Data Science Submission

Edition: 2023 2A

Project: TCNER
Primary Topic: IENLP
Secondary Topic: DM
Team: DS- Project Group ~53
Members: Giulia Pais, Vani Ramesh
Last update: 26/04/2024, 15:27:32

Motivation

Be it a researcher eager to share their groundbreaking discoveries, or a professional striving to stay ahead in their industry, both facing a wide range of conferences all over the world and each promising a unique opportunity. Our system enables them to effortlessly match their papers to the most relevant conference by analysing the title of their paper.
This not only saves time but also ensures that researchers are connected closely with conferences that align with their interests and area of expertise. Also, by analysing patterns in the title it gives the researcher, the conference with the best chance of acceptance. Moreover, this project facilitates greater collaboration by encouraging the researcher to participate in a certain conference that might have otherwise gone unnoticed.
In essence, we will employ machine learning techniques in this project to develop an automated Conference Recommendation System.

Business/Research questions

The objectives of our project are:

1. Specific - Design and develop a conference recommendation system that utilises machine learning algorithms to analyse the title of a paper and classify it to the most relevant conference
2. Measurable - Metrics that can be employed to measure the system's performance
3. Achievable - Choice of algorithms relevant to the situation, time and computational resources available
4. Relevant - In today's knowledge-driven society, help researchers to streamline their conference selection process, saving time and ensuring better alignment with their interests.
5. Time-bound - The project aims to complete the development and testing of the conference recommendation system within the quartile 2A, depending on the scope and resources available.

Source data

In our project, we were provided with two distinct datasets for training and testing, each comprising titles of technical papers, short keywords or phrases labelled with the corresponding conference. The training set contained 21,643 samples. However, an initial analysis revealed a non-uniform distribution of samples across categories, with a significant skew towards the "ISCAS" category.

With the predominance of one category, we faced a potential class imbalance issue. Class imbalance is a common problem in machine learning, where some classes are overrepresented compared to others, leading to a model biased towards the majority class. However, instead of resampling techniques, as a starting point, we opted to address this challenge through strategic algorithm selection, favoring methods known for handling imbalances effectively.

To better understand the underlying patterns and to guide our feature engineering, we conducted a detailed exploratory analysis. This included:

- Frequency Analysis: We plotted the top ten most frequent words in each category to identify commonalities and differences in language use across different conferences. This analysis was instrumental in tailoring our feature extraction to capture essential attributes.
- Title Length Distribution: We examined the distribution of title lengths both in terms of characters and words. This analysis helped us understand the typical structure and verbosity of technical titles, which could impact the performance of our text classification models.

By observing the lengths distributions we were able to identify a small subset of data points that contained just one or two words, generally being acronyms, websites domains or a keyword particularly relevant for the topic of the conference. This might suggest that, in the vectorization phase, it might be appropriate to retain also the very infrequent n-grams as they might be relevant for the classification.

From the frequency analysis per class we can observe that for some classes there are words that are more discriminant with respect to all the others (e.g. "web" for WWW, "networks" for INFOCOM, "data" and "database" for VLDB), while there is not a clear prevalence of some terms for other classes or the separation is not so clear-cut (e.g. SIGGRAPH, ISCAS).

