# Automated Tag Classification and Question Routing on AI Stack Exchange Using Classical NLP Techniques

Bojan Dimovski
*Student, 3rd year, Computer Technologies and Engineering*
*Faculty of electrical engineering and information technologies*
Skopje, Macedonia
kti252022@feit.ukim.edu.mk

Tomi Nikoloski
*Student, 3rd year, Computer Technologies and Engineering*
*Faculty of electrical engineering and information technologies*
Skopje, Macedonia
kti1032022@feit.ukim.edu.mk

***Abstract*—This project investigates automated tag classification for AI Stack Exchange posts using classical Natural Language Processing techniques. The objective was to develop a multilabel classification system capable of predicting relevant tags for question routing and content organization. The dataset consists of 8,981 posts containing 684,383 words, with analysis focused on the top 10 most frequent tags covering approximately 75% of all posts (>12,000). Three vectorization approaches were implemented: Bag of Words with StandardScaler, TF-IDF vectorization, and Singular Value Decomposition with RobustScaler. Multiple machine learning algorithms were evaluated including Naive Bayes, Random Forest, and K-Nearest Neighbors using both binary relevance and classifier chain methods. Feature engineering included lemmatization, chi-square feature selection, and optimization of 2000 features following Zipf's law distribution. The best performing model, Random Forest with Bag of Words representation and chi-square feature selection, achieved an F1-micro score of 0.6127 and subset accuracy of 39.9% while also lessening the hamming loss significantly from the base loss (from 0.15 to 0.09). Real-world validation using a custom AI-related question demonstrated successful prediction of relevant tags - 'deep-learning' for the use-case scenario. However, classical NLP techniques showed fundamental limitations, like the inability to capture semantic context, challenges with sparse high-dimensional feature spaces, and difficulties handling severe class imbalance inherent in multilabel classification. While these methods provide a viable baseline for automated question routing, the results highlight the necessity for more sophisticated approaches such as transformer-based models to achieve production-level performance in complex multilabel text classification scenarios.**

***Index Terms*—Multi-label classification, NLP, Vectorization, Singular value decomposition**

## I. Introduction

Natural Language Processing (NLP) is a field of artificial intelligence that helps computers understand and work with human language. One important application of NLP is text classification, where we automatically sort text documents into different categories. This is especially useful for online communities like Stack Exchange, where thousands of questions are posted daily and need to be organized with appropriate tags. Stack Exchange is a network of question-and-answer websites covering various topics, and is a parent-site of many popular forums such as Stack Overflow, Math Stack Exchange, with AI Stack Exchange being one community focused on artificial intelligence questions. When users post questions, they must add tags that describe the topic of their question. These tags help other users find relevant questions and help experts in specific areas locate questions they can answer. However, choosing the right tags can be challenging for users, and sometimes questions end up with incorrect or missing tags. The goal of this project is to build an automated system that can predict appropriate tags for AI Stack Exchange questions using classical NLP techniques. This would help improve question routing by automatically suggesting relevant tags, making it easier for experts to find questions in their area of expertise and for users to find similar questions that have already been answered.

## II. Data collection and initial processing

The data for this project comes from the Stack Exchange Data Dump [1], which provides complete archives of all Stack Exchange sites in XML format. These data dumps are updated regularly and contain the full history of posts, comments, users, and other site information. The raw data consists of several XML files, with the most important being Posts.xml, which contains all questions and answers posted on the site. Each post entry in the XML file includes various attributes such as the post ID, creation date, score, view count, and most importantly for our project, the post title, body content, and associated tags. To extract and process this data, we used Python's xml.etree.ElementTree library to parse the XML files. The parsing process involved iterating through each post element and extracting key information:

Title: The question headline that summarizes the main topic Body: The detailed question content, often containing code examples and technical explanations Tags: The classification labels assigned by the original poster User ID: The identifier of the person who posted the question

After parsing the XML data, we converted it into a pandas DataFrame for easier manipulation and analysis. This allowed us to use familiar data science tools for filtering, cleaning, and exploring the dataset. The DataFrame structure made it simple to query specific subsets of posts, analyze tag distributions, and prepare the data for machine learning algorithms. The initial dataset contained over 12,000 posts, but after filtering for posts that contain tags that are frequent enough to be

trainable, we focused on 8,981 posts that had clear titles, substantial body content, and **at least one tag assignment**. This preprocessing step was crucial because machine learning models perform better with clean, consistent data rather than trying to handle missing or corrupted entries. This data extraction and initial processing phase laid the foundation for all subsequent analysis and model development, ensuring we had a reliable dataset that accurately represents the types of questions and tagging patterns found on AI Stack Exchange.

