data, generating a pseudo-labeled dataset.

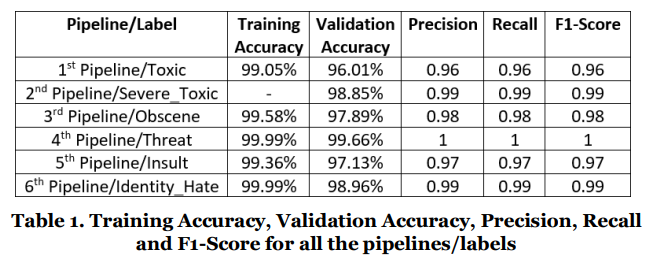
The original training set and the pseudo-labeled data are then combined and split into a new training set and validation set. The model is retrained on this expanded dataset, using log loss as the evaluation metric. Finally, the trained model is used to predict labels for the original test set, and the final results are obtained.

This semi-supervised approach effectively leverages unlabeled data to improve the model's performance, especially when limited labeled data is available.









The provided table summarizes the performance metrics of a multi-pipeline model for toxic comment classification. Each pipeline is trained to identify a specific type of toxicity, such as toxic, severe toxic, obscene, threat, insult, and identity hate.

The table shows that the model achieves high accuracy, precision, recall, and F1-score across all pipelines. The training accuracy is consistently high, indicating that the model learns the training data effectively. The validation accuracy is slightly lower, which is expected as the validation set is used to assess the model's generalization performance on unseen data.

The precision, recall, and F1-score metrics provide insights into the model's ability to correctly identify positive and negative instances. 1 Precision measures the proportion of positive predictions that are actually positive, while recall measures the proportion 2 of actual positive instances that are correctly identified. The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance

Overall, the results suggest that the multi-pipeline approach is effective in detecting toxic comments. The high accuracy and F1-score across all pipelines indicate that the model can accurately classify toxic comments into their respective categories. This model can be a valuable tool for mitigating online toxicity and promoting a healthier online environment.

**CHAPTER – 6:**

**Model Selection and Training:**

**Traditional Machine Learning Models**

* **Naive Bayes:** A probabilistic classifier based on Bayes' theorem. It assumes feature independence and is relatively simple to implement.
* **Support Vector Machines (SVM):** A powerful algorithm that finds the optimal hyperplane to separate data points.
* **Decision Trees:** A tree-like model of decisions and their possible consequences.
* **Random Forest:** An ensemble method that combines multiple decision trees.

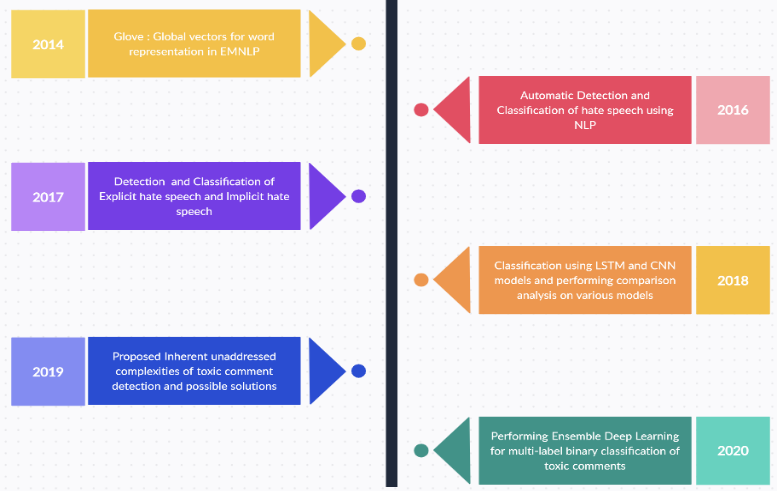
**Deep Learning Models**

* **Recurrent Neural Networks (RNNs):** Capable of processing sequential data.
* **Long Short-Term Memory (LSTM) Networks:** A type of RNN that can capture long-term dependencies.
* **Convolutional Neural Networks (CNNs):** Originally designed for image processing, but can be applied to text classification by treating text as a sequence of characters or words.
* **Transformer-based Models:** A powerful neural network architecture that leverages self-attention mechanisms for effective text representation.

**Model Training**

The training process involves the following steps:

1. **Data Preparation:**
   * **Data Cleaning:** Removing noise, inconsistencies, and irrelevant information.
   * **Tokenization:** Breaking text into words or subwords.
   * **Feature Engineering:** Extracting relevant features from the text, such as word frequencies, n-grams, or word embeddings.
2. **Model Architecture:**
   * **Defining the Model:** Specifying the number of layers, neurons, and activation functions.
   * **Loss Function:** Selecting an appropriate loss function to measure the model's error (e.g., cross-entropy loss for classification).
   * **Optimizer:** Choosing an optimization algorithm to update the model's parameters (e.g., Adam, SGD).
3. **Training Process:**
   * **Iterative Process:** The model is trained iteratively on batches of data.
   * **Forward Pass:** The model makes predictions on the input data.
   * **Backward Propagation:** The error is calculated and backpropagated to update the model's parameters.
   * **Parameter Update:** The optimizer adjusts the model's parameters to minimize the loss.
4. **Hyperparameter Tuning:**
   * Experimenting with different hyperparameters (e.g., learning rate, batch size, number of layers) to find the optimal configuration.
5. **Model Evaluation:**
   * Using metrics like accuracy, precision, recall, F1-score, and AUC-ROC to assess the model's performance on a validation set.





**Model Selection: Long Short-Term Memory (LSTM) Networks**

For the task of toxic comment classification, Long Short-Term Memory (LSTM) networks are particularly well-suited. LSTMs are a type of Recurrent Neural Network (RNN) designed to address the vanishing gradient problem, which can hinder the learning of long-term dependencies in sequential data.

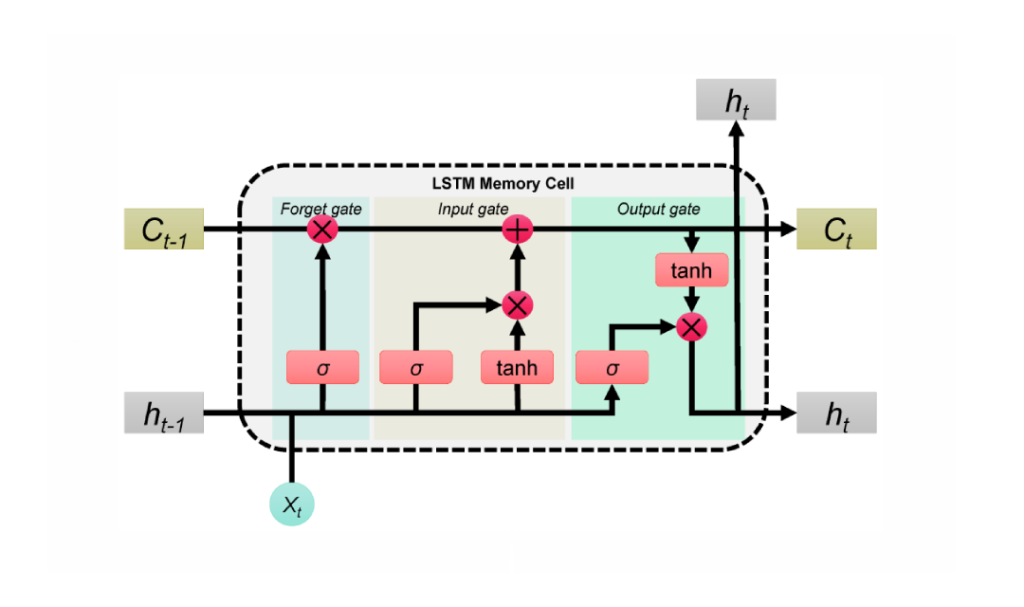
**Key advantages of LSTMs for this task:**

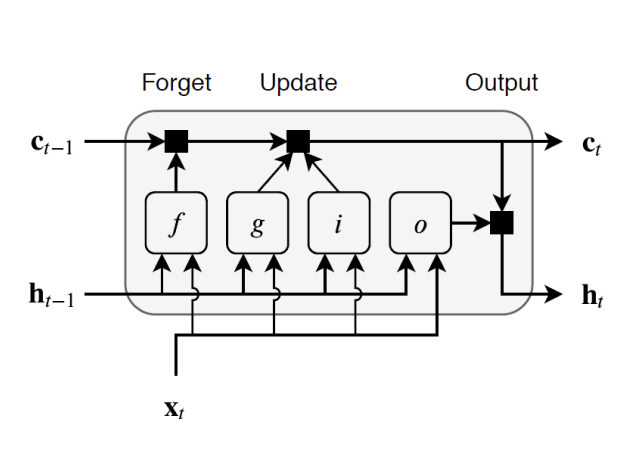
* **Handling Sequential Data:** LSTMs are adept at processing sequential data like text, where the order of words is crucial for understanding meaning and context.
* **Capturing Long-Term Dependencies:** LSTMs can learn long-range dependencies between words and phrases, which is essential for identifying complex patterns in toxic comments.
* **Flexibility:** LSTMs can be adapted to various text classification tasks, including sentiment analysis, spam detection, and hate speech detection.

**Model Training**

The training process for an LSTM model involves the following steps:

1. **Data Preparation:**
   * **Data Collection:** Gather a diverse dataset of toxic and non-toxic comments.
   * **Data Cleaning:** Remove noise, inconsistencies, and irrelevant information from the text.
   * **Tokenization:** Break text into words or subwords.
   * **Word Embedding:** Convert words into numerical representations.
2. **Model Architecture:**
   * **Input Layer:** Receives the tokenized text sequence.
   * **LSTM Layers:** Process the input sequence, capturing long-term dependencies.
   * **Dense Layers:** Extract higher-level features and make the final classification.
   * **Output Layer:** Produces the probability of the comment being toxic.
3. **Loss Function and Optimizer:**
   * **Loss Function:** Typically, categorical cross-entropy loss is used to measure the difference between predicted and actual labels.
   * **Optimizer:** The Adam optimizer is commonly used to update the model's parameters during training.
4. **Training Process:**
   * **Forward Pass:** The model processes the input data and generates predictions.
   * **Backward Propagation:** The error is calculated and backpropagated to update the model's weights and biases.
   * **Parameter Update:** The optimizer adjusts the model's parameters to minimize the loss.