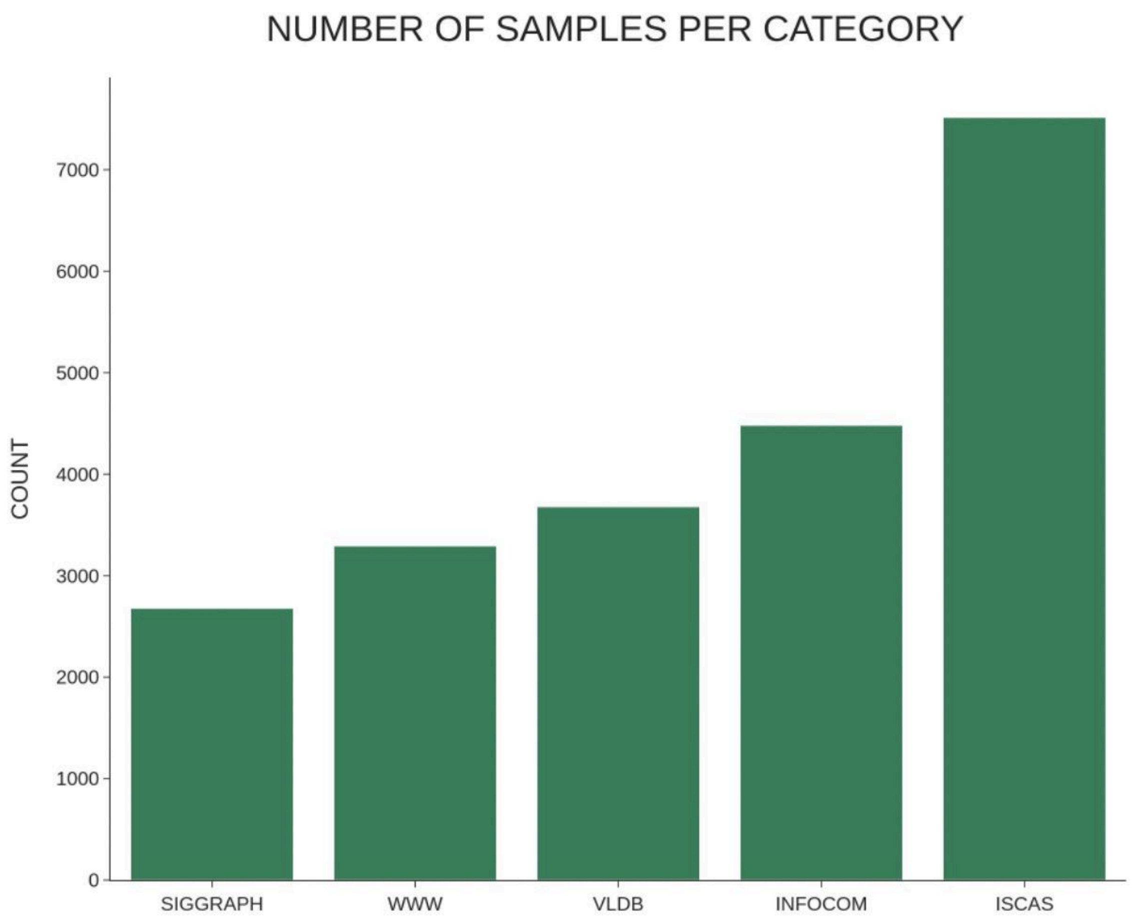


Figure 1. *Distribution of samples across different categories.*

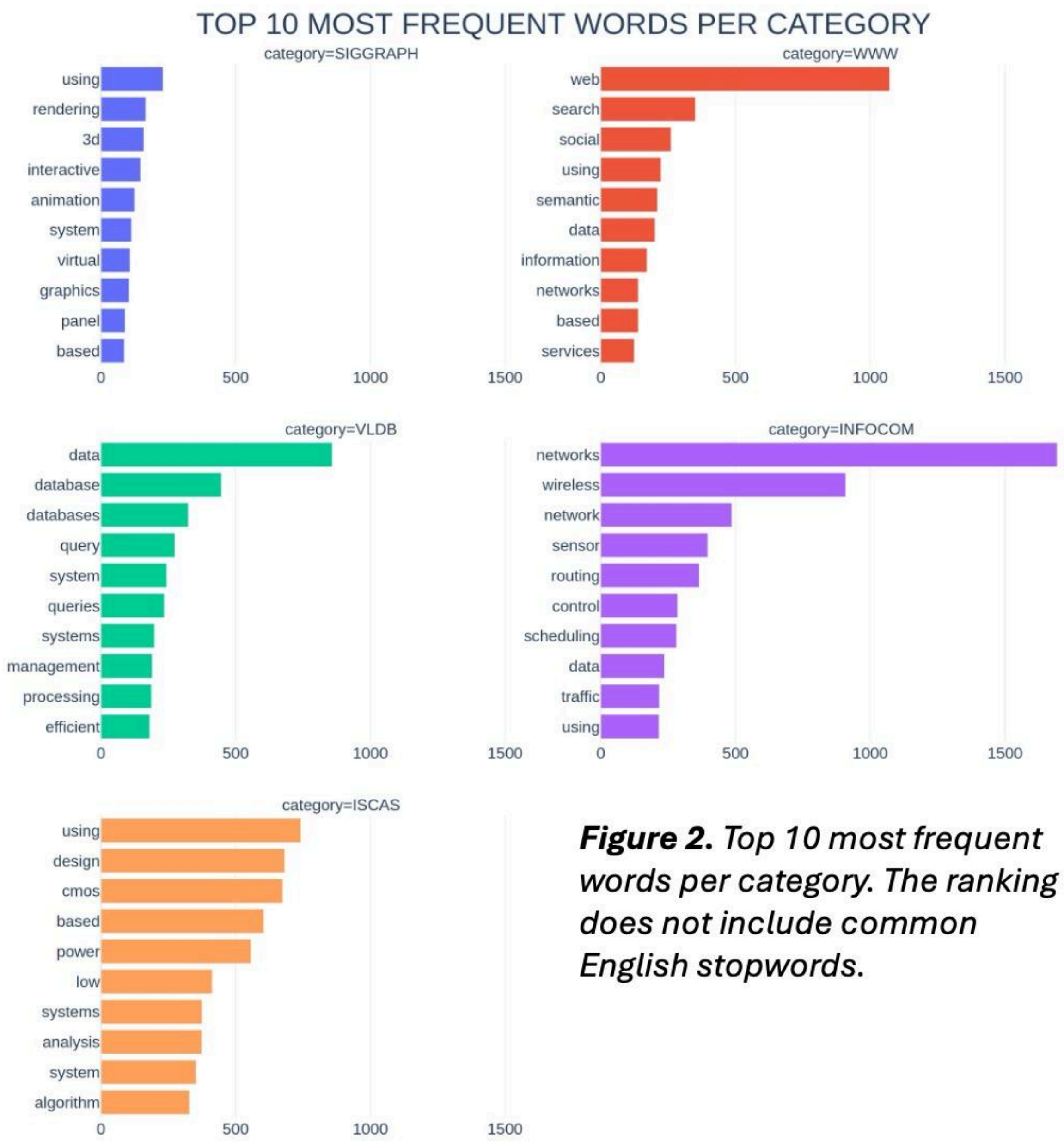


Figure 2. *Top 10 most frequent words per category. The ranking does not include common English stopwords.*