## III. Tag selection

After inspecting, it is found that the dataset contains a total of 986 unique tags assigned to posts on AI Stack Exchange. However, many of these tags are used very infrequently, making them unsuitable for inclusion in a machine learning model. For example, while some tags appear in only a handful of posts, the most frequent tag, neural-networks, appears in over 2,600 posts. This imbalance presents a challenge, as models trained on extremely rare labels tend to perform poorly and introduce noise into the classification task.

To better understand the distribution of tag usage, a tag frequency analysis was performed. The results clearly show an exponential decay pattern, where only a small number of tags are highly frequent, and the majority appear in very few posts. This is consistent with the long-tail distribution commonly observed in language and user-generated content.

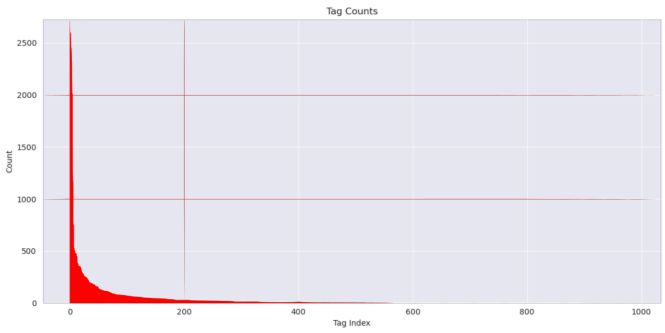


Fig. 1: Frequency of observed tags across all questions posted on the forum

The frequency curve demonstrates that while the top few tags are very common, the majority of the 986 tags occur too infrequently to be viable as model labels. To reduce noise and improve model performance, a method was needed to select only the most relevant and frequently occurring tags. A decision was made to determine a cut-off point based on how many posts could be effectively covered by the top N most frequent tags.

To make this decision, a simple form of the elbow method was applied. A plot was created showing how the number of posts covered increases as more top tags are included. Specifically, for each N, the plot measures how many posts have at least one of the top N tags. The idea was to find a point where adding more tags leads to diminishing returns in terms of post coverage.

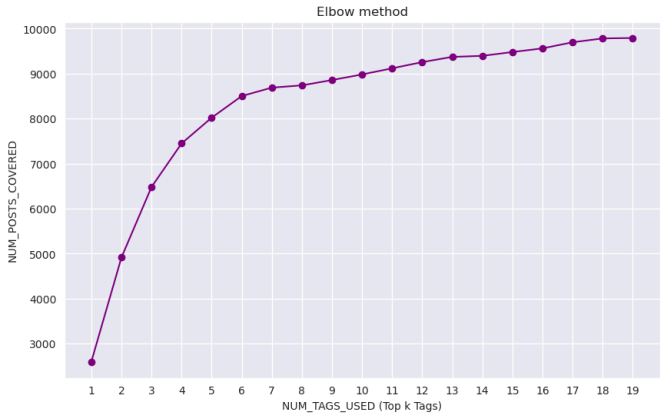


Fig. 2: The elbow method. X axis: number of tags, Y axis: number of posts

Based on this analysis, it was found that the top 10 most frequent tags cover a total of 8,981 posts. This accounts for approximately three quarters of all posts in the dataset. Additionally, only 10 out of the 986 total tags appear in more than 500 posts, reinforcing the decision to limit the label set to these top tags. Tags beyond this threshold contribute very little additional coverage and occur too rarely to be effective as model targets.

By focusing on the 10 most common tags, the dataset becomes more balanced and manageable for multilabel classification, while still preserving a large and diverse portion of the original data. This selection process forms the basis for all subsequent steps in model training and evaluation.

## IV. Data preprocessing

Before applying any machine learning techniques, the raw dataset required careful preprocessing to ensure consistency and relevance. This phase focused on cleaning the text data, handling missing values, and filtering the dataset to align with the chosen label space of the top 10 tags.

To begin with, missing values were addressed by combining the post title and body into a single column named content. Although some posts might have incomplete fields, every post in the dataset has at least a non-empty title or body. By concatenating these two fields, a complete text representation was ensured for every entry, eliminating issues with missing values.

