Analysis of the Classification of Research Papers dataset

Michele Intrevado, Matteo Leoncini, Domenico Santone
Università degli studi dell'Aquila, Italy
Emails: michele.intrevado@student.univaq.it
matteo.leoncini@student.univaq.it
domenico.santone@student.univaq.it

Academic year: 2023–2024

Contents

Introduction

Scientific paper classification in machine learning has emerged as a critical and relevant issue within the research community.
This challenge is especially useful in the face of the vast amount of information available, where researchers often struggle to keep up with the sheer volume of publications. Classification of scientific papers has the potential to improve information retrieval and facilitate knowledge discovery. In addition, effective classification systems can have a significant impact on research efficiency by providing researchers with more focused and relevant information. Reducing the complexities of this kind of classification through machine learning algorithms has the potential to accelerate the way researchers navigate and contribute to scientific knowledge.
Numerous research papers have been published online as well as offline with the increasing advance of computer and information technologies, which makes it difficult for users to search and categorize their interesting research papers for a specific subject. Therefore, it is desired that these huge numbers of research papers are systematically classified with similar subjects so that users can find their interesting research papers easily and conveniently. Typically, finding research papers on specific topics or subjects is a time-consuming activity. For example, researchers usually spend a long time on the Internet to find interesting papers and are bored because the information they are looking for is not retrieved efficiently because the papers are not grouped in their topics or subjects for easy and fast access.[1]
To develop our analysis we used the following tools:

- Jupyter Notebook
- Python
- Pandas (data manipulation)
- Keras & Tensorflow (model implementation, model evaluation)
- Scikit learn (model evaluation)
- Seaborn (data visualization)

Dataset

Our dataset contains several scientific papers, which were previously classified into six different categories: Computer Science, Physics, Mathematics, Statistics, Quantitative Biology, and Quantitative Finance. Each paper can belong to one or more categories; therefore, the task at hand was NLP (Natural Language Processing), specifically a multi-label classification.
The first time it is opened, the dataset looks as shown in Figure 1.

	TITLE	ABSTRACT	Computer Science	Physics	Mathematics	Statistics	Quantitative Biology	Quantitative Finance
0	Reconstructing Subject-Specific Effect Maps	Predictive models allow subject-specific inf...	1	0	0	0	0	0
1	Rotation Invariance Neural Network	Rotation invariance and translation invarian...	1	0	0	0	0	0
2	Spherical polyharmonics and Poisson kernels fo...	We introduce and develop the notion of spher...	0	0	1	0	0	0
3	A finite element approximation for the stochas...	The stochastic Landau--Lifshitz--Gilbert (LL...	0	0	1	0	0	0
4	Comparative study of Discrete Wavelet Transfor...	Fourier-transform infrared (FTIR) spectra o...	1	0	0	1	0	0

Figure 1: Raw dataset structure

The `'TITLE'` feature contains the title of the paper, the `'ABSTRACT'` feature contains the content of the abstract, and then category membership is represented with a one-hot-encoded vector.

Analysis

We did some analysis on the dataset to better interpret the data and we noticed, by plotting the distribution of categories, that the dataset is unbalanced because, as is visible in Figure 2, there is a large disparity between the categories.