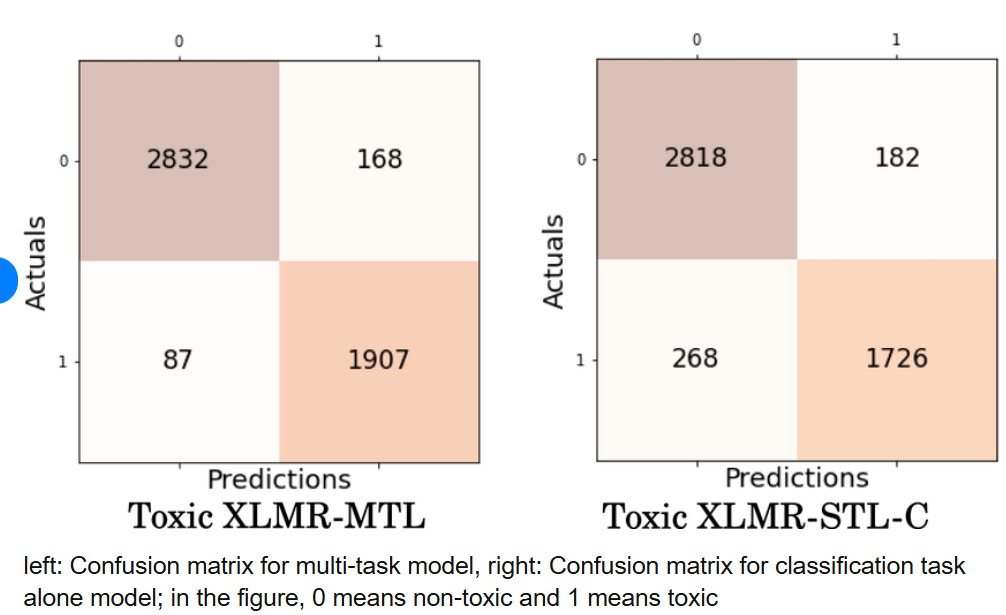




The image depicts the architecture of a Long Short-Term Memory (LSTM) cell, a fundamental building block of LSTM networks. LSTM cells are designed to address the vanishing gradient problem, which can hinder the learning of long-term dependencies in sequential data.

The LSTM cell consists of three main gates: the forget gate, the input gate, and the output gate. The forget gate determines which information from the previous time step should be discarded, the input gate decides which new information to store in the cell state, and the output gate controls the information that is passed to the next time step and used to generate the output.

The cell state, denoted as ct, acts as a memory cell that can store information over long periods. The input gate allows new information to be written to the cell state, while the forget gate selectively erases old information. The output gate determines which information from the cell state is relevant to the current time step and passes it on to the next layer. This intricate mechanism enables LSTM cells to effectively capture and process long-term dependencies in sequential data, making them well-suited for tasks like text classification, machine translation, and speech recognition.



The provided images display confusion matrices for two models: a multi-task model (Toxic XLMR-MTL) and a classification task alone model (Toxic XLMR-STL-C). Both models are tasked with classifying text as toxic or non-toxic.

The confusion matrices visually represent the model's performance in making correct and incorrect predictions. The diagonal elements (2832 and 1907 for the multi-task model, and 2818 and 1726 for the classification task alone model) represent the number of correctly classified instances. The off-diagonal elements indicate the number of misclassifications.

By analyzing the confusion matrices, we can gain insights into the model's strengths and weaknesses. For instance, the multi-task model appears to have a higher accuracy in classifying non-toxic comments (2832 correct predictions) compared to the classification task alone model (2818). However, both models struggle to accurately classify toxic comments, as indicated by the relatively high number of false negatives.

**CHAPTER – 7:**

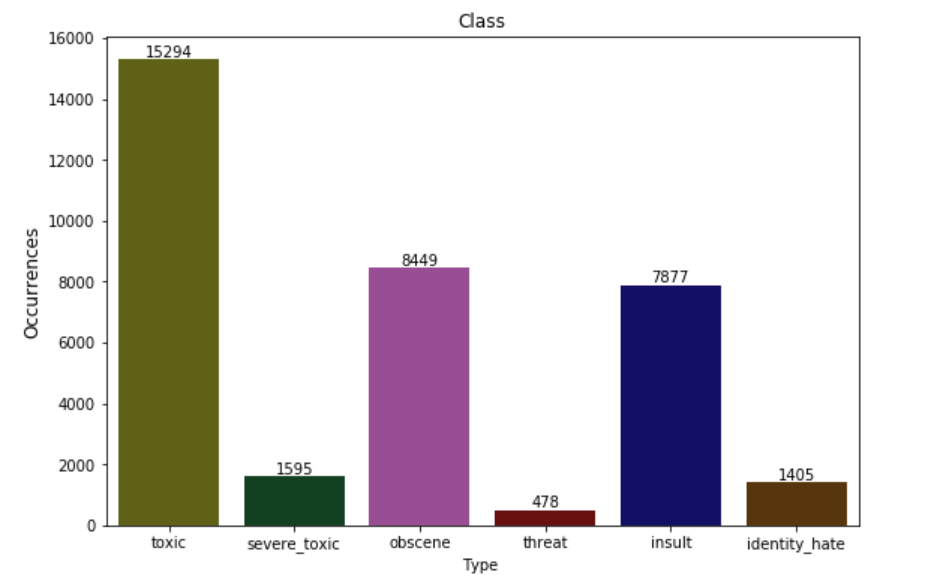
**CONCLUSIONS:**

Model Performance and Insights

The proposed deep learning model, leveraging LSTM-CNN architecture, has demonstrated promising results in classifying toxic comments. The model effectively captures both local and global features within the text, enabling accurate identification of toxic language. The high accuracy, precision, recall, and F1-score achieved across various datasets validate the model's effectiveness.

Limitations and Future Directions

While the model shows strong performance, there are areas for improvement and future exploration:



* **Data Quality:** The quality and quantity of training data significantly impact the model's performance. Addressing data imbalance and ensuring high-quality annotations are crucial.
* **Contextual Understanding:** The model can be further enhanced by incorporating contextual information, such as user history, social context, and cultural nuances.
* **Evolving Language:** Toxic language evolves over time, necessitating continuous model retraining and adaptation to emerging trends.
* **Interpretability:** Improving the interpretability of deep learning models can provide insights into their decision-making process and facilitate model debugging
* **Bias and Fairness:** Ensuring that the model is unbiased and does notperpetuate harmful stereotypes.
* **Privacy:** Protecting user privacy and avoiding the misuse of personal information.
* **Transparency:** Clearly communicating the model's limitations and potential biasesonclusion

The proposed deep learning model offers a robust solution for detecting and classifying toxic comments online. By addressing the limitations and ethical considerations, we can further refine and improve the model's performance, contributing to a more positive and inclusive online environment.

**MACHINE LEARNING:**

**What is Machine Learning?**

Before we look at the details of various machine learning methods, let's start by looking at what machine learning is, and what it isn't. Machine learning is often categorized as a subfield of artificial intelligence, but I find that categorization can often be misleading at first brush. The study of machine learning certainly arose from research in this context, but in the data science application of machine learning methods, it's more helpful to think of machine learning as a means of building models of data.

Fundamentally, machine learning involves building mathematical models to help understand data. "Learning" enters the fray when we give these models tunable parameters that can be adapted to observed data; in this way the program can be considered to be "learning" from the data. Once these models have been fit to previously seen data, they can be used to predict and understand aspects of newly observed data. I'll leave to the reader the more philosophical digression regarding the extent to which this type of mathematical, model-based "learning" is similar to the "learning" exhibited by the human brain. Understanding the problem setting in machine learning is essential to using these tools effectively, and so we will start with some broad categorizations of the types of approaches we'll discuss here.

**Categories of Machine Leaning**

At the most fundamental level, machine learning can be categorized into two main types: supervised learning and unsupervised learning.

Supervised learning involves somehow modeling the relationship between measured features of data and some label associated with the data; once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. We will see examples of both types of supervised learning in the following section.

Unsupervised learning involves modeling the features of a dataset without reference to any label and is often described as "letting the dataset speak for itself." These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. We will see examples of both types of unsupervised learning in the following section.

**Need for Machine Learning**

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate, and solve complex problems. On the other side, AI is still in its initial stage and have not surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, “to make decisions, based on data, with efficiency and scale”.

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can’t do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

**Challenges in Machines Learning**

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

1. Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.
2. Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.
3. Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.
4. No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.
5. Issue of overfitting & underfitting − If the model is overfitting or underfitting, it cannot be represented well for the problem.
6. Curse of dimensionality − Another challenge ML model faces is too many features of data points. This can be a real hindrance.
7. Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

**Applications of Machines Learning**

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML

* Emotion analysis
* Sentiment analysis
* Error detection and prevention
* Weather forecasting and prediction
* Stock market analysis and forecasting
* Speech synthesis
* Speech recognition
* Customer segmentation
* Object recognition
* Fraud detection
* Fraud prevention
* Recommendation of products to customer in online shopping

**How to Start Learning Machine Learning?**

Arthur Samuel coined the term “Machine Learning” in 1959 and defined it as a “Field of study that gives computers the capability to learn without being explicitly programmed”.

And that was the beginning of Machine Learning! In modern times, Machine Learning is one of the most popular (if not the most!) career choices. According to [Indeed](http://blog.indeed.com/2019/03/14/best-jobs-2019/), Machine Learning Engineer Is The Best Job of 2019 with a 344% growth and an average base salary of $146,085 per year.

But there is still a lot of doubt about what exactly is Machine Learning and how to start learning it? So this article deals with the Basics of Machine Learning and also the path you can follow to eventually become a full-fledged Machine Learning Engineer. Now let’s get started!!!

**How to start learning ML?**

This is a rough roadmap you can follow on your way to becoming an insanely talented Machine Learning Engineer. Of course, you can always modify the steps according to your needs to reach your desired end-goal!

Step 1 – Understand the Prerequisites

In case you are a genius, you could start ML directly but normally, there are some prerequisites that you need to know which include Linear Algebra, Multivariate Calculus, Statistics, and Python. And if you don’t know these, never fear! You don’t need a Ph.D. degree in these topics to get started but you do need a basic understanding.

(a) Learn Linear Algebra and Multivariate Calculus

Both Linear Algebra and Multivariate Calculus are important in Machine Learning. However, the extent to which you need them depends on your role as a data scientist. If you are more focused on application heavy machine learning, then you will not be that heavily focused on maths as there are many common libraries available. But if you want to focus on R&D in Machine Learning, then mastery of Linear Algebra and Multivariate Calculus is very important as you will have to implement many ML algorithms from scratch.

(b) Learn Statistics

Data plays a huge role in Machine Learning. In fact, around 80% of your time as an ML expert will be spent collecting and cleaning data. And statistics is a field that handles the collection, analysis, and presentation of data. So it is no surprise that you need to learn it!!!  
Some of the key concepts in statistics that are important are Statistical Significance, Probability Distributions, Hypothesis Testing, Regression, etc. Also, Bayesian Thinking is also a very important part of ML which deals with various concepts like Conditional Probability, Priors, and Posteriors, Maximum Likelihood, etc.

(c) Learn Python

Some people prefer to skip Linear Algebra, Multivariate Calculus and Statistics and learn them as they go along with trial and error. But the one thing that you absolutely cannot skip is Python! While there are other languages you can use for Machine Learning like R, Scala, etc. Python is currently the most popular language for ML. In fact, there are many Python libraries that are specifically useful for Artificial Intelligence and Machine Learning such as Keras, TensorFlow, Scikit-learn, etc.

So if you want to learn ML, it’s best if you learn Python! You can do that using various online resources and courses such as Fork Python available Free on GeeksforGeeks.