Following this, the text data was cleaned to remove noise and irrelevant formatting. Since the body of Stack Exchange posts often contains HTML markup, two stages of HTML cleanup were performed. First, the html.unescape function was used to decode any HTML entities, such as < or &, into their corresponding characters. Then, the BeautifulSoup library was applied to strip away any remaining HTML tags and extract the plain textual content.

In addition to HTML cleanup, regular expressions were used to remove other forms of unwanted characters. These included anything that did not contribute meaningfully to the natural language content of the post. This step helped standardize the text across posts and reduced vocabulary noise during feature extraction.

Another critical part of preprocessing involved ensuring that the tags assigned to each post matched the reduced label set selected earlier. For each post, the original list of tags was intersected with the top 10 most frequent tags. Only tags that appeared in this intersection were retained. Posts that did not have any tags belonging to the top 10 set were removed from the dataset entirely. This filtering step was essential to maintain a consistent and valid multilabel classification setup.

The result of this preprocessing pipeline was a clean and well-structured dataset of 8,981 posts, each represented by a consolidated and cleaned text field, and labeled with at least one of the 10 selected tags.

## V. Exploratory data analysis

To better understand the characteristics of the dataset and guide later modeling decisions, an exploratory data analysis (EDA) was conducted. The dataset contains a total of 8,981 posts. After text preprocessing, the combined word count across all posts was 684,383. This corresponds to a vocabulary size of 38,172 unique words, with an average of 76.20 words per post. These initial statistics suggest a moderate post length and a rich but not overwhelmingly large vocabulary.

A closer analysis of post lengths reveals a significant variation in the number of words per post. The shortest post in the dataset contains only 6 words, while the longest extends to 1,519 words. The median word count is 59.0, and the standard deviation is approximately 69.99. This wide spread indicates the presence of both very brief and highly detailed questions. Most posts, however, tend to be concise.

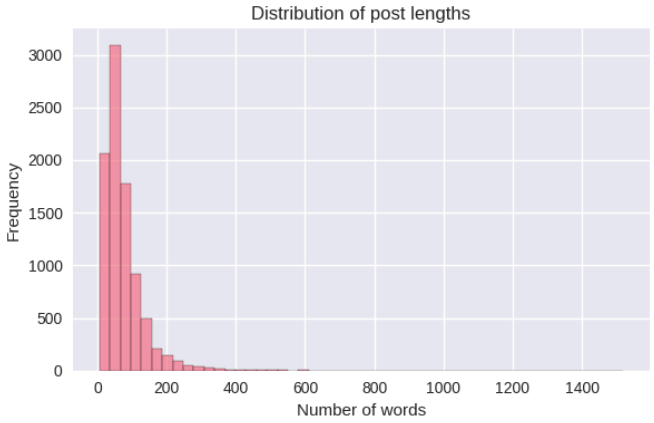


Fig. 3: Histogram showcasing the post length distribution (number of words)

As shown in the histogram, the distribution of post lengths is heavily right-tailed, with the majority of posts falling under 100 words. This skew justifies the choice of cosine similarity as the preferred similarity metric in downstream vector space modeling. Unlike Euclidean distance, cosine similarity is not influenced by the absolute magnitude of the document length and instead focuses on the direction of the feature vectors, making it more appropriate for comparing text of varying lengths.

In terms of vocabulary usage, the most frequently occurring words were examined. Common terms such as network, model, neural, image, function, and datum appear with particularly high frequency. These words reflect the technical focus of AI Stack Exchange discussions. The top 20 most frequent words collectively make up a significant proportion of the overall word count.

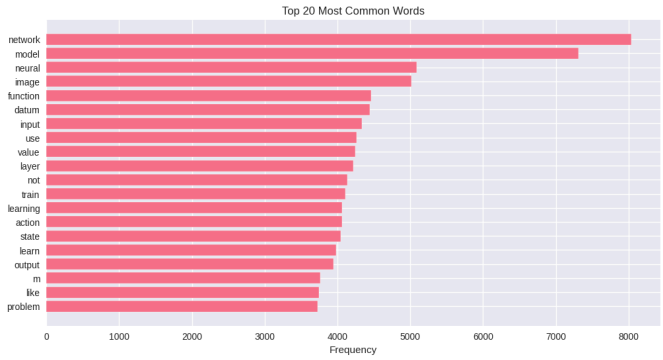


Fig. 4: Bar plot showcasing the 20 most frequently used words across posts

A bar chart visualizes the top 20 most common words in the dataset. Together, these words highlight the domain-specific nature of the content.

