# Entire Literature Review of Automating News Article Classification with Machine Learning Based on Document Classification Aspects


School of Computing, Engineering & Digital Technologies

Teesside University

Middlesbrough TS1 3BA

Author: Michael Zats (18361063/2)

Supervisor: PhD Bohus Ziskal

# Contents

# List of Figures

# 1 Entire Literature Review

## 1.1 Previous Work

The landscape of document classification has evolved significantly, paralleling advancements in computational techniques and the increasing complexity of data. Initially, the field was dominated by rule-based systems, which, while effective for straightforward tasks, quickly showed their limitations in scalability and adaptability, particularly as the volume and variety of digital content burgeoned (Sebastiani, 2002). The transformation was catalyzed by the adoption of machine learning (ML) models, which brought a paradigm shift in how documents were categorized. Support Vector Machines (SVM) and Decision Trees emerged as frontrunners, celebrated for their ability to efficiently navigate the high-dimensional space of text data (Joachims, 1998; Quinlan, 1986).

As the digital era progressed, the advent of deep learning marked a new horizon for document classification. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) demonstrated unparalleled proficiency in extracting complex patterns from text, setting new benchmarks for accuracy and efficiency in classification tasks (LeCun et al., 2015; Hochreiter & Schmidhuber, 1997). This transition was not just technical but also conceptual, shifting the focus from manual feature engineering to models capable of learning hierarchical representations of text data.

The relevance of these advancements to the news industry cannot be overstated. The proliferation of online news platforms has necessitated more sophisticated approaches to

content management and distribution, making efficient document classification systems essential. The sector has seen a specific interest in leveraging ML and deep learning models to categorize news content accurately, facilitating improved information retrieval and user experience (Hinton et al., 2012).

However, the journey is far from complete. The field continues to grapple with challenges such as handling imbalanced datasets, which can skew model performance, and the ever-present concerns of data privacy and ethical use of algorithms (Chawla et al., 2002; Barocas & Selbst, 2016). Moreover, the dynamic nature of news content, with its constantly evolving topics and formats, presents unique challenges that demand ongoing innovation in classification techniques.

This research builds upon the foundational work laid out by predecessors, including my own preliminary studies, which explored the effectiveness of SVM in text classification. It aims to delve deeper into the capabilities of various ML models, considering the unique context and requirements of the news industry. The goal is to not only identify the most effective model for current needs but also to contribute to the body of knowledge that will support future advancements in document classification (Michael Zats, 2022).

## 1.2 Machine Learning in Document Classification

Machine learning has fundamentally transformed the field of document classification by introducing methods that learn from data, rather than relying on explicit programming. This shift allowed for the effective handling of complex and voluminous datasets. Initially, algorithms such as Naive Bayes, Decision Trees, and k-Nearest Neighbors laid the groundwork for this transformation, with Naive Bayes being particularly noted for its efficiency in managing high-dimensional text data through probabilistic modeling (McCallum & Nigam, 1998; Quinlan, 1986).

The emergence of Support Vector Machines (SVM) marked a significant evolution in

text classification tasks. SVM's ability to delineate classes with an optimal hyperplane, coupled with the kernel trick, enabled effective binary and multi-class classification, accommodating non-linear data distributions (Cortes & Vapnik, 1995). This period underscored the critical role of feature representation in the success of ML models. Traditional approaches like Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) provided foundational techniques for vectorizing text, albeit with limitations in capturing semantic meaning (Salton & Buckley, 1988).

The introduction of word embeddings, notably such as Word2Vec, represented a leap forward in feature representation. By situating words in dense vector spaces, these models encapsulated semantic relationships, significantly enhancing the nuanced understanding of text for classification purposes (Mikolov et al., 2013; Pennington et al., 2014).

Deep learning, an advanced subset of machine learning, further expanded the horizons of document classification. Neural network architectures, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have demonstrated exceptional capability in learning hierarchical features from text, setting new benchmarks across classification benchmarks without the need for manual feature engineering (LeCun et al., 2015; Hochreiter & Schmidhuber, 1997).

In summary, the progression from basic ML algorithms to sophisticated deep learning models has significantly enriched the landscape of document classification. This evolution not only improved the accuracy and efficiency of classification systems but also opened avenues for more nuanced and scalable solutions. Ongoing research in machine learning and deep learning is expected to continue this trajectory, offering even more refined approaches to document classification.

## 1.3 Naive Bayes

Naive Bayes classifiers, grounded in the principles of Bayes' theorem, have been instrumental in advancing document classification. They operate under the assumption that features (e.g., words in a document) are independent of each other, simplifying the calculation of a document's likelihood belonging to a certain class as McCallum & Nigam, 1998 and Manning, Raghavan, & Schütze, 2008 claim. This approach, despite its simplicity, has proven remarkably effective, particularly in handling high-dimensional datasets typical in text classification tasks. The mathematical foundation of Naive Bayes is expressed through the following: a Naive Bayes classifier $B$ can be defined as a tuple $(C, F, P)$, where:

- $C$ is a set of class labels.
- $F$ is a set of features used to describe instances, with each feature $f \in F$ having a domain of possible values.
- $P$ represents the probability model, consisting of:
  - The prior probabilities $P(c)$ for each class $c \in C$, representing the probability of each class occurring in the dataset.
  - The conditional probabilities $P(f|c)$ for each feature $f \in F$ given each class $c \in C$, representing the probability of observing feature value $f$ given that the instance belongs to class $c$.

Under the Naive Bayes assumption, the features are considered to be conditionally independent given the class label. Thus, the probability of observing a set of features $f_1, f_2, \ldots, f_n$ given a class $c$ can be expressed as:

$$P(f_1, f_2, \ldots, f_n|c) = \prod_{i=1}^{n} P(f_i|c)$$

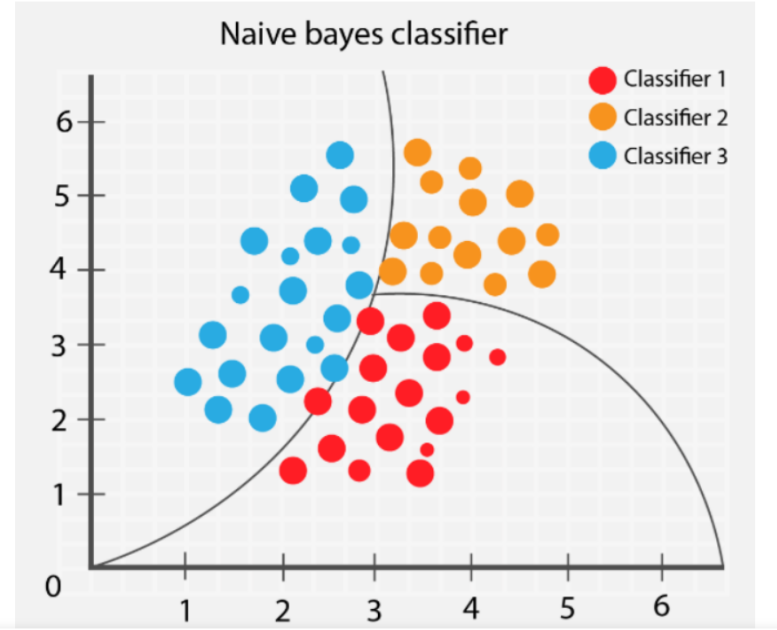



Figure 1.1: Naive Bayes Classifier Algorithm

Given a new instance with a set of features, the classifier assigns it to the class $C_{\max}$ that maximizes the posterior probability:

$$C_{\max} = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(f_i|c)$$

The efficiency and scalability of Naive Bayes come from its ability to quickly process large volumes of text, leveraging probability calculations to classify documents even in the face of vast feature spaces. However, its "naive" assumption of feature independence is not always accurate in real-world scenarios, where word contexts and relationships can significantly affect meaning. Despite this, techniques such as Laplace smoothing have been developed to address potential pitfalls like the zero-frequency problem, ensuring more reliable classification outcomes.

