University Of Geneva

Machine Learning

Stackoverflow Tag Classification

Professor:

Sebastian Engelke

Teacher Assistants:

Dongmei Liu

Manuel Hentschel

Christophe Valvason

Authors:

Ahmad Hamad

David Camela

Bogdan Secas

Contents

1 Introduction

This project focuses on Natural Language Processing (NLP), a specialized domain within Machine Learning dedicated to comprehending, forecasting, and processing words, sentences, and the intricacies of ”natural” language.

Our project aims to enhance the user experience on Stack Overflow, a programmer's invaluable resource, by leveraging machine learning techniques to predict missing tags in posts. Stack Overflow has initiated the analysis of internal data spanning the past 15 years. Unfortunately, some posts lack crucial data—specifically, the tag or subject of the post is absent. In response, we, as machine learning specialists, have been tasked with constructing highly accurate ML models for predicting the missing tags of these posts. To achieve this, Stack Overflow has provided a data set comprising posts and their associated tags. As ML specialists, our primary challenge involves effectively processing these text data using natural language processing (NLP) methodologies.

The project centers around the development of machine learning models capable of predicting the diverse tags (categories) assigned to given posts. These posts, contributed by Stack Overflow users, predominantly cover programming languages and comprise a blend of natural language and code snippets in various programming languages.

The goal of the ML model is to be as accurate as possible in predicting the labels. The accuracy is defined as follows:

$$Accuracy = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Challenges in Natural Language Processing (NLP)

Natural Language Processing (NLP) encounters numerous challenges due to the inherent complexities of human language. The following key challenges are commonly faced in the field:

1) **Ambiguity and Polysemy:** Natural language is rich with words and phrases that have multiple meanings based on context. Ambiguity arises when a single word or phrase can be interpreted in different ways. Polysemy, the coexistence of multiple meanings, further complicates interpretation (7; 5).

2) **Syntax and Grammar Variations:** Natural languages exhibit diverse syntactic structures and grammatical variations, making it challenging to develop models that generalize across different language patterns (7; 5).

3) **Contextual Understanding:** Effective language comprehension requires an understanding of context, as the meaning of words and phrases often relies on the surrounding text. Capturing context accurately is challenging, particularly in situations where meaning evolves with changing contexts (7; 5).

4) **Named Entity Recognition (NER):** Identifying and classifying named entities, such as names of people, organizations, and locations, is intricate due to the wide array of possible entities and the constant evolution of new terms, especially in fields like programming languages (8).

5) **Lack of Standardization:** Natural languages lack uniform standards, leading to variations in spelling, grammar, and vocabulary usage. This lack of standardization complicates model training and hinders generalization (7; 5).

6) **Data Sparsity:** Obtaining labeled datasets for training NLP models is often challenging and expensive. Additionally, certain language phenomena are rare, resulting in sparse data for specific linguistic patterns (7; 5).

7) **Inconsistent Sizes of Input:** Texts in the corpus are almost never the same in size, posing a challenge compared to other machine learning tasks like image processing (7; 5).

Description of the data-set

The corpus is made of 28000 examples of human written posts on the platform Stack Overflow with their associated tags. Our analysis shows that there are 20 different tags, all related to programming languages or IT.

The tags found are the following: 'c#', 'asp.net', 'objective-c', '.net', 'python', 'angularjs', 'iphone', 'ruby-on-rails', 'ios', 'c', 'sql', 'jquery', 'css', 'java', 'android', 'mysql', 'c++', 'javascript', 'html', 'php'.

The average length of the posts is : 1093 characters.

2 Data Exploration and Preliminary Analysis

Upon importing the dataset, it was observed to contain three columns: 'Id' serving as a unique user identifier, 'post' representing the questions posted by users, and 'tags,' which are the labels intended for prediction in new, unseen data. The dataset has a shape of 28,000 rows and 3 columns.

An initial inspection using the `.info()` method revealed that two columns, 'post' and 'tags,' are of object `data type` .

2.1 Response Variable Distribution

To gain a better understanding of the dataset, an essential step is to examine the distribution of the response variable ('tags'). There are a total of 20 unique labels in the dataset. Visualization tools such as Seaborn and Matplotlib.pyplot were employed to facilitate this exploration.

A count plot was employed to illustrate the distribution of labels, given the non-numerical nature of the variable. The plot revealed an equal distribution of labels, with each label appearing approximately 1,400 times in the dataset. This balanced distribution signifies that the dataset contains a similar number of instances for each label.

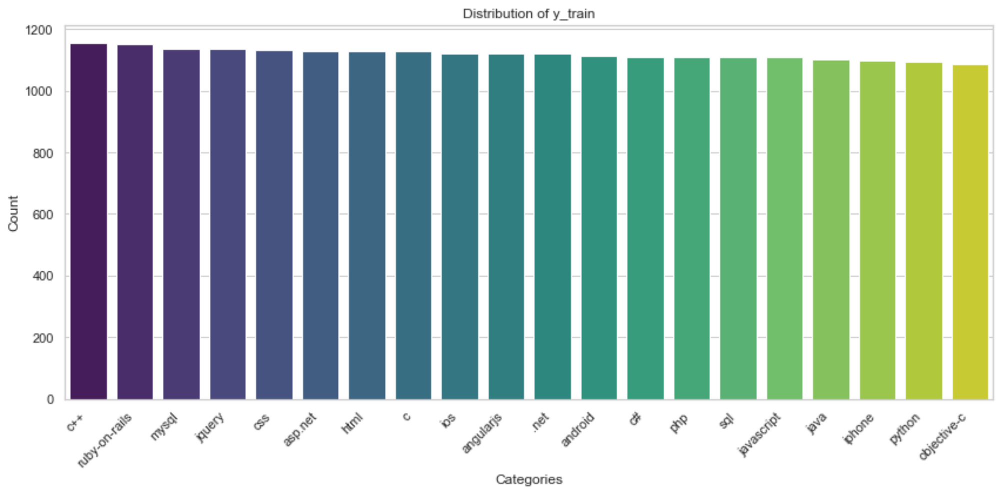


Figure 1: Count plot distribution of the response variable

The balanced distribution of labels is advantageous as it ensures that the machine learning model has exposure to an adequate number of instances for each category during training. This balance can enhance the model's ability to generalize well across different labels, contributing to a more robust and reliable predictive performance. Additionally, a balanced distribution minimizes the risk of the model being biased towards specific labels, promoting fair and unbiased predictions across the entire spectrum of labels in the dataset.

2.2 Handling Duplicates

A crucial step involves checking for and eliminating any duplicate entries in the dataset. Duplicates can introduce biases and distort the model's learning process, impacting the overall performance.

Eliminating duplicates ensures that each unique instance contributes distinct information to the model, promoting robustness and unbiased predictions. In this dataset, no duplicates were found, indicating that each record is unique, and there is no need for duplicate elimination.

2.3 Handling Missing Values

Although missing values are typically associated with numerical data, it is prudent to check for missing values even in columns with object data types. In the context of text data, missing values might manifest as empty strings or other placeholders. To address this, a function was implemented to identify and handle any missing values using the `dropna()` method.
Fortunately, no missing values were detected in the dataset. This not only streamlines the preprocessing steps but also presents an advantage in terms of data completeness. The absence of missing values ensures that the dataset is robust and ready for subsequent text processing and machine learning model development.

2.4 Post Content Analysis

To gain insights into the characteristics of the textual content within the 'post' column, a distribution of word counts for each post was generated, and the lengths of the posts were visualized using histogram plots.

