PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

Ministry of Higher Education and Scientific Research

University of Blida1

Faculty of Sciences

Computer Science Department

By

Mr. AHMED SERIR Aymen and

Mr. HADJ RAMDANE Said

For obtaining the Master's degree

Field: Mathematics and Computer Sciences

Speciality: Natural Language Processing

Contribution to a Transfer Learning Approach for a Multilabel Biomedical Text Classification

Members of the thesis committee:

KAMECHE H	University of Blida 1	President
CHERIF-ZAHAR A	University of Blida 1	Examinator
MEZZI M	University of Blida 1	Supervisor

Blida, September 28th, 2022

Acknowledgment

I would like to thank several people for their help and support during the production of this thesis.

I would like to express my deepest gratitude to my advisor, Mrs. Mezzi Melyara, whose sincerity and encouragement I will never forget.

Mrs. Mezzi Melyara has been an inspiration as I hurdled through the path of this Masters's degree. She is the true definition of an ideal teacher and the ultimate role model. This thesis would not have been possible without her, whose guidance from the initial step of research enabled me to develop an understanding of the subject. I am thankful for the extraordinary experiences she arranged for me and for providing opportunities for me to grow professionally. It is an honour to learn from her.

Also, I greatly appreciate my research partner Aymen, for his professional great work, excellent encouragement, rigor, and sense of responsibility. Thanks a lot, Aymen.

I am grateful for my parents whose constant love and support keep me motivated and confident. My accomplishments and success are because they believed in me. My Deepest thanks to my brothers and family, who keep me grounded, remind me of what is important in life, and are always supportive of my adventures. Finally, I owe my deepest gratitude to my friends, I am forever thankful for the unconditional support throughout the entire thesis process and every day.

Said

Acknowledgment

First and foremost, I would like to recognize the invaluable assistance of our advisor Mrs. Mezzi and thank her for her supervision and support. and kindly express my gratitude for her effort, without her assistance this thesis would have not seen the light of day, she was very focused on helping us through this, and I wish her the very best in her life.

My thanks go to my research partner and new friend Said for his very professional teamwork and his exceptional skills in planning and organizing. I learned a lot of things from him and I hope this experience will lead to more work together in the future.

I would also like to thank my parents for their support and encouragement, and I wish their son made them proud.

Finally, I would like to thank my friends for sharing the best moods during this as it was very needed and appreciated.

Aymen

Abstract

In the age of big data, textual data is more important than ever, with an ever-increasing size and an abundant production of digital documents, particularly in the biomedical field as a consequence of the convergence between medical computer science and bioinformatics. In addition to the fact that these textual data are usually expressed in an unstructured form (i.e., natural language), which makes their automated processing more difficult. Moreover the rapid growth of the biomedical literature, makes the manual indexing approaches more complex, time-consuming and error-prone. Thus, automated classification is essential. Despite the many efforts, classification complete biomedical texts according to segments specific to these texts, such as their title and summary, remains a real challenge.

In this thesis we investigate state of the art approaches in classifying biomedical texts so that we can compare with pre-trained models that we have tested. After performing tests on different artificial intelligence models: BioBERT, Roberta, XLNet, we found out that the ideal model for classifying biomedical texts is BioBERT with an average F1 score of 85,1% which was very similar to the roBERTa model with a score of 85% which unlike BioBERT, was not pre-trained on biomedical texts and with XLNet performing slightly worse with a score of 83%.

Finally, we deployed the three above-mentioned models and developed an Online User Interface on the Hugging Face Platform in order to test and show the classification results clearly and easily.

Keywords: *Automatic Text Classification, Multilabel Classification, Automatic Medical Language Processing, Deep Learning*.

ملخص

في عصر البيانات الضخمة، أصبحت البيانات النصية أكثر أهمية من أي وقت مضى، مع حجم متزايد باستمرار وإنتاج وفير للوثائق الرقمية، لا سيما في مجال الطب الحيوي كنتيجة للتقارب بين علوم الكمبيوتر الطبية والمعلوماتية الحيوية. بالإضافة إلى حقيقة أن هذه البيانات النصية يتم التعبير عنها عادةً في شكل غير منظم (أي لغة طبيعية)، مما يجعل معالجتها الآلية أكثر صعوبة. علاوة على ذلك، فإن النمو السريع للأدب الطبي الحيوي، يجعل منهجيات الفهرسة اليدوية أكثر تعقيدًا، وتستغرق وقتًا طويلاً، وعرضة للخطأ. وبالتالي، فإن التصنيف الآلي ضروري. على الرغم من الجهود العديدة، إلا أن تصنيف النصوص الطبية الحيوية الكاملة وفقًا لأجزاء خاصة بهذه النصوص، مثل العنوان والملخص، لا يزال يمثل تحديًا حقيقيًا.

في هذه الأطروحة نتحرى عن أحدث الأساليب في تصنيف النصوص الطبية الحيوية حتى نتمكن من المقارنة مع النماذج المدربة مسبقًا التي اختبرناها. بعد إجراء اختبارات على نماذج ذكاء اصطناعي مختلفة: BioBERT، و Roberta، و XLNet ، اكتشفنا أن النموذج المثالي لتصنيف النصوص الطبية الحيوية هو BioBERT بمتوسط درجة F1 يبلغ 85.1٪ والتي كانت مشابهة جدًا لنموذج roBERTa مع مجموع نقاط. 85٪ منها على عكس BioBERT لم يتم تدريبها مسبقًا على النصوص الطبية الحيوية وكان أداء XLNet أسوأ قليلاً بنسبة 83٪.

أخيرًا، قمنا بنشر النماذج الثلاثة المذكورة أعلاه وقمنا بتطوير واجهة مستخدم عبر الإنترنت على منصة Hugging Face Platform من أجل اختبار نتائج التصنيف وإظهارها بوضوح وسهولة.

الكلمات الرئيسية: *التصنيف التلقائي للنص، التصنيف متعدد الملصقات، المعالجة التلقائية للغة الطبية، التعلم العميق*

Résumé

À l'ère du big data, les données textuelles sont plus importantes que jamais, avec une taille toujours croissante et une production abondante de documents numériques, notamment dans le domaine biomédical, conséquence de la convergence entre l'informatique médicale et la bioinformatique. Outre le fait que ces données textuelles sont généralement exprimées sous une forme non structurée (c'est-à-dire en langage naturel), ce qui rend leur traitement automatisé plus difficile. De plus, la croissance rapide de la littérature biomédicale rend les approches d'indexation manuelle plus complexes, chronophages et sujettes aux erreurs. Ainsi, la classification automatisée est essentielle. Malgré les nombreux efforts, la classification des textes biomédicaux complets selon des segments propres à ces textes, tels que leur titre et leur résumé, reste un véritable défi.

Dans cette thèse, nous étudions des approches de pointe en matière de classification de textes biomédicaux afin de pouvoir les comparer avec des modèles pré-entraînés que nous avons testés. Après avoir effectué des tests sur différents modèles d'intelligence artificielle : BioBERT, Roberta, XLNet, nous avons découvert que le modèle idéal pour classer les textes biomédicaux est BioBERT avec un score F1 moyen de 85,1 %, ce qui était très similaire au modèle roBERTa avec un score de 85 % qui, contrairement à BioBERT, n'étaient pas pré-formés sur les textes biomédicaux et avec XLNet, les performances étaient légèrement inférieures avec un score de 83 %.

Enfin, nous avons déployé les trois modèles mentionnés ci-dessus et développé une interface utilisateur en ligne sur la plateforme Hugging Face afin de tester et d'afficher clairement et facilement les résultats de la classification.

Mots-clés : *Classification Automatique des Textes, Classification Multi-étiquettes, Traitement Automatique du Langage Médical, Deep Learning.*

Table of Content

List of Figures

Table of Tables

GENERAL

INTROCUTION

1 General Introduction

1.1 General Context

Biomedical Textual data is increasing rapidly as a consequence of the convergence between medical computer science and bioinformatics (scientific articles biomedical reports, medical reports, patient discharge summaries, etc..), In addition, one problem is that these textual data are usually expressed in an unstructured form (i.e. natural language), which makes their automated processing increasingly difficult. Thus, effective access to useful information is difficult. To do this, an appropriate representation of the document's texts is crucial. controlled and hierarchically-organized vocabulary, such as the Medical Subject Heading (MeSH®) Thesaurus that was produced by the National Library of Medicine, are widely used to index biomedical texts to facilitate access to useful information.

The rapid growth of the biomedical literature, makes the manual indexing approaches more complex, time-consuming and error-prone. Thus, automated indexation is essential. Despite the many efforts, indexing complete biomedical texts according to segments specific to these texts, such as their title and summary, remains a real challenge. Moreover, with large amounts of data, using partial information to annotate documents is promising. However, the classification of texts in the medical field is difficult because of two main problems: first, it has some orthographic and grammatical errors, and second, the medical text contains complex medical vocabularies, medical measures, and acronyms which has problems with high-dimensionality and data sparsity.

1.2 Research Problematic

With the advent of deep learning, such as convolutional neural networks (CNNs) and recurrent neurons (RNN) which are nowadays widely used in images, signals and other applications, interest in testing these techniques in the field of medical texts manifested itself and it is precisely in this context that we place in order to test and compare deep learning methods that would be effective for biomedical text classification.

1.3 Research Objective

This paper will attempt at understanding and making a comparative research study on three state of the art automatic biomedical text classification theses and the methods employed in them, separately we will test three pre-trained models and compare their results, in the hopes of studying and understanding the current progress in the field and to give a future perspective on biomedical text classification.

1.4 Thesis organization

This document is organized as follows:

The first chapter deals with explaining automatic text classification. The chapter begins by presenting the famous techniques and principles of automatic text classification including: KNN, decision trees, random forests, LR and SVM, and lastly the statistical evaluation metrics used in order to evaluate a model.

In the next chapter we explore automatic text classification using deep learning neural networks like ANN, CNN, RNN, some deep learning applications and the challenges faced in the field.

In the related works section we study three new research theses relevant to solving our problem with the classification of biomedical texts, we'll do a comparative study of deep learning algorithms that were applied in the theses to get an overall review of state of the art methods.

The next chapter deals with conceptualizing our approach to the problem, we will give a clear look at the dataset used and it's pre-processing, we will explain the concept behind transfer learning and detail the BERT model as well as the pre-trained models used in this thesis and the difference between them.

The next chapter deals with implementing the approach: the hardware and software resources used, the implementation code of each model and finally evaluating the models, later we compare the results with the approaches followed by

the theses explored in the related works chapter, we showcase the models we've trained in a user friendly interface.

And finally a conclusion that includes a summary of the whole process and our perspective on it.

CHAPTER I :

Automatic Classification

Chapter 1: Automatic Classification

1. Introduction

Automatic classification is a process for managing text and unstructured information by categorizing or clustering text. By labeling natural language texts with relevant categories from a predefined set, automatic document classification enables users to organize content quickly and efficiently.

While manual document classification may be highly detailed and accurate, it is time-consuming and subjective. Automatic document classification is faster, scalable, and more objective. It provides organizations with a more systematic and consistent classification and can be useful in more complex, nuanced contexts, such as business-specific content. Machine learning and Artificial Intelligence can boost the speed and efficiency of automatic document classification.

In this chapter, we are dealing with defining automatic classification and briefly explaining some basic concepts such as: Machine Learning in Text Classification, Deep Learning, and finally evaluation Metrics.

2. Machine Learning in Text Classification

Also called a classifier, model, or hypothesis, the classification must assign a Boolean value to each pair $(dj, ci) \in D \times C$ where D is a set of documents and C a set of categories. For the learning phase, the algorithm requires examples of which we know the choice of the expert. From the characteristics extracted from each of the examples, it infers classification rules that are contained in the values of the parameters of its model. The classifiers are distinguished by [1]:

- The complexity of their model (multi-layer neural networks have a high separation capacity.
- The interpretability of their parameters (decision trees provide rules understandable by a human).
- Their performance (the naive Bayesian classifier often gives fewer good results).
- Their scaling up (the size of the training data limits the wide-margin separator classifier (SVM)).

- Their speed in the learning and classification phases (the k-nearest-neighbor (KNN) method that queries the data is sometimes longer in the classification but does not require learning).
- The complexity of the model must be adapted to the amount of data available for its training. Indeed, a complex model can offer better results, but since it is more sensitive to noise than a simpler model, its performance may collapse due to over-fitting. In machine learning theory, the VC dimension (Vapnik Chervonenkis) measures this complexity.

Next, we give a brief introduction of the classical classifiers in the litterature.

2.1. KNN (K-nearest-neighbor)

2.1.1. Definition

This method is based on the assumption that similar documents in the content are classified in the same categories. Related documents or neighbors of the document to be classified are retrieved to extract their categories.

The notion of proximity remains to be defined: For the algorithm to work best on a particular dataset we need to choose the most appropriate distance metric accordingly, here we will tackle some of the distance metrics available, but we are only going to talk about a few widely used ones. Euclidean distance function is the most popular one among all of them as it is set default in the SKlearn KNN classifier library in python [2]. So here are some of the distances used:

2.1.2. Minkowski Distance

It is a metric intended for real-valued vector spaces. We can calculate Minkowski distance only in a normed vector space, which means in a space where distances can be represented as a vector that has a length and the lengths cannot be negative [3].

There are a few conditions that the distance metric must satisfy:

Non-negativity: $d(x, y) >= 0$
Identity: $d(x, y) = 0$ if and only if $x == y$
Symmetry: $d(x, y) = d(y, x)$
Triangle Inequality: $d(x, y) + d(y, z) >= d(x, z)$

$$\left(\sum_{i=1}^{n} |x_i - y_i|^P \right)^{1/P}$$

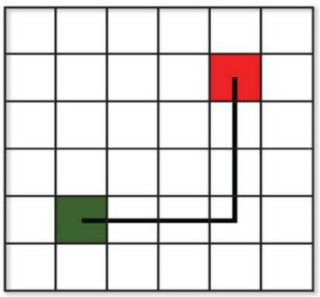
This above formula for Minkowski distance is a generalized form and we can alter it to get different distance metrics.

The p value in the formula can be manipulated to give us different distances like:

- p = 1, when p is set to 1 we get Manhattan distance
- p = 2, when p is set to 2 we get Euclidean distance

2.1.3. Manhattan Distance

This distance is also known as taxicab distance or city block distance, because of the way this distance is calculated. The distance between two points is the sum of the absolute differences of their Cartesian coordinates.



Manhattan Distance

As we know we get the formula for Manhattan distance by substituting p=1 in the Minkowski Distance formula:

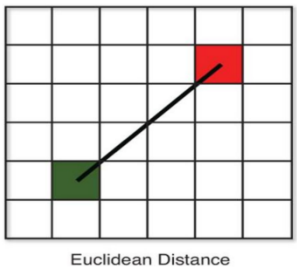
$$d = \sum_{i=1}^{n} |x_i - y_i|$$

This distance is preferred over the Euclidean distance when we have a case of high dimensionality [3].

2.1.4. Euclidean Distance

This distance is the most widely used one as it is the default metric that SKlearn library of Python uses for K-Nearest Neighbor. It is a measure of the true straight line distance between two points in Euclidean space [3].