Critically, the evolution of Naive Bayes within the broader landscape of machine learning

models for document classification underscores a balance between computational simplicity and the nuanced demands of text data analysis. Its continued application and development reflect an ongoing pursuit of models that can adeptly navigate the complexities of language, providing a foundational tool for researchers and practitioners alike in the field of document classification (McCallum & Nigam, 1998; Manning, Raghavan, & Schütze, 2008).

## 1.4 Decision Trees

Decision Trees stand out in the machine learning landscape for their intuitive approach to document classification, representing decisions as a series of branching rules leading to specific outcomes. Rooted in the principle of partitioning the dataset into subsets based on feature values, these models construct paths from a root to leaf nodes, each leaf representing a classification category (Quinlan, 1986; Breiman et al., 1984). This hierarchical structure allows for easy interpretation of the classification process, mimicking human decision-making logic and offering transparency in model predictions. The mathematical foundation of Decision Trees is shown as follows: a decision tree $T$ can be defined as a tuple $(N, E)$, where:

- $N$ is a set of nodes, consisting of decision nodes $D$ and leaf nodes $L$ such that $N = D \cup L$ and $D \cap L = \emptyset$.
- $E$ is a set of edges, where each edge $e \in E$ connects a pair of nodes and represents the outcome of a decision or a test.

Each decision node $d \in D$ represents a test on an attribute, functioning as $f_d : X \to Y$, where $X$ is the attribute's domain and $Y$ represents possible outcomes.

Each leaf node $l \in L$ represents a final decision or classification, associated with a class label $c_l$.

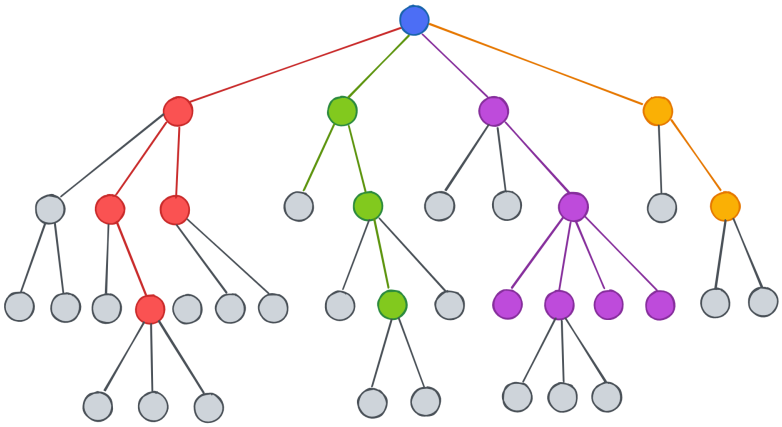


Figure 1.2: Decision Tree Algorithm

Despite their advantages, Decision Trees can be susceptible to overfitting, particularly when dealing with the high-dimensional data characteristic of text classification tasks. Overfitting occurs when the model becomes too complex, capturing noise rather than the underlying pattern, leading to poor generalization on unseen data. Pruning strategies and setting constraints on tree depth are common practices to combat this, ensuring the model remains robust and applicable to new data (Breiman et al., 1984).

Furthermore, Decision Trees lay the groundwork for more complex ensemble methods like Random Forests, which aggregate the predictions of multiple trees to improve accuracy and mitigate the risk of overfitting. This ensemble approach leverages the strength of individual trees while compensating for their vulnerabilities, showcasing the versatility and adaptability of Decision Trees in evolving machine learning applications (James et al., 2013).

In the context of document classification, Decision Trees distinguish themselves by segmenting datasets into subsets based on textual features, essentially making hierarchical decisions that culminate in document categorization. This method's transparency is particularly valued for its straightforward interpretability, allowing one to trace through

the decision points to understand the classification outcome. Addressing the tendency for overfitting in high-dimensional text data, pruning and tree depth constraints enhance model generalization.

The balance between simplicity, interpretability, and predictive performance makes Decision Trees a valuable tool in the text classification toolkit. Their continued relevance in machine learning research and application underscores the importance of foundational models in advancing the field.

## 1.5 Random Forest

The Random Forest algorithm, an ensemble learning technique, has emerged as a pivotal advancement in document classification, skillfully navigating the challenges presented by high-dimensional and sparse text data. As claimed by Breiman (2001), Random Forest integrates the strengths of multiple Decision Trees to form a robust classification system. This method significantly mitigates overfitting—a common drawback in single Decision Trees—by employing bagging (Bootstrap Aggregating) to train each tree on a random subset of the data, thus ensuring a diverse ensemble (Breiman, 2001; Liaw & Wiener, 2002).

At its core, Random Forest utilizes a majority voting system among its Decision Trees to classify documents. Each tree, trained on a bootstrapped dataset, contributes a vote towards the final classification outcome, enhancing prediction accuracy and model robustness (Breiman, 2001; James et al., 2013). The mathematical representation of Random Forest's decision-making process underscores the aggregation of individual tree predictions to achieve a collective classification decision, a strategy that proves highly effective in handling the intricate patterns within text data. The corresponding mathematical explanation of a random forest algorithm is the following:

a Random Forest $R$ can be defined as an ensemble of decision trees $\{T_i\}_{i=1}^{N}$, where:

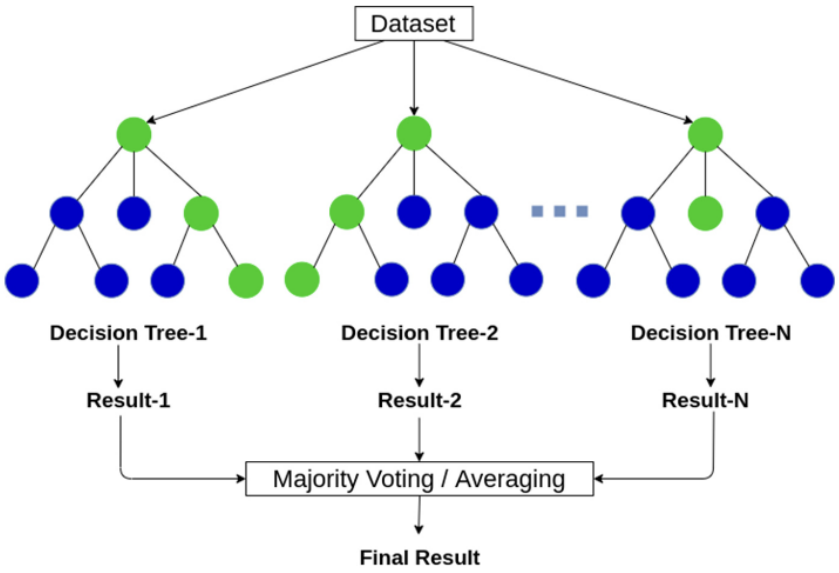



Figure 1.3: Random Forest Algorithm

- Each decision tree $T_i$ is constructed from a bootstrap sample of the training data.
- At each node of the tree, a subset of features is randomly chosen, and the best split is found among those features.