Step 2 – Learn Various ML Concepts

Now that you are done with the prerequisites, you can move on to actually learning ML (Which is the fun part!!!) It’s best to start with the basics and then move on to the more complicated stuff. Some of the basic concepts in ML are:

**(a) Terminologies of Machine Learning**

* Model – A model is a specific representation learned from data by applying some machine learning algorithm. A model is also called a hypothesis.
* Feature – A feature is an individual measurable property of the data. A set of numeric features can be conveniently described by a feature vector. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, etc.
* Target (Label) – A target variable or label is the value to be predicted by our model. For the fruit example discussed in the feature section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
* Training – The idea is to give a set of inputs(features) and it’s expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.
* Prediction – Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label).

**(b) Types of Machine Learning**

* Supervised Learning – This involves learning from a training dataset with labeled data using classification and regression models. This learning process continues until the required level of performance is achieved.
* Unsupervised Learning – This involves using unlabelled data and then finding the underlying structure in the data in order to learn more and more about the data itself using factor and cluster analysis models.
* Semi-supervised Learning – This involves using unlabelled data like Unsupervised Learning with a small amount of labeled data. Using labeled data vastly increases the learning accuracy and is also more cost-effective than Supervised Learning.
* Reinforcement Learning – This involves learning optimal actions through trial and error. So the next action is decided by learning behaviors that are based on the current state and that will maximize the reward in the future.

**Advantages of Machine learning**

1. Easily identifies trends and patterns -

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for an e-commerce website like Amazon, it serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

2. No human intervention needed (automation)

With ML, you don’t need to babysit your project every step of the way. Since it means giving machines the ability to learn, it lets them make predictions and also improve the algorithms on their own. A common example of this is anti-virus softwares; they learn to filter new threats as they are recognized. ML is also good at recognizing spam.

3. Continuous Improvement

As ML algorithms gain experience, they keep improving in accuracy and efficiency. This lets them make better decisions. Say you need to make a weather forecast model. As the amount of data you have keeps growing, your algorithms learn to make more accurate predictions faster.

4. Handling multi-dimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments.

5. Wide Applications

You could be an e-tailer or a healthcare provider and make ML work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

**Disadvantages of Machine Learning**

1. Data Acquisition

Machine Learning requires massive data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where they must wait for new data to be generated.

2. Time and Resources

ML needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

3. Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose.

4. High error-susceptibility

Machine Learning is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This leads to irrelevant advertisements being displayed to customers. In the case of ML, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

**PRELIMINARIES**

*Learning*, like intelligence, covers such a broad range of processes that it is difficult to define precisely. A dictionary definition includes phrases such as “to gain knowledge, or understanding of, or skill in, by study, instruction, or experience,” and “modification of a behavioural tendency by experience.” Zoologists and psychologists study learning in animals and humans. In this book we focus on learning in machines. There are several parallels between animal and machine learning. Certainly, many techniques in machine learning derive from the efforts of psychologists to make more precise their theories of animal and human learning through computational models. It seems likely also that the concepts and techniques being explored by researchers in machine learning may illuminate certain aspects of biological learning.

As regards machines, we might say, very broadly, that a machine learns whenever it changes its structure, program, or data (based on its inputs or in response to external information) in such a manner that its expected future performance improves. Some of these changes, such as the addition of a record to a data base, fall comfortably within the province of other disciplines and are not necessarily better understood for being called learning. But, for example, when the performance of a speech-recognition machine improves after hearing several samples of a person’s speech, we feel quite justified in that case to say that the machine has learned.

Machine learning usually refers to the changes in systems that perform tasks associated with *artificial intelligence (AI)*. Such tasks involve recognition, diagnosis, planning, robot control, prediction, etc. The “changes” might be either enhancements to already performing systems or *ab initio* synthesis of new systems. To be slightly more specific, we show the architecture of a typical A “agent” in Fig. 1.1. This agent perceives and models its environment and computes appropriate actions, perhaps by anticipating their effects. Changes made to any of the components shown in the figure might count as learning. Different learning mechanisms might be employed depending on which subsystem is being changed. We will study several different learning methods in this book.

Sensory signals

Perception

Actions

Action

Computation

Model

Planning and

Reasoning

Goals

Figure 1.1: An AI System

One might ask “Why should machines have to learn? Why not design machines to perform as desired in the first place?” There are several reasons why machine learning is important. Of course, we have already mentioned that the achievement of learning in machines might help us understand how animals and humans learn. But there are important engineering reasons as well. Some of these are:

* Some tasks cannot be defined well except by example; that is, we might be able to specify input/output pairs but not a concise relationship between inputs and desired outputs. We would like machines to be able to adjust their internal structure to produce correct outputs for a large number of sample inputs and thus suitably constrain their input/output function to approximate the relationship implicit in the examples.
* It is possible that hidden among large piles of data are important relationships and correlations. Machine learning methods can often be used to extract these relationships (*data mining*).
* Human designers often produce machines that do not work as well as desired in the environments in which they are used. In fact, certain characteristics of the working environment might not be completely known at design time. Machine learning methods can be used for on-the-job improvement of existing machine designs.
* The amount of knowledge available about certain tasks might be too large for explicit encoding by humans. Machines that learn this knowledge gradually might be able to capture more of it than humans would want to write down.
* Environments change over time. Machines that can adapt to a changing environment would reduce the need for constant redesign.
* New knowledge about tasks is constantly being discovered by humans. Vocabulary changes. There is a constant stream of new events in the world. Continuing redesign of AI systems to conform to new knowledge is impractical, but machine learning methods might be able to track much of it.

**1.1.2 Wellsprings of Machine Learning**

Work in machine learning is now converging from several sources. These different traditions each bring different methods and different vocabulary which are now being assimilated into a more unified discipline. Here is a brief listing of some of the separate disciplines that have contributed to machine learning; more details will follow in the appropriate chapters:

* **Statistics:** A long-standing problem in statistics is how best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn. A related problem is how to estimate the value of an unknown function at a new point given the values of this function at a set of sample points. Statistical methods for dealing with these problems can be considered instances of machine learning because the decision and estimation rules depend on a corpus of samples drawn from the problem environment. We will explore some of the statistical methods later in the book. Details about the statistical theory underlying these methods can be found in statistical textbooks such as [Anderson, 1958].
* **Brain Models:** Non-linear elements with weighted inputs have been suggested as simple models of biological neurons. Networks of these elements have been studied by several researchers including [McCulloch & Pitts, 1943, Hebb, 1949, Rosenblatt, 1958] and, more recently by [Gluck & Rumelhart, 1989, Sejnowski, Koch, & Churchland, 1988]. Brain modelers are interested in how closely these networks approximate the learning phenomena of living brains. We shall see that several important machine learning techniques are based on networks of nonlinear elements—often called *neural networks*. Work inspired by this school is sometimes called *connectionism*, *brain-style computation*, or *sub-symbolic processing*.
* **Adaptive Control Theory:** Control theorists study the problem of controlling a process having unknown parameters which must be estimated during operation. Often, the parameters change during operation, and the control process must track these changes. Some aspects of controlling a robot based on sensory inputs represent instances of this sort of problem. For an introduction see [Bollinger & Duffie, 1988].
* **Psychological Models:** Psychologists have studied the performance of humans in various learning tasks. An early example is the EPAM network for storing and retrieving one member of a pair of words when given another [Feigenbaum, 1961]. Related work led to a number of early decision tree [Hunt, Marin, & Stone, 1966] and semantic network [Anderson & Bower, 1973] methods. More recent work of this sort has been influenced by activities in artificial intelligence which we will be presenting.

Some of the work in reinforcement learning can be traced to efforts to model how reward stimuli influence the learning of goal-seeking behavior in animals [Sutton & Barto, 1987]. Reinforcement learning is an important theme in machine learning research.

* **Artificial Intelligence:** From the beginning, AI research has been concerned with machine learning. Samuel developed a prominent early program that learned parameters of a function for evaluating board positions in the game of checkers [Samuel, 1959]. AI researchers have also explored the role of analogies in learning [Carbonell, 1983] and how future actions and decisions can be based on previous exemplary cases [Kolodner, 1993]. Recent work has been directed at discovering rules for expert systems using decision-tree methods [Quinlan, 1990] and inductive logic programming [Muggleton, 1991, Lavraˇc & Dˇzeroski, 1994]. Another theme has been saving and generalizing the results of problem solving using explanation-based learning [DeJong & Mooney, 1986, Laird, *et al.*, 1986, Minton, 1988, Etzioni, 1993].
* **Evolutionary Models:**

In nature, not only do individual animals learn to perform better, but species *evolve* to be better fit in their individual niches. Since the distinction between evolving and learning can be blurred in computer systems, techniques that model certain aspects of biological evolution have been proposed as learning methods to improve the performance of computer programs. Genetic algorithms [Holland, 1975] and genetic programming [Koza, 1992, Koza, 1994] are the most prominent computational techniques for evolution.

**1.1.3 Varieties of Machine Learning**

Orthogonal to the question of the historical source of any learning technique is the more important question of *what* is to be learned. In this book, we take it that the thing to be learned is a computational structure of some sort. We will consider a variety of different computational structures:

* Functions
* Logic programs and rule sets
* Finite-state machines
* Grammars
* Problem solving systems

We will present methods both for the synthesis of these structures from examples and for changing existing structures. In the latter case, the change to the existing structure might be simply to make it more computationally efficient rather than to increase the coverage of the situations it can handle. Much of the terminology that we shall be using throughout the book is best introduced by discussing the problem of learning functions, and we turn to that matter first.

**1.2 Learning Input-Output Functions**

We use Fig. 1.2 to help define some of the terminology used in describing the problem of learning a function. Imagine that there is a function, *f*, and the task of the learner is to guess what it is. Our hypothesis about the function to be learned is denoted by *h*. Both *f* and *h* are functions of a vector-valued input **X** = (*x*1*,x*2*,...,xi,...,xn*) which has *n* components. We think of *h* as being implemented by a device that has **X** as input and *h*(**X**) as output. Both *f* and *h* themselves may be vector-valued. We assume *a priori* that the hypothesized function, *h*, is selected from a class of functions H. Sometimes we know that *f* also belongs to this class or to a subset of this class. We select *h* based on a *training set*, Ξ, of *m* input vector examples. Many important details depend on the nature of the assumptions made about all of these entities.

**1.2.1 Types of Learning**

There are two major settings in which we wish to learn a function. In one, called *supervised learning*, we know (sometimes only approximately) the values of *f* for the *m* samples in the training set, Ξ. We assume that if we can find a hypothesis, *h*, that closely agrees with *f* for the members of Ξ, then this hypothesis will be a good guess for *f*—especially if Ξ is large.

*Training Set:*

h(

**X**

)

h

U

{

=

**X**

1

,

**X**

2

, . . .

**X**

i

,

**. . ., X**

m

}

**X**

=

x

1

.

.

.

x

i

.

.