The histogram plot depicting word counts per post reveals that the majority of posts are succinct, containing fewer than 200 words. This observation suggests that the content within our dataset tends to be concise and to the point, possibly reflecting the nature of Stack Overflow questions where users strive to articulate their queries succinctly.

Expanding on this analysis, the distribution of text lengths, measured in characters, was also examined. The histogram highlights that a significant portion of posts has a length of less than 1000 characters. This finding aligns seamlessly with the earlier observed word count distribution, reinforcing the notion that the majority of posts in the dataset are relatively short.

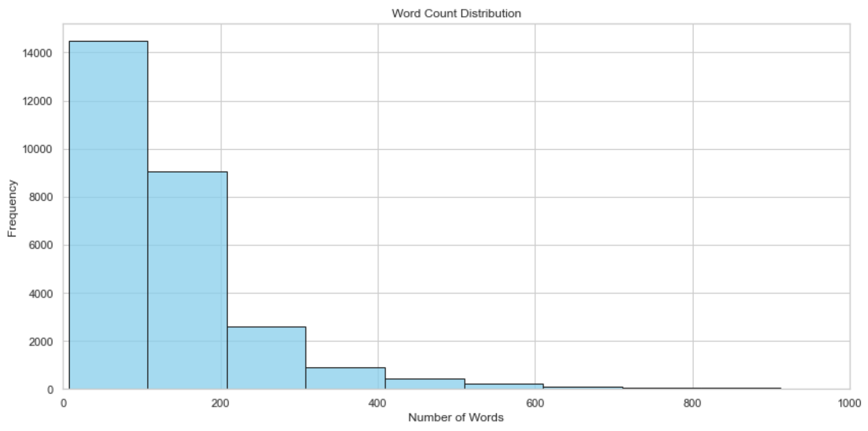


Figure 2: Word Count Distribution in Posts

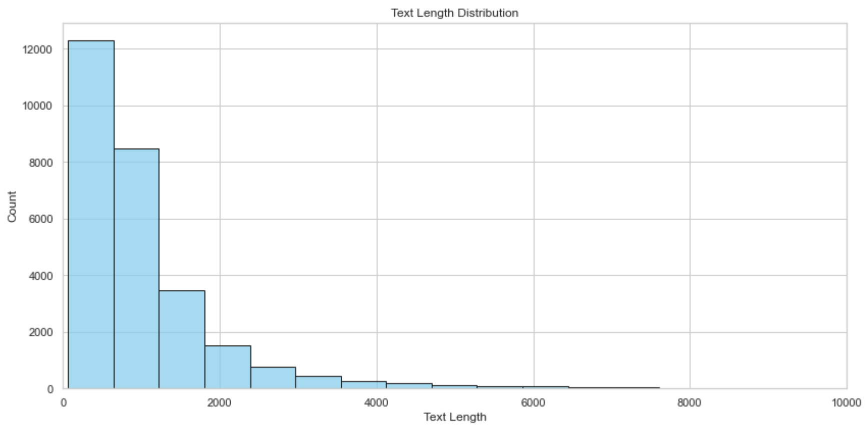


Figure 3: Post Length Distribution

Understanding the distribution of word counts and post lengths is pivotal as it provides valuable insights into the typical length and complexity of questions posted on Stack Overflow. This information could be instrumental in shaping the preprocessing steps for text data and may guide the selection of appropriate models for predicting tags based on the content of the posts.

3 Preprocessing

3.1 Text Cleaning: HTML Parsing

In the preprocessing phase, it is essential to clean the text data, especially when dealing with content extracted from HTML, such as Stack Overflow posts. HTML tags can introduce noise and interfere with the subsequent natural language processing steps. To address this, the BeautifulSoup library was employed to clean the HTML text.

The cleaning process involves decoding the HTML entities and removing any HTML tags. Two parsers, namely "lxml" and "html.parser," were considered for this task. The code uses a try-except block to use the "lxml" parser first, with a fallback to the "html.parser" in case of any issues.

Table 1: Comparison of lxml and html Parsers

Parser	**Pros**	**Cons**
lxml Parser	• Generally faster and more efficient. • Supports both HTML and XML parsing	• External dependency.
html Parser	• Comes bundled with Python. • More friendly with poorly formed HTML.	• Slower compared to "lxml."

This implementation ensures a robust HTML cleaning process, taking advantage of the speed of "lxml" while gracefully falling back to "html.parser" if needed. The choice of parser can be adjusted based on specific requirements and the nature of the HTML content encountered in Stack Overflow posts.

3.2 Text Preprocessing

Text preprocessing plays a pivotal role in shaping the input data for machine learning models. In this section, we describe the key steps taken to preprocess the Stack Overflow post data.

To prepare the text for further analysis, a series of preprocessing steps were applied using the `preprocess_text` function. This function performs the following tasks:

- **Lowercasing**: The entire text is converted to lowercase to ensure uniformity and prevent case-specific variations from affecting the model's performance.

- **Symbol Replacement**: Some symbols are replaced with spaces to separate words and facilitate subsequent tokenization.

- **Symbol Removal**: Any remaining symbols that are not alphanumeric, '+' or '_' are removed from the text.

- **Tokenization**: The text is tokenized using the NLTK library's word tokenizer.

- **Stopword Removal**: Common English stopwords, such as 'the', 'is', and 'and', were removed from the tokenized words to focus on meaningful content.

These preprocessing steps collectively contribute to creating a cleaner and more standardized textual representation of the Stack Overflow posts, facilitating subsequent feature extraction and model training processes.

Observation	Original Post	Processed Post
70	how to ask while to stop from another class there is the class: ¡pre¿¡code¿public class m16 public boolean on = true; void shoot () system.out.println(bang); void pulltrigger() while(on) shoot(); </code></pre> and i want to set my flag on after i call pulltrigger() method. like this: ¡pre¿¡code¿ m16 m = new m16(); m.pulltrigger(); thread.sleep(2000); m.on = false; <code><pre> it is simple question but i dont know what i need to do.	ask stop another class class public class public boolean true void shoot systemoutprintln bang void pulltrigger shoot want set flag call pulltrigger method like new mpulltrigger threadsleep mon false simple question dont know need

Table 2: Original vs. Processed Stack Overflow Posts

The table above contrasts the original posts with their processed counterparts, illustrating the transformative impact of the implemented preprocessing functions. Key Observations:

1. HTML Tag Removal (`clean_html`):
   - The entire HTML structure, including '<pre>', '<code>', and their contents, has been removed. This simplifies the text and eliminates code-related tags.
2. Text Preprocessing (`preprocess_text`):
   - All text has been converted to lowercase for uniformity.
   - Symbols and characters not contributing to the linguistic meaning, such as '() [ ] | @ , ;', have been removed.
   - Stopwords have been removed to focus on essential content.

   - All text has been converted to lowercase for uniformity.

The preprocessing steps have effectively transformed the original Stack Overflow post. The removal of HTML tags has streamlined the content, while text preprocessing has enhanced readability and focused on the core linguistic elements. This processed text is now more suitable for natural language processing tasks, such as machine learning-based tag prediction, without the noise introduced by HTML tags and unnecessary symbols.

Other preprocessing methods, including stemming and lemmatization, were explored to enhance the text data from Stack Overflow. Stemming involves reducing words to their root form by removing suffixes, while lemmatization considers the context and meaning of words to reduce them to their base or dictionary form. However, our experiments revealed that the performance of our models was more favorable without the inclusion of stemming and lemmatization.