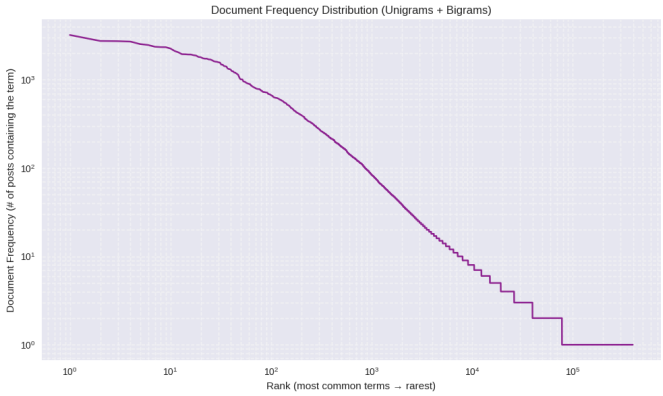


Fig. 5: Line plot showing the frequency of terms - from most to least common

In addition, a Zipf's law plot of the word frequency distribution was generated. The resulting curve follows the expected power-law pattern, where a small number of words occur very frequently, and the majority of words occur infrequently. This observation supports the use of statistical feature selection techniques to prioritize high-impact words while discarding rare and potentially noisy terms.

Further analysis of the dataset's lexical richness was conducted using the Type-Token Ratio (TTR), which was calculated as 0.0558. This indicates that, on average, for every 100 words in the dataset, approximately 5.6 are unique. Moreover, 21,711 of the total 38,172 unique words appear only once, meaning that 56.88% of the vocabulary consists of single-occurrence words. This level of lexical variety is typical of large user-generated datasets.

Tag distribution was also examined in more detail. Each post has, on average, 1.49 of the 10 selected tags. This relatively low number suggests that posts are not heavily multi-labeled, and that any multilabel classifier trained on

this dataset may need to be conservative. There is a risk that models might default to under-predicting tags. This could result in high false negative rates unless actively mitigated by tuning thresholds or using more expressive models.

Finally, a word cloud was generated to provide an intuitive visualization.

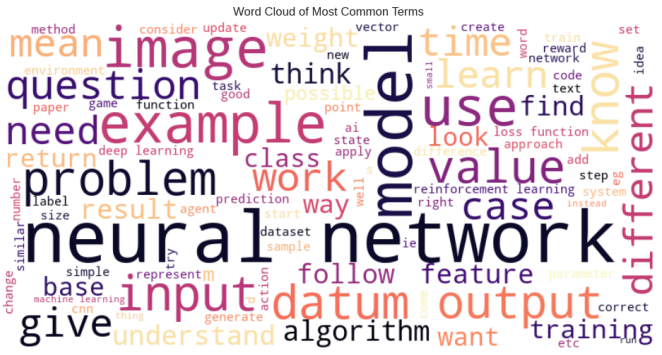


Fig. 6: Word cloud - a high-level summary of the previously showcased plots

## VI. Vectorization

Before any machine learning model can be applied to text data, it must be transformed into a numerical representation. This transformation is commonly known as vectorization, where each post is mapped into a high-dimensional feature space. The vectorization pipeline in this project begins with standard natural language preprocessing techniques: tokenization, stopword removal, and lemmatization. Common English stopwords, such as "the", "is", or "and", were removed to reduce noise and focus on more meaningful terms. While stemming is a widely used technique for reducing words to their base forms, lemmatization was preferred in this project. Unlike stemming, which often produces non-words and can truncate words too aggressively, lemmatization reduces tokens to their dictionary form while preserving grammatical integrity. This results in cleaner and more semantically consistent features.

In addition to unigrams, bigrams were included in the feature space to better capture meaningful phrases that appear frequently in the dataset. For instance, "neural network" is by far the most common term, and treating it as a single feature allows the model to better leverage its semantic importance. This decision is also supported by the Zipfian distribution of word frequencies observed earlier, which suggests that compound terms often dominate domain-specific text.

Three different vectorization approaches were explored, each producing a distinct feature matrix for modeling:

The first approach is Bag of Words (BoW) with 2,000 features selected via chi-squared feature selection. This representation captures word frequencies across the corpus. After the BoW matrix was generated, it was scaled using StandardScaler. The motivation for scaling is to reduce the influence of disproportionately large feature values and normalize the distribution of inputs for algorithms sensitive to feature magnitude.