The prediction of the Random Forest for a new sample $x$ is obtained by aggregating the predictions of all the individual trees. This aggregation can be done by majority voting for classification tasks or averaging for regression tasks:

- For classification: $\hat{y} = \text{mode}\{T_i(x)|i = 1, \ldots, N\}$
- For regression: $\hat{y} = \frac{1}{N}\sum_{i=1}^{N} T_i(x)$

This ensemble approach reduces the variance of the model, making Random Forest more robust and accurate than individual decision trees.

This algorithm stands out in the realm of document classification for its capacity to process the intricate and often sparse feature space typical of textual content. By constructing multiple trees that focus on different aspects of the data, Random Forest captures a broad spectrum of patterns, outperforming many baseline classifiers in accuracy and handling unbalanced datasets (Liaw & Wiener, 2002; Strobl et al., 2007). Additionally, its feature importance evaluation mechanism offers valuable insights into the dataset, highlighting the most significant features for classification and shedding light on the underlying structure of the text.

However, the adoption of Random Forest in document classification is not devoid of challenges. The complexity introduced by the ensemble approach can escalate computational demands, and while the model provides a more interpretable framework than some alternatives, the collective decision-making process may obscure individual tree contributions, complicating the interpretability compared to a singular Decision Tree approach.

In conclusion, Random Forest has solidified its position as a formidable tool for document classification, offering a harmonious balance between accuracy, robustness, and interpretability. Its adeptness at navigating the complexities of language and textual features makes it a cornerstone in the landscape of machine learning algorithms for text analysis, continuously supported by ongoing research and application in the field (Breiman, 2001; James et al., 2013; Liaw & Wiener, 2002).

## 1.6 K-Nearest Neighbors

The k-Nearest Neighbors (k-NN) algorithm, a simple yet powerful technique in machine learning, has been effectively applied in the domain of document classification, illustrating the versatility and adaptability of instance-based learning methods. The essence of k-NN lies in its straightforward approach: classify documents based on the majority label of

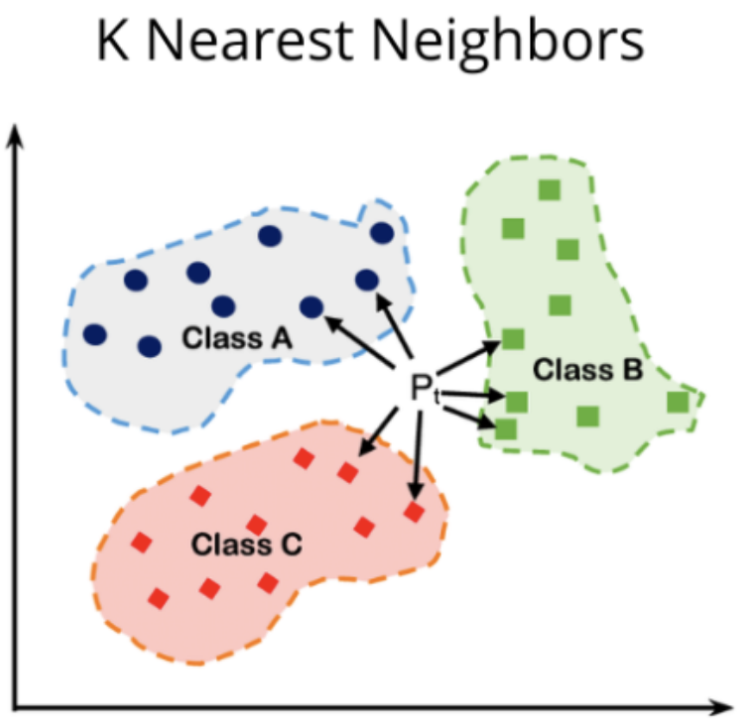



Figure 1.4: K-Nearest Neighbors Algorithm

their k nearest neighbors in the feature space. This method, relying on the proximity of data points, underscores the intuitive principle that documents within similar topics or categories are likely to cluster together in a high-dimensional space (Cover & Hart, 1967; Aha, Kibler, & Albert, 1991).

In the realm of text classification, k-NN's reliance on distance metrics—such as Euclidean distance or cosine similarity—enables it to effectively gauge the similarity between documents. By considering the 'k' closest documents in the feature space, typically represented through vectors obtained from TF-IDF or word embeddings, k-NN makes classification decisions that inherently capture the contextual nuances of textual data (Salton & McGill, 1983; Manning, Raghavan, & Schütze, 2008). The algorithm's

simplicity, requiring no explicit model training phase, allows for dynamic classification, where the decision boundary is determined solely by the distribution of documents in the feature space. In terms of mathematical background, the algorithm can be described as follows:

Given:

- A positive integer $K$ which is the number of nearest neighbors to consider.
- A distance metric $d(x, x')$ to measure the distance between any two data points $x$ and $x'$.
- A set of $n$ labeled training data points $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where $x_i$ represents the features of the $i^{th}$ data point and $y_i$ its associated label.

For a new data point $x$:

1. Compute the distance $d(x, x_i)$ between $x$ and all $x_i$ in the training set.
2. Identify the $K$ training data points $x_{i1}, ..., x_{iK}$ that are closest to $x$, based on the distance metric $d$.
3. For classification, assign $x$ the most frequent label among the $K$ closest points. For regression, assign $x$ the average label of the $K$ closest points:
   - Classification: $\hat{y} = \text{mode}\{y_{i1}, ..., y_{iK}\}$
   - Regression: $\hat{y} = \frac{1}{K} \sum_{j=1}^{K} y_{ij}$

The choice of $K$ and the distance metric $d$ significantly influences the algorithm's performance.

Despite its intuitive appeal and ease of implementation, the performance of k-NN in document classification is heavily influenced by the choice of 'k' and the distance metric used. Selecting an optimal 'k' value is crucial; too small a value makes the

algorithm sensitive to noise, while too large a value may blur the distinctions between different categories (Dasarathy, 1991; Hastie, Tibshirani, & Friedman, 2009). Moreover, the high-dimensionality and sparsity of text data can challenge the efficacy of distance measures, potentially impacting classification accuracy.