The decision not to use stemming and lemmatization in our preprocessing pipeline is rooted in the unique characteristics of Stack Overflow posts. These posts often feature a substantial amount of technical jargon and domain-specific terms related to programming and software development. Stemming and lemmatization, designed for general-purpose language processing, might alter these terms, potentially leading to a loss of specificity and clarity in the representation of programming concepts.

3.3 Tokenization

Tokenization is a foundational process in Natural Language Processing (NLP) that plays a pivotal role in transforming raw text into a structured format. It involves the segmentation of text into individual units, known as tokens, which are typically words or phrases. This breakdown facilitates subsequent analysis and enables the extraction of meaningful information from the text data.(9)

In the context of the provided code, a custom tokenizer is employed for preprocessing. This custom tokenizer is designed to identify and preserve specific symbols that we believe are specific to some programming languages.

The selection of special words in the custom tokenizer is deliberate. These special words align with the labels present in the 'tags' column, capturing essential programming languages and technology concepts within the text. This tailored approach ensures that the tokenizer accounts for domain-specific terms, enhancing its ability to represent the content accurately.

3.4 Vectorization

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) is a feature extraction technique used to represent the importance of words in a document relative to a corpus. It assigns a weight to each term based on its frequency in the document (Term Frequency) and inversely proportional to its frequency in the entire corpus (Inverse Document Frequency)(10). The TF-IDF vectorizer, applied in the code, generates sparse matrices where each row corresponds to a document, and each column represents a unique term. The formula for TF-IDF is given by:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

where:

- $\text{TF}(t, d)$ is the term frequency of term t in document d.
- $\text{IDF}(t, D)$ is the inverse document frequency of term t in the entire document set D.

The term frequency $\text{TF}(t, d)$ is usually calculated as the number of times term t appears in document d divided by the total number of terms in d.

The inverse document frequency $\text{IDF}(t, D)$ is often computed as the logarithm of the ratio of the total number of documents in D to the number of documents containing term t.

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in } D}{\text{Number of documents containing term } t + 1}\right)$$

The addition of 1 in the denominator (smoothing) is to avoid division by zero.

Count Vectorizer

Count Vectorizer is another feature extraction method that converts a collection of text documents to a matrix of token counts. Each document is represented as a vector, and the elements of the vector correspond to the count of each term in the document. The resulting matrix, as seen in the code, captures the term frequency information. The mathematical formula for the count of a term t in a document d is straightforward:

$$\text{Count}(t, d) = \text{Number of times term } t \text{ appears in document } d$$

In our work, we observed distinct behaviors in the performance of Random Forest and Support Vector Classifier (SVC) when employing different feature extraction techniques. Notably, Random Forest demonstrated improved performance when combining TF-IDF and Count Vectorizer techniques. On the other hand, SVC, while benefiting from TF-IDF, exhibited a significant performance boost with an improvement of 6.29%, achieving an accuracy of 79.29% compared to the initial 73%. These findings underscore the impact of feature engineering choices on model performance, highlighting the need for careful consideration and experimentation in the selection of appropriate techniques for different classifiers Alternative methods, such as Bag of Words and embedding techniques like GloVe and Word2Vec was tried. Despite their potential in capturing semantic relationships, these methods exhibit lower performance in comparison to the combined TF-IDF and Count Vectorizer approach.

4 Clasification Models

4.1 Random Forest

Before getting deeper into the definition of Random Forest classifier, we must understand first the role of Decision Trees in Machine Learning. Decision trees are non-parametric supervised learning algorithms, which can be used to classify or do regression tasks. It has hierarchichal, tree structure, which consists of a root node, branches, internal nodes, and leaf nodes. As shown in the following diagram (3).

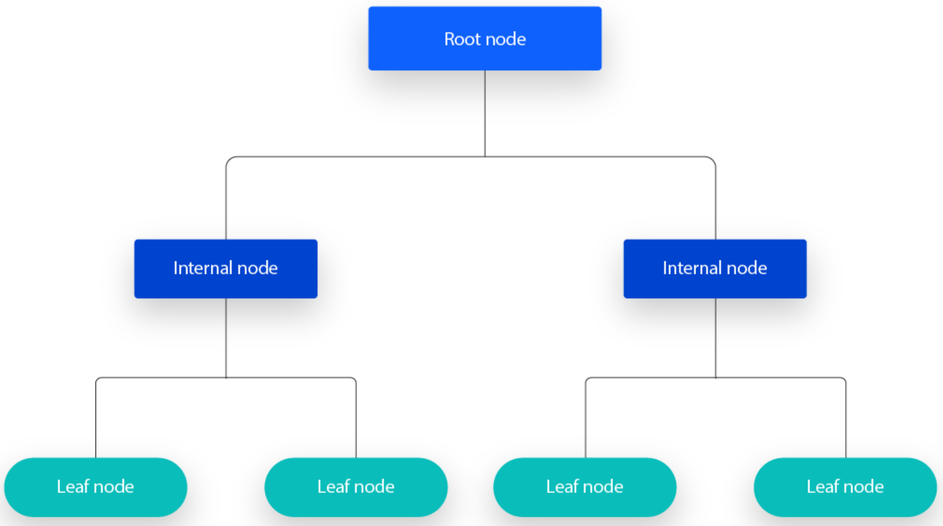


Figure 4: Decision Tree Structure. The outgoing branches from the root node feed into the internal nodes, also known as decision nodes (3).

Random Forest is an extended version of a decision tree that can predict future instances with different classifiers instead of using a single one to reach an appropriate accuracy and correctness

of the prediction (13).
Its ease of use and flexibility have made this algorithm one of the most used lately, as it handles classification and regression problems.
Random Forest is an extension of the bagging method since it uses bagging and feature randomness to develop an uncorrelated forest of decision trees, this is the reason for the name of the algorithm (4).
A key concept for the Random Forest algorithm is **the random subspace method**, which generates a random subset of features, which helps to ensure a low correlation among decision trees. This is key, since instead of using all the possible feature splits, random forest is able to select only a subset of these features.

How does it work?

The random forest algorithm is made up of a collection of decision trees, and each one of these trees is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Additional randomness is introduced through feature bagging, further enhancing the diversity within the dataset and diminishing the correlation among individual decision trees.

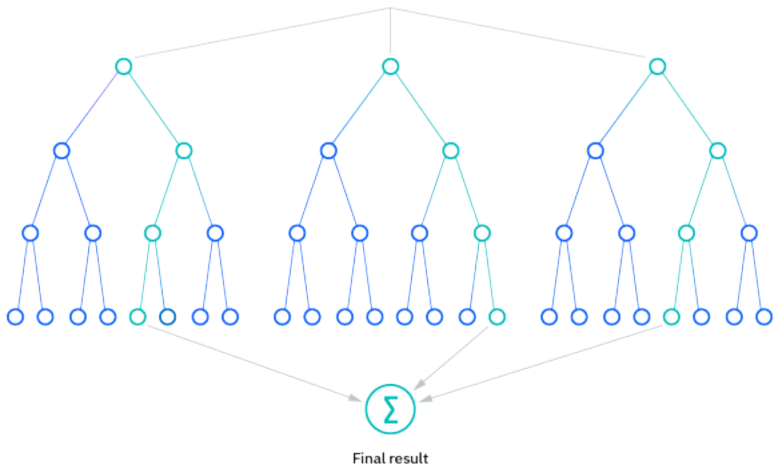


Figure 5: Random Forest Diagram. (4)