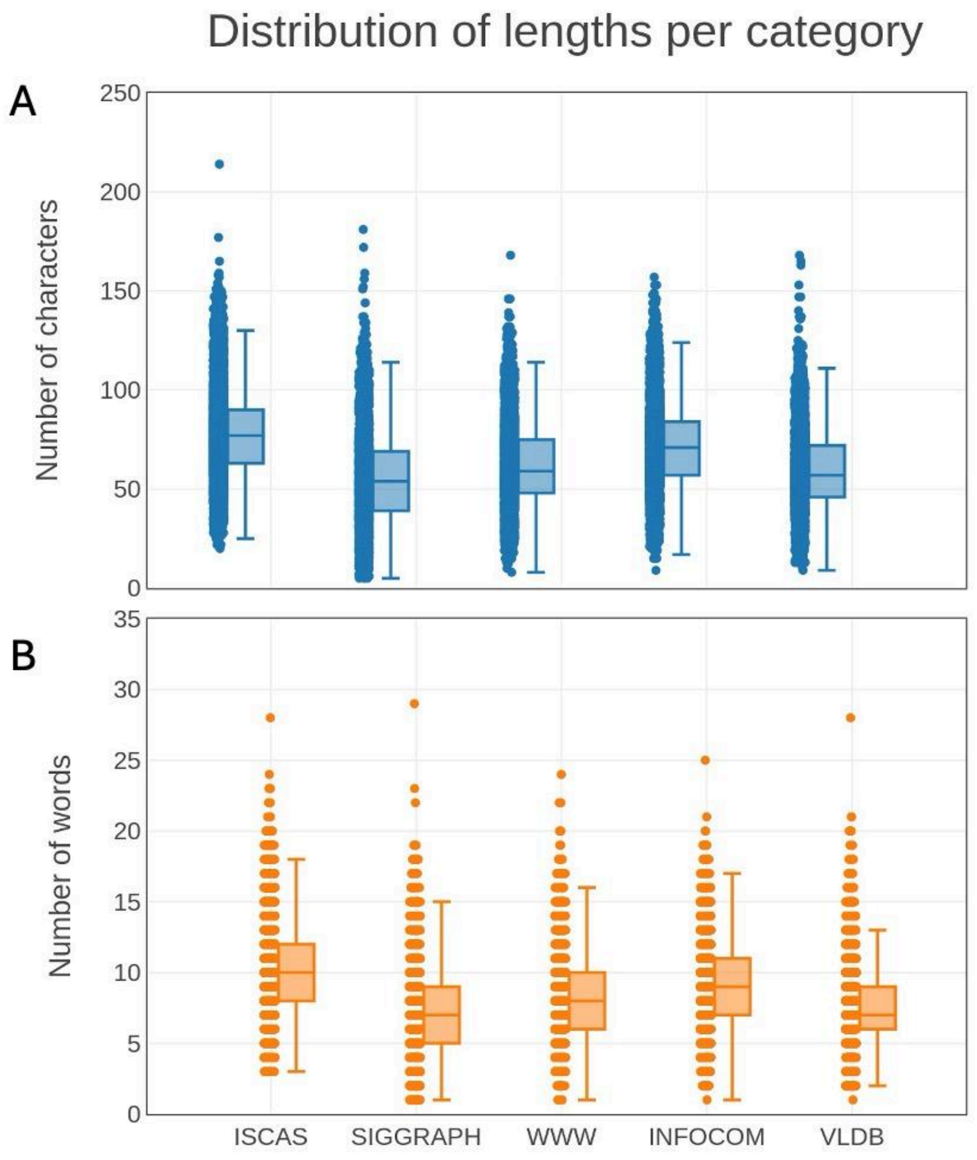


Figure 3. *Distribution of data points by length in terms of number of characters (A) and number of words (B) for each category.*

Method

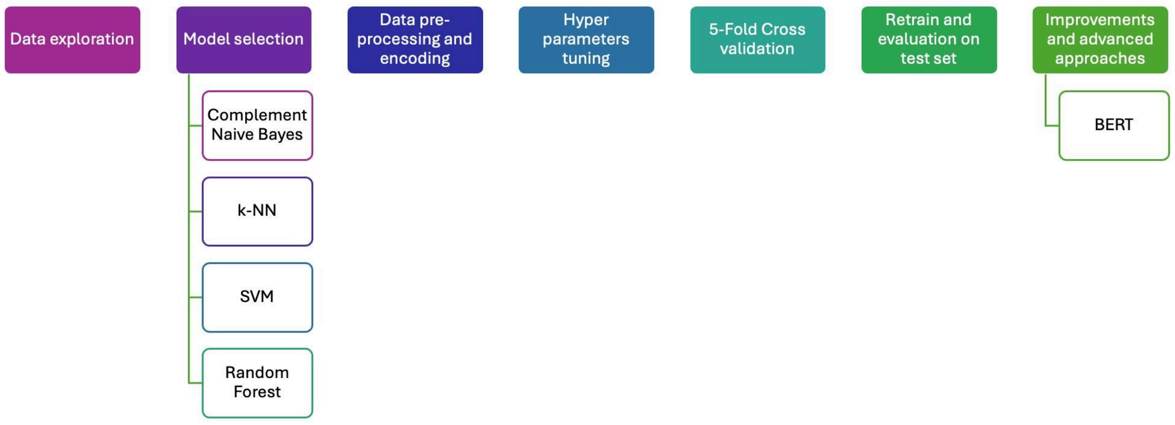


Figure 4. *Graphical representation of the steps followed in our metodology.*

Since data exploration was detailed previously, we now describe subsequent steps.

Model selection
The choice of models was driven by the need to address class imbalance while ensuring good performance and reasonable computation time. We selected:

- Complement Naive Bayes [1]
- K-Nearest Neighbors
- Support Vector Machines
- Random Forest

A more detailed discussion on the choices of these algorithms is presented in the "Models" section.

Data pre-processing and encoding
We used TF.IDF vectors via scikit-learn's TfIdfVectorizer, which often perform better in text classification due to their effective capture of term relevance. More details are in the "Vectorization" section. Parameter choices for the vectorizer were coupled with model hyperparameter tuning. Lowercasing was consistently applied across all models.

Hyperparameters tuning
For each of the four models we performed hyperparameter tuning with nested-cross validation by using the functions available in *scikit-learn*. Here we report more in detail the parameters tuned for each model.