The second approach uses TF-IDF (Term Frequency–Inverse Document Frequency) vectorization, also with 2,000 selected features. TF-IDF adjusts raw word counts by penalizing terms that are common across many documents, thereby giving more weight to distinctive words. Unlike the BoW approach, TF-IDF was not scaled further, since it already embodies a form of weighting and normalization by design. Applying additional scaling to TF-IDF could distort these built-in relationships.

The third and final approach involves dimensionality reduction using Singular Value Decomposition (SVD). This method was applied on top of the TF-IDF matrix. The number of components was chosen such that approximately 90% of the total variance was preserved, which resulted in a reduction from 2,000 features to 1,158. This effectively halves the dimensionality while retaining most of the information. Since SVD produces continuous-valued features that can contain influential outliers, RobustScaler was used to scale the resulting matrix. Unlike standard scaling, robust scaling uses the median and interquartile range, making it more appropriate for skewed distributions. In this context, preserving outliers may actually be beneficial, as the original interpretability of features is lost after SVD, and rare but extreme values might still carry meaningful signals.

These three vector representations - BoW, TF-IDF, and reduced-dimension SVD, formed the basis for all subsequent classification experiments, allowing a comparison of models across different feature spaces.

## VII. Performance metrics

Evaluating a multilabel classification model requires specialized metrics that account for the fact that each instance may be associated with multiple labels simultaneously. Unlike single-label classification, where accuracy alone may suffice, multilabel tasks demand a more nuanced view of performance.

Several metrics [2] were used in this project to evaluate model predictions:

### A. *Hamming loss*

Hamming loss measures the fraction of incorrect labels relative to the total number of labels. It penalizes both false positives (predicting a label that is not actually present) and false negatives (failing to predict a label that is present). Formally, it is defined as:

$$\text{Hamming loss} = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} \mathbf{1}\left[y_{ij} \neq \widehat{y_{ij}}\right] \qquad (1)$$

where $N$ is the number of instances, $L$ is the number of possible labels, $y_{ij}$ is the true label and $\widehat{y_{ij}}$ is the predicted label.

In this project, the baseline hamming loss is approximately 0.15, which corresponds to the performance of a naive model that predicts an empty tag list for every post. This is because no single tag appears in more than half of the posts, so such a model would predict the majority label presence (which is that the label is absent) - as this would minimize false positives at the cost of missing all true labels. So, we aim to reduce the hamming loss below 0.15.

### B. Subset accuracy

Subset accuracy is a strict metric that considers a prediction correct only if all predicted labels exactly match the true set of labels for a given instance. It is defined as:

$$\text{Subset accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\left[Y_i = \hat{Y}_i\right] \tag{2}$$

where $Y_i$ and $\hat{Y}_i$ are respectively the true and predicted label sets of each instance. This metric is especially harsh in multilabel scenarios, since even a single incorrect label causes the entire prediction to be counted as incorrect. However, it is useful for measuring how often the model is completely correct, which is important for tasks like tag recommendation systems where users expect coherent and fully relevant outputs.

### C. Precision, recall and F1-score

To capture the trade-off between false positives and false negatives in a multilabel setting, precision, recall, and F1 scores are computed using two averaging techniques:

- Micro averaging aggregates the contributions of all labels across all instances before calculating the metrics. It gives equal weight to every label occurrence.

$$\text{Micro precision} = \frac{\sum_i \text{TP}_i}{\sum_i \text{TP}_i + \sum_i \text{FP}_i} \tag{3}$$

$$\text{Micro recall} = \frac{\sum_i \text{TP}_i}{\sum_i \text{TP}_i + \sum_i \text{FN}_i} \tag{4}$$

$$\text{Micro F1} = \frac{2 \cdot \text{Micro precision} \cdot \text{Micro recall}}{\text{Micro precision} + \text{Micro recall}} \tag{5}$$

- Macro averaging calculates the metric independently for each label and then takes the average, treating all labels equally regardless of how frequently they appear.

$$\text{Macro precision} = \frac{1}{L}\sum_j \frac{\text{TP}_j}{\text{TP}_j + \text{FP}_j} \tag{6}$$

$$\text{Macro recall} = \frac{1}{L}\sum_j \frac{\text{TP}_j}{\text{TP}_j + \text{FN}_j} \tag{7}$$

$$\text{Macro F1} = \frac{1}{L}\sum_j \frac{2 \cdot \text{Precision}_j \cdot \text{Recall}_j}{\text{Precision}_j + \text{Recall}_j} \tag{8}$$

Macro metrics are sensitive to performance on rare labels, which is especially important in this project given the long-tailed distribution of tag frequencies. Micro metrics, on the other hand, tend to favor the more frequent labels and give a better sense of the model's overall behavior across the dataset.