It can be used by setting the value of p equal to 2 in Minkowski distance metric:



Euclidean Distance

2.1.5. Cosine Distance

This distance metric is used mainly to calculate similarity between two vectors. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in the same direction. It is often used to measure document similarity in text analysis. When used with KNN this distance gives us a new perspective to a business problem and lets us find some hidden information in the data which we didn't see using the above two distance matrices.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

It is also used in text analytics to find similarities between two documents by the number of times a particular set of words appear in it.

The formula for Cosine distance is:

$$\cos\theta = \frac{\vec{a}.\vec{b}}{\|\vec{a}\|.\|\vec{b}\|}$$

Using this distance, we get values between 0 and 1, where 0 means the vectors are 100% similar to each other and 1 means they are not similar at all [3].

2.1.6. Jaccard Distance

The Jaccard coefficient is a similar method of comparison to the Cosine Similarity due to how both methods compare one type of attribute distributed among all data. The Jaccard approach looks at the two data sets and finds the incident where both values are equal to 1. So, the resulting value reflects how many 1 to 1 matches occur in comparison to the total number of data points. This is also known as the frequency that 1 to 1 match, which is what the Cosine Similarity looks for, how frequent a certain attribute occurs. It is extremely sensitive to small sample sizes and may give erroneous results, especially with very small data sets with missing observations.

The formula for Jaccard index is:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

2.1.7. Hamming Distance

The Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, the Hamming distance is the number of bit positions in which the two bits are different. The Hamming distance method looks at the whole data and finds when data points are similar and dissimilar one to one. The Hamming distance gives the result of how many attributes were different.

This is used mostly when you one-hot encode your data and need to find distances between the two binary vectors.

Suppose we have two strings “ABCDE” and “AGDDF” of same length and we want to find the hamming distance between these. We will go letter by letter in each string and see if they are similar or in our example the first letters of both strings are similar, then the second is not similar and so on:

ABC**D**E and **A**GD**D**F

When we are done doing this we will see that only two letters were similar and three were dissimilar in the strings. Hence, the Hamming Distance here will be 3. Note that larger the Hamming Distance between two strings, more dissimilar will be those strings (and vice versa) [3].

2.2. Decision Trees

2.2.1. Definition

From a geometric point of view, the idea is to cut the space up to obtain regions containing a homogeneous classification. A region is said to be pure when all the documents contained therein are classified in the same category. For a new document to be classified, we place it in the representation space and we look in which region it is located.

Initially, space consists of a single region, the entropy is then maximum.

Let $Cj, j = 1 \ldots n,$

The space is sliced successively so that the entropy reduction (gain of information) is maximum at each step. If, in the end, all the regions are pure then the entropy is zero and the information gain is equal to the initial entropy. To avoid obtaining regions containing too few examples, which represents a risk of over-learning, the tree is pruned.

2.2.2. Types of Decision Trees

Types of decision trees are based on the type of target variable we have. It can be of two types [4]:

- *Categorical Variable Decision Tree:* Decision Tree which has a categorical target variable then it is called a Categorical variable decision tree.
- *Continuous Variable Decision Tree*: Decision Tree has a continuous target variable then it is called Continuous Variable Decision Tree.
- *Entropy:* Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.

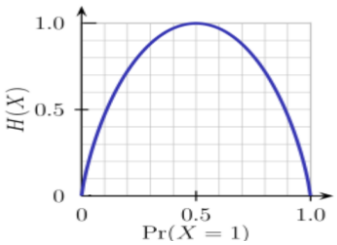


Figure 1: Entropy Graph.

From the above graph, it is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance of perfectly determining the outcome.

- Information Gain: Information gain is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.

$$Information\ Gain(T, X) = Entropy(T) - Entropy(T, X)$$

Where T: Current state and X: Selected attribute.

- *Gini Index*: the Gini index is a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of

each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.

$$Gini = 1 - \sum_{i=1}^{C} (P_i)^2$$

2.3. Random Forest

2.3.1. Definition

Random forest is an algorithm that is used widely in Classification and Regression problems.

The main flaw of the decision tree is that it is very sensitive to the initial data set. Removing or adding a few examples can completely change the tree. To take this instability into account, the random forest algorithm introduces a part of chance in its construction. Its principle is to build a large number of decision trees by bootstrap and to take the vote of each tree to classify. The classification therefore does not depend on a single tree but on several trees (forest) created from the same dataset but different from each other (random) [4].

2.3.2. Bagging (Bootstrap Aggregation)

Decision trees are very sensitive to the data they are trained on, small changes to the training data set can result in a significantly different tree structure. The random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees.

2.3.3. Difference between random forest and decision tree

The critical difference between the random forest algorithm and decision tree is that decision trees are graphs that illustrate all possible outcomes of a decision using a branching approach. In contrast, the random forest algorithm output are a set of decision trees that work according to the output [4].

Table 1: Difference between Decision Tree and Random Forest.

Decision Tree	Random Forest
A decision tree is a tree-like model of decisions along with possible outcomes in a diagram.	A classification algorithm consisting of many decision trees combined to get a more accurate result as compared to a single tree.
There is always a scope for over-fitting, caused by the presence of variance.	Random forest algorithm avoids and prevents over-fitting by using multiple trees.
The results are not accurate.	This gives accurate and precise results.
Decision trees require low computation. Thus, reducing time to implement and carrying low accuracy.	This consumes more computation. The process of generation and analyzing is time consuming.
It is easy to visualize. The only task is to fit the decision tree model.	This has complex visualization as it determines the pattern behind the data.

2.4. Linear Regression

Linear regression is a simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression. If there is a single input variable (x), such linear regression is called simple linear regression. And if there is more than one input variable, such linear regression is called multiple linear regression. The linear regression model gives a sloped straight line describing the relationship within the variables.

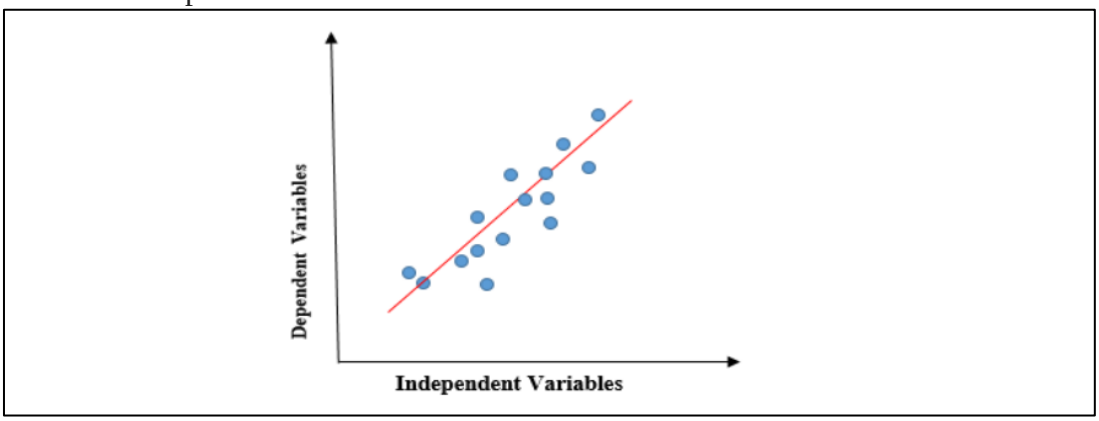


Figure 2: Linear Regression.

The above graph presents the linear relationship between the dependent variable and independent variables. When the value of x (independent variable) increases, the value of y (dependent variable) is likewise increasing. The red line is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best.

2.5. Support Vector Machines

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes.

SVMs are more commonly used in classification problems SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes, as shown in the image below.

Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set [5].

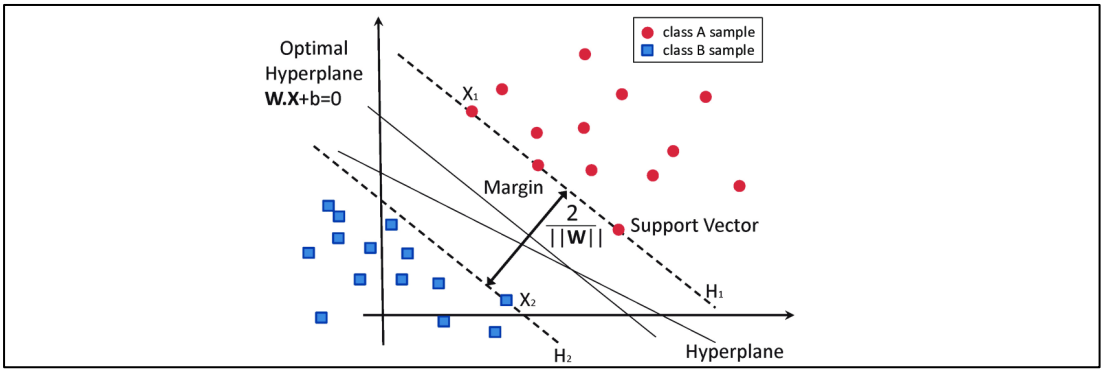


Figure 3: Support Vector Machines.

3. Statistical evaluation metrics

3.1. Definition

Statistical evaluation metrics are used to measure the quality of the statistical or machine learning model. Evaluating machine learning models or algorithms is essential for any project. There are many different types of evaluation metrics

available to test a model. These include classification accuracy, logarithmic loss, confusion matrix, and others.

3.2. Classification Accuracy

Classification Accuracy is what we usually mean when we use the term accuracy. It is the ratio of the number of correct predictions to the total number of input samples:

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

3.3. Logarithmic Loss

Logarithmic Loss or Log Loss, works by penalizing the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples. Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below:

Where: y_{ij}, indicates whether sample *i* belongs to class *j* or not p_{ij}, indicates the probability of sample *i* belonging to class *j*

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

In general, minimizing Log Loss gives greater accuracy for the classifier.

3.4. Confusion Matrix

The Confusion Matrix gives us a matrix as output and describes the complete performance of the model.

Let's assume we have a binary classification problem. We have some samples belonging to two classes: True or False. Also, we have our own classifier which predicts a class for a given input sample. On testing our model on 165 samples, we get the following result:

Table 2: Confusion Matrix.

n=165	Predicted:False	Predicted:True
Actual:False	50	10
Actual:True	5	100

- *True Positives:* Cases where we predicted True and the actual output was also true.
- *True negatives:* The cases in which we predicted False and the actual output was False.
- *False Positives:* Cases where we predicted True and the actual output was False.
- *False negatives:* The cases in which we predicted False and the actual output was true.

Accuracy for the matrix can be calculated by:

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalSample}$$

3.5. F1 Score

F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances). High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model. Mathematically, it can be expressed as:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

The F1 Score tries to find the balance between precision and recall.

Precision: It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TruePositives(TP)}{TruePositives(TP) + FalsePositives(FP)}$$

Recall: It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

$$Recall = \frac{TruePositives(TP)}{TruePositives + FalseNegatives(FN)}$$

3.6. Mean Absolute Error

The Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don't give us any idea of the direction of the error i.e., whether we are under-predicting the data or over-predicting the data. Mathematically, it is represented as:

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N} |y_i - \hat{y}_j|$$

3.7. Mean Squared Error

The Mean Squared Error (MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the square of the difference between the original values and the predicted values. The advantage of MSE being that it is easier to compute the gradient, whereas Mean Absolute Error requires complicated linear programming tools to compute the gradient. As we take square of the error, the effect of larger errors become more pronounced than smaller errors; hence the model can now focus more on the larger errors. Mathematically, it is represented as:

$$MeanSquaredError = \frac{1}{N}\sum_{j=1}^{N} |y_i - \hat{y}_j|$$

3.8. Label Based Evaluation

A classifier which has been employed for a multi label text classification task might predict all the expected labels, a subset of them, or none of the expected labels. Hence, those cases should be considered in order to evaluate the classifier. The metrics for a multilabel text classification problem are organized in two main categories which are example based and label-based evaluation.

Example-based evaluation techniques consider a defined experiment with P positive instances and N negative instances for some condition. The four predicted outcomes can be classified into True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) as seen in some of the measures above. Label-based measures instead, evaluate each label separately and then averages over all labels. All the measures from the example-based evaluation can be used for label evaluation [6].

- *Micro averaged measures*: Any of the example-based evaluation metrics can be computed on individual class labels first and then averaged over all classes.
- *Macro averaged measures:* Any of the example-based evaluation metrics can be computed globally over all samples and all class labels.
- *Weighted average measures*: Any of the example-based evaluation metrics can be computed on individual class labels first and then averaged over all classes with their corresponding class weights. These class weights are given by the distribution of the test data

4. Conclusion

In this chapter we dealt with explaining some of the most famous machine learning algorithms for automatic classification such as KNN, Decision trees, Random Forest, Linear Regression, and finally Support Vector Machines. We also

explained the statistical evaluation metrics that are used to evaluate a model such as: The Confusion Matrix, F-measure, the accuracy, Precision and Recall, and more.

In the next chapter we will explore Deep Learning and understand its relevance to the problem we're attempting to solve.

CHAPTER II :

Deep Learning

Chapter 2: Deep learning

1. Introduction

Deep learning is a set of learning methods attempting to model data with complex Architectures combining different nonlinear transformations. The elementary bricks of deep learning are the neural networks forming deep neural networks.

These techniques have enabled significant progress in sound and image processing, including facial recognition, speech recognition, computer vision, automated language processing, and text classification (for example, spam recognition).

Potential applications are countless. Besides, there exist several types of architectures for neural networks: The multilayer perceptron which are the eldest and simplest ones. Then, The Convolutional Neural Networks (CNN), particularly adapted for image processing. After that, the Recurrent Neural Networks (RNN) which are used for sequential data such as text or time series.

This chapter defines deep learning and briefly explains the architectures for neural networks used in text classification.

2. Neural networks

Neural networks are a set of algorithms modeled loosely after the human brain designed to recognize patterns [1-2]. They interpret sensory data through machine perception, labeling, or raw clustering input. The ways they recognize are numerical, contained in vectors, into which all real-world data must be translated, be it images, sound, text, or time series.

Neural networks help cluster and classify. They can be a clustering and classification layer on top of stored and managed data. They help to group unlabeled data and organize data. (Neural networks can extract features fed to other algorithms for clustering and classification; it consists of deep neural networks as components of larger machine-learning applications.).

In a classification problem, those outcomes are labels that could be applied to data: spam or not_spam in an email filter, good guy or bad guy in fraud detection, angry_customer or happy_customer in customer relationship management. Other problems include anomaly detection (useful in fraud detection and predictive

maintenance of manufacturing equipment) and clustering, which is helpful in recommendation systems that surface similarities.

For example, in classification problems, labeled data is needed. And publicly available or possible to be created. In this example, spam emails would be marked as spam, and the labels would enable the algorithm to map from inputs to the classifications. It is difficult to identify a suitable dataset until the tests are done on it.

Deep learning maps inputs to outputs. It finds correlations. It is known as a "universal approximator" because it can learn to approximate an unknown function $f(x) = y$ between any input x and any output y, assuming they are related at all (by correlation or causation, for example). In the process of learning, a neural network finds the right f, or the correct manner of transforming x into y, whether that be $f(x) = 3x + 2$ or $f(x) = 9x - 0.1$.

2.1. Predictive Analytics Regressions

With classification, deep learning can establish correlations between pixels in an image and a person's name. It might be called a static prediction. Exposed to enough of the correct data, deep learning can establish correlations between current events and future events. It can run regression between the past and the future. The forthcoming event is like the label in a sense. Deep learning doesn't necessarily care about time or the fact that something hasn't happened yet. Given a time series, a deep understanding may read a string of numbers and predict the number most likely to occur next.