- **Complement Naive Bayes (CNB):** parameters ngram_range, min_df, max_df, stop_words for the TfIdfVectorizer. Search method employed was grid search.
- **K-NN:** parameters ngram_range, min_df, max_df, stop_words for the TfIdfVectorizer. Parameters n_neighbors, metric, weights for the model. Search method employed was grid search.
- **SVM:** parameters kernel, C and gamma for the model. Since we noticed the best hyperparameters for the vectorizer seemed to be consistent for the previous models and SVM is computationally more intense, to reduce the search space we used those instead of tuning them again. Search method employed was grid search. We fix the parameter class_weight of the model to 'balanced'.
- **Random Forest:** parameters n_estimators, criterion, max_features, max_depth, min_samples_split, min_samples_leaf, bootstrap, class_weight, ccp_alpha for the model. Search method employed was randomised search with 10 trials per fold due to higher computational demands of the algorithm.

We used a micro-averaged F1-score to select the best combination of hyperparameters, since it takes into account both precision and recall and accuracy is not the best metric for our dataset due to class imbalance.

5-fold cross-validation
Models were subsequently evaluated with 5-fold cross validation with the best hyperparameters found.

Final training and evaluation
Once assessed the model performance through cross validation, we re-trained the models on the full training set and obtained predictions using the separate test set.

Improvements and advanced approaches
After obtaining preliminary results, we decided to apply some advanced techniques like BERT, which is better described in the "Technical Depth" section, and to further investigate how some pre-processing steps could affect performance.
More in detail:

- We assessed for one of the best performing models, Complement Naive Bayes, if stemming with different kinds of stemmers (offered by nltk package) would impact the performance
- We tried employing a different strategy for encoding documents by using pre-trained vectors with GloVe and using SVM for the classification (see Technical Depth for details)
- For Complement Naive Bayes we tried using also CountVectorizer

Results

Hyperparameter tuning

The optimal combination of hyperparameters for the TF.IDF vectorizer, with a F1-score of 86.6%, is:

- ngram_range = (1, 2)
- min_df = 1
- Max_df = 1.0
- stop_words = list of stop words provided by nltk package

As previously noted, this combination proved optimal for the K-NN model as well and was subsequently adopted for both the SVM and Random Forest models. The Document-Term Matrix generated from this configuration comprises 21,643 documents and 106,917 features.

Here we report the other hyperparameter configurations found for each model.
K-NN, best score 82.8%:

- metric = 'cosine'
- n_neighbors = 11
- weights = 'distance'

SVM, best score 87.2%:

- C = 1,
- gamma = 'auto'
- kernel = 'linear'

Random forest, best score 83.2%:

- n_estimators = 300
- min_samples_split = 10
- min_samples_leaf = 1
- max_features = 'sqrt'
- max_depth = None
- criterion = 'entropy'
- class_weight = 'balanced_subsample'
- ccp_alpha = 0.0
- bootstrap = False

Cross validation

From the observation of the learning curves in **Figure 5**, it seems like all models are slightly overfitting, with K-NN and Random Forest having the worst generalisation capabilities.

- CNB and SVM: the models start to generalise better when there are more examples. However, from the tendency of the test curve to plateau, only increasing the size of the train set could potentially provide a limited benefit to performance.
- K-NN and Random Forest: lower scores might indicate other underlying issues such as a sub-optimal choice of hyperparameters or a difficulty to operate with high dimensional vectors

Overall, the best performing models are CNB and SVM (**Figure 6**) with mean F1-scores respectively 87.3% and 87.1%, followed by Random forest (83.7%) and K-NN (82.9%).

