Natural Language Processing

Christian Rossi

Academic Year 2024-2025

Abstract

This course introduces students to the challenges and methodologies related to the analysis and production of natural language sentences, both written and spoken. The course explores the current role of stochastic models and Deep Learning, as well as new opportunities to combine traditional, formally based models with stochastic models. Topics covered include morphology, syntax, semantics, pragmatics, voice, prosody, discourse, dialogue, and sentiment analysis.

The course includes practical exercises where students can test the models and techniques presented in the lectures. Applications explored during the course will include: human-machine and human-human interaction analysis based on language; linguistic and prosodic analysis and generation for rehabilitation; pattern recognition and research for sentiment analysis in critical interactions; complexity analysis in text and overall spoken communication; and the development of user profiles that account for expression preferences in forensic, educational, and clinical contexts.

Contents

CHAPTER 1

Introduction

1.1 Natural language

The origins of spoken language are widely debated. Estimates range from as early as 2.5 million years ago to as recent as 60,000 years ago, depending on how one defines human language.

The development of written language, however, is more clearly documented. The first known writing systems emerged in Mesopotamia (modern-day Iraq) around 3500 BCE. Initially, these were simple pictograms representing objects, but over time, they evolved into abstract symbols representing sounds, paving the way for more complex communication.

The characteristics of human language are as follows:

- *Compositional*: language allows us to form sentences with subjects, verbs, and objects, providing an infinite capacity for expressing new ideas.
- *Referential*: we can describe objects, their locations, and their actions with precision.
- *Temporal*: language enables us to convey time, distinguishing between past, present, and future events.
- *Diverse*: thousands of languages are spoken worldwide, each with unique structures and expressions.

1.1.1 Natural Language Processing

One reason to care about Natural Language Processing is the sheer volume of human knowledge now available in machine-readable text. With the rise of conversational agents, human-computer interactions increasingly rely on language understanding. Furthermore, much of our daily communication is now mediated by digital platforms, making Natural Language Processing more relevant than ever.

However, Natural Language Processing is a challenging field. Human language is highly expressive, allowing people to articulate virtually anything—including ambiguous or nonsensical statements. Resolving this ambiguity is one of the core difficulties in computational linguistics. Moreover, meaning can be influenced by pronunciation, emphasis, and context, making interpretation even more complex. Fortunately, language is often redundant, allowing for error correction and inference even when mistakes occur.

1.1.2 History

The field of Natural Language Processing has its roots in linguistics, computer science, speech recognition, and psychology. Over time, it has evolved through various paradigms, driven by advancements in formal language theory, probabilistic models, and Machine Learning.

During World War II, early work in Natural Language Processing was influenced by information theory, probabilistic algorithms for speech, and the development of finite state automata.

Between 1957 and 1970, two primary approaches emerged. The symbolic approach, based on formal language theory and AI logic theories, focused on rule-based processing. Meanwhile, the stochastic approach leveraged Bayesian methods, leading to the development of early Optical Character Recognition systems.

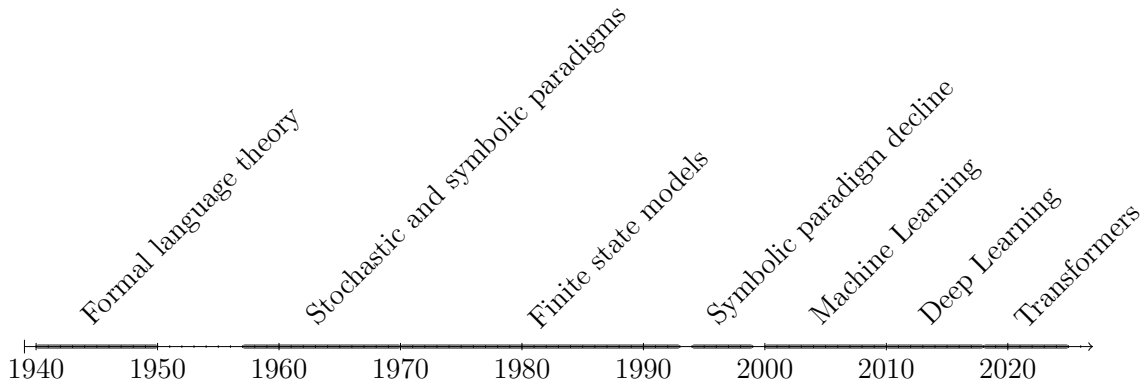
From 1970 to 1993, the focus shifted toward empirical methods and finite-state models. Researchers worked on understanding semantics, discourse modeling, and structural analysis of language.