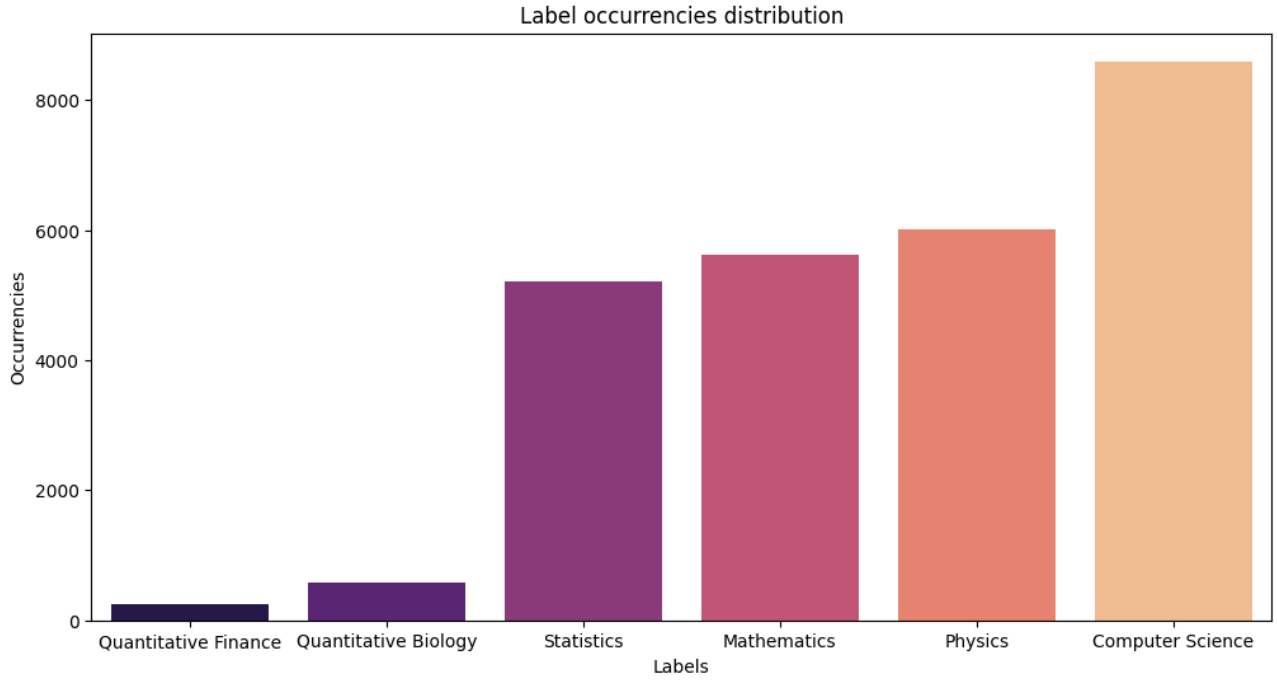

Figure 2: Label occurrences distribution

In particular, the majority of papers are about Computer Science, while a very few papers are about Quantitative Finance and Biology.
Another stage of the analysis involved plotting the most frequent words within the abstracts, as can be seen in Figure 3.