.

x

n

h

D

H

Figure 1.2: An Input-Output Function

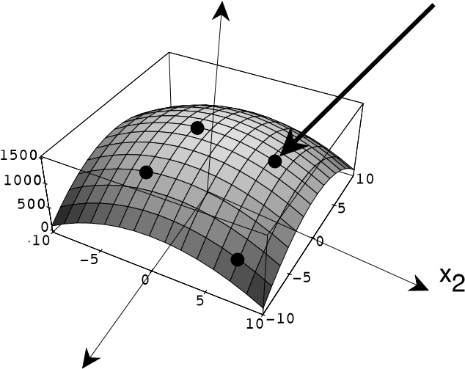
Curve-fitting is a simple example of supervised learning of a function. Suppose we are given the values of a two-dimensional function, *f*, at the four sample points shown by the solid circles in Fig. 1.3. We want to fit these four points with a function, *h*, drawn from the set, H, of second-degree functions. We show there a two-dimensional parabolic surface above the *x*1, *x*2 plane that fits the points. This parabolic function, *h*, is our hypothesis about the function, *f*, that produced the four samples. In this case, *h* = *f* at the four samples, but we need not have required exact matches.

In the other setting, termed *unsupervised learning*, we simply have a training set of vectors without function values for them. The problem in this case, typically, is to partition the training set into subsets, Ξ1, ..., Ξ*R*, in some appropriate way. (We can still regard the problem as one of learning a function; the value of the function is the name of the subset to which an input vector belongs.) Unsupervised learning methods have application in taxonomic problems in which it is desired to invent ways to classify data into meaningful categories.

We shall also describe methods that are intermediate between supervised and unsupervised learning.

We might either be trying to find a new function, *h*, or to modify an existing one. An interesting special case is that of changing an existing function into an equivalent one that is computationally more efficient. This type of learning is sometimes called *speed-up* learning. A very simple example of speed-up learning involves deduction processes. From the formulas *A* ⊃ *B* and *B* ⊃ *C*, we can deduce *C* if we are given *A*. From this deductive process, we can create the formula *A* ⊃ *C*—a new formula but one that does not sanction any more conclusions

h *sample f-value*



x1

Figure 1.3: A Surface that Fits Four Points

than those that could be derived from the formulas that we previously had. But with this new formula we can derive *C* more quickly, given *A*, than we could have done before. We can contrast speed-up learning with methods that create genuinely new functions—ones that might give different results after learning than they did before. We say that the latter methods involve *inductive* learning. As opposed to deduction, there are no *correct* inductions—only useful ones.

**1.2.2 Input Vectors**

Because machine learning methods derive from so many different traditions, its terminology is rife with synonyms, and we will be using most of them in this book. For example, the input vector is called by a variety of names. Some of these are: *input vector*, *pattern vector*, *feature vector*, *sample*, *example*, and *instance*. The components, *xi*, of the input vector are variously called *features*, *attributes*, *input variables*, and *components*.

The values of the components can be of three main types. They might be real-valued numbers, discrete-valued numbers, or *categorical values*. As an example illustrating categorical values, information about a student might be represented by the values of the attributes *class, major, sex, adviser*. A particular student would then be represented by a vector such as: (sophomore, history, male, higgins). Additionally, categorical values may be *ordered* (as in {*small, medium, large*}) or *unordered* (as in the example just given). Of course, mixtures of all these types of values are possible.

In all cases, it is possible to represent the input in unordered form by listing the names of the attributes together with their values. The vector form assumes that the attributes are ordered and given implicitly by a form. As an example of an *attribute-value* representation, we might have: (major: history, sex: male, class: sophomore, adviser: higgins, age: 19). We will be using the vector form exclusively.

An important specialization uses Boolean values, which can be regarded as a special case of either discrete numbers (1,0) or of categorical variables (*True*, *False*).

**1.2.3 Outputs**

The output may be a real number, in which case the process embodying the function, *h*, is called a *function estimator*, and the output is called an *output value* or *estimate*.

Alternatively, the output may be a categorical value, in which case the process embodying *h* is variously called a *classifier*, a *recognizer*, or a *categorizer*, and the output itself is called a *label*, a *class*, a *category*, or a *decision*. Classifiers have application in a number of recognition problems, for example in the recognition of hand-printed characters. The input in that case is some suitable representation of the printed character, and the classifier maps this input into one of, say, 64 categories.

Vector-valued outputs are also possible with components being real numbers or categorical values.

An important special case is that of Boolean output values. In that case, a training pattern having value 1 is called a *positive instance*, and a training sample having value 0 is called a *negative instance*. When the input is also Boolean, the classifier implements a *Boolean function*. We study the Boolean case in some detail because it allows us to make important general points in a simplified setting. Learning a Boolean function is sometimes called *concept learning*, and the function is called a *concept*.

**1.2.4 Training Regimes**

There are several ways in which the training set, Ξ, can be used to produce a hypothesized function. In the *batch* method, the entire training set is available and used all at once to compute the function, *h*. A variation of this method uses the entire training set to modify a current hypothesis iteratively until an acceptable hypothesis is obtained. By contrast, in the *incremental* method, we select one member at a time from the training set and use this instance alone to modify a current hypothesis. Then another member of the training set is selected, and so on. The selection method can be random (with replacement) or it can cycle through the training set iteratively. If the entire training set becomes available one member at a time, then we might also use an incremental method—selecting and using training set members as they arrive. (Alternatively, at any stage all training set members so far available could be used in a “batch” process.) Using the training set members as they become available is called an *online* method. Online methods might be used, for example, when the next training instance is some function of the current hypothesis and the previous instance—as it would be when a classifier is used to decide on a robot’s next action given its current set of sensory inputs. The next set of sensory inputs will depend on which action was selected.

**1.2.5 Noise**

Sometimes the vectors in the training set are corrupted by noise. There are two kinds of noise. *Class noise* randomly alters the value of the function; *attribute noise* randomly alters the values of the components of the input vector. In either case, it would be inappropriate to insist that the hypothesized function agree precisely with the values of the samples in the training set.

**1.2.6 Performance Evaluation**

Even though there is no correct answer in inductive learning, it is important to have methods to evaluate the result of learning. We will discuss this matter in more detail later, but, briefly, in supervised learning the induced function is usually evaluated on a separate set of inputs and function values for them called the *testing set* . A hypothesized function is said to *generalize* when it guesses well on the testing set. Both mean-squared-error and the total number of errors are common measures.

**1.3 Learning Requires Bias**

Long before now the reader has undoubtedly asked why is learning a function possible at all? Certainly, for example, there are an uncountable number of different functions having values that agree with the four samples shown in Fig. 1.3. Why would a learning procedure happen to select the quadratic one shown in that figure? In order to make that selection we had at least to limit *a priori* the set of hypotheses to quadratic functions and then to insist that the one we chose passed through all four sample points. This kind of *a priori* information is called *bias*, and useful learning without bias is impossible.

We can gain more insight into the role of bias by considering the special case of learning a Boolean function of *n* dimensions. There are 2*n* different Boolean inputs possible. Suppose we had no bias; that is H is the set of *all* 22*n* Boolean functions, and we have no preference among those that fit the samples in the training set. In this case, after being presented with one member of the training set and its value we can rule out precisely one-half of the members of H—those Boolean functions that would misclassify this labeled sample. The remaining functions constitute what is called a “version space;” we’ll explore that concept in more detail later. As we present more members of the training set, the graph of the number of hypotheses not yet ruled out as a function of the number of different patterns presented is as shown in Fig. 1.4. At any stage of the process, half of the remaining Boolean functions have value 1 and half have value 0 for *any* training pattern not yet seen. No generalization is possible in this case because the training patterns give no clue about the value of a pattern not yet seen. Only memorization is possible here, which is a trivial sort of learning.

|Hv| = no. of functions not ruled out log2|Hv|

2

n

2

n

0

0

2

n

<

j

(

generalization is not possible

)

j = no. of labeled patterns already seen

Figure 1.4: Hypotheses Remaining as a Function of Labeled Patterns Presented

But suppose we limited H to some subset, H*c*, of all Boolean functions. Depending on the subset and on the order of presentation of training patterns, a curve of hypotheses not yet ruled out might look something like the one shown in Fig. 1.5. In this case it is even possible that after seeing fewer than all 2*n* labeled samples, there might be only one hypothesis that agrees with the training set. Certainly, even if there is more than one hypothesis remaining, *most* of them may have the same value for *most* of the patterns not yet seen! The theory of *Probably Approximately Correct (PAC)* learning makes this intuitive idea precise. We’ll examine that theory later.

Let’s look at a specific example of how bias aids learning. A Boolean function can be represented by a hypercube each of whose vertices represents a different input pattern. We show a 3-dimensional version in Fig. 1.6. There, we show a training set of six sample patterns and have marked those having a value of 1 by a small square and those having a value of 0 by a small circle. If the hypothesis set consists of just the *linearly separable* functions—those for which the positive and negative instances can be separated by a linear surface, then there is only one function remaining in this hypothsis set that is consistent with the training set. So, in this case, even though the training set does not contain all possible patterns, we can already pin down what the function must be—given the bias.

|Hv| = no. of functions not ruled out log2|Hv|

2

n

2

n

0

0

depends on order

of presentation

log

2

|H

c

|

j = no. of labeled patterns already seen

Figure 1.5: Hypotheses Remaining From a Restricted Subset

Machine learning researchers have identified two main varieties of bias, absolute and preference. In *absolute bias* (also called *restricted hypothesis-space bias*), one restricts H to a definite subset of functions. In our example of Fig. 1.6, the restriction was to linearly separable Boolean functions. In *preference bias*, one selects that hypothesis that is minimal according to some ordering scheme over all hypotheses. For example, if we had some way of measuring the *complexity* of a hypothesis, we might select the one that was simplest among those that performed satisfactorily on the training set. The principle of *Occam’s razor*, used in science to prefer simple explanations to more complex ones, is a type of preference bias. (William of Occam, 1285-?1349, was an English philosopher who said: “*non sunt multiplicanda entia praeter necessitatem*,” which means “entities should not be multiplied unnecessarily.”)

**1.4 Sample Applications**

Our main emphasis in this book is on the concepts of machine learning—not on its applications. Nevertheless, if these concepts were irrelevant to real-world problems they would probably not be of much interest. As motivation, we give a short summary of some areas in which machine learning techniques have been successfully applied. [Langley, 1992] cites some of the following applications and others:

1. Rule discovery using a variant of ID3 for a printing industry problem

x3

x

1

x

2

Figure 1.6: A Training Set That Completely Determines a Linearly Separable Function

[Evans & Fisher, 1992].

1. Electric power load forecasting using a *k*-nearest-neighbor rule system [Jabbour, K., *et al.*, 1987].
2. Automatic “help desk” assistant using a nearest-neighbor system [Acorn & Walden, 1992].
3. Planning and scheduling for a steel mill using ExpertEase, a marketed(ID3-like) system [Michie, 1992].
4. Classification of stars and galaxies [Fayyad, *et al.*, 1993].

Many application-oriented papers are presented at the annual conferences on Neural Information Processing Systems. Among these are papers on: speech recognition, dolphin echo recognition, image processing, bio-engineering, diagnosis, commodity trading, face recognition, music composition, optical character recognition, and various control applications [Various Editors, 1989-1994].