By the mid-1990s, symbolic approaches began to decline, and the late 1990s saw a surge in data-driven methods, fueled by the rise of the internet and new application areas.

The 2000s marked a deep integration of Machine Learning into Natural Language Processing. The increasing availability of annotated datasets, collaboration with Machine Learning and high-performance computing communities, and the rise of unsupervised systems solidified empiricism as the dominant paradigm.

From 2010 to 2018, Machine Learning became ubiquitous in Natural Language Processing, with Neural Networks driving major advances in conversational agents, sentiment analysis, and language understanding.

Since 2018, the field has been revolutionized by Transformer architectures. Pretrained language models, such as BERT and GPT, have enabled transfer learning at an unprecedented scale, leading to the rise of massive online language models.



1.2 Text analysis

Before applying any NLP algorithm, it is important to standardize and clean the text. Preprocessing ensures consistency and improves the accuracy of downstream tasks. Common cleaning steps include:

- Before tokenization, removing non-content information such as HTML tags, converting text to lowercase, and eliminating punctuation.
- After tokenization, filtering out stopwords (extremely common words that add little meaning), removing low-frequency words, and applying stemming or lemmatization to reduce vocabulary size.

1.2.1 Text mining

Text may need to be extracted from various sources, each with its own challenges:

- Textual documents, HTML pages, and emails often contain formatting elements that should be removed.
- Binary documents are more complex to process. In PDFs, text may be laid out in multiple columns, requiring careful reconstruction. If all PDFs follow a consistent format, handwritten rules may suffice; otherwise, Machine Learning techniques may be needed.
- Scanned documents require Optical Character Recognition, which relies on Deep Learning to convert images to text. However, Optical Character Recognition is not flawless and can introduce recognition errors.

1.2.2 Characters encoding

When storing and processing text, different character encodings must be considered.

ASCII encoding represents only 128 characters, mapping letters and symbols to numerical values. While sufficient for basic English text, it cannot handle many linguistic symbols.

UTF-8 encoding supports over 149,000 Unicode characters, covering more than 160 languages. Unicode is essential for processing texts that use non-Latin scripts. It also preserves special characters, such as diacritical marks in Italian and English.

1.2.3 Tokens

In many languages, spaces serve as natural boundaries between words, making tokenization straightforward. However, if spaces weren't available, we would need alternative methods to segment text. Since they do exist in most languages, they are commonly used for tokenization.

Despite this, tokenizing text isn't always simple. Hyphenated words can pose challenges, as some languages construct long, compound words that may need to be split for effective processing. In other cases, meaningful units are spread across multiple non-hyphenated words in multiword expressions. Additionally, punctuation cannot always be blindly removed, as certain clitics (words that don't stand alone) depend on them for meaning.

For languages like Chinese, tokenization is even more complex since it does not use spaces to separate words. Deciding what constitutes a word is non-trivial, and a common approach is to treat each character as an individual token. Other languages, such as Thai and Japanese, require sophisticated segmentation techniques beyond simple whitespace or character-based tokenization.

A more advanced method, sub-word tokenization, can be useful for handling long words and capturing morphological patterns within a language. Instead of relying purely on spaces, data-driven techniques determine the optimal way to segment text. This is particularly important for Machine Learning applications, where models benefit from explicit knowledge of a language's structure. A common approach is byte-pair encoding.

In some tasks, text must be split into sentences rather than just words. Sentence segmentation often relies on punctuation marks, which typically indicate sentence boundaries. However, periods are more ambiguous, as they also appear in abbreviations, numbers, and initials. A common approach is to tokenize first and then use rule-based or Machine Learning models to classify periods as either part of a word or a sentence boundary.