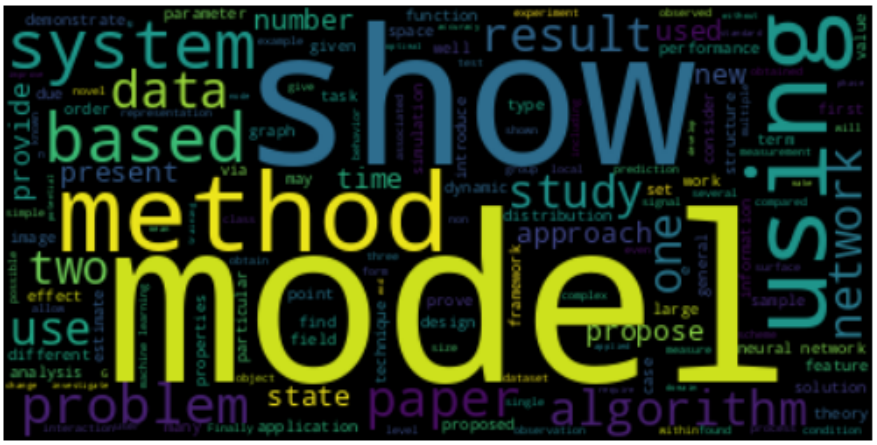

Figure 3: Most frequent words

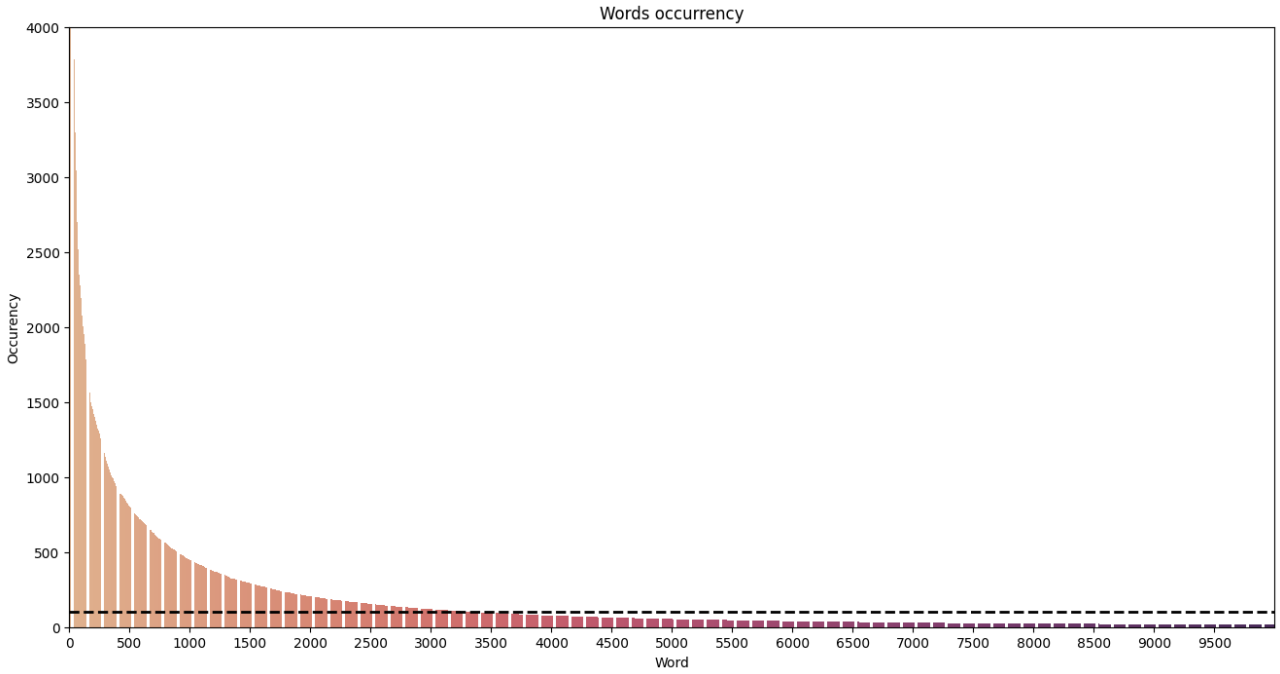

Figure 4: Enter Caption

Preprocessing

Regarding the preprocessing phase, we decided to merge the text of the abstract and the title of the paper into a single feature called ”text” to preprocess and input the resulting text into a vectorizer.
About preprocessing the text, we made all text lowercase and removed quotation marks and all non-alphabet characters. Next, we replaced the words written in contracted form with expanded form using a manually defined dictionary. Finally, we removed the stop-words defined in the ntlk (Natural Language Toolkit Library). The code in which we performed these operations is reported in listing:

```
def prepare_text(text: str):
    text = text.lower()
    text = text.replace('"', '')
    text = re.sub("[^a-zA-Z]", " ", text)
    text = " ".join([contractions_dict.get(word, word) for word in text.split()])
    words = [word for word in text.split() if word not in stop_words and len(word) > 1]
    return " ".join(words).strip()
```

Further Analysis

Once the preprocessing has been done, we analyzed the number of occurrences for each word, as shown in figure 4. As we can see, we have a horizontal line that corresponds to the words repeated at least 100 times, in fact in our model we only will consider the words that are above this threshold.

Then we analyzed the plot in Figure 5 the distribution of paper lengths. We can notice that the average length of papers is around 110.