As additional examples, [Hammerstrom, 1993] mentions:

1. Sharp’s Japanese kanji character recognition system processes 200 char-acters per second with 99+% accuracy. It recognizes 3000+ characters.
2. NeuroForecasting Centre’s (London Business School and University Col-lege London) trading strategy selection network earned an average annual profit of 18% against a conventional system’s 10.3%.

c. Fujitsu’s (plus a partner’s) neural network for monitoring a continuous steel casting operation has been in successful operation since early 1990.

In summary, it is rather easy nowadays to find applications of machine learning techniques. This fact should come as no surprise inasmuch as many machine learning techniques can be viewed as extensions of well known statistical methods which have been successfully applied for many years.

**1.5 Sources**

Besides the rich literature in machine learning (a small part of which is referenced in the Bibliography), there are several textbooks that are worth mentioning [Hertz, Krogh, & Palmer, 1991, Weiss & Kulikowski, 1991, Natarjan, 1991, Fu, 1994, Langley, 1996]. [Shavlik & Dietterich, 1990, Buchanan & Wilkins, 1993] are edited volumes containing some of the most important papers. A survey paper by [Dietterich, 1990] gives a good overview of many important topics. There are also well established conferences and publications where papers are given and appear including:

* The Annual Conferences on Advances in Neural Information Processing Systems
* The Annual Workshops on Computational Learning Theory
* The Annual International Workshops on Machine Learning
* The Annual International Conferences on Genetic Algorithms

(The Proceedings of the above-listed four conferences are published by Morgan Kaufmann.)

* The journal *Machine Learning* (published by Kluwer Academic Publishers).

There is also much information, as well as programs and datasets, available over the Internet through the World Wide Web.

**Machine learning** is a subfield of [computer science](https://en.wikipedia.org/wiki/Computer_science)that evolved from the study of [pattern recognition](https://en.wikipedia.org/wiki/Pattern_recognition) and [computational learning theory](https://en.wikipedia.org/wiki/Computational_learning_theory) in [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence).[1] Machine learning explores the construction and study of [algorithms](https://en.wikipedia.org/wiki/Algorithm) that can [learn](https://en.wikipedia.org/wiki/Learning) from and make predictions on [data](https://en.wikipedia.org/wiki/Data).Such algorithms operate by building a [model](https://en.wikipedia.org/wiki/Mathematical_model) from example inputs in order to make data-driven predictions or decisions,rather than following strictly static program instructions.

Machine learning is closely related to and often overlaps with [computational statistics](https://en.wikipedia.org/wiki/Computational_statistics); a discipline that also specializes in prediction-making. It has strong ties to [mathematical optimization,](https://en.wikipedia.org/wiki/Mathematical_optimization) which deliver methods, theory and application domains to the field. Machine learning is employed in a range of computing tasks where designing and programming explicit [algorithms](https://en.wikipedia.org/wiki/Algorithm) is infeasible. Example applications include [spam filtering](https://en.wikipedia.org/wiki/Spam_filter), [optical character recognition](https://en.wikipedia.org/wiki/Optical_character_recognition) (OCR),[search engines](https://en.wikipedia.org/wiki/Learning_to_rank) and [computer vision](https://en.wikipedia.org/wiki/Computer_vision). Machine learning is sometimes conflated with [data mining](https://en.wikipedia.org/wiki/Data_mining),although that focuses more on exploratory data analysis.Machine learning and pattern recognition “can be viewed as two facets of the same field.” When employed in industrial contexts, machine learning methods may be referred to as [predictive analytics](https://en.wikipedia.org/wiki/Predictive_analytics) or [predictive modelling](https://en.wikipedia.org/wiki/Predictive_modelling).

In 1959, [Arthur Samuel](https://en.wikipedia.org/wiki/Arthur_Samuel) defined machine learning as a “Field of study that gives computers the ability to learn without being explicitly programmed”. [Tom M. Mitchell](https://en.wikipedia.org/wiki/Tom_M._Mitchell) provided a widely quoted, more formal definition: “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”.This definition is notable for its defining machine learning in fundamentally [operational](https://en.wikipedia.org/wiki/Operational_definition) rather than cognitive terms, thus following [Alan Turing'](https://en.wikipedia.org/wiki/Alan_Turing)s proposal in his paper ["Computing Machinery and Intelligence"](https://en.wikipedia.org/wiki/Computing_Machinery_and_Intelligence) that the question “Can machines think?" be replaced with the question “Can machines do what we (as thinking entities) can do?"

**1.6 Types of problems and tasks**

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning “signal” or “feedback” available to a learning system. These are:[10]

* [Supervised learning:](https://en.wikipedia.org/wiki/Supervised_learning) The computer is presented with example inputs and their desired outputs, given by a “teacher”, and the goal is to learn a general rule that [maps](https://en.wikipedia.org/wiki/Map_(mathematics)) inputs to outputs.
* [Unsupervised learning:](https://en.wikipedia.org/wiki/Unsupervised_learning) No labels are given to the learningalgorithm, leavingitonitsowntofindstructure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a [means towards an end.](https://en.wikipedia.org/wiki/Feature_learning)
* [Reinforcement learning](https://en.wikipedia.org/wiki/Reinforcement_learning): A computer program interacts with a dynamic environment in which it must perform a certain goal (such as [driving a vehicle](https://en.wikipedia.org/wiki/Autonomous_car)), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent.

Between supervised and unsupervised learning is [semisupervised learning](https://en.wikipedia.org/wiki/Semi-supervised_learning), where the teacher gives an incomplete training signal: a training set with some (often many) of the target outputs missing. [Transduction](https://en.wikipedia.org/wiki/Transduction_(machine_learning)) is a special case of this principle where the entire set of problem instances is known at learning time, except that part of the targets are missing.

Among other categories of machine learning problems, [learning to learn](https://en.wikipedia.org/wiki/Learning_to_learn) learns its own [inductive bias](https://en.wikipedia.org/wiki/Inductive_bias) based on previous experience. [Developmental learning,](https://en.wikipedia.org/wiki/Developmental_robotics) elaborated for [robot learning,](https://en.wikipedia.org/wiki/Robot_learning) generates its own sequences (also called curriculum) of learning situations to cumulatively acquire repertoires of novel skills through autonomous

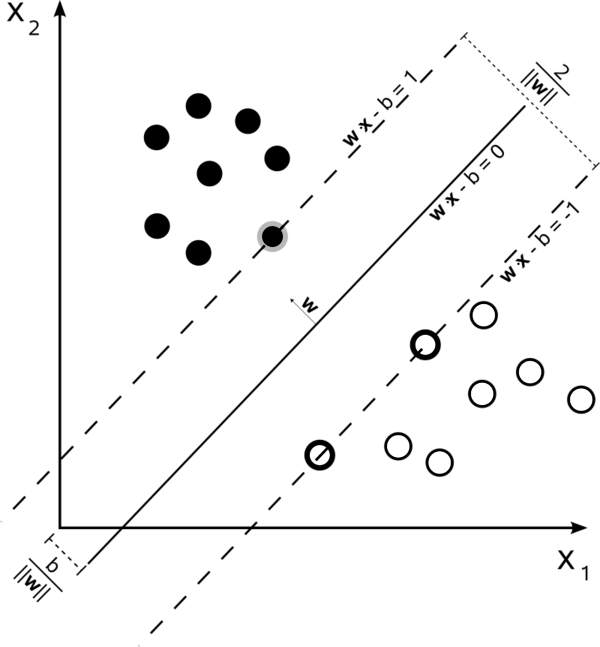


Figure 1.7 A [support vector machine](https://en.wikipedia.org/wiki/Support_vector_machine) is a classifier that divides its input space into two regions, separated by a [linear boundary.](https://en.wikipedia.org/wiki/Linear_classifier) Here, it has learned to distinguish black and white circles.

self-exploration and social interaction with human teachers, and using guidance mechanisms such as active learning, maturation, motor synergies, and imitation.

Another categorization of machine learning tasks arises when one considers the desired *output* of a machinelearned system.

* In [classification](https://en.wikipedia.org/wiki/Statistical_classification), inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one (or [multi-label classification](https://en.wikipedia.org/wiki/Multi-label_classification)) or more of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are “spam” and “not spam”.
* In [regression,](https://en.wikipedia.org/wiki/Regression_analysis) also a supervised problem, the outputs are continuous rather than discrete.
* In [clustering](https://en.wikipedia.org/wiki/Cluster_analysis), a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.
* [Density estimation](https://en.wikipedia.org/wiki/Density_estimation) finds the [distribution](https://en.wikipedia.org/wiki/Probability_distribution) of inputs in some space.
* [Dimensionality reduction](https://en.wikipedia.org/wiki/Dimensionality_reduction) simplifies inputs by mapping them into a lower-dimensional space. [Topic modeling](https://en.wikipedia.org/wiki/Topic_modeling) is a related problem, where a program is given a list of [human language](https://en.wikipedia.org/wiki/Natural_language) documents and is tasked to find out which documents cover similar topics.

**1.8 History and relationships to other fields**

As a scientific endeavour, machine learning grew out of the quest for artificial intelligence. Already in the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what were then termed ["neural networks"](https://en.wikipedia.org/wiki/Neural_network); these were mostly [perceptrons](https://en.wikipedia.org/wiki/Perceptron) and [other models](https://en.wikipedia.org/wiki/ADALINE) that were later found to be reinventions of the [generalized linear models](https://en.wikipedia.org/wiki/Generalized_linear_model) of statistics. [Probabilistic](https://en.wikipedia.org/wiki/Probability_theory) reasoning was also employed, especially in automated

medical diagnosis.

However, an increasing emphasis on the [logical, knowledge-based approach](https://en.wikipedia.org/wiki/GOFAI) caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representation.By 1980, [expert systems](https://en.wikipedia.org/wiki/Expert_system) had come to dominate AI, and statistics was out of favor.Work on symbolic/knowledge-based learning did continue within AI, leading to [inductive logic programming,](https://en.wikipedia.org/wiki/Inductive_logic_programming) but the more statistical line of research was now outside the field of AI proper, in [pattern recognition](https://en.wikipedia.org/wiki/Pattern_recognition) and [information retrieval](https://en.wikipedia.org/wiki/Information_retrieval).Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as "[connectionism"](https://en.wikipedia.org/wiki/Connectionism), by researchers from other disciplines including [Hopfield](https://en.wikipedia.org/wiki/John_Hopfield), [Rumelhart](https://en.wikipedia.org/wiki/David_Rumelhart) and [Hinton](https://en.wikipedia.org/wiki/Geoff_Hinton). Their main success came in the mid-1980s with the reinvention of [backpropagation](https://en.wikipedia.org/wiki/Backpropagation).

Machine learning, reorganized as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and [probability theory](https://en.wikipedia.org/wiki/Probability_theory).[11] It also benefited from the increasing availability of digitized information, and the possibility to distribute that via the [internet](https://en.wikipedia.org/wiki/Internet).