## VIII. Model performance analysis

To evaluate the performance of various classification approaches, the dataset was divided into a training set and a test set using an 80-20 split. This stratification ensured that the distribution of labels was preserved across both subsets, allowing for a realistic simulation of how the model might perform on unseen data.

The multilabel classification task was tackled using the Binary Relevance method as the baseline strategy. In this approach, the multilabel problem [3] is decomposed into $L$ separate binary classification tasks, where $L$ is the number of possible labels (in this case, 10 tags). A distinct binary classifier is trained independently for each tag to determine its presence or absence in a post.

Each of the three feature representations — Bag of Words with StandardScaler, TF-IDF, and SVD-reduced TF-IDF with RobustScaler — was used to train three different classifiers:

Naive Bayes, Random Forest and K-Nearest Neighbors.

The performances of the 9 models were evaluated using the multilabel metrics previously described. The results are summarized in the following table:

TABLE I: Classification report for Binary relevance method

| | f1_macro | f1_micro | avg_recall | avg_prec | hamming | subset_acc |
|---|---|---|---|---|---|---|
| NB_BoW | 0.3968 | 0.5288 | 0.3308 | 0.5452 | 0.1149 | 0.2955 |
| NB_TF | 0.3326 | 0.5114 | 0.2721 | 0.5905 | 0.1087 | 0.3200 |
| NB_SVD | 0.4221 | 0.4626 | 0.5566 | 0.3521 | 0.2026 | 0.1124 |
| RF_BoW | 0.4016 | 0.5906 | 0.3417 | 0.7510 | 0.0960 | 0.3756 |
| RF_TF | 0.3978 | 0.5866 | 0.3367 | 0.7224 | 0.0959 | 0.3806 |
| RF_SVD | 0.2002 | 0.3723 | 0.1499 | 0.5395 | 0.1176 | 0.2270 |
| KNN_BoW | 0.2776 | 0.3936 | 0.2130 | 0.4960 | 0.1391 | 0.2176 |
| KNN_TF | 0.3878 | 0.5008 | 0.3283 | 0.5432 | 0.1243 | 0.3116 |
| KNN_SVD | 0.1957 | 0.3217 | 0.1709 | 0.4149 | 0.1544 | 0.2020 |

A more sophisticated dependency-aware method was implemented as well - the Classifier Chain technique. This method enables the model to capture correlations between tags, which is particularly relevant in cases where certain tags tend to co-occur.

The same three classifiers were used for this method, leading to another 7 model variants - excluding the SVD dataframe for the Random forest and KNN models. Their evaluation metrics are shown in the table below:

TABLE II: Classification report for Classifier chain method

| | f1_macro | f1_micro | avg_recall | avg_prec | hamming | subset_acc |
|---|---|---|---|---|---|---|
| NB_BoW | 0.4170 | 0.5448 | 0.3605 | 0.5348 | 0.1147 | 0.3038 |
| NB_TF | 0.3547 | 0.5331 | 0.3009 | 0.5882 | 0.1087 | 0.3383 |
| NB_SVD | 0.3797 | 0.4149 | 0.5539 | 0.3932 | 0.2304 | 0.0484 |
| RF_BoW | 0.4091 | 0.5987 | 0.3478 | 0.7388 | 0.0948 | 0.3845 |
| RF_TF | 0.3934 | 0.5868 | 0.3373 | 0.6973 | 0.0963 | 0.3784 |
| KNN_BoW | 0.2797 | 0.4140 | 0.2380 | 0.4675 | 0.1452 | 0.2732 |
| KNN_TF | 0.3849 | 0.5116 | 0.3456 | 0.5036 | 0.1275 | 0.3400 |

Across both the binary relevance and classifier chain methods, there are no drastic differences in performance between vectorization strategies. Overall, Random Forest with either BoW

or TF-IDF consistently yields the best results, achieving the highest micro F1 scores (≈0.59) and the lowest Hamming losses (≈0.09). KNN, in contrast, struggles across all configurations.