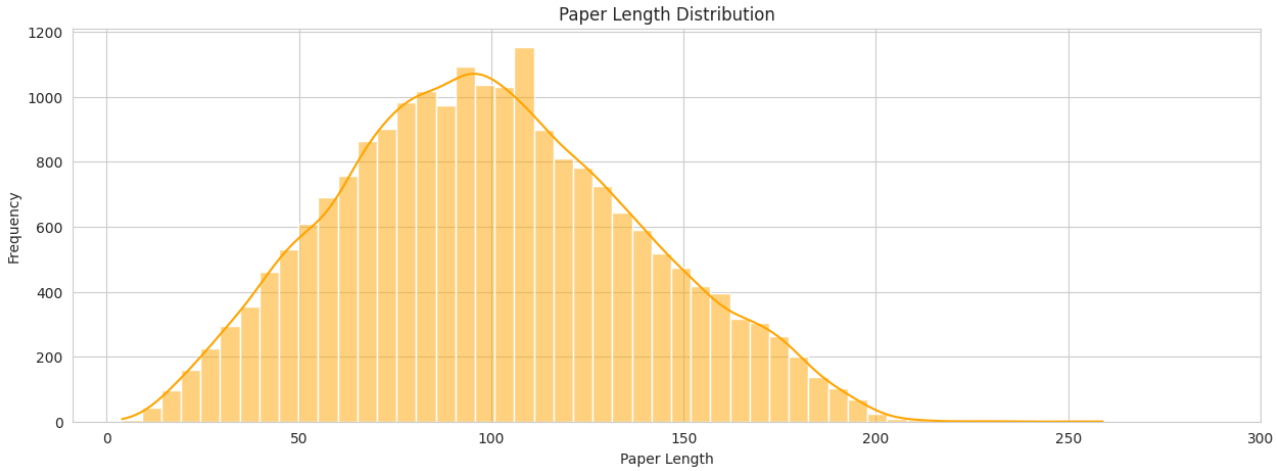

Figure 5: Paper length distribution

Given the plots above we decided to set the hyperparameters for the vectorization as follows:

- **MAX_FEATURES** that represents the number of unique words to be considered to 4000
- **MAX_SEQUENCE** that represent the maximum sequence length to 200

Vectorization

Text vectorization is the process of converting text into numerical representations (or “vectors”) that can be understood by machine learning models. It transforms unstructured text into structured numeric data of representing the semantic meaning of text in a mathematical format.
This process allows for a variety of NLP tasks like document classification, sentiment analysis, enhancing search engines, and so on. The implementation is the following:

```
MAX_FEATURES = 4000
MAX_SEQUENCE = 200 # Maximum length of sequences
tv = TextVectorization(max_tokens=MAX_FEATURES, output_sequence_length=MAX_SEQUENCE, output_mode='
    int')
tv.adapt(X_data)

pickle.dump({'config': tv.get_config(),
             'weights': tv.get_weights()}
            , open("vectorization.pkl", "wb"))

X = np.array(tv(X_data))

X_train, X_test, y_train, y_test = train_test_split(X, y_data, test_size=0.2, random_state=10)
```

The previously identified MAX_FEATURES and MAX_SEQUENCE parameters are passed to the `TextVectorization` method. Next, with the adapt() method the layer will build a vocabulary of all string tokens seen in the dataset, sorted by occurrence count, with ties broken by sort order of the tokens. Since we have set the max_tokens parameter, the vocabulary will be truncated to the max_tokens size.
After saving the model to a file for later use, we performed a split on the vectorized data.

Model Explanation

We implemented a function that creates and returns our Keras model that will be trained with our dataset, as shown in figure 6.

