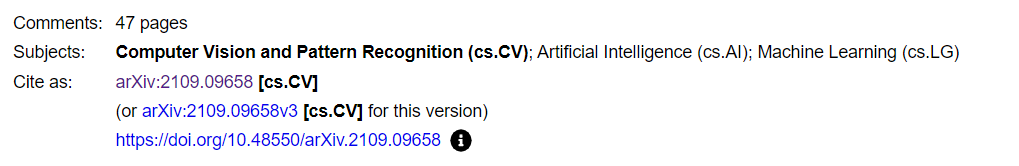
**Methodology:**

**1.1 Dataset:**

About 52,000 publicly available research paper metadata records, which included the title, abstract, and research area or subjects, were gathered for this study from arxiv.org. There were more than 100 unique labeled subjects in total among the metadata, and most of the research papers had at least one subject (example: **cs.CV**). However, we confined our study to 88 unique subjects as we were only able to map 88 out of 100 to their actual category names (example: **cs.CV** to Computer Vision and Pattern Recognition).



Example: Metadata From Arxiv

**1.2 Preprocessing:**

Processing raw metadata presented significant challenges in terms of data preprocessing as the classification of tile and abstract was a multi-label classification problem. In this step, we attempted to transform the data so that it could be easily parsed by a machine. We went through a few data preprocessing steps to accomplish that, we removed the rows with missing subjects.

Stop words, which are frequently used in the abstract and were eliminated using the NLTK library \cite{nltk}. The practice of reducing word variants to their shared basic form stem is known as stemming which enables a reduction in vocabulary while improving recall \cite{(Darwish and Magdy, 2014)}, we used Porter stemmer from NLTK library. Further, we used NLTK’s \cite{} lemmatizer to clean up and standardise the text. It allowed us to group together various inflected forms of a word, such as various tenses and plurals, and treat them as a single entity.

We used the scikit-learn library's MultiLabelBinarizer class to convert the labels for each sample to one hot encoding format. This is useful for multi-label classification because it allows the classifier to predict multiple labels for each sample instead of just one.

**2. Methodology:**

In this study, we trained models on Word2Vec \cite{mikolov} and TF-IDF features, further we fine-tuned pretrained uncased BERT \cite{devlin} on our dataset.

**2.1. TF-IDF Features:**

In our work, we applied the term frequency-inverse document frequency (TF-IDF) method to convert textual data into a numerical feature matrix using the TF-IDF Vectorizer from the Scikit-Learn library. This technique transforms each text in the dataset into a vector of values that represents the weight of each word in the text relative to the entire dataset.

We set the minimum document frequency (min\_df) to 0.8, which helped us weed out uncommon terms that might not be relevant for the classification task by ensuring that they appear in at least 80% of the documents. Additionally, we set max features to 10000, which tells the TfidfVectorizer to only maintain the top 10,000 words based on their TF-IDF values. This helped the classifier perform better by lowering the complexity of the feature matrix.

**2.1.1 Results with TF-IDF Features:**

| **Model** | Hamming Loss | Precision(Macro) | Precision(Sample) | Precision(Average) | Recall (Macro) | Recal (Sample) | Recall (Average) | F1 Score  (Macro) | F1 Score (Micro) | F1 Score (Average) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Log. Reg | 0.01 | 0.09 | 0.95 | 0.82 | 0.03 | 0.77 | 0.66 | 0.03 | 0.78 | 0.7 |
| GaussianNB | 0.02 | 0.6 | 0.6 | 0.78 | 0.29 | 0.74 | 0.69 | 0.35 | 0.61 | 0.69 |
| **RandomFores** | **0.01** | **0.71** | **0.96** | **0.96** | **0.28** | **0.85** | **0.78** | **0.38** | **0.86** | **0.83** |
| XGBoost | 0.01 | 0.56 | 0.92 | 0.84 | 0.17 | 0.78 | 0.68 | 0.24 | 0.77 | 0.71 |

**Table 1. Feature - TF-IDF**

**2.2. Word2Vec Features:**

The Word2Vec technique creates a ***d*** dimensional vector representation of all **N** word in the text that accurately captures its meaning and context. Word2Vec technique tries to predict the target word from a set of context words. From a list of context words, the algorithm tries to predict the target word. In order to do this analytically, select the necessary rows from **W0** and then compute the total of the context word embeddings. These "word embeddings" are vector representations that can be created in a variety of ways. We used *glove-wiki-gigaword-200* from Gensim library’s API to get word embedding of the word in the text. For **N** words in the text we had **N** x 200 sized vector, to create a single vector representation of a text we took the average of a series of word embeddings. By taking the mean or average of the word embeddings in a document we can get a single vector representation that captures the overall meaning of the document.

| **Model** | Hamming Loss | Precision(Macro) | Precision(Sample) | Precision(Average) | Recall (Macro) | Recal (Sample) | Recall (Average) | F1 Score  (Macro) | F1 Score (Micro) | F1 Score (Average) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Log. Reg | 0.01 | 0.02 | 0.6 | 0.58 | 0.01 | 0.46 | 0.39 | 0.01 | 0.52 | 0.46 |
| GaussianNB | 0.12 | 0.03 | 0.13 | 0.56 | 0.39 | 0.67 | 0.64 | 0.04 | 0.16 | 0.56 |
| **RandomFores** | **0.01** | **0.71** | **0.84** | **0.86** | **0.28** | **0.79** | **0.73** | **0.38** | **0.78** | **0.75** |
| XGBoost | 0.01 | 0.38 | 0.77 | 0.68 | 0.11 | 0.68 | 0.58 | 0.15 | 0.65 | 0.57 |

**Table 2. Feature - Word2Vec**

**2.3. Fine Tuning BERT:**

In this study, we used pretrained uncased BERT base model and fine tuned it on our dataset. We used the Adam optimizer to converge more quickly and avoid oscillation by changing the learning rate for each parameter based on its previous gradients. Considering the sequence/text length of **256**, which determines the maximum number of tokens that can be processed by the model in a single input sequence we trained model for **4** epochs**.** Furthermore, we used weight decay of **0.01** to prevent overfitting by adding a penalty to the weights of the model and learning rate of **1e-05** for stable training and better convergence.

| **Model** | Hamming Loss | Precision(Macro) | Precision(Sample) | Precision(Average) | Recall (Macro) | Recal (Sample) | Recall (Average) | F1 Score  (Macro) | F1 Score (Micro) | F1 Score (Average) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BERT | 0.02 | 0.06 | 0.95 | 0.79 | 0.02 | 0.78 | 0.67 | 0.03 | 0.78 | 0.70 |

**Table 3. BERT Fine-Tuned**

**3. Evaluation Metrics:**

There are several different evaluation metrics that can be used for multi-label classification. However, we used the most appropriate metric for our task.

1. Hamming loss: The Hamming loss is a metric that measures the average number of label errors made by the model.
2. Precision (macro):