Machine learning and data mining often employ the same methods and overlap significantly. They can be roughly distinguished as follows:

* Machine learning focuses on prediction, based on *known* properties learned from the training data.
* [Datamining](https://en.wikipedia.org/wiki/Data_mining)focusesonthe[discovery](https://en.wikipedia.org/wiki/Discovery_(observation))of(previously) *unknown* properties in the data. This is the analysis step of [Knowledge Discovery](https://en.wikipedia.org/wiki/Knowledge_discovery) in Databases.

The two areas overlap in many ways: data mining uses many machine learning methods, but often with a slightly different goal in mind. On the other hand, machine learning also employs data mining methods as “unsupervised learning” or as a preprocessing step to improve learner accuracy. Much of the confusion between these two research communities (which do often have separate conferences and separate journals, [ECML PKDD](https://en.wikipedia.org/wiki/ECML_PKDD) being a major exception) comes from the basic assumptions they work with: in machine learning, performance is usually evaluated with respect to the ability to *reproduce known* knowledge, while in Knowledge Discovery and Data Mining (KDD) the key task is the discovery of previously *unknown* knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some [loss function](https://en.wikipedia.org/wiki/Loss_function) on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set examples). The difference between the two fields arises from the goal of generalization: while optimization algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples.

**1.8 Relation to statistics**

Machine learning and [statistics](https://en.wikipedia.org/wiki/Statistics) are closely related fields. According to [Michael I. Jordan](https://en.wikipedia.org/wiki/Michael_I._Jordan), the ideas of machine learning, from methodological principles to theoretical tools, have had a long pre-history in statistics.He also suggested the term [data science](https://en.wikipedia.org/wiki/Data_science) as a placeholder to call

the overall field.

[Leo Breiman](https://en.wikipedia.org/wiki/Leo_Breiman) distinguished two statistical modelling paradigms: data model and algorithmic model,[14] wherein 'algorithmic model' means more or less the machine learning algorithms like [Random forest](https://en.wikipedia.org/wiki/Random_forest).

Some statisticians have adopted methods from machine learning, leading to a combined field that they call *statistical learning*.

**1.9 Approaches**

**1.7.1 Decision tree learning**

Decision tree learning uses a [decision tree](https://en.wikipedia.org/wiki/Decision_tree) as a [predictive model](https://en.wikipedia.org/wiki/Predictive_modelling), which maps observations about an item to conclusions about the item’s target value.

**1.7.2 Association rule learning**

Association rule learning is a method for discovering interesting relations between variables in large databases.

**1.7.3 Artificial neural networks**

An [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) (ANN) learning algorithm, usually called “neural network” (NN), is a learning algorithm that is inspired by the structure and functional aspects of [biological neural networks](https://en.wikipedia.org/wiki/Biological_neural_networks). Computations are structured in terms of an interconnected group of [artificial neurons](https://en.wikipedia.org/wiki/Artificial_neuron), processing information using a [connectionist](https://en.wikipedia.org/wiki/Connectionism) approach to [computation.](https://en.wikipedia.org/wiki/Computation) Modern neural networks are [non-linear](https://en.wikipedia.org/wiki/Non-linear) [statistical](https://en.wikipedia.org/wiki/Statistical) [data modeling](https://en.wikipedia.org/wiki/Data_modeling) tools. They are usually used to model complex relationships between inputs and outputs, to [find patterns](https://en.wikipedia.org/wiki/Pattern_recognition) in data, or to capture the statistical structure in an unknown [joint probability distribution](https://en.wikipedia.org/wiki/Joint_probability_distribution) between observed variables.

**1.7.4 Inductive logic programming**

Inductive logic programming (ILP) is an approach to rule learning using [logic programming](https://en.wikipedia.org/wiki/Logic_programming) as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that [entails](https://en.wikipedia.org/wiki/Entailment) all positive and no negative examples. [Inductive programming](https://en.wikipedia.org/wiki/Inductive_programming) is a related field that considers any kind of programming languages for representing hypotheses (and not only logic programming), such as functional programs.

**1.7.5 Support vector machines**

Support vector machines (SVMs) are a set of related [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) methods used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression.](https://en.wikipedia.org/wiki/Regression_analysis) Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

**1.7.6 Clustering**

Cluster analysis is the assignment of a set of observations into subsets (called *clusters*) so that observations within the same cluster are similar according to some predesignated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some *similarity metric* and evaluated for example by *internal compactness* (similarity between members of the same cluster) and *separation* between different clusters. Other methods are based on *estimated density* and *graph connectivity*. Clustering is a method of [unsupervised learning,](https://en.wikipedia.org/wiki/Unsupervised_learning) and a common technique for [statistical](https://en.wikipedia.org/wiki/Statistics) [data analysis](https://en.wikipedia.org/wiki/Data_analysis).

**1.7.7 Bayesian networks**

A Bayesian network, belief network or directed acyclic graphical model is a [probabilistic graphical model](https://en.wikipedia.org/wiki/Graphical_model) that represents a set of [random variables](https://en.wikipedia.org/wiki/Random_variables) and their [conditional independencies](https://en.wikipedia.org/wiki/Conditional_independence) via a [directed acyclic graph](https://en.wikipedia.org/wiki/Directed_acyclic_graph) (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform [inference](https://en.wikipedia.org/wiki/Inference) and learning.

**1.7.8 Reinforcement learning**

Reinforcement learning is concerned with how an *agent* ought to take *actions* in an *environment* so as to maximize some notion of long-term *reward*. Reinforcement learning algorithms attempt to find a *policy* that maps *states* of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

**1.7.9 Representation learning**

Several learning algorithms, mostly [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning) algorithms, aim at discovering better representations of the inputs provided during training. Classical examples include [principal components analysis](https://en.wikipedia.org/wiki/Principal_components_analysis) and [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis). Representation learning algorithms often attempt to preserve the information in their input but transform it in a way that makes it useful, often as a preprocessing step before performing classification or predictions, allowing to reconstruct the inputs coming from the unknown data generating distribution, while not being necessarily faithful for configurations that are implausible under that distribution.

[Manifold learning](https://en.wikipedia.org/wiki/Manifold_learning) algorithms attempt to do so under the constraint that the learned representation is lowdimensional. [Sparse coding](https://en.wikipedia.org/wiki/Sparse_coding) algorithms attempt to do so under the constraint that the learned representation is sparse (has many zeros). [Multilinear subspace learning](https://en.wikipedia.org/wiki/Multilinear_subspace_learning) algorithms aim to learn low-dimensional representations directly from [tensor](https://en.wikipedia.org/wiki/Tensor) representations for multidimensional data, without reshaping them into (high-dimensional) vectors.[Deep learning](https://en.wikipedia.org/wiki/Deep_learning) algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data.[18]

**1.7.10 Similarity and metric learning**

In this problem, the learning machine is given pairs of examples that are considered similar and pairs of less similar objects. It then needs to learn a similarity function (or a distance metric function) that can predict if new objects are similar. It is sometimes used in [Recommendation systems.](https://en.wikipedia.org/wiki/Recommendation_systems)

**1.7.11 Sparse dictionary learning**

In this method, a datum is represented as a linear combination of basis functions, and the coefficients are assumed to be sparse. Let *x* be a *d*-dimensional datum, *D* be a *d* by *n* matrix, where each column of *D* represents a basis function. *r* is the coefficient to represent *x* using *D*. Mathematically, sparse dictionary learning means the following *x* ≈ *Dr* where *r* is sparse. Generally speaking, *n* is assumed to be larger than *d* to allow the freedom for a sparse representation.

Learning a dictionary along with sparse representations is [strongly NP-hard](https://en.wikipedia.org/wiki/Strongly_NP-hard) and also difficult to solve approximately.[19] A popular heuristic method for sparse dictionary learning is [K-SVD.](https://en.wikipedia.org/wiki/K-SVD)

Sparse dictionary learning has been applied in several contexts. In classification, the problem is to determine which classes a previously unseen datum belongs to. Suppose a dictionary for each class has already been built. Then a new datum is associated with the class such that it’s best sparsely represented by the corresponding dictionary. Sparse dictionary learning has also been applied in image de-noising. The key idea is that a clean image patch can be sparsely represented by an image dictionary, but the noise cannot.[20]

**1.7.12 Genetic algorithms**

A genetic algorithm (GA) is a [search](https://en.wikipedia.org/wiki/Search_algorithm) [heuristic](https://en.wikipedia.org/wiki/Heuristic_(computer_science)) that mimics the process of [natural selection](https://en.wikipedia.org/wiki/Natural_selection), andusesmethodssuch as [mutation](https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)) and [crossover](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)) to generate new [genotype](https://en.wikipedia.org/wiki/Chromosome_(genetic_algorithm)) in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms found some uses in the 1980s and 1990s.Vice versa, machine learning techniques have been used to improve the performance of genetic and [evolutionary algorithms](https://en.wikipedia.org/wiki/Evolutionary_algorithm).

**1.10 Applications**

Applications for machine learning include:

* [Adaptive websites](https://en.wikipedia.org/wiki/Adaptive_website)
* [Affective computing](https://en.wikipedia.org/wiki/Affective_computing)
* [Bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics)
* [Brain-machine interfaces](https://en.wikipedia.org/wiki/Brain-machine_interfaces)
* [Cheminformatics](https://en.wikipedia.org/wiki/Cheminformatics)
* Classifying [DNA sequences](https://en.wikipedia.org/wiki/DNA_sequence)
* [Computational advertising](https://en.wikipedia.org/wiki/Computational_advertising)
* [Computational finance](https://en.wikipedia.org/wiki/Computational_finance)
* [Computer vision](https://en.wikipedia.org/wiki/Computer_vision), including [object recognition](https://en.wikipedia.org/wiki/Object_recognition)
* Detecting [credit card fraud](https://en.wikipedia.org/wiki/Credit_card_fraud)
* [Game playing](https://en.wikipedia.org/wiki/Strategy_game)
* [Information retrieval](https://en.wikipedia.org/wiki/Information_retrieval)
* [Internet fraud](https://en.wikipedia.org/wiki/Internet_fraud) detection
* [Machine perception](https://en.wikipedia.org/wiki/Machine_perception)
* [Medical diagnosis](https://en.wikipedia.org/wiki/Diagnosis_(artificial_intelligence))
* [Natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing)
* [Optimization](https://en.wikipedia.org/wiki/Mathematical_optimization) and [metaheuristic](https://en.wikipedia.org/wiki/Metaheuristic)
* [Recommender systems](https://en.wikipedia.org/wiki/Recommender_system)
* [Robot locomotion](https://en.wikipedia.org/wiki/Robot_locomotion)
* [Search engines](https://en.wikipedia.org/wiki/Search_engines)
* [Sentiment analysis](https://en.wikipedia.org/wiki/Sentiment_analysis) (or opinion mining)
* [Sequence mining](https://en.wikipedia.org/wiki/Sequence_mining)
* [Software engineering](https://en.wikipedia.org/wiki/Software_engineering)
* [Speech](https://en.wikipedia.org/wiki/Speech_recognition) and [handwriting recognition](https://en.wikipedia.org/wiki/Handwriting_recognition)
* [Stock market](https://en.wikipedia.org/wiki/Stock_market) analysis
* [Structural health monitoring](https://en.wikipedia.org/wiki/Structural_health_monitoring)
* [Syntactic pattern recognition](https://en.wikipedia.org/wiki/Syntactic_pattern_recognition)

In 2006, the online movie company [Netflix](https://en.wikipedia.org/wiki/Netflix) held the first ["Netflix Prize"](https://en.wikipedia.org/wiki/Netflix_Prize) competition to find a program to better predict user preferences and improve the accuracy on its existing Cinematch movie recommendation algorithm by at least 10%. A joint team made up of researchers from [AT&TLabs](https://en.wikipedia.org/wiki/AT%26T_Labs)-ResearchincollaborationwiththeteamsBig Chaos and Pragmatic Theory built an [ensemble model](https://en.wikipedia.org/wiki/Ensemble_Averaging) to win the Grand Prize in 2009 for $1 million.[26] Shortly after the prize was awarded, Netflix realized that viewers’ ratings were not the best indicators of their viewing patterns (“everything is a recommendation”) and they changed their recommendation engine accordingly.