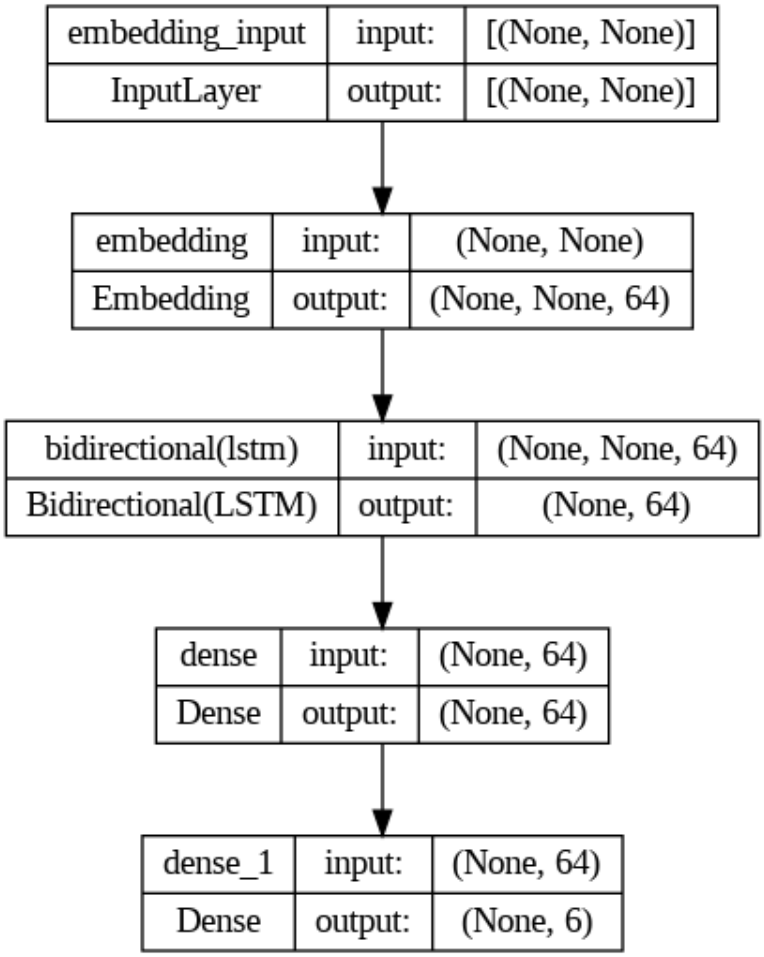

Figure 6: Model Architecture

As we can notice, the used model has been implemented with the following architecture:

- One Embedding layer: this layer requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Keras Tokenizer API also provided with Keras. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.
- One Bidirectional LSTM layer: this kind of layer processes the embedded sequences in both forward and backward directions. It trains two layers on the input sequence: one LSTM layer on the input sequence and another LSTM layer on the reversed copy of the input sequence.
- Two Dense layer: this kind of layer is a fully connected layer that connects every output of the previous layer to every perceptron in the current dense layer. In particular, regarding the last layer we have used the sigmoid activation function that returns a value between 0 and 1. This comes in our help, because this kind of function returns the probability that the text given to our deep neural network belongs to each category of the dataset.

As last aspect of our architecture we can consider the number of parameters for each layer, that are shown in the figure 7. In the first layer we have 256064 obtained by the following formula $(4000 + 1) * 64$ neurons. In the second layer we have 24832 parameters obtained by multiplying $4 * ((32 + 1) * 64 + 64^2)$, where 32 is the size of input, 64 is the size of output (perceptrons of the next layer). In the third layer we have 4160 parameters that is obtained by $(64 + 1) * 64$. In the last layer we have 390 parameters obtained by $(64 + 1) * 6$.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, None, 64)          256064

 bidirectional_1 (Bidirectio  (None, 64)                24832
 nal)

 dense_2 (Dense)              (None, 64)                4160

 dense_3 (Dense)              (None, 6)                 390

=================================================================
Total params: 285,446