2.2. Neural Network Elements

Deep learning is the name we use for "stacked neural networks"; networks are composed of several layers.

The layers are made of *nodes*. A node is just a place where computation happens, loosely patterned on a human brain neuron that fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights that either amplify or dampen that input, thereby assigning significance to inputs concerning the task the algorithm is trying to learn, e.g. which information is most helpful in classifying data without error? These input-weight products are summed, and then the sum is passed through a node's so-called activation function to

determine whether and to what extent that signal should progress further through the network to affect the outcome, say, an act of classification. If the signal passes through, the neuron has been “activated”.

A neural network has some key components which constitute its anatomy as figure 4 depicts. Those components are [6]:

- Layers which are combined into a network.
- Loss function which measures how far the output is from the expected value.
- Optimizer which helps the network to update itself based on the data it sees and its loss function.

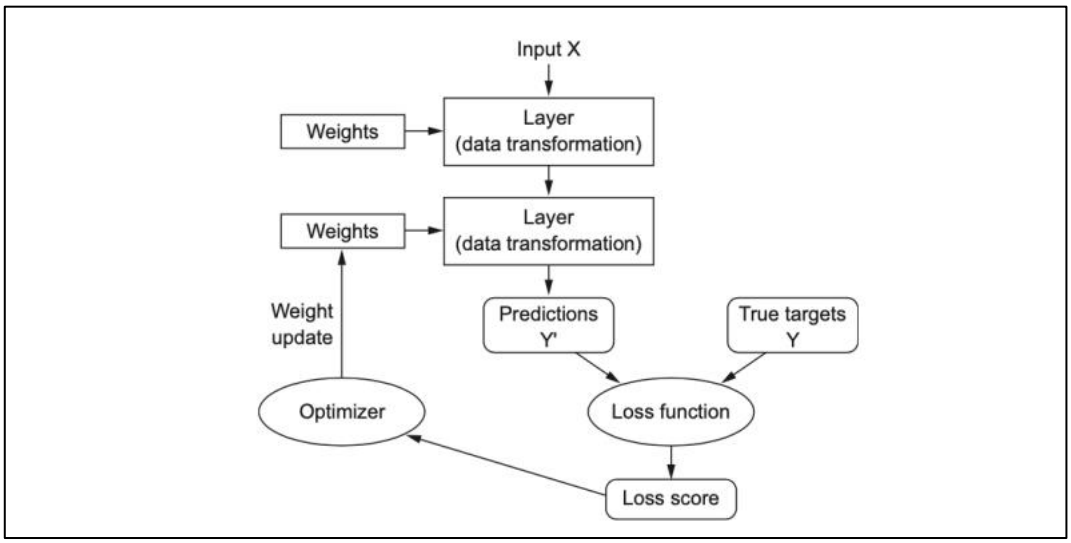


Figure 4: Anatomy of a Neural Network *[6]*.

Initially, the weights have random values, the output is far from the expected value and the loss score is high. The network starts processing the samples (text input), the weights are adjusted a little in the correct direction and the loss score decreases. A network with a minimal loss is one for which the outputs are as close as they can be to the targets and is called trained network. For handling a text classification task one more important key component is the last layer activation. The activation function constrains the network’s output. For a text classification task, the predicted values should be between 0 and 1. The most popular activation functions for this purpose are *sigmoid*, *relu* and *softmax*, where the *sigmoid* function is the most suitable for a multi label text classification problem.

2.3. Critical Concepts of Deep Neural Networks

Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their depth, that is, the number of node layers through which data must pass in a multistep pattern recognition process.

Earlier versions of neural networks such as the first perceptrons were shallow, composed of one input, one output layer, and at most one hidden layer. More than three layers (including input and output) qualify as "deep" learning. In this regard, *deep* is a strictly defined term that means more than one hidden layer.

In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer's output. The further it advances into the neural net, the more complex the features nodes can recognize since they aggregate and recombine.

3. Artificial Neural Network

This section provides an overview of Deep Learning and Artificial Neural Networks (ANN) architecture and discusses some key terminology.

As shown in the following figure, each perceptron is made up of the following parts:

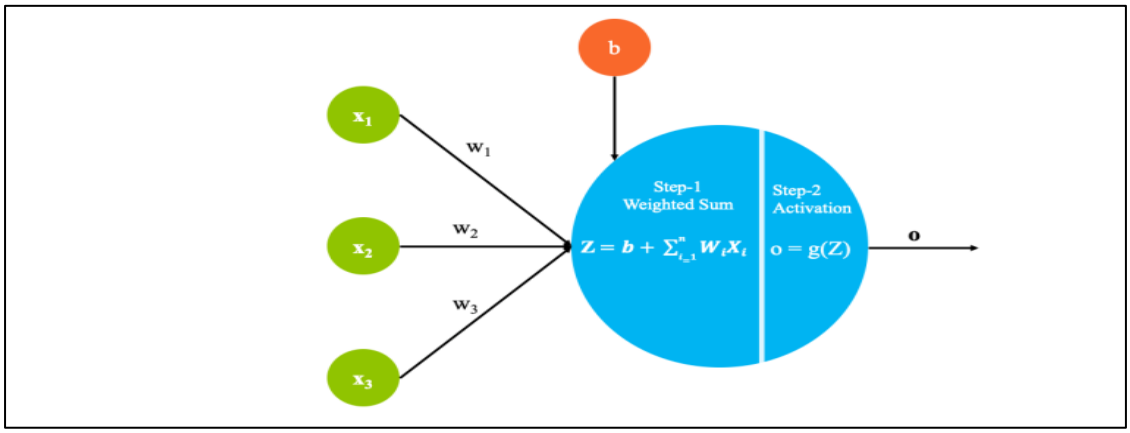


Figure 5: Artificial Neural Networks.

Calculate the weighted sum Inputs x_1 through x_n, which a vector X. X_i can also denote represents the i^{th} entry from the dataset. Each entry from the data set contains *n* dependent variables.

Weights w_1 through w_n, which can be denoted as a matrix WA bias term b, which is a constant.

3.1. Activation function

The output of step 1 is now passed through an *activation function*. The activation function *g* transforms the works to a desired non-linear format before sending them to the next layer. It maps the summation result to the selected range.

For example, a sigmoid function maps values to the range [0,1], which is helpful if the desire of system to predict probabilities. Doing so leads to modeling complex, non-linear decision boundaries.

3.2. Shallow neural network

In the most basic form, a neural network contains three layers: input, hidden, and output. The following figure shows that a network with just one hidden layer is termed an external neural network.

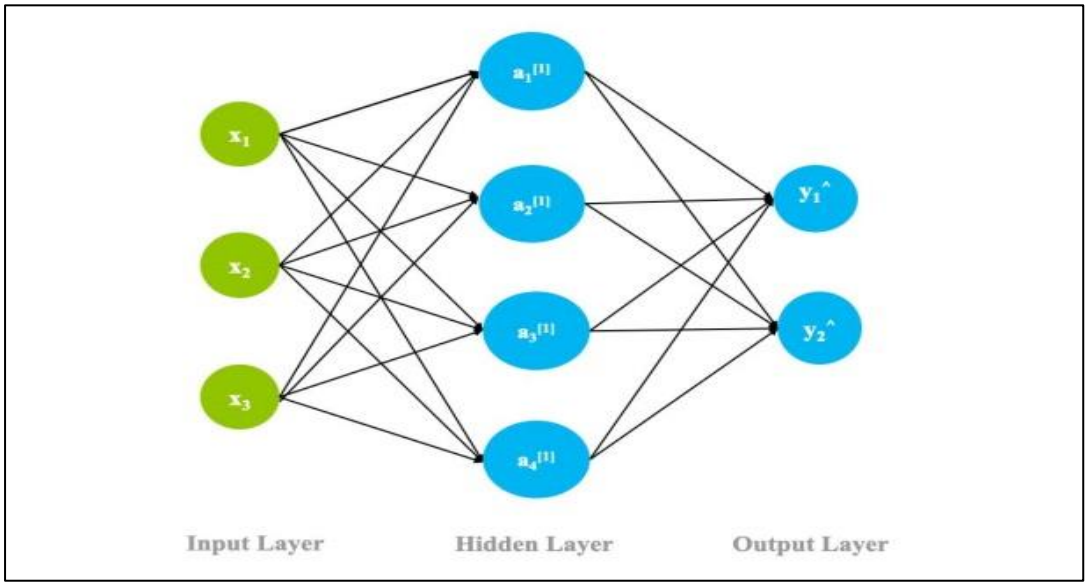


Figure 6: Shallow Neural Network.

The computations discussed in the previous sections happen for all neurons in a neural network, including the output layer, and one such pass is known as forwarding propagation. After one forward pass is completed, the output layer must compare its results to the actual ground truth labels and adjust the weights based on the differences between the ground truth and the predicted values. This process is a backward pass through the neural network known as back-propagation. While the

mathematics behind back-propagation are outside the scope of this article, the basics of the process can be outlined as follows:

The network works to minimize an objective function, for example, the error incurred across all points in a data sample. At the output layer, the network must calculate the total error (difference between actual and predicted values) for all data points and take its derivative concerning weights at that layer. The product of the error function concerning consequences is called the gradient of that layer.

The weights for that layer are then updated based on the gradient. This update can be the gradient itself or a factor of it. This factor is known as the learning rate, and it controls how significant the steps are taken to change the weights. The process is then repeated for One layer before it and continues until the first layer is reached.

During this process, values of gradients from previous layers can be reused, making the gradient computation efficient.

The result of one pass of forwarding propagation and back-propagation is a change to the network layers' weights. It brings the system closer to modeling the data set provided to it. Because this process uses the gradient to minimize the overall error, converging the neural networks' parameters to the optimum is called gradient descent.

3.3. Deep neural networks

A deep neural network is an external neural network with more than one hidden layer. Each neuron in the hidden layer is connected to many others. Each arrow has a weight property, which controls how much that neuron's activation affects the others attached to it.

Selecting the number of hidden layers depends on the problem's nature and the data set's size. The following figure shows a deep neural network with two hidden layers [7].

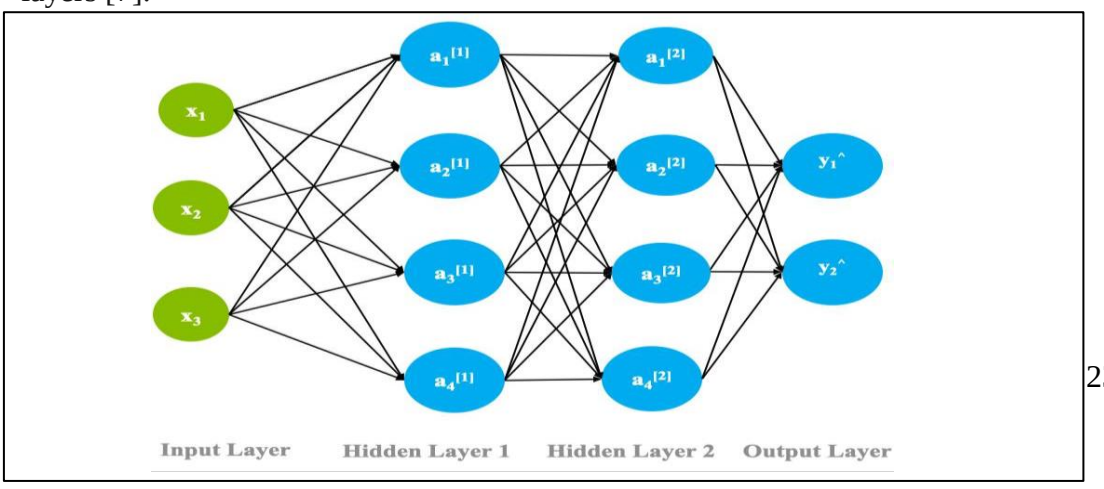


Figure 7: deep neural network with two hidden layers.

4. Convolutional Neural Networks

In recent years, neural networks have covered deep learning, machine learning, and many other areas. Neural networks mimic how the human brain solves complex problems and finds patterns in a particular dataset.

In machine learning, a Convolutional Neural Network (CNN or ConvNet) is an Artificial Neural Network most commonly used to analyze visual images. This is an input from deep learning algorithms for ideas that effectively weight objects and distinguish some photos from others. The network embeds deep neural networks such as neural networks in images and other data.

It has existed for several decades and has proven to be very powerful when large labeled data sets such as images, videos, and other data are used.

CNNs are complex neural networks capable of recognizing complex features in data. They are used in everything from vision-powered robots to self-driving vehicles. They can also be used to identify and classify images and objects such as faces, animals, road signs, objects in a scene, and other things. The effectiveness of coil webs in image recognition is one of the main reasons why the efficacy of deep learning has shaken the world. Unlike image functions, which are learned with small squares of input data, coils preserve the relationship between pixels. An image classifier can be usefully confused in myriad ways, for example, to classify cats and dogs or to determine whether an image of the brain contains tumors.

Convolutional Neural Networks (CNNs) create a matrix of word embeddings for each sentence to which convolutional filters called kernels are applied with all possible windows of words in order to extract different features [8]. After that, a max pooling strategy is applied in order to reduce the layers' output dimensionality and to keep them in a fixed size. Deep convolutional networks are created by stacking combinations of these convolutional layers. CNNs are competitive with RNNs and can be a much faster alternative to RNNs for tasks like text classification. Therefore, multiple studies have used CNNs for multi-label text classification. It is also suited for large scale text classification and Multi-label Classification.

5. Recurrent Neural Networks

RNNs are a powerful and robust type of neural network and belong to the most promising algorithms because they are the only ones with internal memory.

Like many other deep learning algorithms, recurrent neural networks are relatively old. They were initially created in the 1980s, but we have only seen their true potential in recent years.

Because of their internal memory, RNNs can remember important things about the input they received, which means the model keeps memory of the previous elements with respect to the target. This memory is crucial for RNNs since the semantic meaning of sequences often relies on previous elements. Another important aspect of RNNs is their availability to handle inputs of variable length, which is something that CNNs cannot do and can have an impact on text classification tasks. However, RNNs have the vanishing gradient problem, which means that in practice when a RNN is run many times the gradient becomes so small that the weights of the first layers do not update anymore, so the model forgets information that is far away from the current feature. To solve that Long Short-Term Memory was proposed by Hochreiter and Schmidhuber [8].

Like all other deep learning algorithms, a feed-forward neural network assigns a weight matrix to its inputs and then produces the output. Note that RNNs apply weights to the current and previous information. Furthermore, a recurrent neural network will also tweak the weights for gradient descent and back-propagation over time (BPTT). [9]

Also, while feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation), and many to one (classifying a voice) [10].

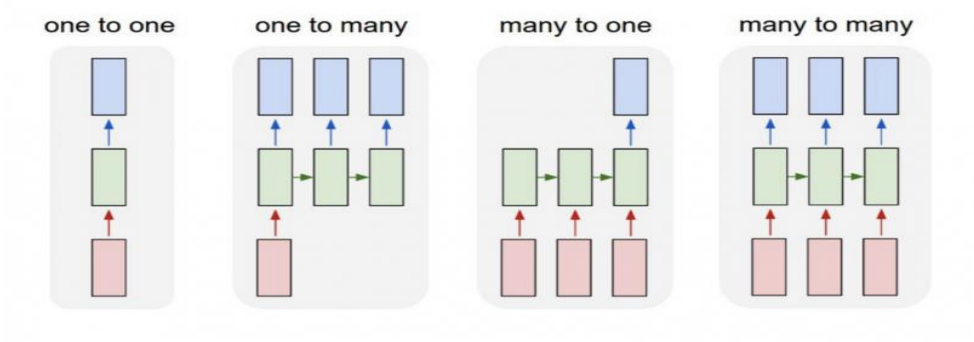


Figure 8: Different mappings of RNN *[10]*.