1.2.4 Text normalization

In many applications, such as web search, all letters are converted to lowercase. This process significantly reduces the vocabulary size and improves recall by ensuring that variations in capitalization do not affect search results. Since users often type queries in lowercase, this normalization helps retrieve more relevant documents.

For classification tasks, removing case can simplify the learning process by reducing the number of distinct tokens. With fewer parameters to learn, models can generalize better even with limited training data.

However, case folding is not always beneficial. In some contexts, capitalization carries meaningful information. Machine translation and information extraction may also benefit from preserving case distinctions.

Beyond case folding, word normalization involves converting words or tokens into a standard format, ensuring consistency in text processing. This step is particularly crucial in applications like web search, where variations in word forms should not hinder retrieval performance.

Stopwords Stopwords are the most frequently occurring words in a language. They typically have extremely high document frequency scores but carry little discriminative power, meaning they do not contribute much to understanding the main topic of a text. Removing stopwords can sometimes improve the performance of retrieval and classification models, mainly by reducing computational and memory overhead. Eliminating common words can also speed up indexing by preventing the creation of excessively long posting lists. However, stopword removal is not always beneficial. In some cases, stopwords play an important role in understanding meaning and context.

1.2.5 Morphology and lemmatization

Morphology, a fundamental concept in linguistics, refers to the analysis of word structure. At its core, it involves breaking words down into their smallest meaningful units, known as morphemes.

Definition (*Morpheme*)**.** A morpheme is the smallest linguistic unit that carries meaning.

A morpheme can be a root (base form) or an affix, which can appear as a prefix, infix, or suffix.

Definition (*Lexeme*)**.** A lexeme is unit of lexical meaning that exists regardless of inflectional endings or variations.

Definition (*Lemma*)**.** A lemma is the canonical form of a lexeme.

Definition (*Lexicon*)**.** A lexicon is the set of all lexemes in a language.

Definition (*Word*)**.** A word is an inflected form of a lexeme.

Lemmatization Lemmatization is the process of reducing words to their lemma, or base form. By normalizing words to a common root, it helps deal with complex morphology, which is essential for many languages with rich inflectional systems.

Stemming Stemming is a simpler approach that removes affixes based on predefined rules, often without considering the actual meaning or structure of the word. Unlike lemmatization, stemming does not require a lexicon.

Porter stemming algorithm (1980) is one of the most widely used stemming algorithms, it applies a set of rewriting rules to reduce words to their stems. While computationally efficient, stemming can introduce errors such as collisions (different words may be reduced to the same stem) and over-stemming (some words may be shortened excessively, losing meaning).

While stemming is computationally cheaper, lemmatization provides more linguistically accurate results, making it preferable for tasks requiring precise language understanding.

1.3 Regular expressions

Text documents are fundamentally just sequences of characters. Regular expressions provide a powerful way to search within these sequences by defining patterns that match specific character sequences. Regular expressions are useful for:

- *Pattern detection*: determine whether a specific pattern exists within a document.
- *Information extraction*: locate and extract relevant information from a document whenever the pattern appears.

Name	**Formula**	**Description**
Exact match	`aaa`	Matches the exact sequence `aaa`
Sequence choice	`(aaa|bbb)`	Matches either `aaa` or `bbb`
Wildcard	`.`	Matches any single character except for a newline
Character choice	`[]`	Matches any one character inside the square brackets
Newline	`\n`	Represents a newline character
Tab	`\t`	Represents a tab character
Whitespace	`\s`	Matches any whitespace character
Non-whitespace	`\S`	Matches any non-whitespace character
Digit	`\d`	Matches any digit (`[0-9]`).
Word character	`\w`	Matches any word character (`[a-zA-Z0-9]`)
Zero or more times	`*`	Matches the preceding character zero or more times
One or more times	`+`	Matches the preceding character one or more times
Zero or one time	`?`	Matches the preceding character zero or one time
Exactly n times	`{n}`	Matches n occurrences of the preceding character
From n to m times	`{n,m}`	Matches between n and m occurrences of the preceding character

1.3.1 Regular expressions in text mining

Regular expressions offer a powerful way to define patterns that can extract specific content from text documents. This allows for highly customizable and efficient text processing.