Trainable params: 285,446

Non-trainable params: 0

_________________________________________________________________
```

Figure 7: Model Summary

Model Evaluation

Once the model has been trained on the dataset (by using the 10-fold cross validation) we eveluated its performances by executing some predictions on the test set.
We obtained the following results:

Metric	Result
Accuracy	90.21%
Precision	76.28%
Recall	77.46%
F-1 Score	76.67%

To better understand the prediction results, we compared the results for the individual categories using the following confusion matrices shown in figure 8

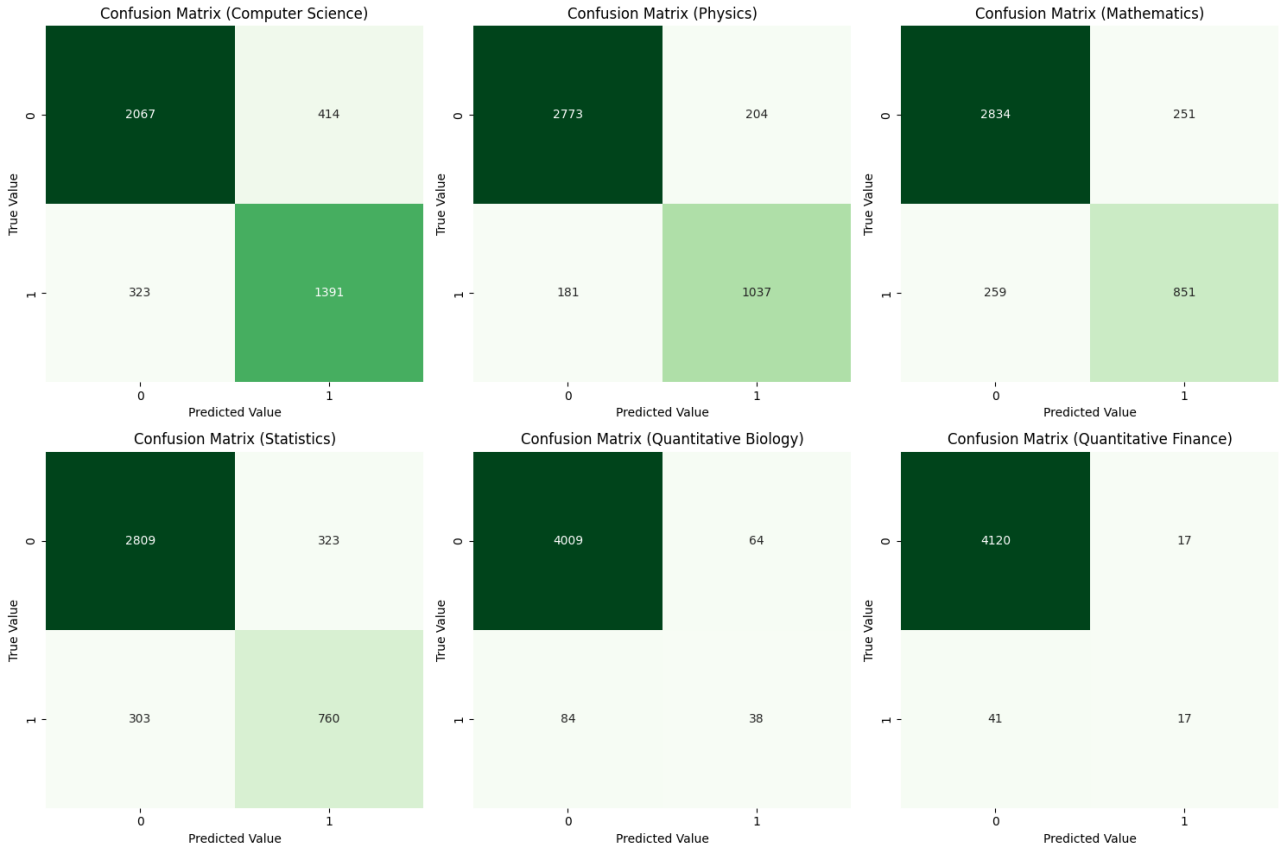

Figure 8: Confusion Matrices

For those classification problems that have a severe class imbalance, the default threshold can result in poor performance. As such, a possible approach to improve the performance of a classifier that predicts probabilities on an imbalanced classification problem is to tune the threshold used to map probabilities to class labels.
For each category we calculated the threshold used to consider each paper belonging to it. Each threshold has been computed by considering the precision and the recall metrics and trying to find the couple of values on the precision-recall curve that minimize the absolute difference. [2]

Model Testing Using IEEE Explore

To show an interactive use of our model we defined a web scraping function that given an IEEE Explore link can retrieve the title and abstract of the paper and use them to predict the categories of the paper as you can see below:

```
import requests
from bs4 import BeautifulSoup

def extract_title_and_abstract(ieee_link):
    # Make a request to the IEEE link
    headers = {'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36'}
    response = requests.get(ieee_link, headers=headers)

    if response.status_code == 200:
        # Parse the HTML content
        soup = BeautifulSoup(response.text, 'html.parser')
        # Extract title
        title_element = soup.find("meta", property="og:title")["content"]
        title = title_element.strip() if title_element else None

        # Extract abstract
        abstract_element = soup.find("meta", property="og:description")["content"]
        abstract = abstract_element.strip() if abstract_element else None

        return title, abstract
    else:
        print(f"Failed to retrieve content. Status code: {response.status_code}")
        return None, None

while True:
  ieee_link = input("Insert the link of the paper from IEEE (type quit to block the execution): ")
  if ieee_link == "quit":
    break
  title, abstract = extract_title_and_abstract(ieee_link)
  if title and abstract:
      print("Title:", title)
      print("Abstract:", abstract)
      print("-"*50)
      print(format_prediction(title, abstract))
      print()
  else:
      print("Error extracting title and abstract.")
```

Web Dashboard

We implemented a web Dashboard using the Python library Dash [3] in order to use the model directly from a web interface. You can see the code inside the `dashboard.py` file. The interface is the following:

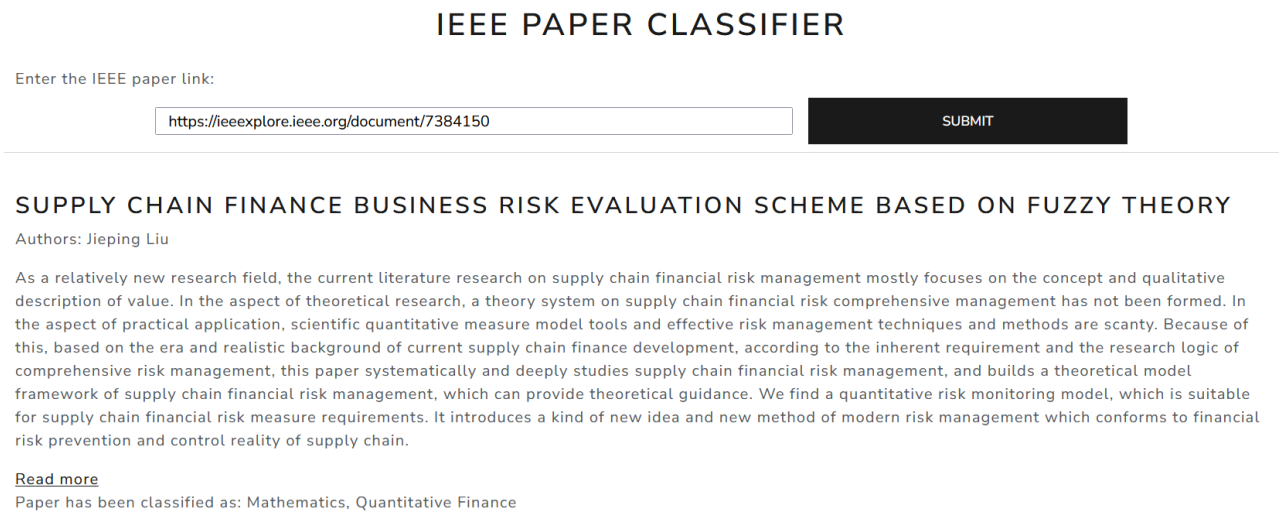

Figure 9: Dashboard Interface

To run it you can simply type `python dashboard.py` and the web server will be accessible at the address `http://localhost:8050`.
The logic for retrieving the prediction and printing the output is the same of the jupyter notebook with a little bit of interactivity given from the web interface. The follwing code defines the layout:

```
app.layout = html.Div([
    html.H1("IEEE Paper Classifier", className="text-center my-4"),
    html.Div([
        html.Label("Enter the IEEE paper link:", className="form-label mx-2"),
        dcc.Input(id="ieee-link",
                  type="text",
                  placeholder="Enter the IEEE paper link",
                  className="mx-2 mb-2 w-50"),
        html.Button("Submit", id="submit-button", className="btn btn-primary mx-2 mb-2 w-25"),
        html.Hr(className="w-100 d-block"),
        dcc.Loading(
            id="loading-1",
            type="default",
            children=[
                html.Div(id="loading-output-1", className="my-4 d-flex justify-content-center")
            ],
            color="#00ccbc"
        )
    ], className="row justify-content-center align-items-center"),
], className="container my-4 justify-content-center")
```

And this callback is the one that retrieves the data from the website and performs the calls to the functions to get the prediction and update the components accordingly:

```
@app.callback(
    Output("loading-output-1", "children"),
    Input("submit-button", "n_clicks"),
    State("ieee-link", "value"),
    prevent_initial_call=True
)
def update_output(n_clicks, ieee_link):
    # Attempt to extract title and abstract
    logging.info(f"Extracting content from {ieee_link}")
    title, abstract, authors = extract_title_and_abstract(ieee_link)

    if title and abstract and authors:
```

```
        # Classify the paper and create the output content
        classification = classify_paper(title, abstract)
        output_content = html.Div([
            html.H3(title),
            html.P(children=[html.Strong("Authors: "), ", ".join([author["content"] for author in authors])]),
            html.P(abstract),
            html.A("Read more", href=ieee_link, target="_blank", id="ieee-link-output"),
            html.P(classification)
        ])
        return [output_content]
    else:
        # Display error message if extraction fails
        logging.error(f"Failed to retrieve content from {ieee_link}. Please check the link and try again.")
        return [html.P("Failed to retrieve content. Please check the link and try again.")]
```

Comparison With Baselines

Since we didn't had dedicated baselines to compare with we tried to submit our solution to the hackaton website [4] and we obtained the following score:

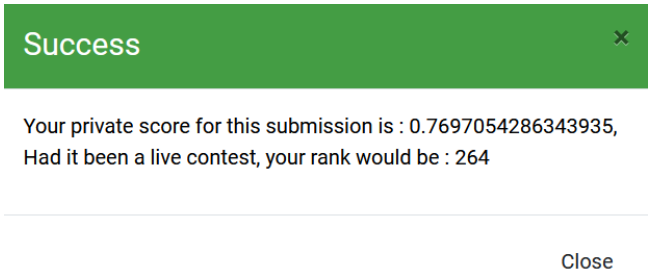

Figure 10: Score from the Analytics Vidhya Hackaton

Our score however was not very distant from the top partecipants that got about 0.86 points so we're pretty satisfied with the job we've done. Probably with more knowledge about natural language processing techniques and pretrained models (the top are all using Bert variations) we'll be able to do better in the future.

Conclusions

In conclusion we reached successfully the object to implement a machine learning model to classify scientific papers and we obtained a pretty good result compared to the baseline.
We were in our first ML experience and we hope to learn how to improve this type of models in the future with more specific and advanced techniques.
It was very challenging for us since this was our very first course about Neural Networks, Machine Learning and Natural Language Processing techniques.

References

[1] Sang-Woon Kim and Joon-Min Gil. Research paper classification systems based on tf-idf and lda schemes. *Human-centric Computing and Information Sciences*, 9:1–21, 2019.

[2] Takaya Saito and Marc Rehmsmeier. The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. *PLOS ONE*, 10(3):e0118432, March 2015.

[3] Dash plotly documentation. `https://dash.plotly.com/`. Accessed: 2024-02-03.

[4] Janatahack: Independence day 2020 ml hackathon. `https://datahack.analyticsvidhya.com/contest/janatahack-independence-day-2020-ml-hackathon`. Accessed: 2024-02-04.