Several other algorithms are pretty popular these days. Some of them are:

- LSTMS: Long Short-Term Memory Networks
- MLPs: Multilayer Perceptrons
- RBFNs: Radial Basis Function Networks
- DBNs: Deep Belief Networks
- GANs: Generative Adversarial Networks
- RBMs: Restricted Boltzmann Machines
- SOMs: Self-Organizing Maps
- Autoencoders

6. Transfer Learning

Inductive transfer learning [8] is a research problem of machine learning that deals with how knowledge can be stored, reused and transferred from previously trained models. Whereas transfer learning has been extensively used in computer vision with important success, it has not been until recent years that the NLP community has managed to successfully apply transfer learning to text. For NLP, transfer learning refers more specifically to pre-trained language representations — general models that are trained on large corpora and contain a considerable amount of world-knowledge. It has proven to be very effective to train models for specific downstream tasks on top of these pre-trained language representations. This effectiveness seems to be caused by that general knowledge provided by the pre-trained models which otherwise cannot be inferred in many cases from the training set of the task at hand. As presented, in NLP there are two main types of pre-trained language representations [8]: feature-based and fine-tuning models.

Whereas the first are often used to initialize the first layer of a neural network, the latter are fine-tuned as a whole model for a specific downstream task. In the next sections, we will give an overview on pre-trained language representations.

6.1. Word embeddings

An example of feature-based pre-trained language representations are word embeddings, word representations in vectors that have been learned from their context, following the hypothesis that similar words have similar contexts. Three of the most emblematic word embeddings architectures are Word2vec, GloVe and ELMo, which we will explain hereafter [8]:

- *Word2vec*: Even though there had been previous approaches to word embeddings, it was Word2vec that mostly popularized them. Word2vec is based on the distributional hypothesis, which refers to the fact that the meaning of the words can be inferred by its context. Word2vec can be trained with two model architectures in order to obtain the word representations — Continuous Bag-of-Words (CBOW) and Skip-gram model. The first one learns to predict a word given a window of k words around it, where the order is not taken into account. The second architecture learns to predict words before and after a certain target word using a *log-linear classifier* with continuous projection layer. Words that are further from the target word are given less weight since they are usually less related to the target word.
- *GloVe*: Global Vectors for Word Representation (GloVe) was proposed by Pennington et al. (2014) as a *log-bilinear* regression model that uses the occurrence counts of the words in the entire corpus when training, instead of only using a shallow window-based method such as that of Word2vec.
- ELMo Some of the limitations of Word2vec are that word embeddings cannot represent a combination of two or more words

like idioms. They are also limited by the window size of their training, and have problems representing polysemic words. To deal with these issues, the Embeddings from Language Models (ELMo) was introduced, deep contextualized word representations that model both how words are used (syntax and semantics) and contextual variances (like polysemy). The representations are learned from the internal states of a deep bidirectional language model (biLM) trained with a LSTM on a large corpus of 30 million sentences and instead of returning a representation per word, they return a representation per contextual word.

6.2. Fine-tuning language models

From 2018 to 2019 there has been a proliferation of multiple fine-tuning pretrained language models, such as GPT (Radford et al., 2018), GPT-2 (Radford et al., 2019), Transformer-XL (Dai et al., 2019), XLNet (Yang et al., 2019) and XLM (Lample and Conneau, 2019). In the next sections we will introduce two of the earliest approaches — ULMFiT and BERT.

- *ULMFiT:* The Universal Language Model Fine-tuning (ULMFiT), proposed by Howard and Ruder (2018), is an inductive transfer learning method which consists of three stages — 1) train a language model on a general domain corpus, 2) then fine-tune that model on a more specific corpus related to the downstream task, 3) and finally train a classifier on top of that model. This approach is very useful for tasks with little labelled data and some unlabelled data.
- *BERT*: Another important pre-trained representation model, based on the Transformer architecture introduced by Vaswani et al. (2017), is Bidirectional Encoder Representations from Transformers (BERT) by Devlin et al. (2018). The authors argue that the main limitation of pre-training models is the unidirectionality, which means the representations are only learnt from left to right, therefore missing cataphoric references which are important for certain tasks like question answering. They propose the Masked Language Model

(MLM) with a deep bidirectional Transformer, which randomly masks parts of the unlabelled input so that the model learns how to predict the masked elements from both directions. Additionally, the model also learns how to predict the next sentence with Next Sentence Prediction (NSP). Unlike ULMFiT, BERT has only two steps — pre-training and fine-tuning. Multiple successful examples of using BERT have been proposed for text classification.

In the Modeling chapter, we will speak further about the Transfer Learning Approaches that we retained in our contribution.

7. Frameworks

The deep learning frameworks offer reusable code blocks. These reusable code blocks provide various modules and help abstract the logical blocks. The modules are handy and can be easily used to develop any deep learning model.

For better understanding, the frameworks can be classified into:

7.1. Low-Level Frameworks:

Low-level frameworks give the basic abstraction block. They are flexible and can be customized as per the requirement. Some of the popular learning frameworks are:

- MxNet
- Tensor Flow
- PyTorch

7.2. High-Level Frameworks:

High-level frameworks simplify the work by aggregating the abstraction further. Low-level frameworks are the back ends of high-level frameworks. The source is converted into a required low-level framework before executing them. Some of the popular learning frameworks are as follows:

- Gluon
- Keras

8. Challenges in deep learning

Though deep learning methods have gained immense popularity in the last ten years, the idea has been around since the mid-1950s when Frank Rosenblatt invented the perceptron on an IBM® 704 machine. It was a two-layer-based electronic device that could detect shapes and do reasoning. Advancements in this field in recent years are primarily because of the increase in computing power and high-performance graphical processing units (GPUs), coupled with the significant increase in the wealth of data these models have at their disposal for learning, as well as interest and funding from the community for continued research. Though deep understanding has taken off in the last few years, it comes with challenges that the community is working hard to resolve.

9. Conclusion

Deep learning helps computers to derive meaningful links from a plethora of data and make sense of unstructured data. Here, the mathematical algorithms are combined with a lot of data and robust hardware to get qualified information. This method allows information from digital data to be automatically extracted, classified and analyzed.

In the next chapter, we will describe some related works to our Master project.

CHAPTER III :

Related Works

Chapter 3: Related Works

1. Introduction

This chapter will present related works that can give valuable insights into the biomedical document text classification method.

We will summarize each work, problem, and objective; we will examine the datasets and the techniques the authors have used, the proposed solutions, and the obtained results. Furthermore, we made a comparative study of the methods to evaluate their performance and their advantages/disadvantages.

2. Related Works

2.1. Medical Text Classification using Convolutional Neural Networks

The goal behind this work [11] is to apply machine learning approaches to build models that allow an automatically generated context-based and rich representation of health-related information. Convolutional neural networks (CNNs) have dramatically improved the approaches to many active research problems. One of the critical differentiators between CNNs and traditional machine learning approaches is the ability of CNNs to learn complex feature representations. A CNN-based approach to categorizing text fragments is applied, at a sentence level, based on the emergent semantics extracted from a corpus of medical text. The process is compared with three other methods: Sentence Embedding, Mean Word embedding, and Word Embeddings with BOW (bag-of-word). The results indicate that the CNN-based approach outperforms the different techniques by at least 15% in terms of accuracy in the task of classification.

2.1.1. Datasets

Two datasets were procured from the medical domain. This approach makes use of the Word2vec algorithm. A dataset from PubMed1 was procured for training Word2vec models. A collection of 15k clinical research papers was used to train it.

The Word2vec model described in this paper was prepared using this PubMed collection.

For sentence-level classification, it was necessary to gather training data pre-classified by medical professionals. Merck Manual2 dataset contains articles on topics like Brain, Cancer, etc. Each article is classified under a parent header representing specific medical issues and conditions. The dataset consisted of 26 medical classes, and 4000 sentences were chosen randomly for each category extracted from the Merck articles to use as training data and to ensure balance across all types. The validation dataset consisted of 1000 sentences from each of the categories.

The CNN-based approach was applied to automatically learn and classify sentences into one of the 26 categories in the evaluation dataset.

2.1.2. Techniques and Proposed Solutions

Each sentence is converted to a word-level matrix where each row in the matrix is a sentence vector extracted from the Word2vec model. CNNs require input to have a fixed size, and sentence lengths can vary greatly. Therefore, a max word length of 50 allowable for a sentence is chosen, which worked well. During the training phase, a Word2vec hidden layer size of 100 is applied, thus giving the input feature a resolution of 100×50. If a sentence contained less than 50 tokens, a particular stop word was repeatedly appended to the end of the sentence to meet the 50-word requirement. If a conviction had over 50 words, only the first 50 were considered to be representative of that sentence.

During the evaluation, various CNN configurations were tested. A grid search was applied to ascertain the optimal number of filters and filter sizes and experimented with multiple formats of convolutional layers, including 2, 4, and 6. From these experiments, the best-performing CNN model consisted of a configuration of two sets of convolutional layers with each pair followed by a max pooling layer. This model, 256 convolutional filters were used with a filter size of 5 across all convolutional layers. After the second max pooling function, a dropout function is applied to help prevent over-fitting. In this model, a dropout rate of .5 was used. Then append a fully connected layer with a length of 128, followed by a second

dropout function. This is followed by a dense layer with a size of 26 to represent the number of classification classes, with a Softmax function determining the output.

A visual representation of this model can be found in the figure below:

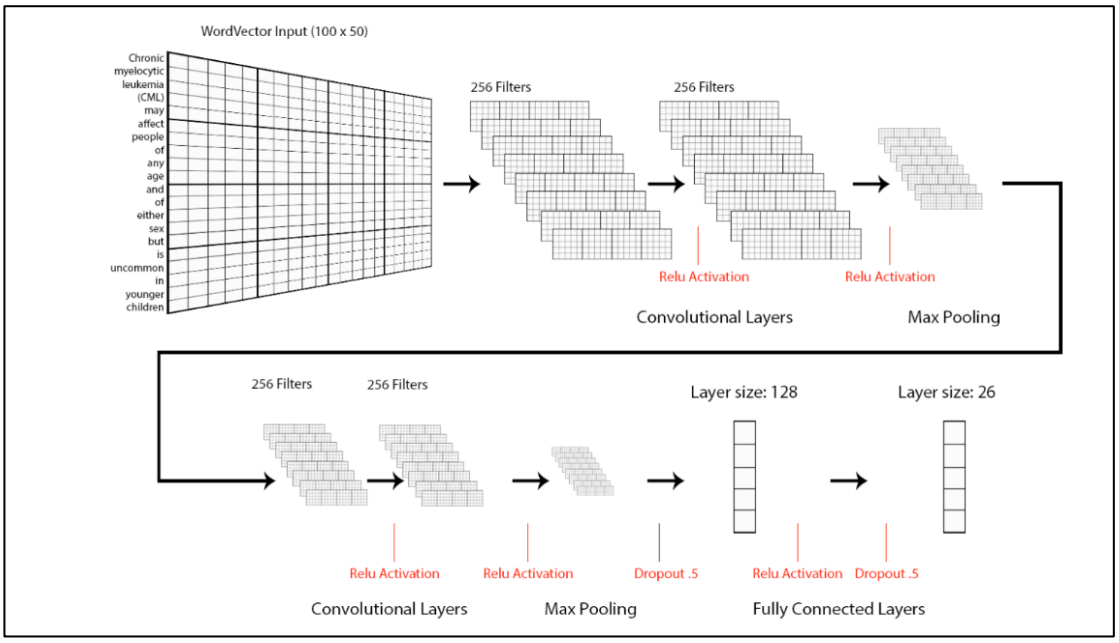


Figure 9: Outline of the CNN model structure.

2.1.3. Evaluation

The model is compared with the following methods:

Sentence Embeddings (LogR+Doc2vec).

Mean Word Embeddings (ZeroMean/ElimMean+Word2vec).

Word Embeddings with BOW(bag-of-words) Features (BOW+LogR).

The figure below shows the accuracy (percentage of sentences classified correctly) of each

The method in the experiments. The first three methods shown in the figure perform worse because the initially pre-trained embeddings do not provide good classification features. The bag-of-word method performs better - probably due to better feature extraction based on the pre-trained word embeddings. The CNN-based approach has the highest accuracy by a wide margin. This could be explained by the fact that the deep learning approach can capture more complex features than the other shallow learning approaches.

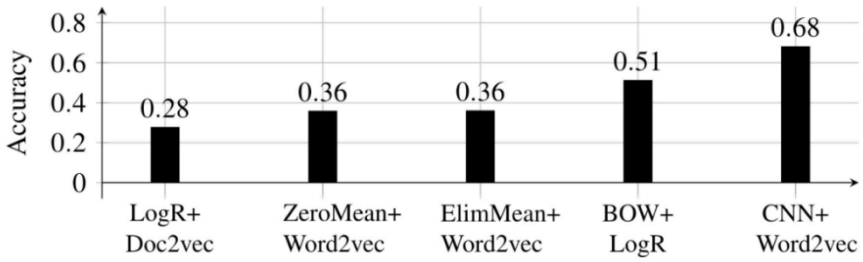


Figure 10: Classification performance.

2.2. Deep Learning Classification of Biomedical Text using Convolutional Neural Network

Convolutional Neural Network is one of the deep neural networks, and it is believed that this neural network can solve the data scarcity problem. It is instrumental in extracting information from raw signals, ranging from computer vision applications to speech recognition. This research [12] focus on using a Convolutional Neural Network to classify biomedical text abstracts and to measure the effectiveness of using Convolutional Neural networks in text classification.

2.2.1. Datasets

The dataset used in this research is the Ohsumed dataset which is the subset of the MEDLINE database. This research has a total number of 11,566 abstracts selected from the Ohsumed dataset.

The Ohsumed datasets might contain different levels from the first level until the fourth level. This research will focus on only 11,566 abstracts from biomedical journals from the first and second levels. All the categories and the number of abstracts for each category used in this research are stated in the table below:

Table 3: List of selected categories with document numbers.

Category Name	Number of Documents
Arrhythmia	1,173
Coronary Disease	4,235
Heart Arrest	513
Heart Defects, Congenital	718
Heart Failure, Congestive	1,295
Heart Valve Diseases	335
Myocardial Diseases	355
Myocardial Infarction	2,942
TOTAL ABSTRACTS USED: 11,566 ABSTRACTS	

During the training process, the dataset is split into the training set and validation set. In this research, the training set consists of 80% of the whole dataset, while the validation set consists of 20% of the whole dataset.