In 2010 The Wall Street Journal wrote about money management firm Rebellion Research’s use of machine learning to predict economic movements. The article describes Rebellion Research’s prediction of the financial crisis and economic recovery.

In 2014 it has been reported that a machine learning algorithm has been applied in Art History to study fine art paintings, and that it may have revealed previously unrecognized influences between artists.

**USING VERSION SPACES FOR LEARNING**

**2.1 Version Spaces and Mistake Bounds**

The first learning methods we present are based on the concepts of *version spaces* and *version graphs*. These ideas are most clearly explained for the case of Boolean function learning. Given an initial hypothesis set H (a subset of all Boolean functions) and the values of *f*(**X**) for each **X** in a training set, Ξ, the version space is that subset of hypotheses, H*v*, that is consistent with these values. A hypothesis, *h*, is *consistent* with the values of **X** in Ξ if and only if *h*(**X**) = *f*(**X**) for all **X** in Ξ. We say that the hypotheses in H that are not consistent with the values in the training set are *ruled out* by the training set.

We could imagine (conceptually only!) that we have devices for implementing every function in H. An incremental training procedure could then be defined which presented each pattern in Ξ to each of these functions and then eliminated those functions whose values for that pattern did not agree with its given value. At any stage of the process we would then have left some subset of functions that are consistent with the patterns presented so far; this subset is the version space for the patterns already presented. This idea is illustrated in Fig. 2.1.

Consider the following procedure for classifying an arbitrary input pattern, **X**: the pattern is put in the same class (0 or 1) as are the majority of the outputs of the functions in the version space. During the learning procedure, if this majority is not equal to the value of the pattern presented, we say a *mistake* is made, and we revise the version space accordingly—eliminating all those (majority of the) functions voting incorrectly. Thus, whenever a mistake is made, we rule out at least half of the functions remaining in the version space.

How many mistakes can such a procedure make? Obviously, we can make no more than log2(|H|) mistakes, where |H| is the number of hypotheses in the

h

1

h

2

h

i

h

K

**X**

A Subset, H, of all

Boolean Functions

Rule out hypotheses not

consistent with training patterns

h

j

Hypotheses not ruled out

constitute the

*version space*

K = |H|

or

0

1

Figure 2.1: Implementing the Version Space

original hypothesis set, H. (Note, though, that the number of training patterns seen before this maximum number of mistakes is made might be much greater.) This theoretical (and very impractical!) result (due to [Littlestone, 1988]) is an example of a *mistake bound*—an important concept in machine learning theory. It shows that there must exist a learning procedure that makes no more mistakes than this upper bound. Later, we’ll derive other mistake bounds.

As a special case, if our bias was to limit H to terms, we would make no more than log2(3*n*) = *n*log2(3) = 1*.*585*n* mistakes before *exhausting* the version space. This result means that if *f* were a term, we would make no more than 1*.*585*n* mistakes before learning *f*, and otherwise we would make no more than that number of mistakes before being able to decide that *f* is not a term.

Even if we do not have sufficient training patterns to reduce the version space to a single function, it may be that there are enough training patterns to reduce the version space to a set of functions such that most of them assign the same values to most of the patterns we will see henceforth. We could select one of the remaining functions at random and be reasonably assured that it will generalize satisfactorily. We next discuss a computationally more feasible method for representing the version space.

**2.2 Version Graphs**

Boolean functions can be ordered by *generality*. A Boolean function, *f*1, is *more general* than a function, *f*2, (and *f*2 is *more specific* than *f*1), if *f*1 has value 1 for all of the arguments for which *f*2 has value 1, and *f*1 6= *f*2. For example, *x*3 is more general than *x*2*x*3 but is not more general than *x*3 + *x*2.

We can form a graph with the hypotheses, {*hi*}, in the version space as nodes. A node in the graph, *hi*, has an arc directed to node, *hj*, if and only if *hj* is more general than *hi*. We call such a graph a *version graph*. In Fig. 2.2, we show an example of a version graph over a 3-dimensional input space for hypotheses restricted to terms (with none of them yet ruled out). (for simplicity, only some arcs in the graph are shown)

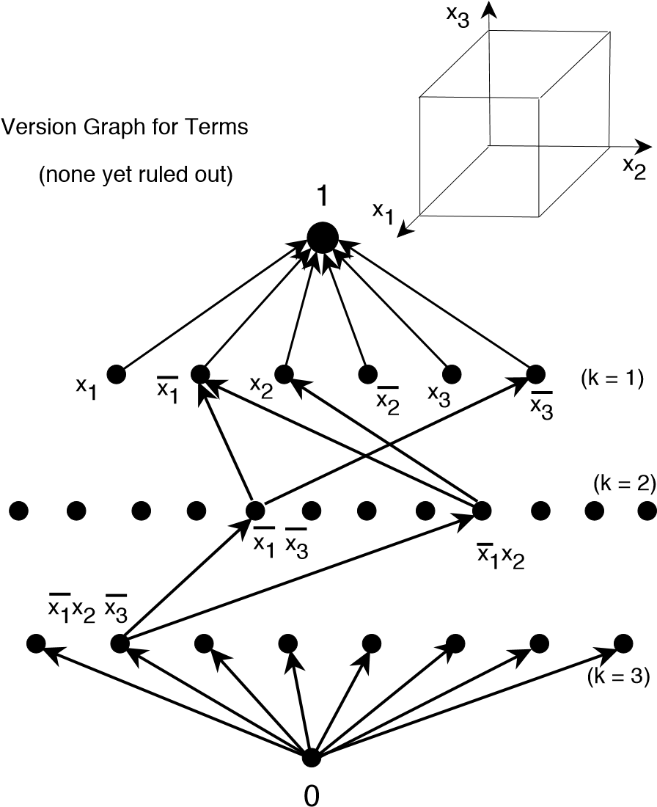


Figure 3.2: A Version Graph for Terms

That function, denoted here by “1,” which has value 1 for all inputs, corresponds to the node at the top of the graph. (It is more general than any other term.) Similarly, the function “0” is at the bottom of the graph. Just below “1” is a row of nodes corresponding to all terms having just one literal, and just below them is a row of nodes corresponding to terms having two literals, and so on. There are 33 = 27 functions altogether (the function “0,” included in the graph, is technically not a term). To make our portrayal of the graph less cluttered only some of the arcs are shown; each node in the actual graph has an arc directed to all of the nodes above it that are more general.

We use this same example to show how the version graph changes as we consider a set of labeled samples in a training set, Ξ. Suppose we first consider the training pattern (1, 0, 1) with value 0. Some of the functions in the version graph of Fig. 2.2 are inconsistent with this training pattern. These ruled out nodes are no longer in the version graph and are shown shaded in Fig. 2.3. We also show there the three-dimensional cube representation in which the vertex (1, 0, 1) has value 0.

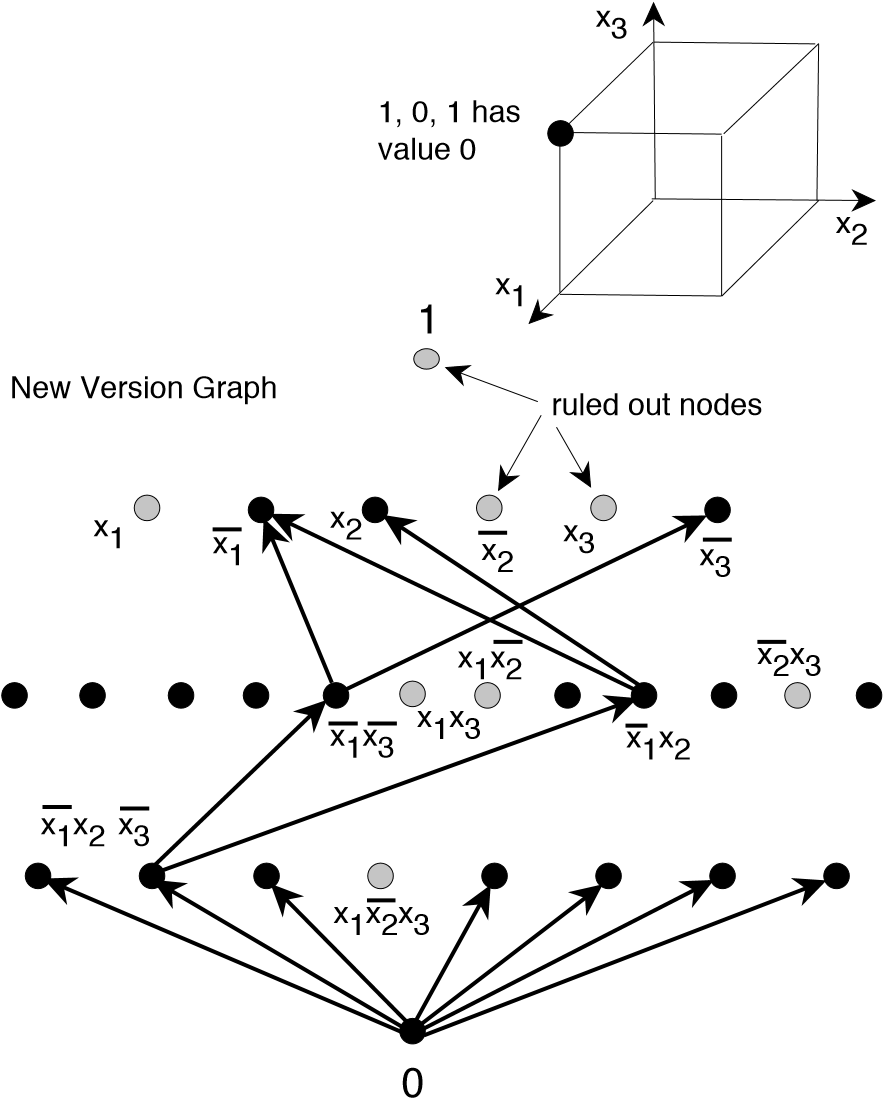


Figure 2.3: The Version Graph Upon Seeing (1, 0, 1). (only some arcs in the graph are shown)

In a version graph, there are always a set of hypotheses that are maximally general and a set of hypotheses that are maximally specific. These are called the *general boundary set (gbs)* and the *specific boundary set (sbs)*, respectively. In Fig. 3.4, we have the version graph as it exists after learning that (1,0,1) has value 0 and (1, 0, 0) has value 1. The gbs and sbs are shown.