The advantages of regular expression based text extraction are:

- *Simplicity*: regular expressions are a straightforward way to specify patterns.
- *Precision*: extraction rules can be finely tuned to target specific patterns, which reduces false positives.

The limitations of regular expression based text extraction are:

- *Manual rule creation*: writing extraction rules usually requires manual effort, which can be time-consuming and complex.
- *False positives*: regular expressions can still yield false positives, where the pattern matches unintended content.
- *False negatives*: false negatives occur when the rules are not broad enough to capture all valid cases.
- *Lack of context awareness*: regular expressions typically work on isolated patterns, without understanding the context in which the pattern appears.

CHAPTER 2

Text classification

2.1 Supervised Learning

Machine Learning involves techniques that help machines become more intelligent by learning from past data to predict future outcomes.

Definition (*Machine Learning*)**.** A computer program is considered to learn from experience E when it improves its performance on a specific task T based on that experience, as measured by a performance metric P.

In supervised learning, each training example is represented as a vector in a feature space. These training examples are labeled with the correct class. The task is to divide the feature space in such a way that the model can make accurate predictions for new, unseen data points.

In practice, however, data is rarely perfectly clean or neatly separable. Often, different classes of data overlap, meaning they may not be linearly separable. Additionally, instances are described by many features, and not all features are equally useful for distinguishing between classes. Some features might provide more meaningful information, while others might be less relevant.

To address this, classifiers divide the feature space into regions, and the boundaries between these regions can either be linear or non-linear. Linear models use simple decision boundaries to separate classes. On the other hand, non-linear models are capable of creating more complex decision boundaries to better fit the data.

Training The process of training a model involves finding a formula that can predict the correct labels for new instances. The learning algorithm takes in the training data and their corresponding labels, and then searches for the best parameters for the model. These parameters are adjusted in order to minimize prediction loss on the training data. The learning algorithm operates based on its own settings, called hyperparameters, which control aspects of the learning process.

Hyperparameters Hyperparameters play a crucial role in determining the model's behavior. These are parameters that govern the learning algorithm itself, and they can influence the complexity of the model.

2.1.1 Overfitting

Overfitting is a common challenge in Machine Learning. As the model becomes more complex, the error on the training data tends to decrease. However, at some point, the model may begin to memorize the training data rather than learning generalizable patterns, leading to poor performance on unseen data. This is known as overfitting. The goal is to find the model that strikes the right balance, one that generalizes well to new, unseen data, rather than simply fitting the training data perfectly.

To prevent overfitting, we need to carefully select hyperparameters that control the model's complexity. Since the training data alone doesn't tell us how well the model will generalize, and the test data should be reserved for final evaluation, we use a separate validation dataset. This dataset is a portion of the training data held out during the training process. The model is trained multiple times with different hyperparameter settings, and its performance is evaluated on the validation set. By comparing how well each configuration generalizes, we can choose the hyperparameters that lead to the best performance.

2.2 Text classification

Text classification involves training a model to assign documents to specific categories. It's a widely-used task across various fields. Classification problems can take different forms, including:

- *Binary classification*: where the output is either one of two possible labels.
- *Ordinal regression*: where the output is an ordered value, representing categories with a natural rank.
- *Multi-class classification*: where the output corresponds to one category from a set of predefined options.
- *Multi-label classification*: where the output is a set of categories that can overlap or be chosen simultaneously.

2.2.1 Feature extraction

Text can be arbitrarily long, so it can't be fed directly into a model. To make text usable for machine learning, we first need to extract meaningful features. Features are the useful signals in the document that help predict its category. To do this, we need to convert text data into a vector of features that a classifier can process.