## IX. Model performance after feature selection

To further evaluate the performance of our models, we conducted feature selection to reduce dimensionality while retaining informative features. An image of the pairwise label dependencies using the Phi coefficient reveals that inter-label correlations are quite weak.


Fig. 7: Heatmap showcasing the phi ($\varphi$) coefficient for inter-label dependency

Given these minimal dependencies, we proceed with the binary relevance approach for its simplicity and computational efficiency, especially since classifier chains did not show a clear performance gain. We apply univariate feature selection using the chi-squared ($\chi^2$) test, selecting the top 500 features from the original 2,000. [4] This feature selection differs from just picking the 500 most frequent words as the $\chi^2$ test looks at dependencies between labels.

The classification report for this setup is shown below.

TABLE III: Classification report for Feature-selected Chi2 dataset

| | f1_macro | f1_micro | avg_recall | avg_prec | hamming | subset_acc |
|---|---|---|---|---|---|---|
| NB_BoW | 0.3588 | 0.5011 | 0.2873 | 0.5504 | 0.1126 | 0.2844 |
| NB_TF | 0.3080 | 0.4817 | 0.2415 | 0.5725 | 0.1087 | 0.2866 |
| NB_SVD | 0.4682 | 0.5103 | 0.5786 | 0.4038 | 0.1718 | 0.1664 |
| RF_BoW | 0.4448 | 0.6127 | 0.3860 | 0.6251 | 0.0959 | 0.3990 |
| RF_TF | 0.4333 | 0.6068 | 0.3766 | 0.6411 | 0.0958 | 0.3918 |
| KNN_BoW | 0.3940 | 0.5092 | 0.3262 | 0.5445 | 0.1220 | 0.3072 |
| KNN_TF | 0.4084 | 0.5222 | 0.3502 | 0.5489 | 0.1208 | 0.3261 |

Feature selection using the chi-squared method led to a slight drop in average precision across most models, reflecting a loss of specificity. However, this trade-off appears to have benefited subset accuracy, the most stringent metric, which improved in several cases—especially for the Random Forest models, reaching almost 0.4. This suggests that although fewer features may slightly blur exact predictions, the overall tag combinations predicted tend to match the true sets more often. It should also be noted that the $\chi^2$ test was applied using a proper train–validation–test split. This ensures that the selection of informative features was evaluated on a separate validation set before final testing. By isolating the test set until the final evaluation stage, we prevent data leakage and avoid overfitting the model to perform well only on the test data. This setup upholds the integrity of the experimental process.

## X. Conclusions

In conclusion, the experiments demonstrate encouraging outcomes within the limits of classical NLP techniques. The base hamming loss of 0.15, representing a dummy model that assigns no tags at all, was successfully lowered to 0.09. This indicates that the trained models were able to make meaningful tag predictions. The strongest configuration reached an F1-micro score of 0.61 on the feature-selected dataset, reflecting solid performance in recognizing relevant tags across posts.

However, recall values remain relatively low, which suggests that the models are somewhat passive in their behavior. This tendency was already visible in the exploratory data analysis, where the average number of tags per post was only 1.49 out of a possible 10. This cautiousness is also evident in a qualitative test case. Given the following example post:

"My model is performing poorly. I want to train it better but training it makes me so tired. Perhaps deep learning models could perform better for such tasks."

the model returned a single tag:

['deep-learning']

even though other reasonable tags like those in the set ['training', 'neural-networks'] could have been expected.

While the results are not groundbreaking, they are solid given that the approach used only classical methods like bag-of-words, TF-IDF, and dimensionality reduction. No deep learning or modern embeddings were used. This baseline can serve as a starting point for future work with more advanced methods like transformer-based models that better capture meaning and context.

## References


[1] S. E. Inc., "Stack Exchange Data Dump." Accessed: Dec. 01, 2024. [Online]. Available: https://archive.org/details/stackexchange

[2] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python." [Online]. Available: https://scikit-learn.org/

[3] G. Tsoumakas and I. Katakis, "Multi-Label Classification: An Overview," *International Journal of Data Warehousing and Mining*, vol. 3, no. 3, pp. 1–13, 2007, doi: 10.4018/jdwm.2007070101.

[4] Y. Yang and J. O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," in *Proceedings of the Fourteenth International Conference on Machine Learning*, Nashville, TN: Morgan Kaufmann, Jul. 1997, pp. 412–420.