The computational cost of k-NN, particularly in large-scale datasets, is another consideration. Since classification involves computing distances to all training instances, the algorithm may become impractically slow as the dataset grows, necessitating efficient indexing and retrieval techniques to maintain scalability (Indyk & Motwani, 1998; Manning, Raghavan, & Schütze, 2008).

Despite these challenges, k-NN remains a favored choice for document classification tasks, especially when the dataset is not overly large or when the simplicity and interpretability of the model are paramount. Its effectiveness in capturing the subtle semantic relationships within textual data, combined with the flexibility to adapt to new data without retraining, positions k-NN as a valuable tool in the arsenal of machine learning algorithms for text analysis. Researchers continue to explore enhancements to k-NN, including dimensionality reduction techniques and advanced distance metrics, to leverage its strengths while mitigating its limitations in handling complex and voluminous text datasets (Hastie, Tibshirani, & Friedman, 2009; Manning, Raghavan, & Schütze, 2008).

## 1.7 Support Vector Machines

Support Vector Machines (SVMs) represent a cornerstone in the evolution of document classification methodologies, encapsulating a sophisticated approach to discerning patterns within high-dimensional text data. Originating from the work of Vapnik and Cortes (1995), SVMs are predicated on the principle of identifying the optimal hyperplane that segregates documents into their respective categories with maximal margin, thereby ensuring a robust classification boundary (Cortes & Vapnik, 1995).

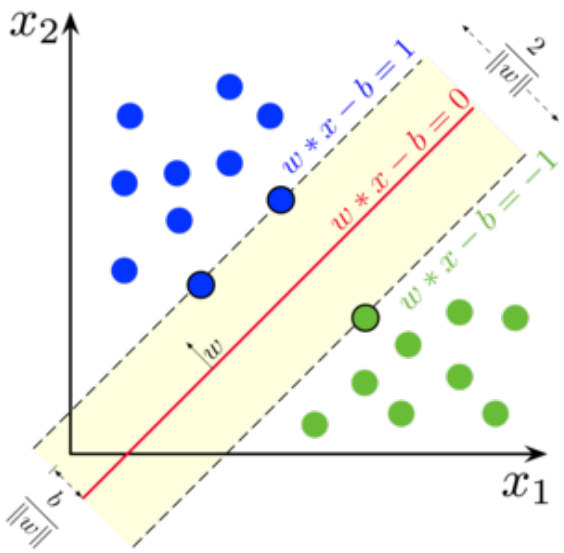



Figure 1.5: Support Vector Machines Algorithm

This foundational concept has positioned SVMs as a powerful tool in the machine learning domain, especially suited to the complexities of text classification. Meanwhile, the mathematical background of the SVM is the following:

Given:

- A training dataset of $n$ points of the form $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where $x_i \in \mathbb{R}^p$ represents the features of the $i^{th}$ data point, and $y_i \in \{-1, 1\}$ its class label.

- A mapping function $\phi : \mathbb{R}^p \to \mathbb{R}^q$ that transforms the input space to a higher-dimensional space (if needed).

The SVM solves the following optimization problem to find the optimal hyperplane:

$$\begin{aligned} &\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to} \quad y_i(\mathbf{w}^T\phi(x_i) + b) \geq 1 - \xi_i, \\ &\qquad\qquad \xi_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

Here, $\mathbf{w}$ represents the weights of the hyperplane, $b$ is the bias term, $\xi_i$ are the slack variables allowing for misclassification of difficult or noisy data points, and $C$ is a regularization parameter that balances the margin maximization and loss minimization.

For non-linearly separable data, SVM uses the kernel trick to implicitly map the input space into a high-dimensional space where a linear separation is possible. Common kernels include:

- Linear: $K(x, x') = x^T x'$
- Polynomial: $K(x, x') = (\gamma x^T x' + r)^d$
- Radial Basis Function (RBF): $K(x, x') = \exp(-\gamma\|x - x'\|^2)$

where $\gamma, r$, and $d$ are kernel parameters.

The decision function for a new data point $x$ is given by:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i K(x, x_i) + b\right)$$

where $\alpha_i$ are the Lagrange multipliers obtained from solving the dual problem of the SVM optimization, with non-zero $\alpha_i$ corresponding to the support vectors.

At the heart of SVMs lies their capacity to transform the feature space using kernel functions, enabling the algorithm to operate effectively in high-dimensional spaces where linear separability may not be inherently present. This kernel trick allows SVMs to classify documents by implicitly mapping their features into a higher-dimensional space, without

the computational burdens typically associated with such transformations. Among the various kernels, the Radial Basis Function (RBF) and polynomial kernels have been widely adopted in text classification tasks, facilitating the handling of nonlinear relationships between textual features (Schölkopf & Smola, 2002; Joachims, 1998).

The application of SVMs in document classification benefits from the algorithm's emphasis on margin maximization, which contributes to its generalization ability and resilience against overfitting. By focusing on the documents closest to the classification boundary (support vectors), SVMs ensure that the model is influenced primarily by the most informative instances, enhancing its predictive performance on unseen data. This aspect is particularly valuable in the context of text data, which often encompasses a vast array of loosely related features that could potentially dilute the predictive power of less discriminative models (Joachims, 1998; Burges, 1998).

However, the deployment of SVMs in document classification is not without challenges. The selection of an appropriate kernel and its parameters requires careful consideration, as these choices significantly impact the model's performance. Furthermore, while SVMs excel in binary classification tasks, extending them to multi-class scenarios necessitates additional strategies, such as one-vs-one or one-vs-all approaches, which can complicate the training and prediction processes (Hsu, Chang, & Lin, 2003).

Despite these considerations, SVMs have consistently demonstrated their efficacy in document classification, offering a compelling balance between accuracy and computational efficiency. Their robustness to high-dimensional spaces and capacity to capture complex patterns make them an indispensable tool in the text classification toolkit. Ongoing advancements in SVM research and their integration with other machine learning techniques continue to expand their applicability and performance in document classification tasks, solidifying their role in the ongoing evolution of text analysis methodologies (Joachims, 1998; Cortes & Vapnik, 1995; Hsu, Chang, & Lin, 2003).