2.2.2. Techniques and Proposed Solutions

This research consists of several components, and all of these components are connected sequentially. For instance, the components in the architecture include the input, biomedical abstracts, word embedding layer, deep network, which is made up of convolutional layers and max-pooling layer, fully connected layer, and the output, which is the classification result. The deep learning text classification model architecture used in this research is shown in the figure below:

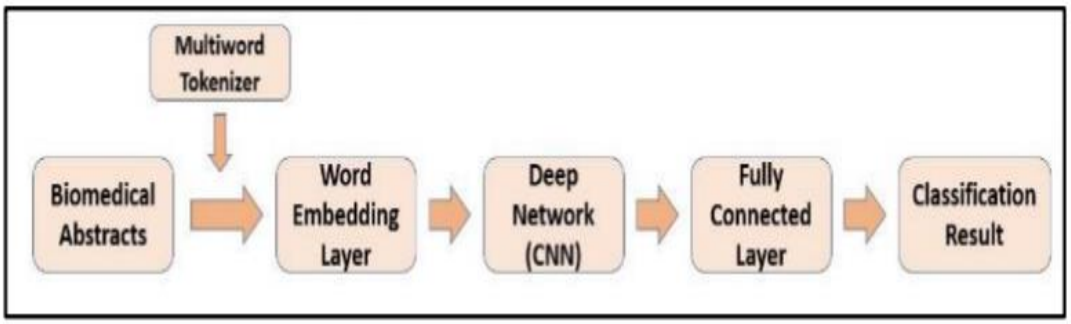


Figure 11: Deep Learning Text Classification Model Architecture.

The input, the biomedical abstracts, will go through the process of tokenizing using the multi-word tokenizer instead of the single-word tokenizer.

Next, the word embedding layer must be set up before the pre-processed text input passes. The purpose of this layer is to transform all the words in the text that have the same or similar meaning to have an equal representation in the form of a vector [13]. In this research, a pre-trained BioASQ word vector is used [14].

The sequence of embedding vectors obtained from the previous process will be converted into a compressed representation; the stack of convolutional layers and max-pooling layer take it as the input.

The convolutional layer consists of trainable kernels, also known as filters, which detect any specific input features. The different convolution processes produce a set of activation maps, which detects different features and passes to the max-pooling layer.

The activation function used in the proposed model is the Rectified Linear Unit (ReLU). Moreover, the max-pooling layer will take the transformed output from the convolutional layer as input. This layer functions to reduce the computation complexity and the spatial dimension without dumping the momentous information. These are the significant features. Next, the fully connected layer is where the classification is performed based on the features extracted from the stack of convolutional layers and max-pooling layers.

In this research, a softmax function with categorical_crossentropy loss function is used; its function is to apply a transformation to the output obtained so that the final output can be interpreted as a probability vector for each class or class scores. Finally, the cross-validation method is used during the model's training process to reduce problems like model over fitting.

3. **Evaluation**

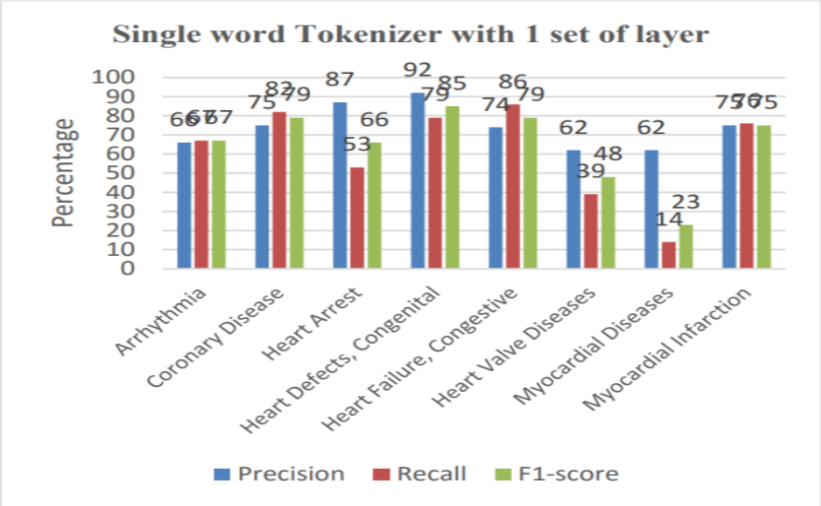


Figure 12: Result of the Experiment using Single Word Tokenizer with 1 Set of Convolution and Max-Pooling Layer.

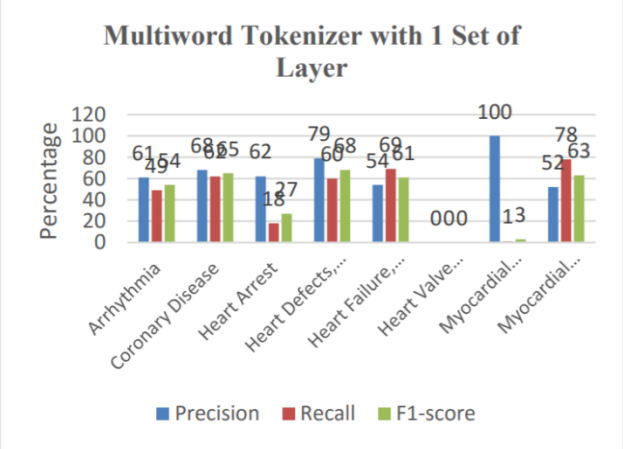


Figure 13: Result of the Experiment using Multiword Tokenizer with 1 Set of Convolution and Max-Pooling Layer.

Table 4: Comparing the results using multi-word and single-word tokenizers.

	Average Accuracy (%)	Average Precision (%)	Average Recall (%)	Average F1-score (%)
Using multiword tokenizer	54.79	61.00	60.00	60.50
Using single word tokenizer	70.64	75.00	75.00	75.00

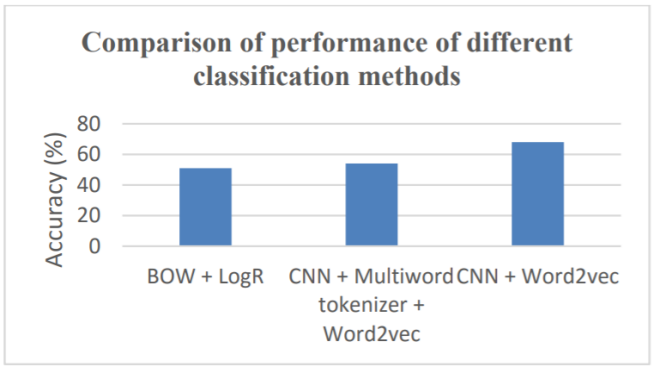


Figure 14: Comparison of Performance of different Classification Methods.

3.1. Improved Convolutional Neural Network for Biomedical Text Classification

Convolutional neural networks have an excellent ability to extract useful features and so are widely used in the field of text classification. This paper proposes a novel approach for biomedical text classification based on improved convolutional neural networks to solve the problem that deep convolutional neural networks have a large amount of computation and cannot perceive the relationship between levels well. This work [15] uses the combination of deep separable convolution and void convolution to improve the convolutional neural network. At the same time, we use the attention mechanism to classify biomedical literature. In addition, the focusing loss function is used to improve the imbalance of biomedical texts. Experiments show that the classification model in this paper is adequate for biomedical texts.

3.1.1. Datasets

The experiment used two datasets publicly available on the Internet. The MEDLINE dataset collects biomedical articles with article titles and abstracts. The dataset contains a training set of 94936 articles and a test set of 48906. The original data set contains more than 20,000 categories. Ten indistinguishable categories (e.g., neurology, gastroenterology, and oncology) were selected for categorizing documents. The Ohsumed dataset is a subset of clinical papers from the MEDLINE database, which contains 23 categories of cardiovascular disease. The Ohsumed dataset has 13,929 documents, with varying numbers of documents per class and a very irregular distribution. If a table is divided into parts, these should be labeled (a), (b), (c), etc., but there should only be one caption for the whole table, not separate ones for each part.

3.1.2. Techniques and Proposed Solutions

The text is preprocessed, and the word vector is represented. Then the multi-head self-attention mechanism is used to extract keywords and capture the global relationship. The support vector machine is used to classify the extracted text features.

Figure 15: Flowchart of Biomedical Text Classification.

In the text Pre-Processing stage, abbreviation expansion, word form restoration, and case conversion operations in addition to word segmentation and removal of stop words. In this way, the preprocessed text can better express the text features. The NLTK module is used to preprocess text data. For preprocessed data, for a particular category, 0 in the first column means that it does not belong to a particular category, and one means that it belongs to a particular category. The second column is the processed title and content.

In the Word embedding stage, the Skip-Gram model represents the preprocessed text data as word vectors. After word2vec word vector training, the word vector representation of each word is obtained, and the complete information of the document cannot be obtained.

In the improved-CNN stage, a combination of void convolution and deep separable convolution is adopted to replace the ordinary convolution to alleviate problems such as model parameter doubling, the large amount of calculation, bloated model, and gradient explosion associated with enlarging the convolution kernel to obtain deeper features.

In the Multi-attention stage, a Multi-head self-attention mechanism is used. The purpose of the Multi-head self-attention mechanism is to capture the critical information of the text sequence from many aspects. Mapping the query matrix (Q), fundamental matrix (K), and value matrix (V) into several subspaces, and the subspaces are calculated separately.

Furthermore, finally in the Focal Loss stage, In order to reduce the influence of simple samples on the model and increase the influence of complex samples, the influence factor was introduced into the traditional cross-entropy loss, and the weight coefficient of balanced, complex samples were increased.

3.1.3. Evaluation

In order to evaluate the performance of the improved CNN method in text classification, the traditional CNN-Softmax method was introduced, and comparative experiments and analyses were carried out. Comparison results of the two algorithms are shown in the Table below. The classification accuracy of the improved CNN method proposed in this paper is better than that of the CNN-Softmax method. The improved CNN-SVM method can obtain deep multi-scale features so that the text classification can improve. Meanwhile, the enhanced CNN-SVM process can alleviate the impact of data imbalance on the classification effect and improve the overall classification effect to a certain extent.

Table 5: Results.

Model	Classification accuracy	Kappa coefficient
CNN-Softmax	86.14%	88.7%
Improved CNN	90.88%	90.56%

4. Comparative Study

Table 6: Comparison of Models by Average Precision.

	CNN+word2vec 1	CNN+word2vec 2	Improved CNN
Average Precision	68%	70,64%	90,88%

After evaluating the result of the most prominent previous works related to the subject of this thesis, the CNN model provides good results with varying degrees of success, most importantly is how the pre-processing plays a huge part in whether a model performs well.

In these cases, the different models were trained on different datasets, but the first two were trained on 2 datasets comparable in size (1st model: 15,000 articles, 2nd model: 11,566 articles) wherein the remaining model was trained on a much larger

dataset (3rd model: 143842 articles) which explains the statistically better results a difference of more than 20% in accuracy.

The improved CNN model consists of having a multi-head self-attention mechanism which was used to extract keywords and capture the global relationship. In addition, a support vector machine is used to classify the extracted text features.

By this we conclude that: the dataset and its pre-processing are very important in training deep learning models especially in the biomedical domain giving that the raw data is very precise and that modifications like the ones applied to the improved CNN model can make a huge difference.

5. Conclusion

In this chapter, we've examined three prominent state of the art works that have tested and developed different solutions for biomedical text classification using convolutional neural networks, we compared their results and explained them.

CHAPTER IV :

Solution Modelling

Chapter 4: Solution Modelling

1.5 Introduction

This chapter will present how our approach works in detail, and a give clear look at the process from start to finish.

It involves explaining our proposed approach and methodology, including dataset information and how dataset preprocessing works, we will also discuss in detail the classification models used.

1.6 Proposed Approach and Methodology

The approach is based on a comparative study using pre-trained biomedical text classification models, and different result from the convolutional neural network with improved pre-processing seen in Chapter 3.

For the Pre-trained models, we first begin by importing the biomedical text dataset and splitting it into two subsets: Training set and Test set.

Then we will do the pre-processing using: BERT Tokenizer [16], RoBERTa Tokenizer and XLNet Tokenizer Respectively for the three pre-trained models used.

Later we load the pre-trained models: BERT, RoBERTa and XLNet and do the training phase.

And lastly, we get the classification results and apply the relevant classification metrics to evaluate.

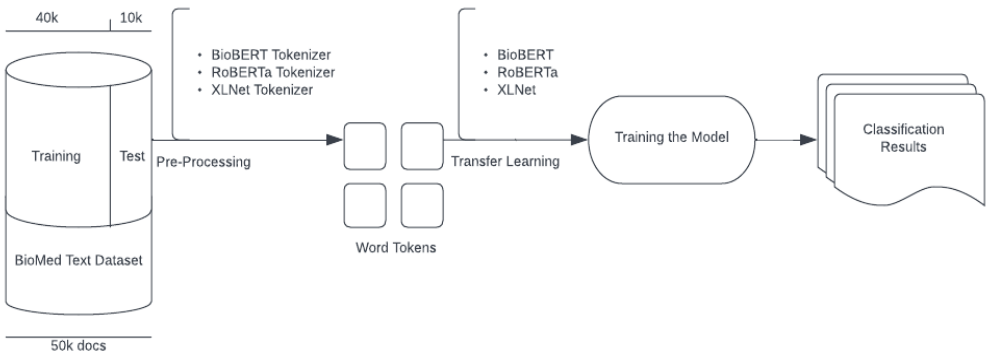


Figure 16: Classification Process using Pre-Trained Models.

1.6.1 Dataset

Original Version of this Dataset contains 15,559,157 Articles from BioASQ Task 9A, more details about the format of the data and the task are available in the Guidelines for Task 9A [17].

The Dataset used currently consists of 50000 collections of research from PubMed repository.

Originally these documents are manually annotated by Biomedical Experts with their MeSH labels and each article are described in terms of 10-15 MeSH labels. In this Dataset we have huge numbers of labels present as a MeSH major which is raising the issue of extremely large output space and severe label sparsity issues. To solve this Issue Dataset has been Processed and mapped to its root as Described in the Figure:

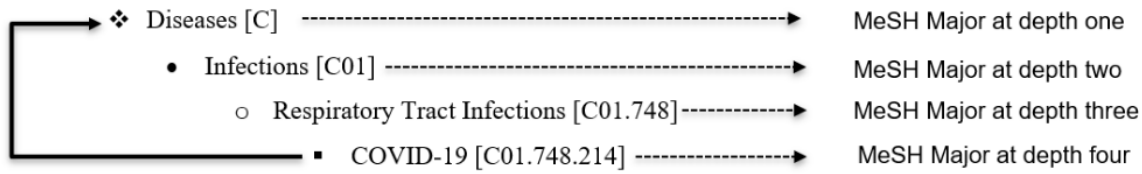


Figure 17: Structure of Processed Dataset.

The training set is served as a JSON string with the following format, where each line is a JSON object that represents a single article:

```
   {"articles": [
   {"title":"title..","abstractText":"text..","meshMajor":["me
sh1",...,"meshN"], "pmid":"PMID","meshid","meshroot"},
   ...,
   {..}
   ]}
```

This dataset has two new columns compared to the original unprocessed BioASQ dataset:

- Meshroot is MeSH Major at depth one.
- Meshid is the MeSH Major at the last depth.

The Dataset has multiple Labels which includes: *Anatomy, Organisms Diseases, Chemicals and Drugs Analytical, Diagnostic and Therapeutic Techniques, and Equipment Psychiatry and Psychology Phenomena and Processes Disciplines and Occupations Anthropology, Education, Sociology, and Social Phenomena Technology, Industry, and Agriculture Information Science Named Groups Health Care, Geographicals..*

Category Name	Number of Abstracts
Anatomy – A	23263
Organisms – B	46577
Diseases – C	26453
Chemicals and Drugs – D	31074
Analytical, Diagnostic and Therapeutic Techniques, and Equipment – E	39202
Psychiatry and Psychology – F	8885
Phenomena and Processes – G	33609
Disciplines and Occupations – H	6069
Anthropology, Education ,Sociology, and Social Phenomena – I	5595
Technology, Industry, and Agriculture – J	5531
Information Science – L	7503
Named Groups – M	21363
Health Care – N	22919
Geographicals – Z	8049

Table 7: Table showing dataset categories and the number of abstracts included in each.