The method of making predictions varies based on the nature of the problem at hand. In a regression scenario, the prediction is derived by averaging the outcomes of individual decision trees. Conversely, in a classification task, the predicted class is determined through a majority vote, representing the most frequent categorical variable. Ultimately, the out-of-bag (oob) sample is used for cross-validation, ultimately solidifying the prediction.(4)

Mathematical Context

In Random Forest classification, the choice of the loss function is closely linked to the specific implementation and the objective of the classification problem.
For classification tasks, the commonly used metrics are *Gini* or *Entropy*. *Gini* and *Entropy* can be used to measure the impurity or disorder within each node when making decisions in decision trees.

$$\mathcal{L}_G(\mathcal{N}_m) = \sum_{k=1}^{k} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{1}$$

1: Loss function *Gini*

$$\mathcal{L}_E(\mathcal{N}_m) = \sum_{k=1}^{k} \hat{p}_{mk} \log \hat{p}_{mk} \tag{2}$$

2: Loss function *Entropy*

These metrics aim to minimize impurity within each node while partitioning the data, resulting in more homogeneous subsets. Lower values of Gini and Entropy indicate reduced impurity or greater homogeneity within nodes.
In practice, within Random Forests, the default criterion for classification tasks is *Gini*. However, users can specify *Entropy* as an alternative. For regression tasks in Random Forests, the loss function is the Mean Squared Error (MSE), which is used to measure the variance between predicted and actual values in regression problems (2).

4.2 Support Vector Machine

Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or non-linear classification, regression, and even outlier detection tasks. Extensions of the methods have been developed by a vast research community, and include methods for regression, clustering, factor analysis, based on the same design principles. (6).
Support vector machine is one of the classical machine learning technologies that can handle to solve big data classification problems. However, the SVM is mathematically complex and computationally expensive.
Based on the nature of the decision boundary. SVM can be divided into two main parts.

- Linear SVM
- Non-Linear SVM

The main objective of SVM is to find the optimal hyperplane in an N-dimensional space, that can separate the data points in different classes in the feature space. The purpose of this hyperplane is to find the maximum margin between the closest points of different classes. The dimension of this hyperplane depends mainly on the number of features. If the number of input features is three, then the hyperplane becomes a 2-D plane, and so on. The plane becomes difficult to imagine when the number of features exceeds three.

How does it work?

The algorithm works by mapping input data points into a higher-dimensional feature space using a kernel function, allowing for nonlinear classification when the data isn't linearly separable in its original space. In this higher-dimensional space, SVM finds the hyperplane that maximizes the margin, thus creating the boundary between different classes.

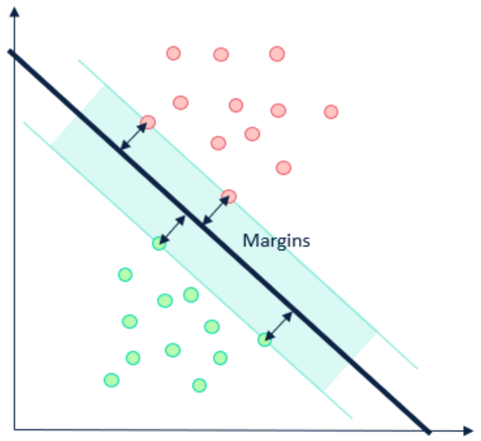


Figure 6: Graphical representation of the distance between the hyperplane and the support vectors are called the margins.(1)

Mathematical Context

In the case of Support Vector Machines (SVMs) when the decision boundary is not linear, the kernel trick is employed. The kernel trick allows SVMs to learn a nonlinear decision boundary by implicitly mapping the input data into a higher-dimensional space where a linear separation might

be possible.(Scikit-Learn)
The general form of the decision function in a nonlinear SVM using the kernel trick is:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b \tag{3}$$

Where:

- $f(x)$ is the decision function for predicting the class of a new sample x
- α_i are the learned coefficients or Lagrange multipliers associated with support vectors
- y_i are the class labels (-1 or 1 in binary classification) of the support vectors
- $K(x, x_i)$ is the kernel function that computes the similarity between input samples x and x_i in the higher-dimensional space

The equation 3 uses a *Kernel function* $K(x, x_i)$ which makes it nonlinear.(11)
The *Kernel function* can be any of the following:

- **Linear**: $K(x, x_i) = x^T x_i$
- **Polynomial**: $K(x, x_i) = (x^T x_i + r)^d$, where d is the degree of the polynomial and r an independent term.
- **Gaussian Radial Basis Function (RBF) Kernel**: $K(x, x_i) = \exp\left(-\frac{\|x-x_i\|^2}{2\sigma^2}\right)$, where σ is the bandwidth parameter.
- **Sigmoid**: $K(x, x_i) = \tanh(\alpha x^T x_i + c)$, where α and c are parameters.

The usage of a support vector machine for classification involves setting parameters and optimizing objectives. These objectives are primarily contingent on the arrangement of classes within the data space. Specifically, classes might exhibit linear separability or non-separability. Notably, while some classes might appear linearly separable, they can still be non-separable in a nonlinear context. Consequently, when defining parameters and optimizing objectives related to the data space, it's crucial to account for these diverse class properties. (11)

5 Comparison of Methods

This section aims to do a comparative analysis of Random Forest (RF) and Support Vector Machine (SVM) classifiers, emphasizing their performance in terms of accuracy, and scalability. Additionally, the optimization of hyperparameters within each model will be explored, leading to the ultimate selection of the most suitable classifier for the specific NLP project at hand.

5.1 Parameters

In Support Vector Machines (SVM) we tried to gridsearch 2 parameters: C and Kernel.

- The **C parameter** regulates the trade-off between accurate classification of training data and the simplicity of the decision boundary: A smaller C value allows more misclassifications, leading to a simpler decision boundary. A larger C value penalizes misclassifications, resulting in a more complex decision boundary that might fit the training data more accurately.

- **Kernel function** (see Mathematical Context of section 4.2) for this parameter the grid had 2 options: 'rbf' and 'linear'.

For Random Forest classifier:

- **n_nestimators**: This parameter defines the number of decision trees in the forest. A smaller value may result in underfitting as the model might not capture the complexity of the data. A larger value can lead to overfitting on the training data.

- **max_features:** It determines the maximum number of features considered for splitting a node. `'log2'`: uses the base-2 logarithm of the total number of features. `'sqrt':` uses the square root of the total number of features.

In summary, adjusting these parameters involves a trade-off: - Choosing a small $n_{\text{estimators}}$ or limiting max_features promotes simpler trees, reducing overfitting. - Larger values allow the model to capture more complex relationships in the data, potentially leading to overfitting if not carefully tuned.

5.2 Accuracy

For the accuracy comparison, we fitted the models using their default parameters setting up the data as follows:

- **Size of train Data:** 80%

- **Size of test Data:** 20%

- **TF-IDF vectorizer:** with `max_features = 15000`

Considering this, we fitted the models (default parameters) and got the following results.

Table 3: Accuracy scores comparison between different Classifiers

Classifier	**Accuracy**
Random Forest	80.20%
SVM RBF	78.95%
SVM Linear	78.63%

These accuracy metrics serve us as a **first-approach** indicator of how effectively the models can classify and predict the tags for the posts. The observed 80.20% accuracy with Random Forest suggests that this model, known for its effectiveness with various datasets, exhibits a higher predictive accuracy in this scenario. on the other hand, the SVM model, which relies on finding optimal hyperplanes, achieved a slightly lower accuracy of 80.20%. In the context of NLP classification, the Random Forest model might have performed better accuracy due to its ability to handle the inherent complexity of the textual data, capture nonlinear relationships more effectively, and mitigate overfitting. However, the SVM model, while getting a slightly lower accuracy, might improve in scenarios where feature representations are well-defined and linearly separable, given appropriate parameter tuning and data preprocessing.