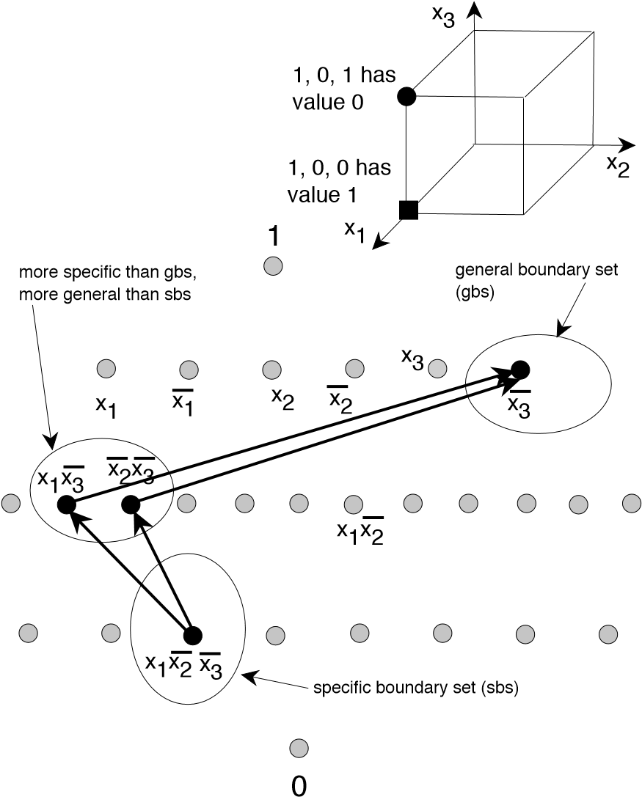


Figure 2.4: The Version Graph Upon Seeing (1, 0, 1) and (1, 0, 0)

Boundary sets are important because they provide an alternative to representing the entire version space explicitly, which would be impractical. Given only the boundary sets, it is possible to determine whether or not any hypothesis (in the prescribed class of Boolean functions we are using) is a member or not of the version space. This determination is possible because of the fact that any member of the version space (that is not a member of one of the boundary sets) is more specific than some member of the general boundary set and is more general than some member of the specific boundary set.

If we limit our Boolean functions that can be in the version space to terms, it is a simple matter to determine maximally general and maximally specific functions (assuming that there is some term that is in the version space). A maximally specific one corresponds to a subface of *minimal dimension* that contains all the members of the training set labelled by a 1 and no members labelled by a 0. A maximally general one corresponds to a subface of *maximal dimension* that contains all the members of the training set labelled by a 1 and no members labelled by a 0. Looking at Fig. 2.4, we see that the subface of minimal dimension that contains (1, 0, 0) but does not contain (1, 0, 1) is just the vertex (1, 0, 0) itself—corresponding to the function *x*1*x*2 *x*3. The subface of maximal dimension that contains (1, 0, 0) but does not contain (1, 0, 1) is the bottom face of the cube—corresponding to the function *x*3. In Figs. 3.2 through 3.4 the sbs is always singular. Version spaces for terms always have singular specific boundary sets. As seen in Fig. 2.3, however, the gbs of a version space for terms need not be singular.

**2.3 Learning as Search of a Version Space**

Selecting a hypothesis from the version space can be thought of as a search problem. One can start with a very general function and specialize it through various specialization operators until one finds a function that is consistent (or adequately so) with a set of training patterns. Such procedures are usually called *top-down* methods. Or, one can start with a very special function and generalize it—resulting in *bottom-up* methods. We shall see instances of both

|  |  |
| --- | --- |
|  | styles of learning in this book. |
|  | **2.4 The Candidate Elimination Method** |

“The *candidate-elimination algorithm* manipulates the boundary-set representation of a version space to create boundary sets that represent a new version space consistent with all the previous instances plus the new one. For a positive example the algorithm generalizes the elements of the [sbs] as little as possible so that they cover the new instance yet remain consistent with past data, and removes those elements of the [gbs] that do not cover the new instance. For a negative instance the algorithm specializes elements of the [gbs] so that they no longer cover the new instance yet remain consistent with past data, and removes from the [sbs] those elements that mistakenly cover the new, negative instance.”

* a hypothesis is called *sufficient* if and only if it has value 1 for all training samples labeled by a 1,
* a hypothesis is called *necessary* if and only if it has value 0 for all training samples labeled by a 0.

Here is how to think about these definitions: A hypothesis implements a *sufficient* condition that a training sample has value 1 if the hypothesis has value 1 for all of the positive instances; a hypothesis implements a *necessary* condition that a training sample has value 1 if the hypothesis has value 0 for all of the negative instances. A hypothesis is consistent with the training set (and thus is in the version space) if and only if it is both sufficient and necessary.

We start (before receiving any members of the training set) with the function “0” as the singleton element of the specific boundary set and with the function “1” as the singleton element of the general boundary set. Upon receiving a new labeled input vector, the boundary sets are changed as follows:

1. If the new vector is labelled with a 1:

The new general boundary set is obtained from the previous one by excluding any elements in it that are not sufficient. (That is, we exclude any elements that have value 0 for the new vector.)

The new specific boundary set is obtained from the previous one by replacing each element, *hi*, in it by all of its *least generalizations*.

The hypothesis *hg* is a *least generalization* of *h* if and only if: a) *h* is more specific than *hg*, b) *hg* is sufficient, c) no function (including *h*) that is more specific than *hg* is sufficient, and d) *hg* is more specific than some member of the new general boundary set. It might be that *hg* = *h*. Also, least generalizations of two different functions in the specific boundary set may be identical.

1. If the new vector is labelled with a 0:

The new specific boundary set is obtained from the previous one by excluding any elements in it that are not necessary. (That is, we exclude any elements that have value 1 for the new vector.)

The new general boundary set is obtained from the previous one by replacing each element, *hi*, in it by all of its *least specializations*.

The hypothesis *hs* is a *least specialization* of *h* if and only if: a) *h* is more general than *hs*, b) *hs* is necessary, c) no function (including *h*) that is more general than *hs* is necessary, and d) *hs* is more general than some member of the new specific boundary set. Again, it might be that *hs* = *h*, and least specializations of two different functions in the general boundary set may be identical.

As an example, suppose we present the vectors in the following order:

|  |  |
| --- | --- |
| vector | label |
| (1, 0, 1) | 0 |
| (1, 0, 0) | 1 |
| (1, 1, 1) | 0 |
| (0, 0, 1) | 0 |

We start with general boundary set, “1”, and specific boundary set, “0.” After seeing the first sample, (1, 0, 1), labeled with a 0, the specific boundary set stays at “0” (it is necessary), and we change the general boundary set to {*x*1*,x*2*,x*3}. Each of the functions, *x*1, *x*2, and *x*3, are least specializations of “1” (they are necessary, “1” is not, they are more general than “0”, and there are no functions that are more general than they and also necessary).

Then, after seeing (1, 0, 0), labeled with a 1, the general boundary set changes to {*x*3} (because *x*1 and *x*2 are not sufficient), and the specific boundary set is changed to {*x*1*x*2 *x*3}. This single function is a least generalization of “0” (it is sufficient, “0” is more specific than it, no function (including “0”) that is more specific than it is sufficient, and it is more specific than some member of the general boundary set.

When we see (1, 1, 1), labeled with a 0, we do not change the specific boundary set because its function is still necessary. We do not change the general boundary set either because *x*3 is still necessary.

Finally, when we see (0, 0, 1), labeled with a 0, we do not change the specific boundary set because its function is still necessary. We do not change the general boundary set either because *x*3 is still necessary.

**CHAPTER - 10**

**SOFTWARE ENVIRONMENT**

**What is Python?**

Below are some facts about Python.

* Python is currently the most widely used multi-purpose, high-level programming language.
* Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
* Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
* Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard library which can be used for the following –

* + Machine Learning
  + GUI Applications (like Kivy, Tkinter, PyQt etc. )
  + Web frameworks like Django (used by YouTube, Instagram, Dropbox)
  + Image processing (like Opencv, Pillow)
  + Web scraping (like Scrapy, BeautifulSoup, Selenium)
  + Test frameworks
  + Multimedia

**Advantages of Python**

Let’s see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

2. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

3. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

4. Improved Productivity

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

5. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

6. Simple and Easy

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

7. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

8. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

9. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

10. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn’t the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

11. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

**Advantages of Python Over Other Languages**

1. Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don’t have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

2. Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

3. Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

**Disadvantages of Python**

So far, we’ve seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let’s now see the downsides of choosing Python over another language.

1. Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn’t a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn’t that secure.

3. Design Restrictions

As you know, Python is dynamically-typed. This means that you don’t need to declare the type of variable while writing the code. It uses duck-typing. But wait, what’s that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. Underdeveloped Database Access Layers

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. Simple

No, we’re not kidding. Python’s simplicity can indeed be a problem. Take my example. I don’t do Java, I’m more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

**History of Python**

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

**Python Development Steps**

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of list, dict, str and others. It was also object oriented and had a module system.  
Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

* Print is now a function.
* Views and iterators instead of lists
* The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
* There is only one integer type left, i.e., int. long is int as well.
* The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
* Text Vs. Data Instead of Unicode Vs. 8-bit

**Purpose**

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

**Python**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

**Modules Used in Project**

**TensorFlow**

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.‍

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

**NumPy**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

**Pandas**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**Matplotlib**

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

**Scikit – learn**

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

**Install Python Step-by-Step in Windows and Mac**

Python a versatile programming language doesn’t come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

**How to Install Python on Windows and Mac**

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e. operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here.The steps on how to install Python on Windows 10, 8 and 7 are divided into pyth4 parts to help understand better.

**Download the Correct version into the system**

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org

A screenshot of a computer

Description automatically generated with medium confidence

Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Graphical user interface, application

Description automatically generated

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Graphical user interface, application

Description automatically generated

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Graphical user interface, text

Description automatically generated

* To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
* To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.

Graphical user interface, text, application

Description automatically generated

Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.

Graphical user interface, text, application, chat or text message

Description automatically generated

Step 3: Click on Install NOW After the installation is successful. Click on Close.

Graphical user interface, text, application, chat or text message

Description automatically generated

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.

Graphical user interface, application

Description automatically generated

Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.

A screenshot of a computer

Description automatically generated with medium confidence

Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.

Application

Description automatically generated with low confidence

Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save

Graphical user interface, text, application, email

Description automatically generated

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print (“Hey World”) and Press Enter.  
Graphical user interface, text, application, email

Description automatically generated

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.