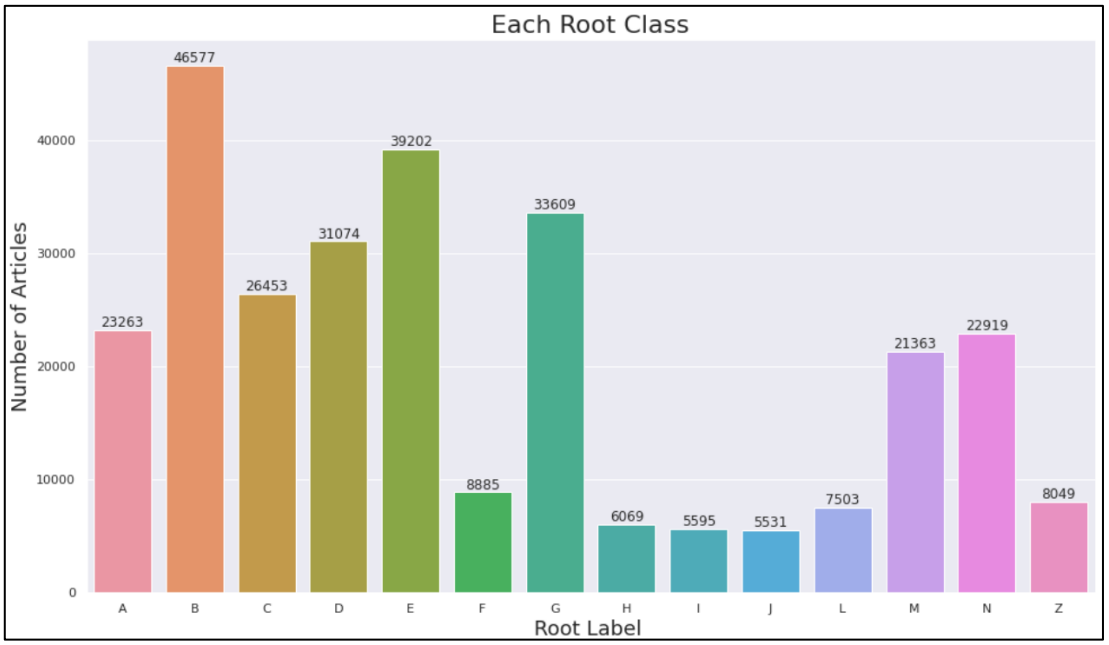


Figure 18: A graph representation of the different categories present in the dataset with the according number of abstracts.

During the training process, the dataset is split into the training set and validation set. In this research, the training set consists of 80% of the whole dataset, while the test validation set consists of 20% of the whole dataset.

1.6.2 Data Pre-Processing

Text preprocessing is used to clean up text data before feeding it to a machine-learning model. Text data contains a variety of noise, such as emotions, punctuation, and text in a different capitalization or in numerical or special character forms. Because machines cannot understand words, they require numbers. And therefore, a fast and efficient way to transform text to numbers is needed.

1.6.2.1 Standard Procedure of Text Pre-Processing

The standard or conventional procedure of pre-processing is a bit tedious and also a user-centric procedure. The below steps are carried out under the hood of standard pre-processing techniques:

- **Lower-casing the corpus**

Although often overlooked, lower-casing is one of the simplest and most effective forms of text preprocessing. It is suitable for most text mining problems (Text Mining) and NLP and contributes significantly to the consistency of the expected output. Capitalization consists of reducing all letters to lowercase. It's often a good practice: this will allow instances as "Product" at the beginning of a sentence to match the query "product", because both have the same meaning, if they are not converted to lowercase then they will constitute dissimilar words in the vector space model. On the other hand, such practice could assimilate words with completely different meanings. Many proper nouns are derived from generic nouns and are therefore only distinguished only under specific circumstances. These words include, for example, personal names (Rose, rose),.., etc.

Raw	Lowercased
Canada CanadA CANADA	canada
TOMCAT Tomcat toMcat	tomcat

- **Normalization**

Before proceeding with the preprocessing of the corpus it first must be normalized. Normalization generally refers to a set of related tasks aimed at put all the text in the same format; i.e. convert the text into a single canonic form. Normalizing the corpus before processing it guarantees the consistency of entry before any operation is performed. However, normalization requires knowing the type of text to be normalized and how to treat it. In our case study, the language used is English. Among the operations included in the standardization process, we find:

- Elimination of duplicate white spaces.
- Removal of punctuation and special characters.
- The substitution of contractions (very common in English, for example: "I'm" → "I am").

- The conversion of numbers into words to keep only the alphabetical characters.
- Removing the stop words: In this phase we will delete all stop words (personal pronouns, prepositions, etc.).

With Stop Words	Without Stop Words
/growing-up-with-hearing-loss/	/growing-hearing-loss/

- **Tokenizing the Corpus**

Tokenization is an NLP task that consist/s of splitting a piece of text into smaller units called “tokens”. A token is an instance of a sequence of characters in a specific document that are combined together as a semantic entity that may be useful for processing. Whether it's dividing a paragraph to sentences, a sentence to words or a word to characters, these tokens are either sentences, words or characters. So tokenization can be generally divided into three categories: the tokenization of sentences, words and characters (n-gram characters).the relevant tokenization here is word tokenizing, as shown in the following figure:

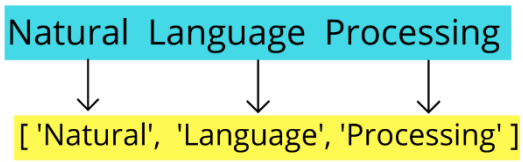


Figure 19: Tokenization.

- **Stemming and Lemmatization**
  - *Stemming*: is the process of reducing infected words to their stem; i.e. removing the last few characters of a given word, to obtain a shorter form, even if that form doesn’t have any meaning.
    For example: History and Historical becomes Histori.
  - *Lemmatization*: is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatisation depends on correctly identifying the intended part of speech and meaning

of a word in a sentence, as well as within the larger context surrounding that sentence.

- **Word Embeddings**

Word embedding is a language modelling technique to represent the words or phrases as vectors of real numbers. The words are grouped together to get similar representation for words with similar meaning. The word embedding learns the relationship between the words to construct the representation. This is achieved by the various methods like co-occurrence matrix, probabilistic modelling, and neural networks. It has become one of the basic knowledge in natural language processing.

One-hot vectors is one of the simple representations of the words. Each word can be represented by one hot vector but as the number of words increases the dimensionality increases. Word Embeddings reduces this dimensionality of the word vectors by using various ways such how the words occur collectively like King , Queen or words which can be used alternatively like car , vehicle etc. So similar words are represented by similar representation of vectors. This reduces dimensionality of the vectors.

Word Embeddings are categorized into two types:

- Frequency based embeddings for example: Count Vector, Co-occurrence vector, HashingVectorizer, TF-IDF.
- Pre-trained word embeddings for example: Word2Vec, GloVe, BERT, fastText.

Usually while approaching any NLP problem, we tend to follow this process and the above process does not ensure any reasonable result if our raw data changes slightly. This means if the data is from a web page there, we need additional work to remove HTML tags. Nowadays all these pre-processing steps can be carried out by using transfer learning modules like BERT.

1.6.2.2 Advanced Procedure of Text Pre-Processing in BERT

Bidirectional Encoder Representations from Transformers or BERT [18] is a deep learning framework, developed by Google that can be applied to NLP. This means that the NLP BERT framework learns information from both the right and left side of

a word (or token in NLP parlance). This makes it more efficient at understanding context.

To learn the contextual relationships between words in a text, BERT utilizes a Transformer, an attention mechanism. A transformer is a neural network that transduces input data into vector representations using self-attention layers with encoder–decoder structure. The transformer's vanilla implementation has two mechanisms: an encoder that receives text input and a decoder that predicts the task. Only the encoder mechanism is required because the purpose of BERT is to construct a language model.

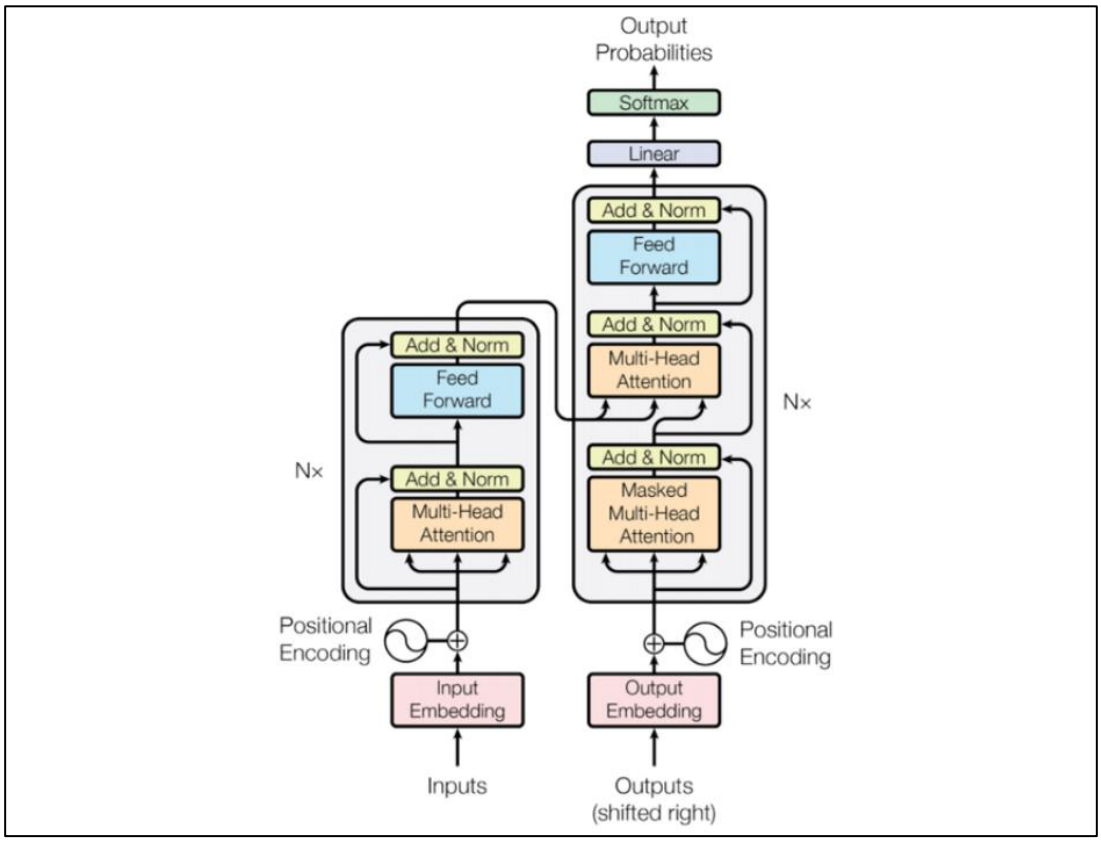


Figure 20: The Transformer network as described in the "Attention is all you need" paper [22].

The Transformer encoder reads the entire sequence of words at once, unlike directional versions that read the text input sequentially. It is classed as bidirectional as a result of this, while the actual term is non-directional. This feature allows the model to learn a word's context based on its surroundings.

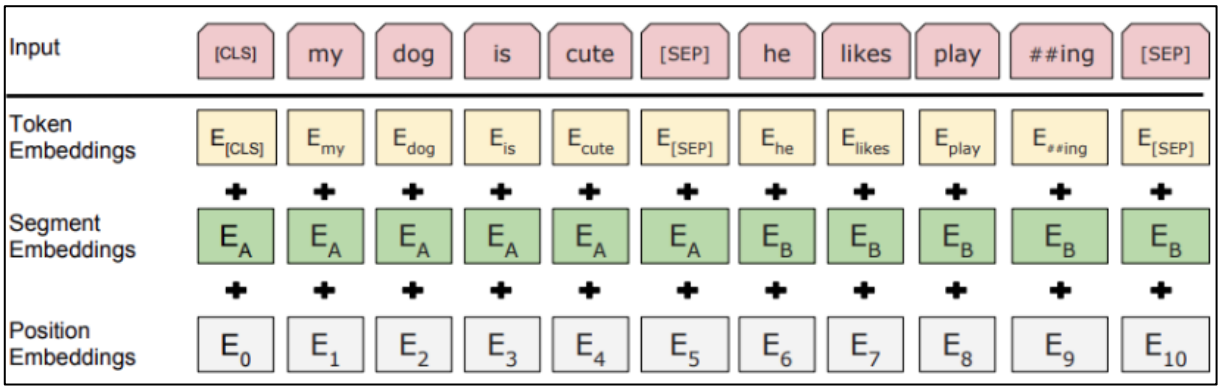


Figure 21: BERT input representation.

The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

1.6.3 Transfer Learning

It is a popular approach in deep learning where pre-trained models are used as the starting point on natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems, this is called Transform Learning.

The basic principle of transfer learning is simple: take a model trained on a large set of data and transfer their knowledge to a smaller dataset.

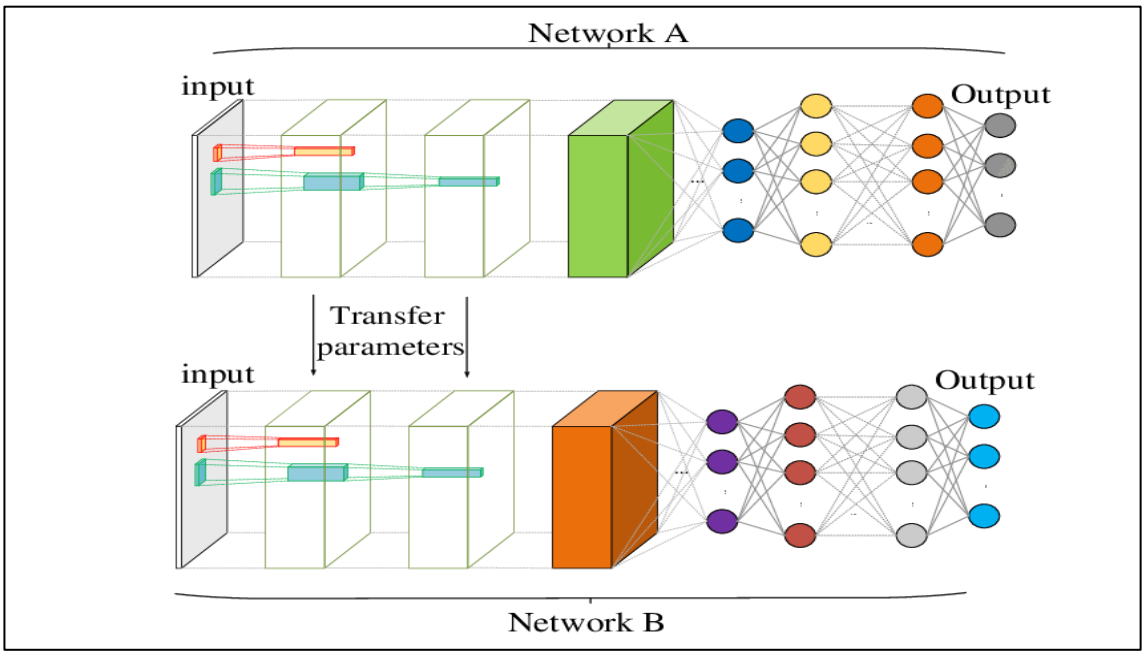


Figure 22: Illustration that shows the principle behind transfer learning [23].