When training data is limited (with only a few documents available), some approaches to feature extraction include:

- *Syntax-based features*: such as the number of capitalized words.
- *Part-of-speech features*: like the count of verbs versus proper nouns.
- *Reading difficulty features*: such as average word or sentence length

However, the most common and effective features to extract are the words themselves. The vocabulary of the document provides significant signals. The frequency of word occurrences offers additional context.

One popular method is the Bag-of-Words (BoW) model. This model represents documents as vectors of word counts. It results in a sparse representation (long vectors with many zero entries).

One-hot encoding One-hot encoding could be an option to create fixed-dimensional feature vectors. However, there are practical limitations: represents each word as a binary feature, creating a vector where each dimension corresponds to a word in the vocabulary. To solve this, we often sum the one-hot encodings. This reduces the number of features to the size of the vocabulary. While this discards word order, it retains the critical information about the vocabulary and word occurrences.

2.2.2 Word frequencies

In text data, certain statistical laws describe how term frequencies behave across documents and collections:

- *Heap's law*: this law states that the vocabulary size grows with the square root of the document or collection length:
$$V(l) \propto l^{\beta}$$
Here, $\beta \approx 0.5$. This means the number of unique words in a document or collection increases slowly as the length of the document or the size of the collection grows.

- *Zipf's law*: this law describes the frequency of a token being inversely proportional to its rank:
$$\text{ctf}_t \propto \frac{1}{\text{rank}(t)^s}$$
Here, $s \approx 1$. In simple terms, a small number of words (the most frequent ones) appear very often, while a large number of words appear very rarely.

Heap's Law is derived from Zipf's Law and can be understood through models like random typing, showing how the vocabulary of a document or collection grows more slowly compared to its length.

Bag of Words The Bag of Words model represents a document as a collection of its terms, ignoring grammar and word order but keeping track of the frequency of each word. In this model:

- The vocabulary of a document is much smaller than the vocabulary of the entire collection, so the terms in the document generally give a good representation of its content.

- The BoW representation typically includes the count of occurrences of each term, although a binary representation (indicating presence or absence of words) can also be used with minimal loss of information.

However, the BoW model has limitations. It produces a sparse representation of the text, meaning most of the values in the vector are zero. The model completely ignores word order, which means it cannot capture the sequence or context in which words appear.

To improve this, we can extend BoW to include n-grams, which capture sequences of words. This can enhance performance, but it significantly increases the number of features, requiring more data to avoid overfitting.

2.2.3 Preprocessing

In natural language processing, preprocessing is a critical step to prepare the text data for machine learning. Common preprocessing tasks include tokenization, spelling correction, and other forms of cleaning the text.

Tokenization The default tokenizer in the scikit-learn toolkit for machine learning in Python is simple and efficient. However, for more complex tokenization, the NLTK (Natural Language Toolkit) offers a tokenizer that uses advanced regular expressions to identify different types of tokens, such as words, numbers, punctuation.

Spelling correction When dealing with text data, misspellings can often occur, especially with user-generated content. Correcting these misspellings can improve model performance. One way to approach spelling correction is by using a probabilistic model.

If we had an enormous corpus of misspellings and their correct versions, we could estimate the probability of a misspelling being corrected in a certain way by using string edit distance, which measures how much one string differs from another. The edit distance counts the minimum number of operations (insertions, deletions, substitutions, or transpositions) needed to convert one string into another.

To apply this to spelling correction, we use Bayes' Rule to reverse the conditional probability:

- The likelihood of a misspelling being corrected to a particular word can be computed using the edit distance between the misspelled word and the candidate correction.
- The prior Pr(correct) represents the popularity or frequency of the word in a large corpus. This can be estimated by counting how often the candidate correction appears in a corpus.
- The likelihood Pr(observed | correct) represents the probability of observing the misspelled word given the correct word.

In practical terms, the prior shows how often a word appears in a large corpus, while the likelihood shows how likely it is that the observed misspelling corresponds to a particular correction, based on string edit distance. Since the denominator in Bayes' Rule is the same for all candidate corrections, we can ignore it during the calculation and normalize probabilities later.