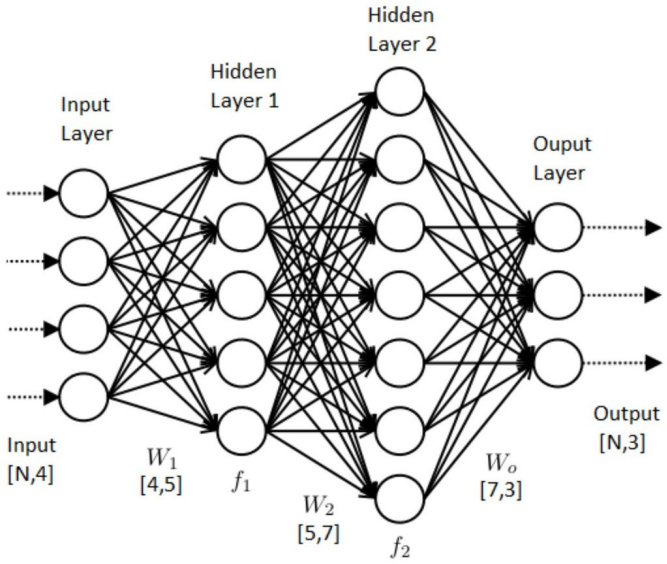



Figure 1.6: Neural Networks Algorithm

## 1.8 Neural Networks

The advent of neural networks, particularly through deep learning paradigms, has marked a transformative phase in the domain of document classification within natural language processing (NLP). These sophisticated architectures excel in discerning complex patterns and relationships inherent in human language, attributing to their design which allows for an intricate modeling of data dynamics (LeCun et al., 2015; Goodfellow et al., 2016).

Convolutional Neural Networks (CNNs), initially conceptualized for visual data analysis, have been remarkably repurposed for text classification endeavors. The convolutional layers within CNNs adeptly identify and extract localized textual features, such as n-grams or specific word arrangements, effectively capturing the essence of textual input. This ability to pinpoint and aggregate pivotal features renders CNNs exceptionally resilient to variations in text, thereby enhancing their predictive accuracy (Kim, 2014; LeCun et al., 2015).

Furthermore, Recurrent Neural Networks (RNNs) and their evolved counterparts, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), offer a tailored approach for sequential data analysis, including textual content. Equipped with memory mechanisms, these networks are capable of preserving and leveraging historical data within a sequence, which is instrumental in comprehending the contextual richness and semantic depth of documents (Hochreiter & Schmidhuber, 1997; Cho et al., 2014). Concerning the mathematical background of neural networks, the following can be seen:

Neural Networks (NN) are a class of models within the field of machine learning designed to mimic the human brain's architecture and function. A basic neural network can be described as follows:

Given:

- An input layer, one or more hidden layers, and an output layer. Each layer consists of a number of neurons (or nodes).
- Each neuron in a layer is connected to all neurons in the next layer, with associated weights $w_{ij}$ indicating the strength of the connection between neuron $i$ and neuron $j$.
- A bias term $b$ for each neuron, adjusting the output independently of its inputs.
- An activation function $\sigma$, such as the sigmoid, tanh, or ReLU function, applied to each neuron's weighted input sum to introduce non-linearity into the model.

For a neuron $j$ in layer $l$, its output $a_j^l$ is given by:

$$a_j^l = \sigma\left(\sum_i w_{ij}^l a_i^{l-1} + b_j^l\right)$$

where $a_i^{l-1}$ is the output of neuron $i$ in the previous layer $l-1$.

The network learns by adjusting the weights and biases to minimize a loss function $\mathcal{L}(y, \hat{y})$, which measures the difference between the predicted output $\hat{y}$ and the actual target $y$. The learning process involves:

1. Forward propagation: Computing the output of the network for a given input by applying the weights, biases, and activation functions sequentially from the input layer to the output layer.

2. Backpropagation: Calculating the gradient of the loss function with respect to each weight and bias in the network by applying the chain rule, moving from the output layer back through the network's layers.

3. Weight update: Adjusting the weights and biases in the direction that minimally reduces the loss, typically using an optimization algorithm like gradient descent.

This process is repeated for many iterations (epochs) over the training data until the network's performance on unseen data stops improving significantly.

Neural networks are highly versatile and can be extended to various architectures, including Convolutional Neural Networks (CNNs) for image processing and Recurrent Neural Networks (RNNs) for sequential data, among others.

The introduction of Transformer models has further revolutionized document classification by employing a self-attention mechanism. This innovation allows the model to evaluate the relevance of different text segments in relation to each other, significantly enhancing the understanding of textual context. Transformer-based models, such as BERT (Bidirectional Encoder Representations from Transformers) and its subsequent iterations, have established new performance standards across various NLP tasks, including the nuanced field of document classification (Vaswani et al., 2017; Devlin et al., 2018).

A notable advantage of leveraging neural networks for document classification is their adeptness at processing raw text. Through the utilization of pre-trained word embeddings

or by cultivating embeddings within the neural network model itself, these systems can encapsulate semantic relationships and nuances between words. This capability fosters the generation of enriched text representations, which, in turn, propels classification accuracy to new heights (Mikolov et al., 2013; Pennington et al., 2014).

In essence, neural networks have substantially broadened the horizon for document classification systems. Their unparalleled ability to extract hierarchical features and decode contextual cues positions them at the forefront of cutting-edge document classification solutions, continuing to push the boundaries of what is achievable in NLP.

## 1.9 Word Embeddings (Pre-trained Embeddings)

The development and integration of Pre-trained Embeddings of Word2Vec into the landscape of natural language processing (NLP) represent a paradigm shift in how machines understand and represent human language. Introduced by Mikolov et al. (2013), Word2Vec is a groundbreaking framework designed to transform words into dense vector spaces, encapsulating semantic relationships in a way that mirrors human cognitive processes. This methodological innovation has been pivotal for a wide array of NLP tasks, including document classification, where capturing the essence of textual content is paramount. Word2Vec operates on the hypothesis that words appearing in similar contexts tend to possess similar meanings, a notion succinctly encapsulated by the distributional hypothesis of linguistics.

Through its two architecturally distinct models—Continuous Bag of Words (CBOW) and Skip-Gram—Word2Vec efficiently learns word embeddings by either predicting a word based on its context (CBOW) or predicting the context given a word (Skip-Gram). These embeddings are then utilized in machine learning models to significantly enhance the performance of document classification tasks by providing a rich, nuanced representation of text (Mikolov et al., 2013; Le & Mikolov, 2014).

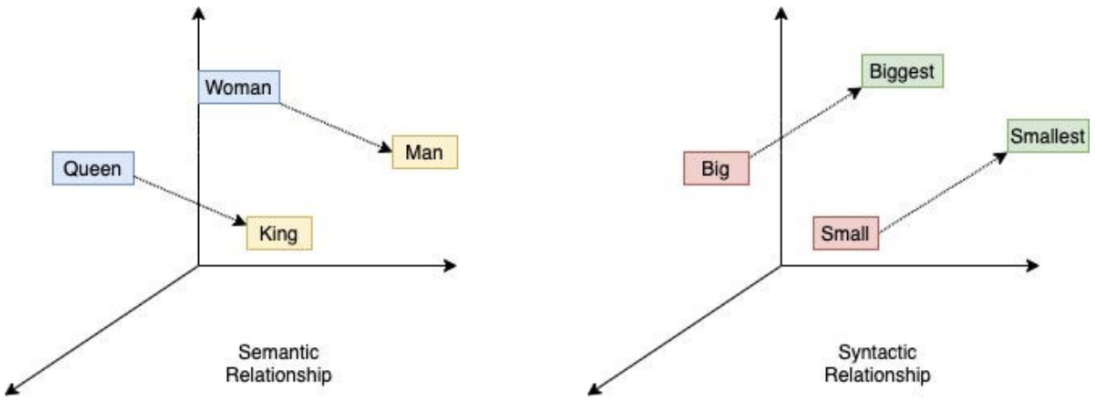



Figure 1.7: Word Embeddings

The mathematical foundation underlying Word2Vec involves optimizing an objective function that measures the prediction accuracy of word contexts. This optimization leads to embeddings where the geometric relationships between vectors capture semantic and syntactic word relationships, such as analogies and categorizations. The effectiveness of Word2Vec in generating these high-quality word embeddings has fundamentally altered the approach to feature extraction in text analysis, shifting from sparse, high-dimensional vectors (like those used in traditional Bag-of-Words models) to dense, low-dimensional, and semantically rich vector representations (Goldberg & Levy, 2014; Rong, 2014).