1.6.3.1 BERT Model

During the BERT training process, pairs of sentences are provided as input to the model, and it learns to predict whether or not the second sentence in the pair is the following sentence in the original document. Half of the inputs during training are pairs where the second sentence is the next sentence in the original document while the other half is a random sentence from the corpus. The underlying assumption is that the second phrase will be unrelated to the first.

During training, as shown above, a [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is introduced at the end of each sentence, with each token containing a sentence embedding indicating Sentence A or Sentence B. Sentence embeddings are essentially similar to token embeddings, but with a two-word vocabulary. Finally, each token is assigned a positional embedding that corresponds to its place in the sequence [18].

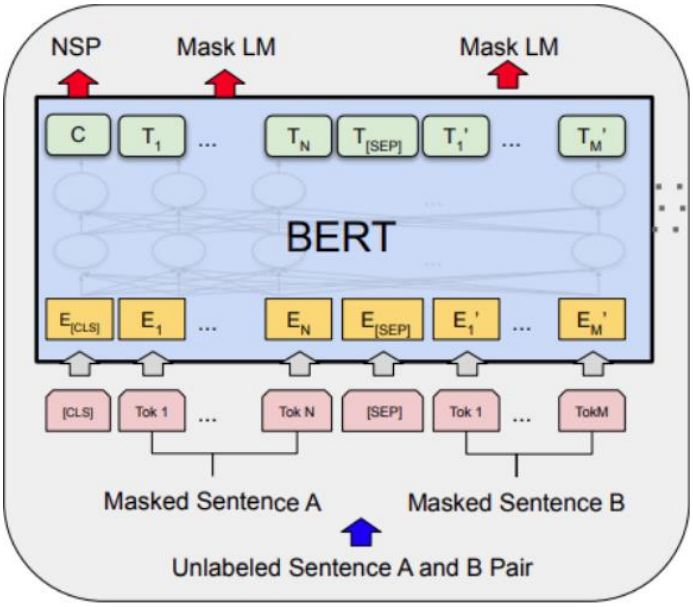


Figure 23: Overall pre-training procedure for BERT.

[CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

Before feeding word sequences into BERT, some part of each sequence is replaced with a [MASK] token. The model then makes an attempt to forecast the original value of the masked words using the context provided by the other, non-masked phrases in the sequence. It is necessary to add a classification layer on top of the encoder output in order to predict the output words. This is followed by multiplying the encoder output vectors by the embedding matrix, transforming them into the vocabulary dimension, and computing the probability of each word in the vocabulary using softmax [18].

The BERT loss function only considers the predictions of the masked values and ignores the predictions of the non-masked words. Consequently, the model converges more slowly than directional models. When learning the BERT model, Masked LM (shown in Figure 22) and Pre-training (shown in Figure 24) are trained jointly in order to minimize the combined loss function of the two techniques [18].

BERT currently has two variants:

- BERT Base: 12 layers, 12 attention heads, and 110 million parameters.
- BER Large: 24 layers, 16 attention heads, and 340 million parameters.

1.6.3.2 BioBERT Model

BioBERT (Bidirectional Encoder Representations from Transformers for Biomedical Text Mining) [19], which is a domain specific language representation model pre-trained on large-scale biomedical corpora. Based on the BERT architecture, BioBERT effectively transfers the knowledge of large amount of biomedical texts into biomedical text mining models. While BERT also shows competitive performances with previous state-of-the-art models, BioBERT significantly outperforms them on three representative biomedical text mining tasks including biomedical named entity recognition (**1.86%** absolute improvement), biomedical relation extraction (**3.33%** absolute improvement), and biomedical question answering (**9.61%** absolute improvement) with minimal task-specific architecture modifications [19].

Table 8: List of text corpora used for BioBERT.

Corpus	Number of words	Domain
English Wikipedia	2.5B	General
BooksCorpus	0.8B	General
PubMed Abstracts	4.5B	Biomedical
PMC Full-text articles	13.5B	Biomedical

1.6.3.3 RoBERTa Model:

Additionally to BioBERT, we used the RoBERTa model [20], RoBERTa builds on BERT's language masking strategy, and wherein the system learns to predict intentionally hidden sections of text within otherwise unannotated language examples. RoBERTa, which was implemented in PyTorch, modifies key hyper-parameters in BERT, including:

- Removing the Next Sentence Prediction (NSP) objective: In the next sentence prediction, the model is trained to predict whether the observed document segments come from the same or distinct documents via an auxiliary Next Sentence Prediction (NSP) loss. The authors experimented with removing/adding of NSP loss to different versions and concluded

that removing the NSP loss matches or slightly improves downstream task performance

- Training with bigger batch sizes & longer sequences: Originally BERT is trained for 1M steps with a batch size of 256 sequences. In this paper, the authors trained the model with 125 steps of 2K sequences and 31K steps with 8k sequences of batch size. This has two advantages, the large batches improves perplexity on masked language modelling objective and as well as end-task accuracy. Large batches are also easier to parallelize via distributed parallel training.
- Dynamically changing the masking pattern: In BERT architecture, the masking is performed once during data preprocessing, resulting in a single static mask. To avoid using the single static mask, training data is duplicated and masked 10 times, each time with a different mask strategy over 40 epochs thus having 4 epochs with the same mask. This strategy is compared with dynamic masking in which different masking is generated every time we pass data into the model.
- This allows RoBERTa to improve on the masked language modeling objective compared with BERT and leads to better downstream task performance.

1.6.3.4 XLNet Model

Another recently developed Model is XLNet, XLNet has a similar architecture to BERT. However, the major difference comes in its approach to pre-training.

XLNet is a generalized auto-regressive (AR) language model that enables learning bidirectional contexts using Permutation Language Modeling. XLNet borrows the ideas from both AE and AR language model while avoiding their limitation. As per paper, XLNet outperforms BERT on 20 tasks, often by a large margin, including question answering, natural language inference, sentiment analysis, and document ranking [21].

XLNet outperforms BERT for mainly two reasons:

- BERT is pre-trained on two unsupervised tasks: sentence reconstruction and next sentence prediction. The reconstruction task involves randomly masking tokens in a sentence, and reconstructing the original sentence from the masked one. The model reconstructs the masked tokens conditionally independently of one another. However, this is not really a valid assumption. An example from the paper uses the sentence, "New York is a city" where "New" and "York" are masked and reconstructed during training. Clearly, if the first word is "New" then the next word is more likely to be "York", XLNet doesn't make this assumption during pre-training.
- BERT also uses a special [mask] token during pre-training, which creates some discrepancy between pre-train and fine-tune stages, where a special token such as [mask] is not used for the latter.

1.7 Conclusion

In this chapter we detailed the dataset used and explained how the pre-processing works in both the standard procedure and the advanced, we furthermore explained the basics behind the concept of Transfer Learning, moreover how the used pre-trained models work including: BERT, RoBERTa and XLNet.

CHAPTER V :

Tests and Evaluation

Chapter 5: Implementation and Evaluation

1.1 Introduction

This chapter deals with implementing the method described in the previous chapter, and gives a thorough look at the entire process.

It will include a view at the environment employed and all the libraries and APIs used, additionally it will showcase the implementation with code detail, moreover a presentation of the classification results, and a comparative study with previous related works. And lastly in the simulation part, we will see the classification in action using a user-friendly interface.

1.2 Hardware Resources

The table hereafter (Table 9) describes the specifications of the computer that has been used for the tests.

Table 9: Computer Specifications.

CPU	I5 8265U 1.60ghz 8 cores
GPU	Mesa Intel UHD graphics 620
Storage	256gb
Memory	8gb
Operating System	Ubuntu 20.04.4 LTS 64bits

1.3 Software Resources

In this section we will present the software resources used during the implementation including: the programming language, frameworks, as well as the libraries.

1.3.1 Programming Language and frameworks

Here are the programming language and Frameworks that we have used.

- **Python Programming Language:**

is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation, Python 3.9.5 is the latest, major stable release of the Python programming language, and contains a number of new features and optimizations, with 111 commits since version 3.9.4.

- **Jupyter Notebook:**

is a web-based interactive computational environment for creating notebook documents. Jupyter Notebook[1] is built using several open-source libraries, including IPython, ZeroMQ, Tornado, jQuery, Bootstrap, and MathJax.

- **Google Collaboratory:**

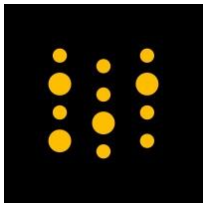
[1] https://jupyter.org/

Or "Colab" for short, is a product from Google Research. Colab[2] allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs.

- **Kaggle:**

A subsidiary of Google, Kaggle[3] allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

- **Weights & Biases:**



WandB[4] is a central dashboard to keep track of your hyper-parameters, system metrics, and predictions so you can compare models live, and share your findings.

1.3.2 Libraries

Hereafter, we describe the libraries that have been used in our project.

[2] https://colab.research.google.com/
[3] https://www.kaggle.com/
[4] https://wandb.ai/site

- **Scikit-learn (for scientific/data models)[5]:**

is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

- **Pandas (for tables)[6]:**

is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

- **Keras (for Neural Networks)[7]:**

is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

- **Tqdm:**

[5] https://scikit-learn.org/
[6] https://pandas.pydata.org/
[7] https://keras.io/

tqdm[8] derives from the Arabic word taqaddum (تقدّم) which can mean "progress," and is an abbreviation for "I love you so much" in Spanish (te quiero demasiado).

Instantly make your loops show a smart progress meter - just wrap any iterable with tqdm(iterable), and you're done!

- **Numpy (for numerical analysis)[9]:**

is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

- **Matplotlib (for plots)[10]:**

is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

- **TensorFlow[11]:**

is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

---

[8] https://github.com/tqdm/tqdm
[9] https://numpy.org/
[10] https://matplotlib.org/
[11] https://www.tensorflow.org/?hl=fr

- **PyTorch[12]:**

is an open-source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing.

- **Transformers[13]**:

Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:

- Natural Language Processing: text classification, named entity recognition, question answering, language modeling, summarization, translation, multiple choice, and text generation.
- Computer Vision: image classification, object detection, and segmentation.
- Audio: automatic speech recognition and audio classification.
- Multimodal: table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

[12] https://pytorch.org/
[13] https://huggingface.co/transformers

- **Gradio[14]:**

  is an open-source python library that allows you to quickly create easy-to-use, customizable UI components for your machine learning model. Gradio allows you to integrate the GUI directly into your Python notebook making it easier to use.

1.4 Implementation and Results

In this section, we will explain how the implementation was carried with lines of code and screenshots of the results of all the architecture process.

1.4.1 Data Loading and Visualization

At first, we start by loading the dataset we're working on and visualize it:

```
dataset='datasets/PubMed Multi Label Text Classification Dataset Processed.csv'

df= pd.read_csv(dataset)
df.head(4)
```

	Title	abstractText	meshMajor	pmid	meshid	meshroot	A	B	C	D	E	F	G	H	I	J	L	M	N	Z
0	Expression of p53 and coexistence of HPV in pr...	Fifty-four paraffin embedded tissue sections f...	['DNA Probes, HPV', 'DNA, Viral', 'Female', 'H...	8549602	[['D13.444.600.223.555', 'D27.505.259.750.600....	['Chemicals and Drugs [D]', 'Organisms [B]', '...	0	1	1	1	1	0	0	1	0	0	0	0	0	0

We display each category and count the number of abstracts in each:

```
counts = []
for mesh_Heading_category in mesh_Heading_categories:
    counts.append((mesh_Heading_category, df[mesh_Heading_category].sum()))
df_count = pd.DataFrame(counts, columns=['Root Label', 'number of Abstract'])
df_count
```

[14] https://gradio.app/

	Root Label	number of Abstract
0	A	23263
1	B	46577
2	C	26453
3	D	31074
4	E	39202
5	F	8885
6	G	33609
7	H	6069
8	I	5595
9	J	5531
10	L	7503
11	M	21363
12	N	22919
13	Z	8049

We split the dataset into 80% (40000) for training and 20% (1000) for testing and then use one hot labels to further visualize our data:

```
df_train, df_test = train_test_split(df, random_state=32, test_size=0.20, shuffle=True)
df_train['one_hot_labels'] = list(df_train[mesh_Heading_categories].values)
df_train.head(3)
```

	Title	abstractText	meshMajor	pmid	meshid	meshroot	A	B	C	D	...	F	G	H	I	J	L	M	N	Z	one_hot_labels
31112	Neurofibromatosis associated with a coronary a...	A case of a patient with type 1 neurofibromato...	['Aged', 'Coronary Aneurysm', 'Humans', 'Male'...	11144803	[['M01.060.116.100'], ['C14.280.647.250.250', ...	['Named Groups [M]', 'Diseases [C]', 'Organism...	0	1	1	0	...	0	0	0	0	0	0	1	0	0	[0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0]

1.4.2 PreTrained Models Implmentation:

We take the same outputs and split the training data into 80 % (32000) for the training and 20 % (8000) for validation, we then convert the data into torch tensors to use it in the training of the model:

In the training we used a number of `epochs = 12` as the original authors of the model recommend between 10 and 20.

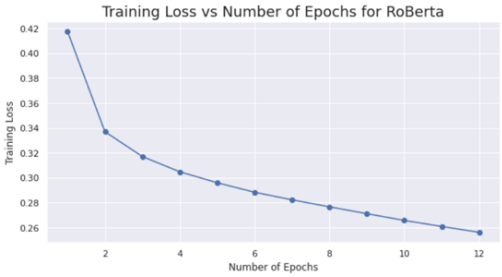
After the Training is over we evaluate our model by calculating the accuracy a `threshold = 0.50` was used.

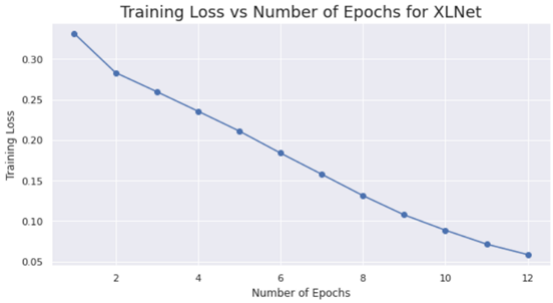
For each Model we used the appropriate Tokenizer, for example for RoBERTa we use the RoBERTa tokenizer.

We used the GPU in order to perform the training.