To improve spelling correction, we can incorporate contextual information. This can be done by considering bigrams (pairs of consecutive words) instead of just individual words (unigrams). By calculating the bigram probabilities Pr(bigram), the model can use both the misspelled word and the previous word in the sentence as features. This approach turns the spelling correction process into a Naïve Bayes model with two features: the misspelled word and the previous word in the sentence.

2.3 Linear classifier

In text classification, where documents are often represented with a bag-of-words model, linear classifiers are commonly used due to the high dimensionality of the feature space. Linear models work by assigning a parameter to each word in the vocabulary, making them highly interpretable. This approach allows us to understand which terms have the greatest impact on the prediction and the extent of their influence.

Decision boundaries Linear classifiers create decision boundaries that are represented as hyperplanes in an n-dimensional vector space. The model includes a weight vector, which is the same size as the feature vector, along with a bias term.

2.3.1 Multinomial Naïve Bayes

Naïve Bayes is one of the oldest and simplest text classification algorithms. It is called naïve because it makes a simplifying assumption: that word occurrences are statistically independent of each other given the class label.

This assumption means that each word contributes independent information about the class. It simplifies the process of calculating the model's parameters, making the algorithm easy to implement. However, in practice, this assumption doesn't hold since words are often correlated with each other. Despite this, Naïve Bayes still produces effective predictions, though the assumption can make the model seem overly confident in certain cases.

If all instances of a word appear exclusively in one class, we can have issues with probability estimation. To avoid this problem, we use smoothing, which consists in adding a small pseudo-count α for each feature. This helps prevent zero probabilities for unseen word-class combinations. The value of α can be selected to optimize performance or set to a default value (if $\alpha = 1$, this technique is known as Laplace smoothing).

Independence assumption The independence assumption is not necessarily a big problem. the assumption simplifies both model estimation and prediction, which, in turn, makes the process more efficient. While this theoretically reduces the model's accuracy slightly, in practice, Naïve Bayes often works well for some tasks.

Advantages	
Speed	Naïve Bayes is incredibly fast to train, requiring just one pass over the training data. No need for complex optimization routines like gradient descent
Stability	It's a reliable model even with limited data. If the conditional independence assumption holds, it provides the best possible performance
Disadvantages	
Scalability	Naïve Bayes doesn't perform as well on large datasets compared to other classifiers because redundant features are counted multiple times
Calibrating probabilities	The predicted probabilities are not well calibrated, meaning they can be less reliable for certain applications

2.3.2 Logistic regression

The farther a point is from the decision boundary, the more confident we are in our prediction. The signed distance of a point from the hyperplane is given by:

$$s(\mathbf{x}) = \boldsymbol{\theta}\mathbf{x} - b$$

To convert the signed distance $s(x)$ into a probability, we need a function that maps the entire range of real values $\mathbb{R}$ to the probability range $[0, 1]$. The standard function used for this purpose is the logistic curve (also known as the sigmoid function):

$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

This function outputs a probability of 0.5 at the decision boundary ($s = 0$). The slope, or the speed of probability change, depends on the magnitude of $\boldsymbol{\theta}$

Advantages	
Well-calibrated probabilities	Logistic regression produces well-calibrated probability estimates
Scalability	It can be trained efficiently and scales well to large numbers of features
Interpretability	The model is explainable, since each feature's contribution to the final score is additive
Disadvantages	
Linearity assumption	Logistic regression assumes feature values are linearly related to log-odds
Sensitivity to assumptions	If the linearity assumption is strongly violated, the model will perform poorly

2.3.3 Support Vector Machines

Imagine a dataset where two classes are clearly separable into two groups. There are many possible positions for the linear decision boundary. We want to select a boundary that generalizes well to new, unseen data, avoiding overfitting.

The SVM approach finds the maximum margin hyperplane that separates the two classes. The margin, denoted γ, is the distance from the hyperplane to the closest points on either side. These closest points are called support vectors. Support vectors are the points that lie exactly on the margin. They prevent the margin from expanding and thus help define the location of the boundary. In a d-dimensional space, you need at least $d + 1$ support vectors to define the hyperplane.