5.3 Scalability

We can also compare the models with a scalability perspective. With this, we aim to evaluate how both models perform across varying dataset sizes. In this evaluation, we subject them to identical training data samples (80%) and measure the fitting time required by each model.

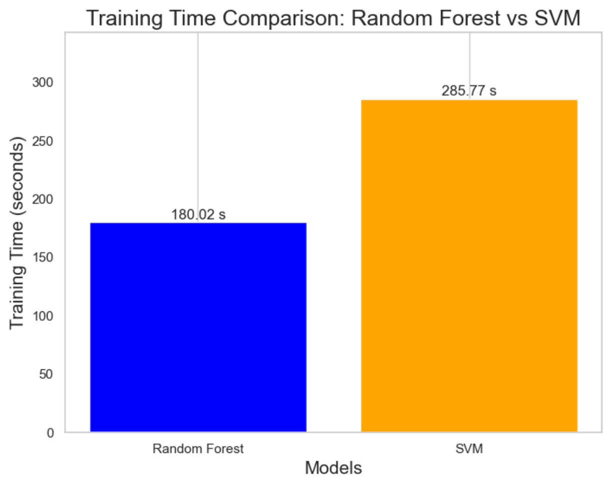
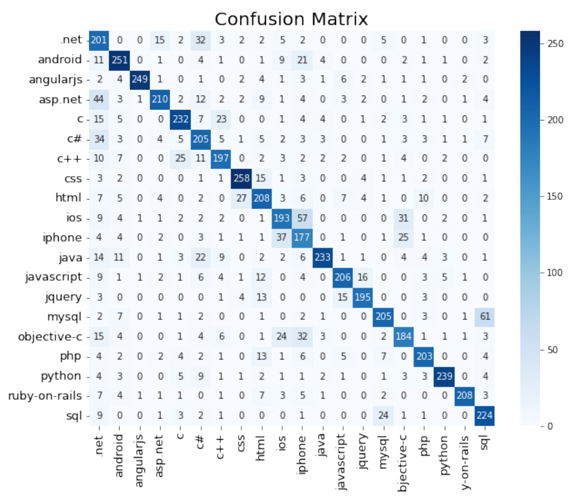


Figure 7: Time training comparison between the two classifiers

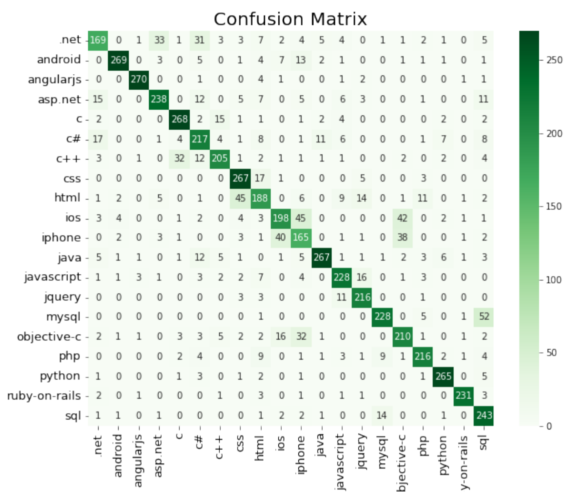
The figure 7 show us that the Random Forest algorithm demonstrated a shorter training period, with a time of 180.02 seconds, while SVM required 285.77 seconds for training.This means that for this specific case, SVM is approximately 58% slower.
These differences show us the fundamental computational contrasts inherent in these two algorithms. Random Forest, as an ensemble technique based on decision trees, tends to train easily due to its capacity for parallel processing and the streamlined construction of individual trees. In contrast, SVM, focuses more on optimizing hyperplanes in high-dimensional spaces, and typically involves more complex computation, resulting in longer training durations.
Last but not least, the observed differences in training times underscore the trade-offs between computational speed and model performance. This highlights the necessity for a balanced consideration of diverse factors when choosing between Random Forest and SVM for a given machine learning task.

5.4 Confusion Matrix

This analysis aims to present and compare the confusion matrices derived from the RF and SVM models. The confusion matrices provide insights into the models' predictive performance, indicating the true positives, false positives, true negatives, and false negatives for each tag category. By examining these matrices, we can understand which classes tend to be confused by the models. So it helps us understand the strengths and potential limitations of each model in accurately predicting specific tags within the Stack Overflow posts.



(a) SVM Classifier



(b) Random Forest Classifier

Figure 8: Confusion Matrix of classifiers

Figure 8 provides us useful insights into the performance differences in the models. If we look at 8(b) it shows us consistently higher values across various tags and overall accuracy metrics compared to the SVM, which suggests that RF might generally perform better for this specific problem of predicting tags from posts on Stack Overflow.
The lower counts for specific tags in the RF confusion matrix could imply that the RF model is more conservative or cautious when predicting these tags. In summary, while RF shows better performance, noticing differences in tag prediction counts between RF and SVM, especially for certain tags like ".net" and "HTML," highlights the different behaviors of these models.

5.5 Fine-Tuning the model

Random Forest Classifier

In this section we make use of the function `GridSearchCV` and `KFolds` from `sklearn` in order to fine-tune the hyperparameters of the model. Initially, we investigated the impact of two crucial parameters: the number of estimators (i.e., the number of decision trees) and the function that determines the maximum number of features (`max_features`) considered when searching for the optimal split.

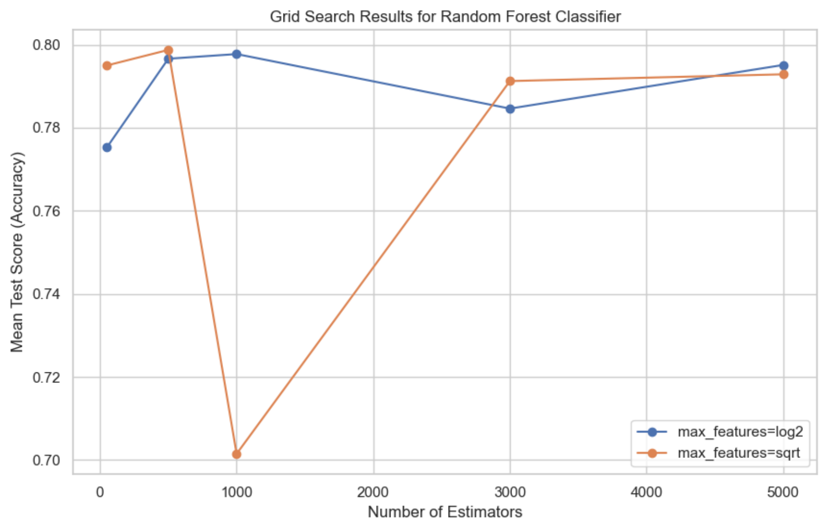


Figure 9: Accuracy trend for the different number of estimators in RF comparing two `max_features` parameters.

The accuracy trend plot compared different numbers of estimators in Random Forest models with `max_features` parameters set as `sqrt` and `log2`. The parameters `max_features = "sqrt"` and `n_estimators = 3000` were considered the best ones for the model performance.