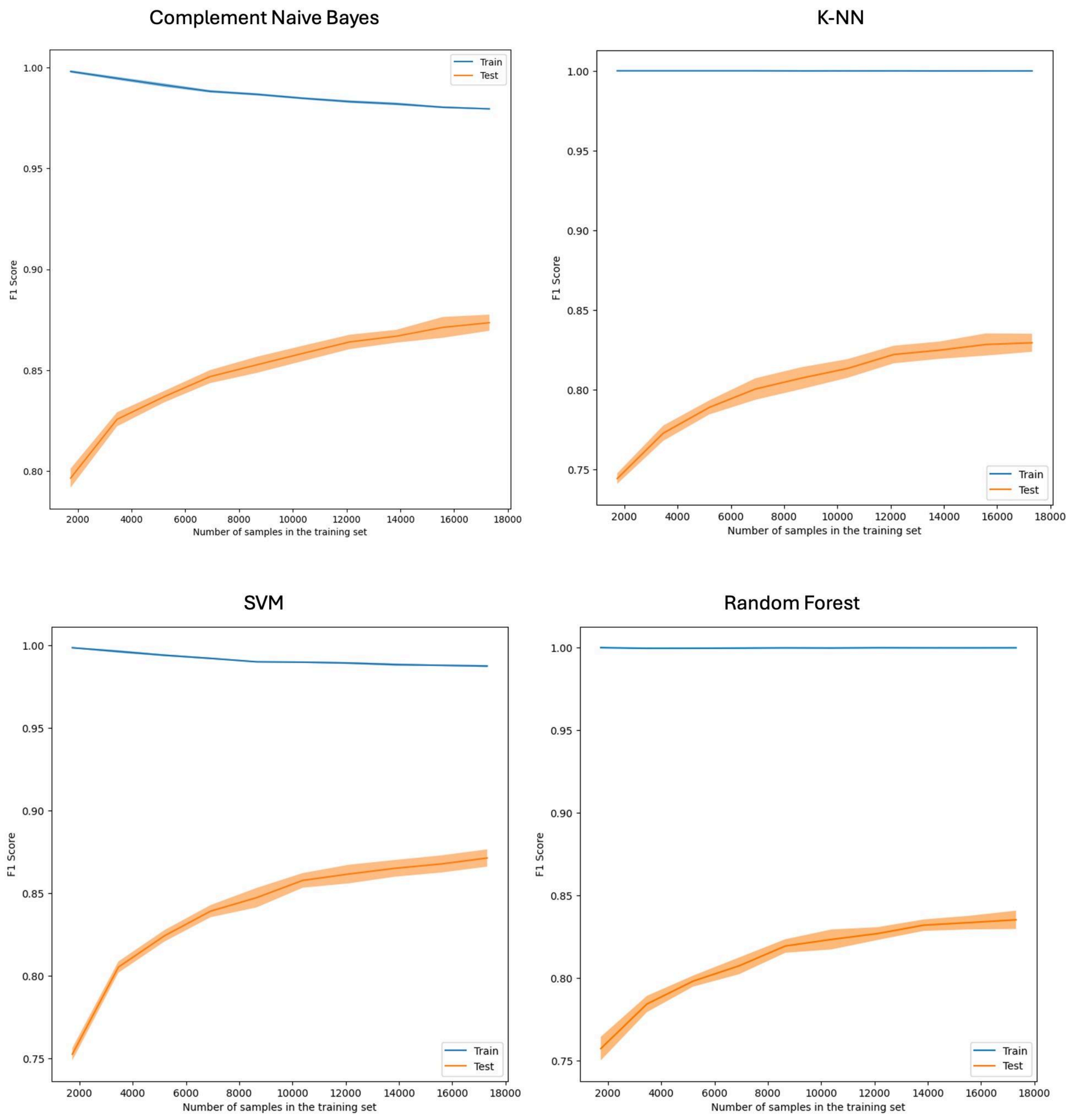


Figure 5. *Learning curves for the four models. All models show signs of overfitting, with training score being systematically higher than test score and a typical plateau tendency of the test curve, more pronounced in K-NN and Random Forest with overall worse scores.*

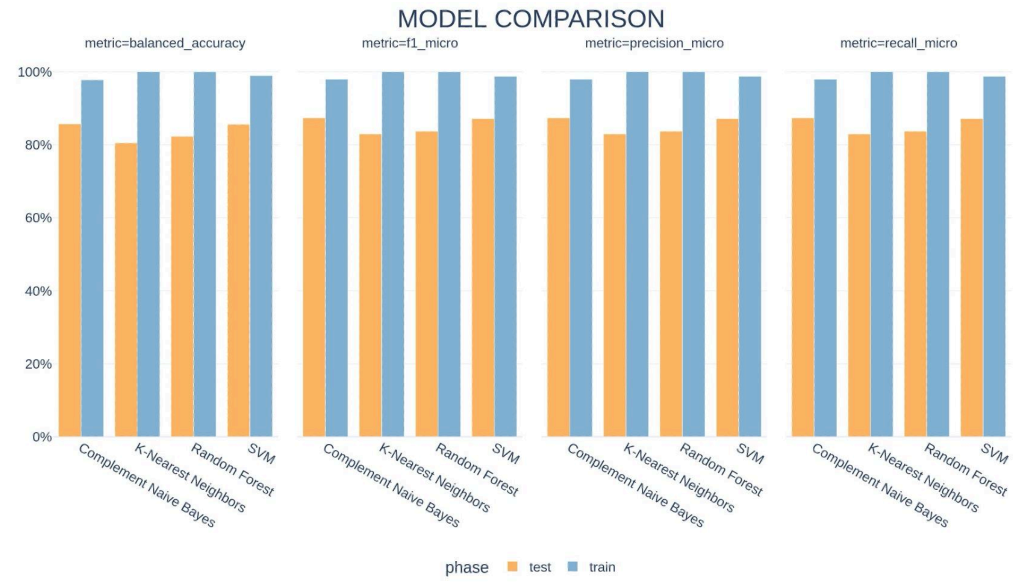
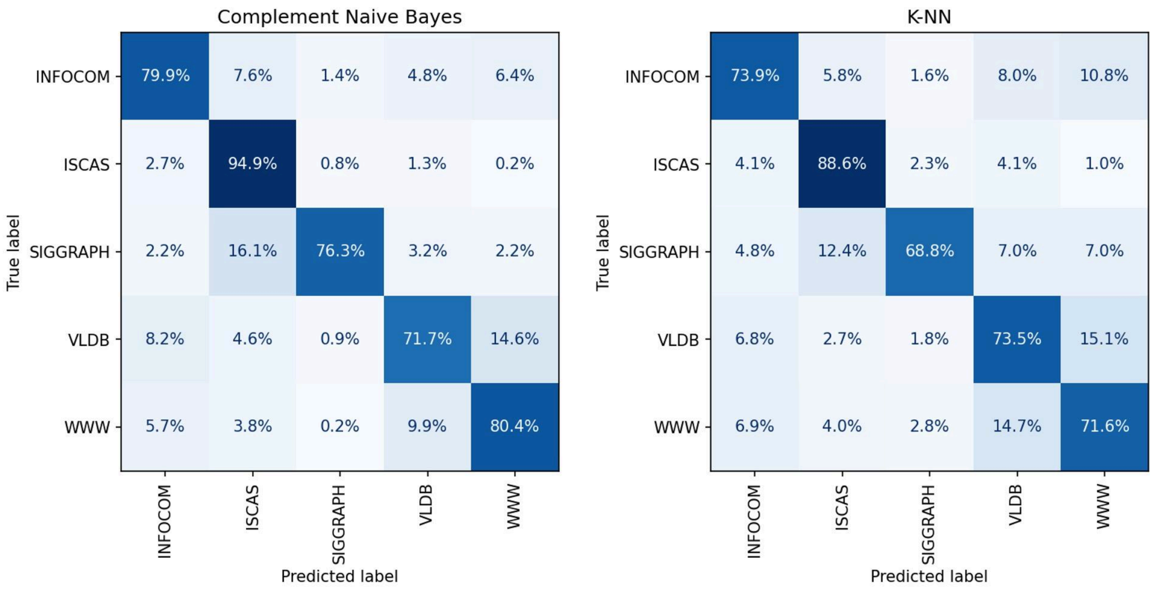


Figure 6. *Comparison of the four models with four different metrics. Reported values are averages across the 5-folds of cross validation, for both train (blue) and validation (orange).*

Final training and evaluation

We report in **Table 1** the scores for each model trained on the whole training set and evaluated on the separate test set. In **Figure 7** we report the confusion matrices: despite the models being able to handle class imbalance, it is evident from the matrices that the best represented classes are also the ones with least classification errors.