In contrast to logistic regression, where the position of the hyperplane depends on the entire dataset, the SVM hyperplane's position is determined only by the closest points. The convex hull of the data points helps define the boundary, and moving internal points does not affect the hyperplane.

Hard margin SVM A basic SVM is also a linear classifier that finds a hyperplane in feature space that best separates the two classes. While logistic regression and Naïve Bayes also find linear decision boundaries, the difference lies in the loss function used to find the model parameters:

- Logistic regression uses negative log-likelihood, which penalizes points based on the probability of incorrect predictions, even if they are correctly classified.
- SVM uses hinge loss, which only penalizes points that are on the wrong side of the margin (or very close to it).

Mathematically, the loss function for SVM is:

$$\mathcal{L}(\mathbf{w}) = \sum_i w_i^2 + \sum_j \varepsilon_j$$

Here, ε_j is the error for a prediction (x_j, y_j), and it is defined as:

$$\varepsilon_j = \max(0, 1 - y_j \mathbf{w} x_j)$$

This formulation reflects the hinge loss, which penalizes misclassified points or those close to the margin.

Soft margin SVM In the case of non linearly separable dataset, SVMs still aim to separate the classes by penalizing points that are on the wrong side of the margin, based on their distance from the hyperplane. Support vectors are now the points that are either misclassified or very close to the margin, contributing a non-zero amount to the loss function.
The objective function to minimize remains similar to the hard margin case, but it includes a penalty for misclassified points:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_i w_i^2 + C \sum_j \varepsilon_j$$

Here, ε_j is the distance from the j-th support vector to the margin and C is a hyperparameter that controls the trade-off between minimizing the margin size and penalizing errors. A large C places more emphasis on minimizing misclassifications, while a smaller C allows a larger margin even at the cost of more misclassifications.

SVM and logistic regression Both SVM and logistic regression are linear classifiers, but they differ in the loss functions they use: Logistic regression uses log-likelihood, which penalizes points based on the probability of incorrect predictions, including those correctly classified. SVM uses hinge loss, which only penalizes points on the wrong side of the margin or those very close to it.

2.4 Model Evaluation

When evaluating a classification model, a common starting point is the confusion matrix, which summarizes the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Several metrics are derived from the confusion matrix to assess the performance of a model.

Accuracy Accuracy is the proportion of correct predictions, and it can be calculated as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Precision Precision measures the proportion of positive predictions that were actually correct. It is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

A high precision indicates that when the model predicts positive, it is likely to be correct.

Recall Recall, also known as sensitivity or true positive rate, measures the proportion of actual positives that were correctly identified by the model. It is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

High recall means the model is good at identifying positive instances, though it might also include more false positives.

F-measure The F1 score combines precision and recall into a single metric by taking the harmonic mean of the two:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1 score is particularly useful when you need a balance between precision and recall, especially when the class distribution is imbalanced.

Area under the ROC curve The Area under the ROC Curve (AuC) is another important metric, often used when dealing with different thresholds for making predictions. It measures the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate against the false positive rate at various thresholds. A higher area under the curve indicates better model performance, as it signifies that the model is better at distinguishing between classes regardless of the threshold.

2.4.1 Multi-class classifiers

When working with multi-class classifiers, the confusion matrix expands to an $n \times n$ matrix, where n is the number of classes. In this case, precision and recall are calculated for each class, treating each class as the positive class in a one-vs-all fashion.

For a given class i, precision is the ratio of true positives for that class to the sum of true positives and false positives. Similarly, recall for class i is the ratio of true positives for that class to the sum of true positives and false negatives.

To combine the precision and recall across all classes, two methods are commonly used:

- *Macro-average*: takes the average of the precision and recall across all classes, treating each class equally, regardless of how many instances belong to that class. This method gives equal importance to all classes, which can be useful when the classes are imbalanced.
- *Micro-average*: aggregates the true positives, false positives, and false negatives across all classes before calculating precision and recall. This method gives more weight to classes with more data points. The micro-average is useful when the number of data points varies significantly between classes.