Although a higher accuracy score was observed at `n_estimators = 500` for `max_features = "sqrt"`, the decision to stick with `n_estimators = 3000` was due to its stability and consistency across different data subsets. While the outlier score at `n_estimators = 500` seemed momentarily higher, the choice for `n_estimators = 3000` prioritized a more robust and generalized model performance, aiming for better overall predictions.

SVM Classifier

In this fine-tuning exploration we focused on analyzing the impact of the regularization parameter `C` on the performance of an SVM model employing an RBF Kernel.
We implemented the following `C` values on the grid search: `C = [1,3,5,7,10]`. The objective was to evaluate the influence of this parameter in the accuracy of SVM in classification tasks. The figure 10 shows us the relationship between the fine-tuned regularization parameter and the model performance.

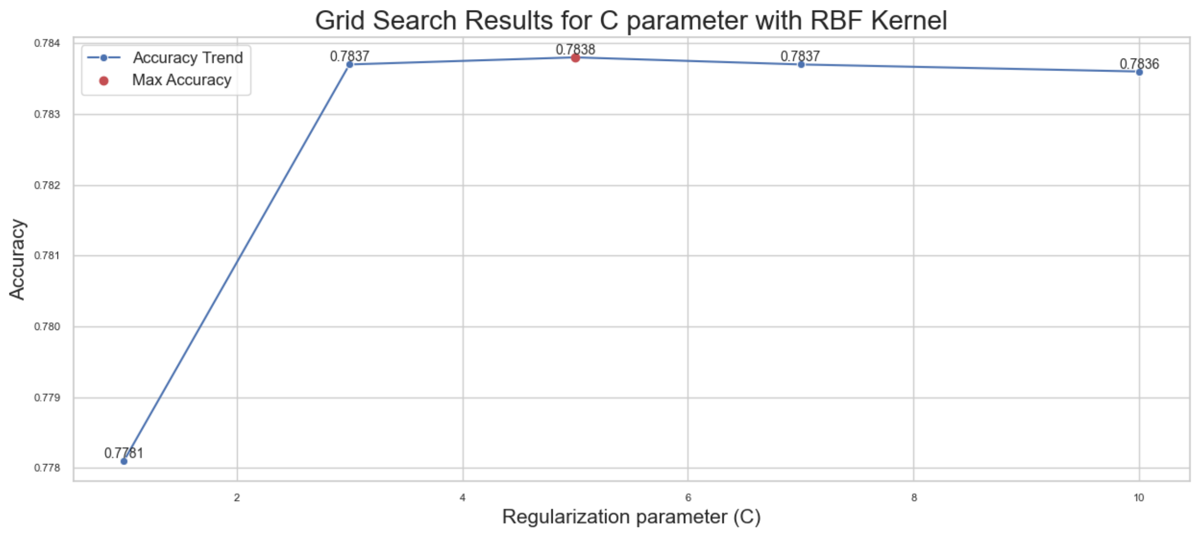


Figure 10: Accuracy trend for C parameter in SVM using RBF Kernel

Viewing the results we can notice a consistent trend of accuracy stability. The accuracy remains relatively steady within a close range, fluctuating between approximately 77.81% and 78.38%. This indicates that the choice of the regularization parameter C in the given range doesn't significantly impact the model's accuracy. Despite altering the parameter C, the model's performance, as measured by accuracy, remains relatively consistent, showcasing robustness to changes in this specific hyperparameter within the evaluated range. Specifically for this dataset, setting the regularization parameter to `C = 5` results in achieving a mean accuracy score of 78.37%.
On the other hand, we also tried a linear kernel. Similar to our previous examination with the RBF Kernel, we fine-tuned the regularization parameter `C` over a range of values that were inferior to the ones we used in our previous analysis. The results can be seen in the following figure.

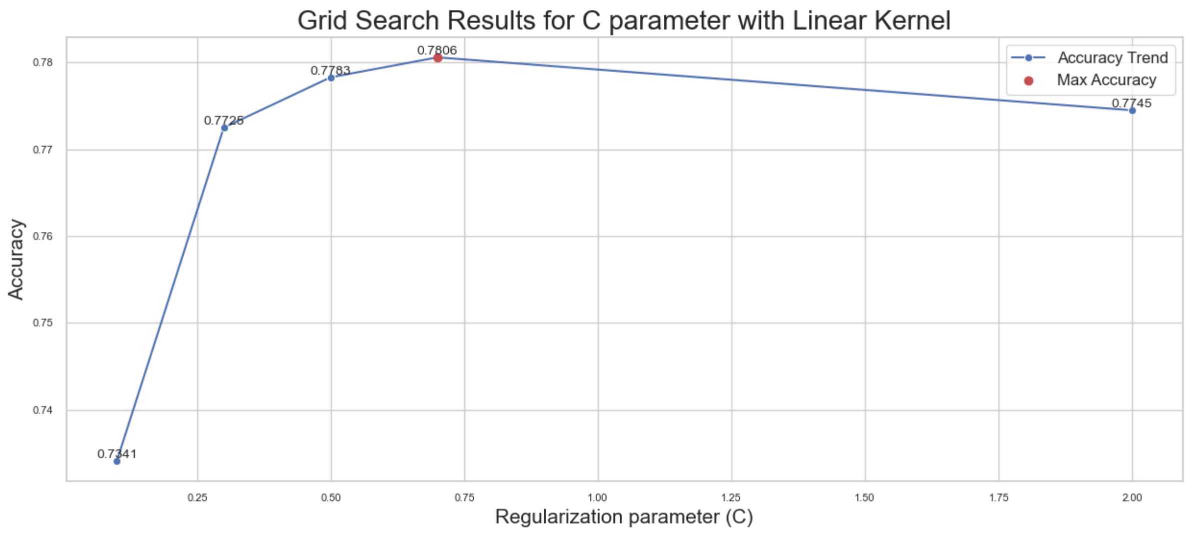


Figure 11: Accuracy trend for C parameter in SVM using Linear Kernel

We note in figure 11 that as `C` increases from $0.1 - 0.7$, the accuracy scores demonstrate a consistent improvement. However, it can be noticed that beyond `C = 0.7` up to `C = 2`, while the accuracy remains high, there's a slight decrease observed in the graphic.
The highest accuracy achieved among the tested `C` values was 0.7806 at `C = 0.7`, suggesting an optimal balance between regularization strength and model performance.
We can conclude that in the context of a linear kernel, values of `C` between 0.5 and 0.7 we get the most favorable accuracy scores. It's crucial to note that the optimal choice of `C` may vary depending on the dataset characteristics.

5.6 Model Selection

Across multiple experiments, the trend consistently revealed that the Random Forest model outperformed SVM in terms of predictive accuracy. Notably, when exploring SVM with RBF and linear kernels, despite meticulous fine-tuning of hyperparameters, the RF model exhibited superior accuracy across different scenarios.
Furthermore, the comparison extended to include an examination of confusion matrices, which confirmed the higher predictive prowess of the Random Forest model over SVM. Additionally, considering the computational aspect, the RF model displayed a more efficient training time, further solidifying its suitability for practical deployment.
All of this evidence helps us to decide to adopt Random Forest as our final classifier for the prediction of the tags. This selection is supported by the analysis and comparison we conducted, affirming

that Random Forest should be the optimal choice for achieving predictions on the StackOverflow tags.