We also plotted feature importances (top 10) retrieved from the Random Forest model (**Figure 8**) to assess which features had the most impact in terms of mean decrease in impurity and verified that those correspond largely with the most frequent words identified in the frequency analysis presented in the "Source Data" section.



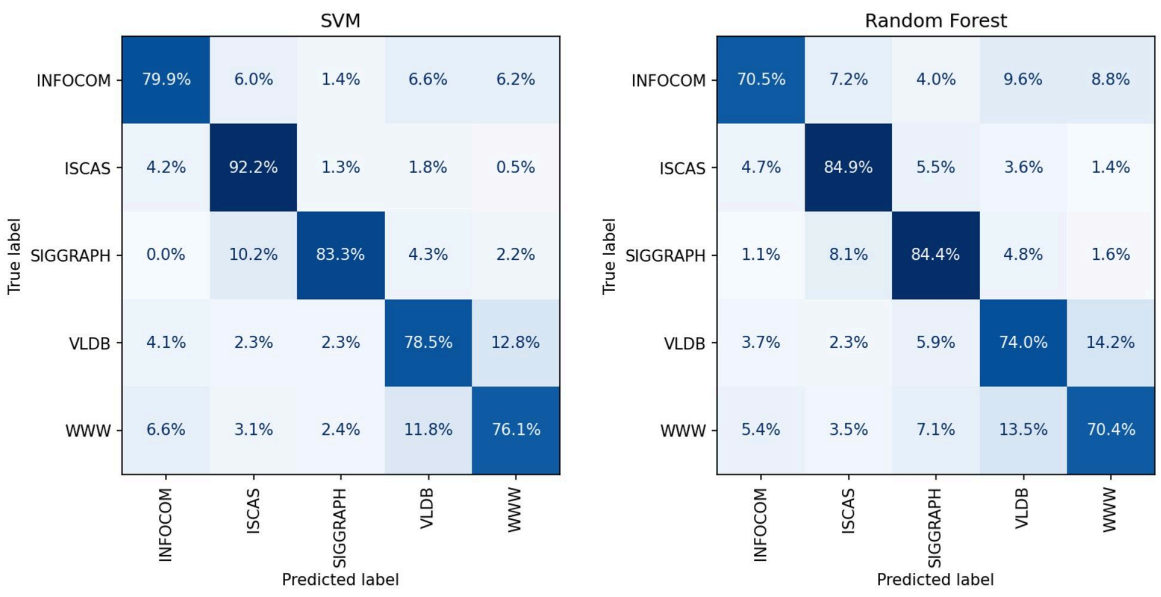


Figure 7. *Confusion matrices for the four models re-trained on the whole training set and tested on a separate test set.*

Balanced accuracy	Precision (micro)	Recall (micro)	F1 Score (micro)	Model
80.64%	88.32%	88.32%	88.32%	CNB
75.30%	82.24%	82.24%	82.24%	KNN
82.02%	86.98%	86.98%	86.98%	SVM
76.84%	80.20%	80.20%	80.20%	RF

Table 1. *Performance scores for the four models re-trained on the whole training set and tested on a separate test set.*

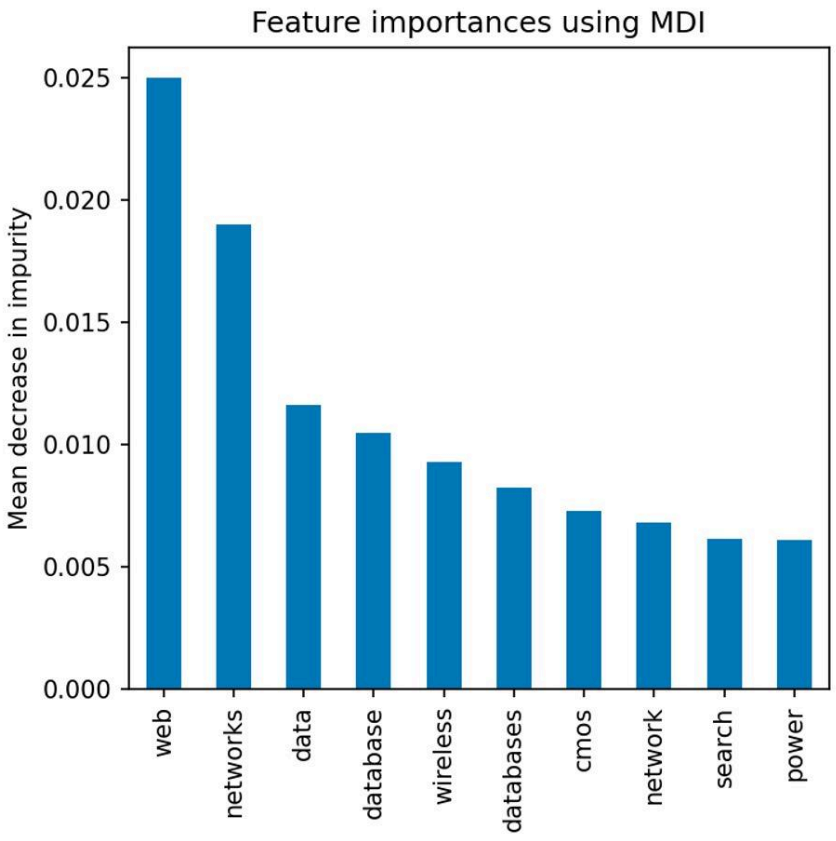


Figure 8. *Feature importances extracted from the Random Forest model. Here we report only the top 10 most important features by mean decrease in impurity.*

Improvements

We selected CNB both for the good performance and the low computational demand to perform another round of hyperparameter tuning to assess if stemming could impact performance significantly. In particular, we tested the effects of 3 different stemmers offered by the nltk package: SnowballStemmer, PorterStemmer and LancasterStemmer. The results suggest that stemming has minimal impact on the performance, with the best configuration having an F1-score of 86.3% versus the 86.6% score of the baseline configuration. We therefore decided to exclude stemming from the pipeline to avoid unnecessary complexity and cut computational costs.