Formally, given a sequence of training words $w_1, w_2, ..., w_T$, the objective function for CBOW can be expressed as:

$$\frac{1}{T}\sum_{t=1}^{T}\log p(w_t|w_{t-n}, ..., w_{t+n})$$

where $n$ specifies the context window size. Similarly, for Skip-Gram, the objective is:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leq j\leq n, j\neq 0}\log p(w_{t+j}|w_t)$$

The probabilities are typically computed using a softmax function:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^\top v_{w_I})}{\sum_{w=1}^{W} \exp({v'_w}^\top v_{w_I})}$$

where $v_w$ and $v'_w$ are the input and output vector representations of word $w$, and $W$ is the vocabulary size.

Word2Vec's resulting word vectors capture many linguistic regularities and patterns. For example, vector operations can uncover relationships such as $vec(\text{"}king\text{"}) - vec(\text{"}man\text{"}) + vec(\text{"}woman\text{"}) \approx vec(\text{"}queen\text{"})$.

In document classification, the introduction of Word2Vec embeddings has facilitated a more profound understanding of textual data. By leveraging these embeddings, classifiers can discern subtle semantic differences between texts, enabling more accurate and nuanced classification outcomes. This capability is particularly beneficial in dealing with large and complex datasets where traditional methods may fall short in capturing the intricacies of language (Mikolov et al., 2013; Le & Mikolov, 2014).

However, while Word2Vec represents a significant advance in text representation, it is not without limitations. The model's reliance on local context does not inherently capture global document context or the relationships between documents. Consequently, subsequent developments in NLP, such as Doc2Vec and contextual embeddings from models like BERT, have sought to address these shortcomings by providing embeddings that account for broader context and document-level semantics (Le & Mikolov, 2014; Devlin et al., 2019).

In summary, Word2Vec has been instrumental in advancing the field of document classification, offering a mechanism to encode semantic relationships within text into a computationally efficient vector space. Pre-trained Word2Vec models, trained on extensive text corpora, offer a convenient means to leverage these embeddings without the need to train from scratch, providing a powerful tool for a wide range of natural language processing tasks.

## 1.10 Feature Engineering (N-grams)

Feature engineering plays a pivotal role in enhancing machine learning models, especially in the realm of text analysis. N-grams, sequences of 'n' contiguous items from a given text, stand out as a fundamental technique for capturing linguistic patterns. This method effectively encodes local context and structural nuances of language, significantly improving the performance of text-related tasks such as document classification, sentiment analysis, and language modeling (Manning & Schütze, 1999; Litman, Durlach & Lesgold, 2015).

N-grams work by considering the probability of a word's occurrence based on the sequence of words preceding it, thus leveraging the Markov property of natural language. Bi-grams and tri-grams are commonly employed variants that capture word dependencies and phrase structures, providing a richer feature set compared to single-word analysis. This capability allows for a more nuanced understanding of text, enabling algorithms to discern context and meaning more effectively (Brown et al., 1990; Mikolov et al., 2013).

The concept of N-grams can be described as follows:

Given:

- A text corpus $C$, consisting of sentences or documents.
- An integer $n$ specifying the size of the N-gram (e.g., $n = 2$ for bigrams, $n = 3$ for trigrams, etc.).

The process of generating N-grams involves:

1. Tokenizing the text corpus into words or characters, depending on the level of N-gram desired.
2. Sliding a window of size $n$ across the tokens to create sequences of $n$ consecutive words or characters.

3. Optionally, applying frequency filtering or other statistical measures to select the most relevant N-grams for the task at hand.

Mathematically, for a sentence $S = [w_1, w_2, ..., w_m]$, where $w_i$ represents the $i^{th}$ word, the set of N-grams $G$ can be represented as:

$$G = \{(w_i, w_{i+1}, ..., w_{i+n-1}) | 1 \leq i \leq m - n + 1\}$$

In the context of document classification, n-grams are particularly valuable. They enable the classification algorithm to recognize patterns that single words on their own may not convey. For instance, the sentiment of phrases like "highly recommend" or "not interested" can be directly captured through bi-grams, improving the model's ability to classify documents based on their content's sentiment and thematic elements. This specificity enhances the algorithm's predictive accuracy, making n-grams a go-to feature for document classification tasks.

However, employing n-grams also introduces challenges such as increased feature dimensionality and computational complexity. High-dimensional feature spaces can lead to the curse of dimensionality, adversely affecting model performance. To address this, techniques like feature selection, dimensionality reduction, and regularization are employed to maintain model efficiency and effectiveness. Moreover, advancements in word embeddings and deep learning offer complementary approaches to capture semantic relationships in a more compact form, providing a promising direction for integrating n-grams into sophisticated text analysis models (Bengio et al., 2000; Manning, Raghavan, & Schütze, 2008).

In summary, n-grams are a crucial component in the text analysis toolkit, especially for document classification. Their ability to capture local context and linguistic structures enriches the feature space, enabling models to achieve higher accuracy in classifying documents. Despite the challenges associated with their use, ongoing research and method-

ological innovations continue to solidify the role of n-grams in extracting meaningful insights from text data.

## 1.11 Feature Selection (Chi-Squared Test)

Feature selection is an essential process in machine learning that involves identifying the most relevant features for use in model training. This step is crucial for improving model performance, reducing complexity, and enhancing interpretability. Among various feature selection techniques, the Chi-Squared ($X^2$) test stands out as a popular method for categorical data, including text. In the domain of document classification, the Chi-Squared test is employed to evaluate the independence of features (e.g., words, terms) with respect to the classes (document categories), thereby selecting those features that have the most significant relationships with the class labels (Yang & Pedersen, 1997; Forman, 2003). The Chi-Squared Test for feature selection can be described as follows:

Given:

- A dataset with $m$ categorical features $F_1, F_2, ..., F_m$, and a categorical target variable $T$.
- Each feature $F_i$ has $R_i$ distinct categories, and the target variable $T$ has $C$ distinct classes.