6 Final Result and Conclusions

As mentioned in the section labeled "Comparison of Methods" (5), we opted to utilize the Random Forest model for predicting tags within the StackOverflow dataset.
Initially, a pre-processing function was employed to manipulate the HTML texts. This function converted all uppercase letters to lowercase, replaced specific symbols with spaces, and eliminated English stopwords. A customized tokenizer was utilized, designed specifically to retain certain special symbols essential for distinguishing between different programming syntaxes, such as parentheses and brackets. Subsequently, the tokenized text served as input for a TF-IDF Vectorizer, employing the parameter setting `max_features= 15000`.The selection of 15000 for `max_features` was a balance between richness in vocabulary representation and computational efficiency.
The selected model for fitting the data, as previously specified, was the Random Forest, configured with parameters `max_features = "sqrt"` and `n_estimators = 3000`. After model fitting and prediction, the accuracy achieved on 20% validation data stood at 80.94%.

6.1 Performance of the model

The following section presents a detailed breakdown of precision, recall, and F1-score values for individual classes to give an overview of the model's performance concerning specific categories. Additionally, the overall accuracy, macro average, and weighted average metrics are examined to look at the model's performance across the entire dataset. This comprehensive analysis aims to provide a refined understanding of the model's efficacy in classifying various textual categories, offering valuable insights for further enhancements and optimizations.
The following table (4) shows the Classification Report obtained.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
.net	0.69	0.63	0.66	273
android	0.95	0.85	0.90	309
angularjs	0.99	0.94	0.96	281
asp.net	0.83	0.76	0.80	303
c	0.84	0.89	0.86	300
c#	0.73	0.72	0.72	286
c++	0.82	0.77	0.79	268
css	0.78	0.93	0.85	293
html	0.74	0.65	0.69	286
ios	0.74	0.66	0.70	306
iphone	0.59	0.61	0.60	258
java	0.88	0.85	0.87	317
javascript	0.80	0.82	0.81	272
jquery	0.83	0.92	0.88	234
mysql	0.88	0.80	0.84	287
objective-c	0.71	0.75	0.73	282
php	0.85	0.83	0.84	254
python	0.92	0.95	0.94	280
ruby-on-rails	0.96	0.94	0.95	244
sql	0.70	0.91	0.79	267
Accuracy	0.81			
Macro Avg	0.81	0.81	0.81	5600
Weighted Avg	0.81	0.81	0.81	5600

Table 4: Classification Report

The previous table gave us some insights into the model performance across the different tags. Some of the key metrics we noticed were the following.

- **Precision:** Helps us to notice high precision scores for categories like "angularjs" (0.99), "python" (0,92), "ruby-on-rails" (0.96), etc. This suggests that the model has a good performance for these categories.

- **Recall**: Represents the ability of the model to identify all relevant instances in the data set. We have high recall for categories like "android" (0.85), "css" (0.93), "mysql" (0.80), etc. Knowing this, we can conclude that the model can effectively capture many instances of these categories.

- **F1-Score:** High F1-scores can be spotted across different categories, which indicates a balance trade-off between precision and recall.

The accuracy of the model across all classes is approximately 81%, which as mentioned at the beginning of the report, represents the proportion of correctly predicted tags out of the total dataset.

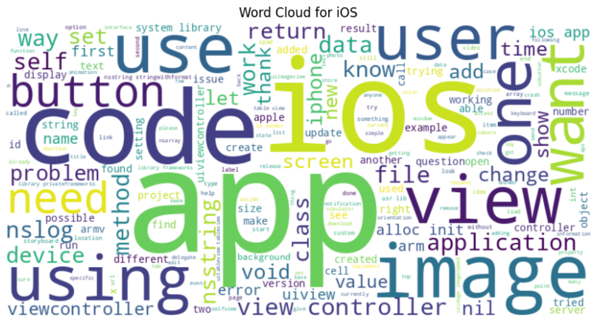
The model also showed to struggle relatively more with classes such as ".net,"ios","iphone" and "sql" as indicated by lower precision, recall, and F1-scores for these categories. Overall, the model shows promising performance. However, there's variability in performance across different classes, indicating potential areas for improvement, especially for classes with lower precision, recall, and F1-scores. Analyzing these metrics helps identify specific areas where the model struggles, providing insights for potential model enhancements or targeted data improvements to boost performance, especially for classes with lower scores.

6.2 Tags Classification

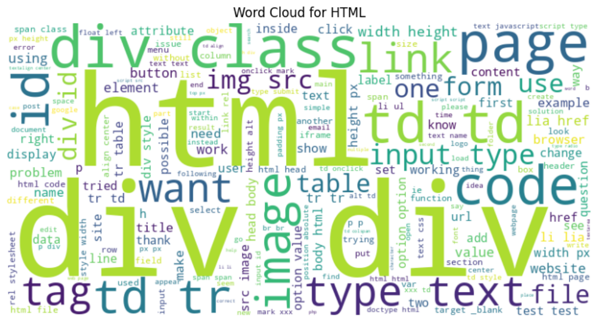
This section aims to explain the reason why certain posts are associated to a specific tag. Three tags have been selected for developing this section: *iOS, HTML* and *Python*.

Word Cloud

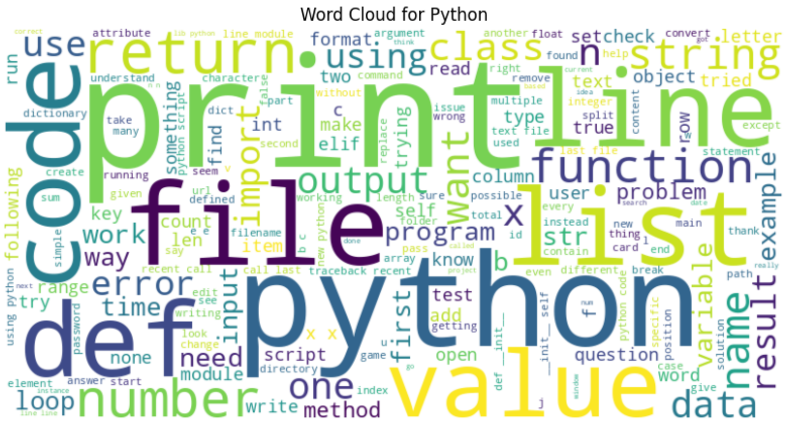
As a first approach, we might want to have a look at the data after it has been preprocessed. To do so, we can make some graphics using a package called `WordCloud`, which will show us the most frequent words that are found in the post depending on each tag. The results are shown below.



(a) iOS tag



(b) HTML tag



(c) Python tag

Figure 12: Word Cloud for different tags.

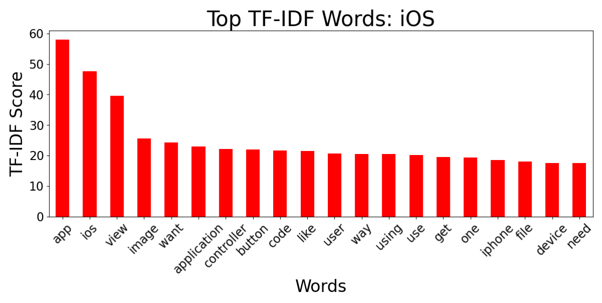
Figures shown in 12 give us a quick overview of the most common terms associated with each tag after applying the `preprocess` function.

TF-IDF and Count Vectorizer Scores

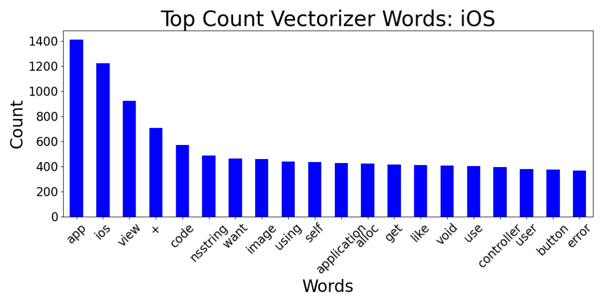
The following figures shown at Fig. (13) present a comparative analysis of word-associated tags obtained through two distinct natural language processing techniques: TF-IDF (Term Frequency-Inverse Document Frequency) and Count Vectorizer.