Again with CNB, we tested an alternative vectorization method by using CountVectorizer instead of TF.IDF, tuning for the same hyperparameters as reported for the TF.IDF vectorizer. We found little difference again, with the best configuration landing an F1-score of 86.9%.

Reliability of results

One of the key limitations affecting reliability is the dependence on the specific models chosen—SVM, k-NN, Random Forest, and Complement Naive Bayes. Each model has intrinsic biases and assumptions that may not perfectly align with the real-world distribution of academic paper titles. For instance, SVM and Random Forest might perform exceptionally well on the training data but could potentially overfit, especially if hyperparameter tuning is not adequately addressed. k-NN's performance is heavily dependent on the choice of the distance metric and the value of k, which can significantly influence its generalisation capability. For the Random forest model, in particular, we opted for a less thorough exploration of the hyperparameter space due to higher computational demands of the model.

In natural language processing, especially in tasks like conference recommendation, class imbalance can severely skew the results. We utilised methods such as adjusting class weights in models and chose models that are naturally best suited to deal with this issue to ensure minority classes were adequately represented. This not only helps in achieving more accurate models but also in enhancing the reliability of predictions across less common categories.

To enhance the reliability of our findings, we implemented several validation techniques. Cross-validation was used extensively to assess model performance more robustly, reducing the risk of overfitting and ensuring that the models are tested across a variety of data scenarios. Additionally, we validated our final models against an independent test set that was not used during the model training or tuning stages.

Technical depth

Vectorization with GloVe embeddings
GloVe is a pre-trained word embedding method that captures global word-word co-occurrence statistics from a corpus, providing a dense vector representation for each word. By integrating GloVe embeddings, we aimed to leverage semantic relationships and contextual information embedded in the text data, which traditional methods like TF-IDF cannot capture. In particular we used the GloVe embeddings "glove.6B.100d" [2] in combination with SVM and each document is represented by the mean vector of all the words present in the title. The results however did not outperform our approach with TF-IDF, since the best hyperparameters for SVM with nested cross validation achieved an F1-score of only 75.3%. This might be due to either 100 features being too few for the classification and/or the fact that embeddings for some keywords like domain names or technical terms might not be available in the pre-trained embeddings, thus providing no representation for those words.

BERT

BERT (Bidirectional Encoder Representations from Transformers) [3] is a state-of-the-art NLP model to understand the context of a text and capture the dependencies within a sentence. It is a pre-trained model and can be fine-tuned to suit our necessities. This was implemented using the PyTorch library. The selected model was bert-base-uncased and trained by optimising F1-score as the dataset had imbalanced classes. It was trained only for 5 epochs as the training time for each epoch took around 6 hours. An F1-score of 0.84 was obtained indicating a good generalisation ability. The classification accuracy ranges from 53% for SIGGRAPH to 91% for ISCAS. A possible explanation for this could be that the imbalanced dataset causes the model to learn features of majority classes much better than the rest.

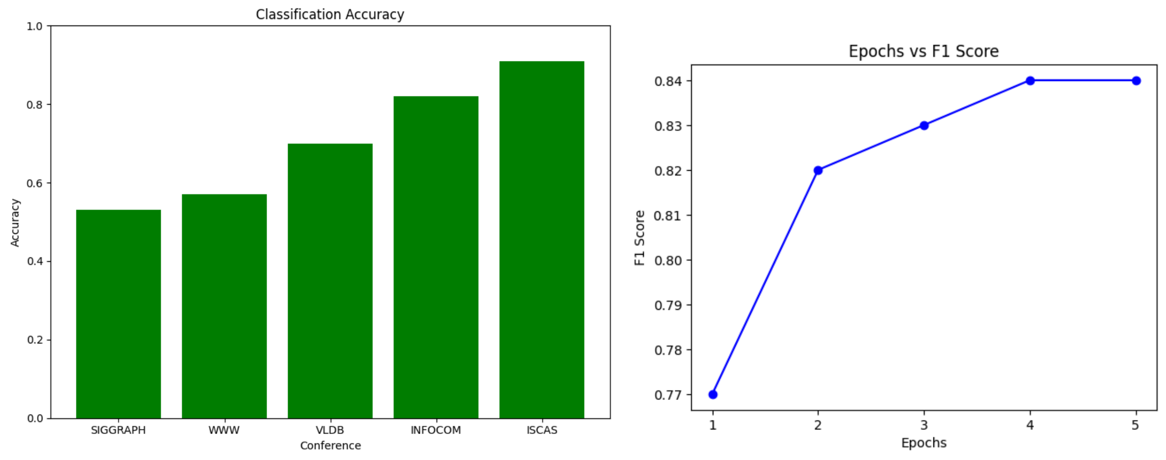


Figure 9. *Classification accuracy and F1 score obtained after training BERT*

Conclusions & recommendations

In the conclusion of our project, we found that Complement Naive Bayes (CNB) and Support Vector Machines (SVM) emerged as the best-performing models for classifying academic paper titles into conference categories. Despite their strong performance, we observed signs of slight overfitting in these models, which indicates a potential need for further refinement in model training and parameter tuning. Models like BERT perform extremely well but are also computationally expensive. Hence, it is recommended to use lighter models like RNN or LSTM.