The Chi-Squared statistic for a feature $F_i$ is calculated by comparing the observed frequencies of each category within $F_i$ against the expected frequencies if $F_i$ and $T$ were independent. The Chi-Squared statistic is defined as:

$$\chi^2 = \sum_{r=1}^{R_i} \sum_{c=1}^{C} \frac{(O_{rc} - E_{rc})^2}{E_{rc}}$$

where:

- $O_{rc}$ is the observed frequency of category $r$ of feature $F_i$ and class $c$ of $T$.

- $E_{rc}$ is the expected frequency of category $r$ and class $c$, calculated under the assumption of independence between $F_i$ and $T$.

The higher the $\chi^2$ value, the more evidence there is against the null hypothesis of independence between $F_i$ and $T$, suggesting that $F_i$ is a relevant feature for predicting $T$.

The essence of the Chi-Squared test in text analysis lies in its ability to measure how expectations compare to actual observed data. Specifically, it calculates the difference between the observed frequency of a feature in a particular class and the frequency expected if that feature were independent of the class. A high Chi-Squared value for a feature indicates that its occurrence is highly dependent on the class, suggesting that the feature is important for classification purposes. This statistical measure effectively highlights terms that are disproportionately frequent in certain categories but not others, making them valuable for distinguishing between document classes (Manning, Raghavan, & Schütze, 2008).

Applying the Chi-Squared test in document classification involves computing the statistic for each term across all categories. The terms are then ranked based on their Chi-Squared scores, and a selection is made to include only the top-scoring terms as features in the model. This process not only reduces the dimensionality of the feature space, mitigating issues like overfitting and computational inefficiency, but also enhances model accuracy by focusing on the most informative aspects of the text (Forman, 2003).

However, it's essential to note that the effectiveness of the Chi-Squared test can be influenced by the size of the dataset and the distribution of the categories. For instance, terms occurring very infrequently may lead to unreliable Chi-Squared scores due to small expected frequencies, potentially skewing feature selection. Thus, careful preprocessing and consideration of term frequency thresholds are advisable to ensure robust feature selection outcomes (Yang & Pedersen, 1997).

In conclusion, the Chi-Squared test serves as a powerful tool for feature selection in

document classification, enabling the identification of terms that significantly contribute to category differentiation. By leveraging statistical measures to focus on the most relevant features, document classification models can achieve higher accuracy and efficiency. Ongoing research continues to refine these processes, exploring more sophisticated statistical methods and their integration with machine learning algorithms to further enhance text classification performance.

## 1.12 Model Ensembling (Bagging)

Model ensembling is a powerful technique in machine learning that combines multiple models to improve overall performance, robustness, and accuracy. Among various ensembling methods, bagging, or Bootstrap Aggregating, is particularly notable for its effectiveness in reducing variance and preventing overfitting. Bagging involves training multiple models on different subsets of the training data, then aggregating their predictions to form a final decision. This approach is especially beneficial in document classification tasks, where the diversity of text data and the complexity of language patterns can make models prone to overfitting on training data (Breiman, 1996).

The core principle of bagging lies in creating multiple subsets of the original training dataset through bootstrap sampling; that is, randomly selecting samples with replacement. Each model in the ensemble is then trained independently on these subsets. In the context of document classification, these models might be decision trees, support vector machines, or any other suitable classifier. The final classification decision for a given document is made by aggregating the predictions from all models, typically through majority voting or averaging. This process enhances the ensemble's ability to generalize, as it mitigates the impact of noise and outliers in the training data (Breiman, 1996; Dietterich, 2000). As a result, the principle and methodology of Bagging can be described as follows:

Given:

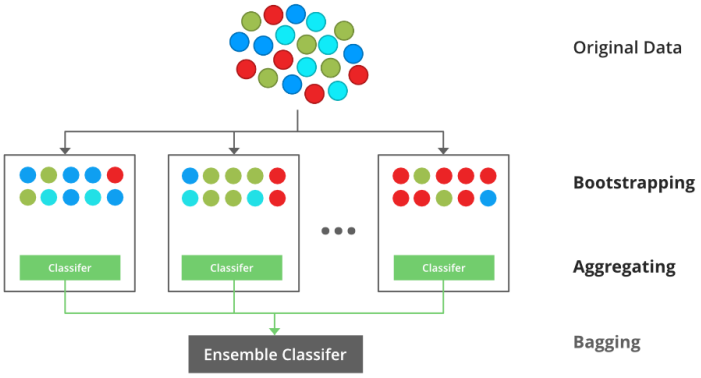



Figure 1.8: Bagging

- A training dataset $D$ with $n$ instances.
- A base learning algorithm $\mathcal{A}$.

The Bagging procedure involves the following steps:

1. For $i = 1$ to $B$ (where $B$ is the number of models in the ensemble):
   a) Create a bootstrap sample $D_i$ of size $n$ by uniformly sampling instances from $D$ with replacement.
   b) Train a model $M_i$ using $\mathcal{A}$ on $D_i$.
2. Aggregate the predictions of all models $\{M_i\}_{i=1}^{B}$ to make the final prediction. For classification, this is often done by majority voting. For regression, averaging the predictions is a common approach:
   - Classification: $\hat{y} = \text{mode}\{M_i(x)|i = 1, \ldots, B\}$
   - Regression: $\hat{y} = \frac{1}{B}\sum_{i=1}^{B} M_i(x)$

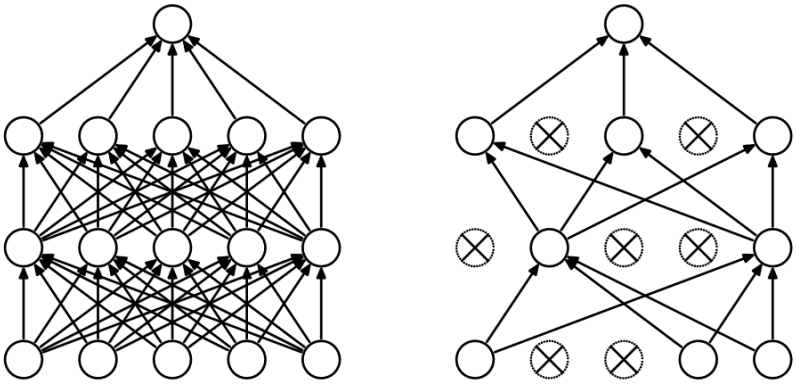
One of the key strengths of bagging is its simplicity and ease of implementation. Moreover, because models are trained independently, bagging can be efficiently paral-

lelized, making it well-suited for handling large datasets common in text analysis tasks. Additionally, bagging is particularly effective with base learners that have high variance and low bias, such as decision trees. By aggregating the predictions of several such models, bagging effectively reduces the variance without significantly increasing bias, leading to more accurate and stable classification performance (Breiman, 1996).

In document classification, where the goal is to categorize documents into predefined classes based on their content, bagging can significantly enhance the robustness of the classification model. For example, variations in language use, writing styles, and document lengths can introduce variability that single models may not handle well. By employing bagging, the ensemble model can better capture and generalize across these variations, leading to improved classification accuracy across diverse document sets (Opitz & Maclin, 1999).