The primary objective of this comparison is to explore and visualize the differences in the top word associations generated by TF-IDF and Count Vectorizer. The word-associated tags or features extracted by these techniques hold significant importance as they play an important role in determining textual content's context and sentiment orientation.

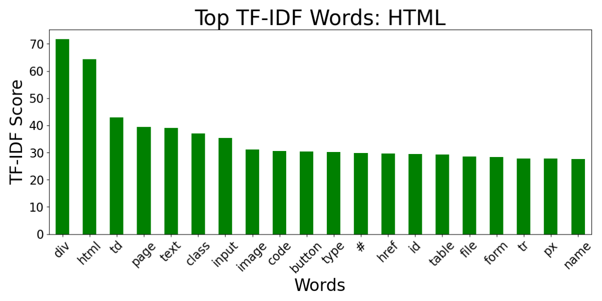
The pictures we'll show highlight the words each method thinks are most useful for figuring out how someone might feel in a sentence. We aim to help understand which method is better at finding the most important words that show feelings in text.



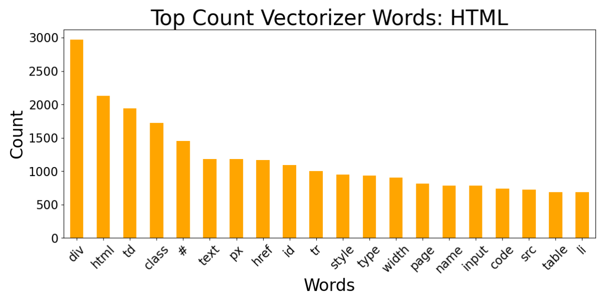
(a) TF-IDF - iOS tag



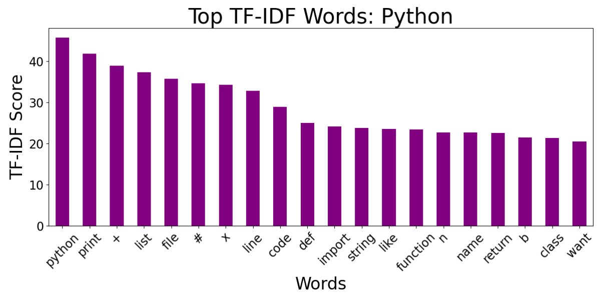
(b) Count Vectorizer - iOS tag



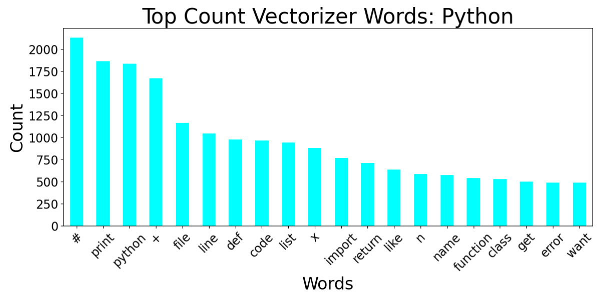
(c) TF-IDF - HTML tag



(d) Count Vectorizer - HTML tag



(e) TF-IDF - Python tag



(f) Count Vectorizer - Python tag

Figure 13: Comparison of Top Word Associations using TF-IDF and Count Vectorizer Techniques

The diagrams show a detailed link between certain words and how they relate to different tags. Examining the *iOS* tag, both TF-IDF and Count Vectorizer methods consistently highlighted "app," "iOS," and "view" as prevalent terms, indicating their recurrent use in discussions related to iOS platforms. In the realm of the *HTML* tag, the recognition of "tag," "html," and "td" by both TF-IDF and Count Vectorizer signifies their crucial role in conversations focused on HTML content structure and formatting conventions.

However, an intriguing divergence arises in analyzing the *Python* tag. TF-IDF emphasizes "Python," "print," and "+" as primary words, while Count Vectorizer assigns importance to "#", "print," and "python." This difference suggests different mechanisms in word prioritization, potentially in-

fluenced by varied contextual interpretations or methodological differences.
These findings show how tricky it can be to understand text analysis methods like TF-IDF and Count Vectorizer. They each pick out important words differently, depending on the context. Understanding these differences helps us know how each method focuses on specific words, which makes it easier to sort out text. Because we saw TF-IDF choosing better words, we went with TF-IDF for our main model. This helped our model make better predictions.

6.3 Conclusion

In this NLP project aimed at predicting tags for StackOverflow posts, our exhaustive analysis led us to select Random Forest as our best classifier. We carefully examined various aspects including text preprocessing, tokenization, accuracy, scalability, confusion matrix analysis, and fine-tuning before finalizing the model selection.

During our exploration, we discovered interesting differences between Support Vector Machines (SVM) and Random Forest in their predictive abilities for certain tags. This diversity in performance motivated us to experiment with ensemble techniques that might help us to improve the accuracy, leading us to employ a VotingClassifier combining fine-tuned SVM and Random Forest models. Despite promising validation results of 94.68% accuracy, the model's performance on Kaggle's public leaderboard fell to 89.81%.

Additionally, we explored into more complex models such as Long Short-Term Memory (LSTM) networks. LSTMs, being good at capturing long-range dependencies in sequential data, showed potential. However, their application did not gave us significant improvements in our specific task.

We also explored the Stacking Classifier, which integrate predictions from multiple base models like SVC and RF. This ensemble method demonstrated an overall accuracy of 83.80%, surpassing the individual accuracies of RF and SVC. It showcased the potency of combining diverse models through stacking, thereby enhancing classification performance compared to using each model independently.

While Random Forest emerged as our primary model, the exploration of ensemble methods like VotingClassifier and Stacking highlighted alternative approaches to weigh the strengths of multiple models for enhanced predictive performance. These alternative models presented promising insights but also highlighted the complexity of effectively combining different algorithms for optimal results in our specific context.

Moving forward, further research into ensemble methods and fine-tuning may unlock additional opportunities to improve model performance for more accurate predictions in classifying StackOverflow post tags.

References

[1] Alteryx Community (Accessed: 2023). Support vector machines. Image source: Alteryx Community.

[2] Gupta, A. (2021). Kernel tricks in support vector machines.

[3] IBM (2023). Decision trees. `https://www.ibm.com/topics/decision-trees#:~:text=A%20decision%20tree%20is%20a,internal%20nodes%20and%20leaf%20nodes`. IBM.

[4] IBM (Accessed: 2023). Random forest. `https://www.ibm.com/topics/random-forest`.

[5] Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing*. Prentice-Hall.

[6] Mammone, A., Turchi, M., and Cristianini, N. (2009). Support vector machines. *WIREs Computational Statistics*, 1(3):283–289.

[7] Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

[8] Nadeau, D. and Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

[9] OpenAI (2022). Chatgpt.

[10] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, (2021). *Scikit-learn TfidfVectorizer Documentation*. Scikit-learn.

[11] Sanchaya, E. (2023). Support vector machine (svm) algorithm.

[Scikit-Learn] Scikit-Learn, D. Support vector machines.

[13] Shaik, A. B. and Srinivasan, S. (2018). A brief survey on random forest ensembles in classification model. *International Conference on Innovative Computing and Communications*, page 253–260.