Class imbalance remains a significant challenge, potentially skewing the predictive performance of our models towards more frequently occurring classes.
To better address this, we recommend experimenting with more sophisticated resampling techniques such as Synthetic Minority Over-sampling Technique (SMOTE) or Adaptive Synthetic (ADASYN) sampling. These methods can generate synthetic samples for minority classes during training, thereby providing a more balanced dataset without losing valuable information. Feature reduction techniques like PCA or t-SNE could also prove beneficial.
Expanding the dataset to include a broader array of disciplines and conferences and continually updating the training data to include new terminology and emerging research

trends will also be crucial to maintaining the relevance and accuracy of the classification system.

We recommend that stakeholders in the academic community, such as conference organisers and academic publishers, consider integrating these machine learning techniques into their systems to automate and enhance the process of sorting and categorising submission entries. This could significantly streamline their operations, reduce manual workload, and increase the accuracy of matching papers to appropriate conferences. Over the course of time, the system can be personalised and adapted to a researcher's area of expertise.

Reflection

Throughout the project, we encountered several challenges, particularly in managing high-dimensional data, addressing class imbalance, and mitigating model overfitting. These issues were tackled using a variety of techniques learned during the course, such as TF-IDF vectorization to manage high-dimensional text data, and machine learning algorithms like Complement Naive Bayes and Support Vector Machines, which are particularly suited to text classification tasks.

Transparency is crucial, so it is important to note that we utilised ChatGPT to assist in various aspects of the project. ChatGPT was primarily used for scientific literature recommendation, code debugging, code snippets generation mainly for plotting purposes and for spell checking and concepts' summarization in this report.

Vectorization

The choice of the Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer for our project leverages its ability to emphasize relevant terms within the dataset. TF-IDF excels in highlighting key words that are frequently mentioned in specific documents but rare across the whole corpus. This capability is crucial for parsing succinct academic titles laden with significant keywords, allowing the vectorizer to enhance text classification by focusing on terms critical to the content and thematic essence of the papers.

However, TF-IDF does have notable limitations. It does not handle out-of-vocabulary words well, potentially diminishing classification performance when encountering documents with unfamiliar vocabulary. Moreover, TF-IDF considers words in isolation, ignoring the context in which they appear, and only captures the order of terms to a limited extent through n-grams. This could lead to a loss in capturing the full semantic complexity of texts, although this is less of a concern in our project due to the brief nature of paper titles. Additionally, TF-IDF produces a sparse representation of text, which is typically manageable in the context of paper titles but might pose challenges in applications with longer documents. These characteristics should be carefully considered when adapting this approach to other applications beyond classifying short academic titles.

Preprocessing

We employed a series of preprocessing techniques designed to optimize the textual data for analysis and classification. These techniques include tokenization, lowercasing, removal of stopwords, and the application of the TF-IDF vectorization.

Tokenization serves as the foundational step, segmenting text into individual terms or tokens. This is essential for creating a manageable and analyzable feature space, as each token becomes a feature for our machine learning models. Lowercasing standardizes the text, reducing complexity by ensuring that words are treated uniformly regardless of their place in a sentence or title. This homogenization prevents the same words in different cases from being interpreted as distinct, thereby reducing feature space dimensionality.

Removal of stopwords is particularly impactful, eliminating commonly used words that offer little to no value in distinguishing between different conferences. By filtering out these stopwords, we enhance the focus on meaningful content, which significantly improves model accuracy and efficiency.

Lastly, we employed TF-IDF vectorization to encode our documents (see "Vectorization").

As previously mentioned, we also examined the impact of stemming on the performance of Complement Naive Bayes, finding little to no improvement, so we decided to omit it from the pipeline to reduce computational load and unnecessary complexity.

Models

The chosen models—Support Vector Machine (SVM), k-Nearest Neighbors (k-NN), Complement Naive Bayes (CNB), Random Forest and BERT—each bring distinct advantages to the table that align well with the specific requirements of text classification.

Complement Naive Bayes (CNB), designed specifically for imbalanced datasets, proves valuable in adjusting the bias typically encountered in standard Naive Bayes when dealing with classes of varying sizes—a common scenario in our dataset of conference titles.

k-Nearest Neighbors (k-NN) excels by measuring distances between text instances, utilizing cosine similarity to assess the thematic similarity between paper titles. This model's reliance on the local interpolation of its nearest neighbors ensures high accuracy when classifiable examples are dense and well-represented.

Support Vector Machines (SVM) are particularly effective due to their ability to handle high-dimensional data, a common feature in text classification where each word or phrase may be treated as a separate feature. SVM's capacity for creating optimal hyperplanes in a multidimensional space makes it ideal for distinguishing between complex patterns found in paper titles.

Random Forest leverages ensemble learning to enhance decision accuracy and control overfitting, a frequent challenge in NLP tasks. It also provides insights into feature importance, helping us understand which terms most significantly influence conference classification.

BERT generates contextual embeddings for each word in a sentence capturing its meaning in the sentence, i.e., the context and other dependencies. It is also a transfer learning approach which enables us to leverage our system's performance using the experience of the pre-trained model. Finally, it is highly flexible and can be fine tuned easily unlike the previous models that would require feature engineering.

References

[1] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the Poor Assumptions of Naive Bayes Text Classifiers".

[2] J. Pennington, R. Socher, and C. Manning, "GloVe: Global Vectors for Word Representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds., Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. doi: 10.3115/v1/D14-1162.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv, May 24, 2019. doi: 10.48550/arXiv.1810.04805.