Despite its advantages, bagging is not without limitations. The technique can increase computational complexity due to the need to train multiple models. Furthermore, if the base learners are too complex or the data is not diverse enough, bagging may not lead to significant improvements over single models. Therefore, careful consideration of the base learners and the characteristics of the dataset is crucial when applying bagging to document classification tasks.

In conclusion, bagging stands as a cornerstone of model ensembling strategies in machine learning, offering a straightforward yet effective approach to enhancing document classification models. Through its capacity to reduce overfitting and improve generalization, bagging enables the development of more accurate, robust, and reliable text classification systems. As research in machine learning progresses, the exploration of advanced bagging techniques and their integration with other ensembling methods continues to offer promising avenues for further improving document classification performance.



(a) Standard Neural Net (b) After applying dropout.

Figure 1.9: Dropout

## 1.13 Regularization Techniques (Dropout)

Regularization techniques are critical in machine learning to prevent overfitting, ensuring that models generalize well to unseen data. Among these techniques, dropout has emerged as a particularly effective method, especially in the context of neural networks applied to tasks such as document classification. Originally introduced by Srivastava et al. (2014), dropout addresses overfitting by randomly "dropping out" a subset of units (neurons) in a neural network during training, effectively thinning the network temporarily. This process prevents units from co-adapting too closely to the training data, encouraging the model to learn more robust features that generalize better to new data.

The core idea behind dropout is to introduce randomness into the training process, simulating a form of model averaging by creating numerous different "thinned" versions of the network. Each forward pass during training uses a different subset of units, while at test time, the entire network is used, but the units' outputs are scaled down appropriately to account for the reduced number of active units during training. This approach can be likened to training a collection of models that share weights, where each model sees only

a partial view of the data. As a result, the ensemble effect of these models helps reduce overfitting, improving the model's ability to perform well on unseen data (Srivastava et al., 2014). The dropout process can be described as follows:

Given:

- A neural network with $L$ layers, each layer $l$ containing $N_l$ neurons.
- A dropout rate $p$, which is the probability of any neuron being dropped in a given training step.

During training, for each forward pass:

1. For each layer $l$, independently set each neuron's output to zero with probability $p$, effectively removing the neuron from the layer for that pass.
2. Forward propagate the input through the modified network to compute the loss.
3. Backpropagate the loss to update the weights of the neurons that were not dropped.

Mathematically, for a layer $l$ with input vector $\mathbf{x}$ and weight matrix $\mathbf{W}$, the output $\mathbf{y}$ without dropout is $\mathbf{y} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$, where $f$ is a non-linear activation function, and $\mathbf{b}$ is the bias vector. With dropout applied, the output becomes:

$$\mathbf{y} = f((\mathbf{W} \odot \mathbf{M})\mathbf{x} + \mathbf{b})$$

where $\odot$ denotes element-wise multiplication, and $\mathbf{M}$ is a mask vector of the same dimension as $\mathbf{x}$, with each element independently drawn from a Bernoulli distribution with parameter $1 - p$.

In the realm of document classification, where models are tasked with categorizing text documents into predefined classes, dropout proves invaluable. Text data is inherently high-dimensional and sparse, making models particularly prone to memorizing training data rather than learning generalizable patterns. By applying dropout to neural network layers

that process textual features, such as those obtained from embeddings or convolutional operations, researchers can significantly enhance the model's robustness and prevent over-reliance on specific parts of the data. This ensures that the classification model captures a broad and generalizable understanding of language patterns, rather than memorizing training examples (Goodfellow, Bengio, & Courville, 2016).

However, the effectiveness of dropout is not without its considerations. The dropout rate—the probability of dropping a unit—must be carefully chosen. A rate too high may lead to underfitting, where the network fails to learn relevant patterns in the data, while a rate too low may not sufficiently regularize the model. Empirical studies often find dropout rates of 0.5 for hidden layers and slightly lower for input layers to be effective, but these values can vary depending on the specific task and network architecture (Srivastava et al., 2014).

Moreover, while dropout is predominantly used in neural networks, its underlying principle of introducing randomness and preventing co-adaptation of features can inspire regularization techniques in other model types. As document classification models evolve, incorporating complex neural architectures and vast datasets, regularization techniques like dropout remain essential for achieving high performance and generalization in text analysis tasks.

In conclusion, dropout stands as a powerful regularization technique that enhances the generalizability of document classification models. By mitigating overfitting through strategic deactivation of network units during training, dropout ensures that neural networks can learn robust and diverse representations of text data. This regularization approach, coupled with its simplicity and effectiveness, makes dropout a staple in the development of advanced neural network models for text classification.

## 1.14 Precision, Recall, F1-Score, and Accuracy

In the evaluation of classification models, several metrics are crucial for understanding the model's performance in various aspects. Precision, Recall, F1-Score, and Accuracy are among the most widely used metrics. These metrics are defined as follows:

Given:

- True Positives ($TP$): The number of positive instances correctly classified.
- False Positives ($FP$): The number of negative instances incorrectly classified as positive.
- True Negatives ($TN$): The number of negative instances correctly classified.
- False Negatives ($FN$): The number of positive instances incorrectly classified as negative.

**Precision** (or Positive Predictive Value) measures the accuracy of positive predictions. It is defined as the ratio of true positive predictions to the total positive predictions made:

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

This metric is pivotal in document classification tasks where the accuracy of identifying relevant documents is prioritized over the model's ability to identify all relevant documents (Davis & Goadrich, 2006; Sokolova & Lapalme, 2009).

**Recall** (or Sensitivity, True Positive Rate) measures the ability of the model to identify all relevant instances. It is defined as the ratio of true positive predictions to the actual number of positive instances:

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

Recall becomes a critical evaluation metric in contexts where missing a relevant document (a false negative) has severe consequences, emphasizing the model's capacity to capture all pertinent information (Powers, 2020; Sokolova & Lapalme, 2009).

**F1-Score** is the harmonic mean of Precision and Recall, providing a single metric that balances both the Precision and Recall, especially useful when there is an uneven class distribution:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

It is particularly useful when the cost of false positives and false negatives is unknown or when classes are imbalanced (Van Rijsbergen, 1979; Powers, 2020).

**Accuracy** measures the overall correctness of the model and is defined as the ratio of correctly predicted instances to the total instances in the dataset:

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

While accuracy is straightforward and universally applicable, its utility can be limited in scenarios with highly imbalanced class distributions, as it may not reflect the performance of the model on the minority class effectively (Japkowicz & Stephen, 2002; Sokolova & Lapalme, 2009).

In document classification, these metrics serve as the cornerstone for evaluating and comparing the performance of various models, guiding the selection of the most appropriate algorithms for specific tasks. Precision, Recall, and F1-Score are particularly crucial in applications where the balance of false positives and false negatives has significant implications, while Accuracy offers a quick snapshot of overall performance. Together, these metrics facilitate a nuanced understanding of model effectiveness, informing the development and refinement of classification systems.