1.4.3 Evaluation

During the training we calculate the Training Loss, F1 validation Accuracy, and the Flat Validation Accuracy of each model, the training loss is a metric used to assess how a deep learning model fits the training data:





After the testing is done we calculate the precision, recall, and f1-score for each model:

BioBERT:

Test F1 Accuracy: 0.8513293827678432
Test Flat Accuracy: 0.1707

	precision	recall	f1-score	support
A	0.78	0.81	0.80	4609
B	0.97	0.98	0.97	9250
C	0.89	0.86	0.88	5206
D	0.91	0.93	0.92	6259
E	0.82	0.94	0.88	7778
F	0.83	0.72	0.77	1767
G	0.84	0.89	0.86	6799
H	0.54	0.24	0.33	1221
I	0.66	0.64	0.65	1068
J	0.72	0.56	0.63	1110
L	0.59	0.57	0.58	1491
M	0.88	0.87	0.88	4232
N	0.82	0.77	0.79	4602
Z	0.71	0.74	0.73	1558
micro avg	0.85	0.86	0.85	56950
macro avg	0.78	0.75	0.76	56950
weighted avg	0.84	0.86	0.85	56950
samples avg	0.85	0.86	0.84	56950

RoBERTa:

Test F1 Accuracy: 0.8500327439423706
Test Flat Accuracy: 0.1676

	precision	recall	f1-score	support
A	0.76	0.83	0.79	4609
B	0.96	0.98	0.97	9250
C	0.88	0.87	0.88	5206
D	0.91	0.93	0.92	6259
E	0.81	0.94	0.87	7778
F	0.81	0.72	0.77	1767
G	0.82	0.90	0.86	6799
H	0.59	0.15	0.24	1221
I	0.68	0.63	0.65	1068
J	0.64	0.58	0.61	1110
L	0.69	0.44	0.54	1491
M	0.86	0.91	0.89	4232
N	0.83	0.75	0.79	4602
Z	0.72	0.72	0.72	1558
micro avg	0.85	0.85	0.85	56950
macro avg	0.78	0.74	0.75	56950
weighted avg	0.84	0.85	0.84	56950
samples avg	0.85	0.86	0.84	56950

XLNet:

```
Test F1 Accuracy:  0.834202672894771
Test Flat Accuracy:  0.133
```

	precision	recall	f1-score	support
A	0.76	0.80	0.78	4609
B	0.96	0.98	0.97	9250
C	0.86	0.87	0.87	5206
D	0.89	0.94	0.92	6259
E	0.84	0.84	0.84	7778
F	0.74	0.78	0.76	1767
G	0.83	0.84	0.84	6799
H	0.39	0.39	0.39	1221
I	0.63	0.65	0.64	1068
J	0.62	0.62	0.62	1110
L	0.59	0.52	0.55	1491
M	0.87	0.89	0.88	4232
N	0.74	0.82	0.78	4602
Z	0.64	0.76	0.69	1558
micro avg	0.82	0.85	0.83	56950
macro avg	0.74	0.76	0.75	56950
weighted avg	0.82	0.85	0.83	56950
samples avg	0.83	0.85	0.82	56950

1.5 Comparisons and Discussion

The BioBERT and RoBERTa models gave comparable F1 accuracy Results, with 85,1% for BioBERT and 85% for RoBERTa. XLNet performed slightly worse with an F1 accuracy result of 83%.

BioBERT was expected to perform the best giving how it was trained on a huge dataset of biomedical texts, interestingly the results of RoBERTa shows the huge potential, and is evidence that it's an improvement over the standard BERT model.

Later we compare these results to the previous results shown in the state of the art section:

Table 10 :Results comparison between proposed approach and state of the art.

Model	CNN+word2vec 1	CNN+word2vec 2	Improved CNN	BioBERT	roBERTa	XLNet
Average Precision	68%	71%	91%	85%	85%	82%

We can see that compared to both CNN+word2vec models the pre-Trained models gave better precisions, the improved CNN model shows that

with the proper amelioration: CNN can show great results comparable to pre-trained models.

1.6 Simulation

1.6.1 Trained dataset using the pre-trainer model

To make visualizations and logging artifacts and comparisons of different models we have thought about integrating Weight and Bias (wandb). It helps us to quickly track experiments, version and iterate on datasets, evaluate model performance, reproduce models, visualizes results and spot regressions, and share findings publicly, following these steps:

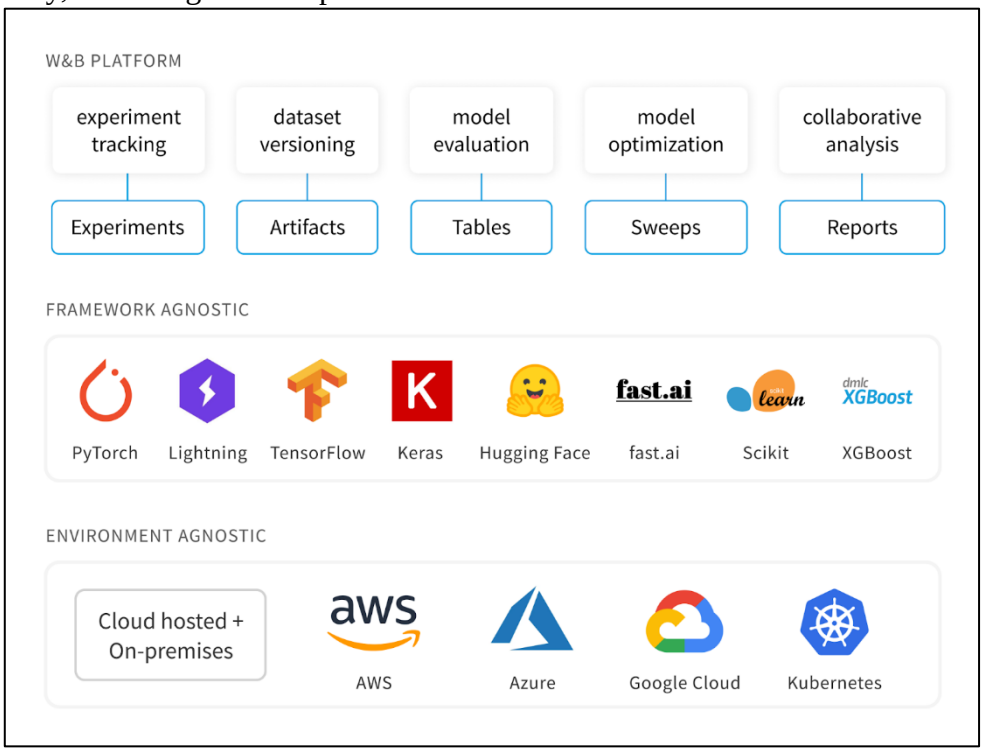


Figure 24: Model Deployment in the HuggingFace.

- We sign up and log in to the wandb account.
- We Install the wandb library on our machine in a Python 3 environment using pip

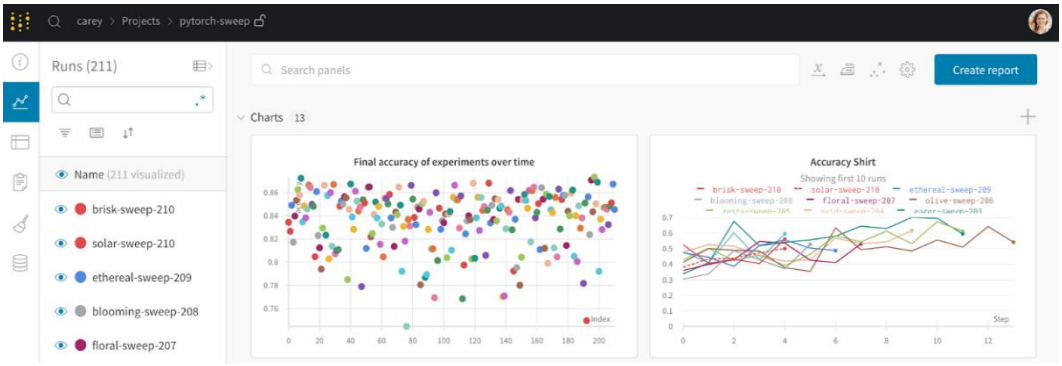
Command Line | Notebook

```
!pip install wandb

wandb.login()
```

- We then, login to the wandb library on the machine and get the app key.

- We Initialize a new run in W&B in Python script. wandb.init() will start tracking system metrics and console logs, right out of the box. Run our code, and we put in Our API key when prompted, and we see the new run appear in W&B.



1.6.2 Storing the trained model

The base classes *PreTrainedModel*, implement the common methods for *loading/saving* a model either from a local file or directory, or from a pre-trained model configuration provided by the library (downloaded from HuggingFace's AWS S3 repository).

PreTrainedModel and TFPreTrainedModel also implement a few methods which are common among all the models such as Tokenizers.

After making the train locally, we saved the pre-trained configuration, Converting Labels to categories before uploading it to HugginFace Hub. Finally, we Upload the model file to the Model Hub while synchronizing a local clone of the repo in repo_path_or_name

1.6.3 Call and Use of the dataset trained

In Transformers 4.20.0, the from_pretrained() method has been reworked to accommodate large models using Accelerate. This requires Accelerate >= 0.9.0 and PyTorch >= 1.9.0. Instead of creating the full model, then loading the pre-trained weights inside it (which takes twice the size of the model in

RAM, one for the randomly initialized model, one for the weights), there is an option to create the model as an empty shell, then only materialize its parameters when the pre-trained weights are loaded.

This option can be activated with low_cpu_mem_usage=True. The model is first created on the Meta device (with empty weights) and the state dict is then loaded inside it (shared by shard in the case of a sharded checkpoint). This way the maximum RAM used is the full size of the model only.

Moreover, we can directly place the model on different devices if it does not fully fit in RAM (only works for inference for now). With device_map="auto", Accelerate will determine where to put each layer to maximize the use of our fastest devices (GPUs) and offload the rest on the CPU, or even the hard drive if we do not have enough GPU RAM (or CPU RAM). Even if the model is split across several devices, it will run as we would normally expect.

1.6.4 Building the Web App in Python

Gradio is an open-source Python library that is used to build data science demos and web applications.

With Gradio, we have created a user friendly interface around our three models such as Bio Bert, Roberta, and XLNet, and let the users try by dragging and dropping in their content, pasting text, and interacting with the demo, all through the browser, using the below link: https://huggingface.co/spaces/saidhr20/pubmed-biobert-text-classification

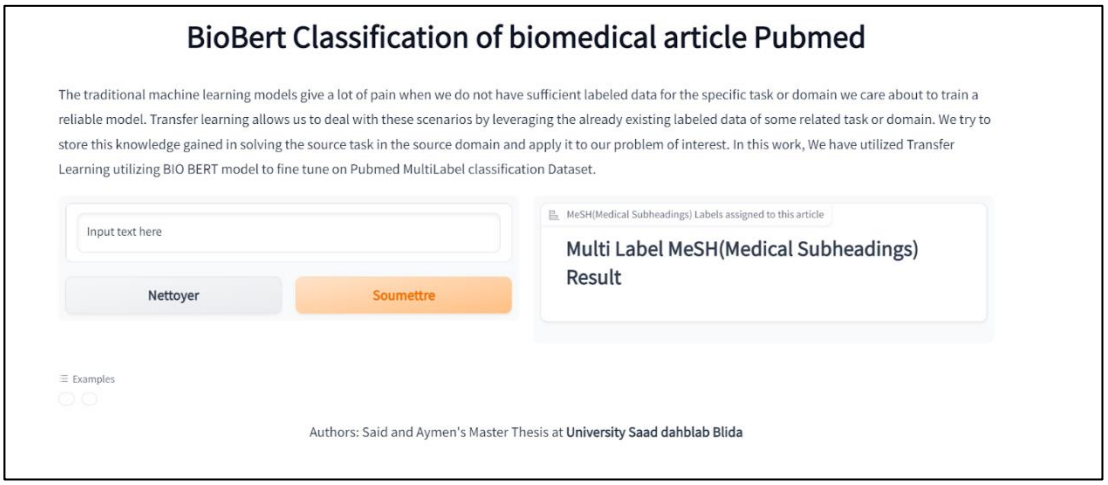
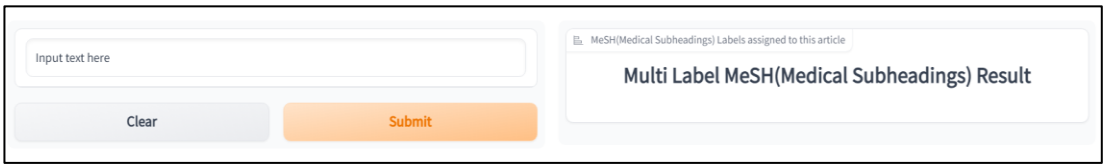


Figure 25: Application Interface.

In order to perform a classification by writing an input text and pressing the submit button, the classification results of that input text will be displayed, the 6 most pertinent categories will be shown, the model used here is BioBERT:



For example, we'll search the content of a biomedical abstract and see the results:

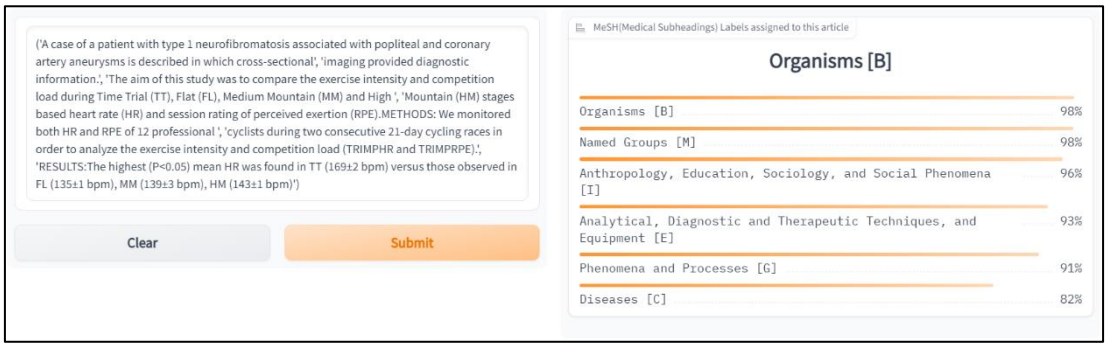


Figure 26: Exemple of a classification with our application.

1.7 Conclusion

In this chapter we dealt with implementing the different models and showing the process of training and testing and evaluation.

Furthermore, we analyzed and compared the results of the 3 models, and later with the state-of-the-art approaches.

Lastly, we showcased the classification results in a user interface, in order to see the results clearly and easily.

GENERAL CONCLUSION

2 General Conclusion

2.1 Conclusion

In this thesis we explored the deep learning field of biomedical text classification we started by understanding basic notions of automatic classification and went further in explaining deep learning concepts, in addition we studied three state of the art theses about biomedical text classification and compared their results which gave us an informed overall review of the biomedical text classification field, furthermore we have tested three pre-trained biomedical text models, and evaluated their results, we compared their results with the ones obtained in the state of the art theses, we found out that the BioBERT model performs best than the other two pre-trained models, and finally we showcased the model in an interface that can be easily used by anyone online to do a biomedical text classification on the go.

The prospective of developing a robust method to do classification is huge, the method can take advantage of pre-trained models as they're showing very good results even ones that are not trained on biomedical text classification like RoBERTa.

2.2 Perspectives

The biomedical text classification field is a vast field and has huge developmental potential, from this research the approach of developing a pre-trained model and taking advantage of the optimizations like the ones used in RoBERTa and training it on huge datasets of biomedical text sounds like it would make great results that can even exceed state of the art biomedical methods developed recently, and that's obviously up to the test, we hope we can see such methods developed in the future.

REFERENCES

References

[1] C. Sebastien, «Classification automatique de textes biomédicaux,» p. 45, 2015.

[2] D. O. Thomas W.Edgar, «Machine Learning».

[3] V. B. S. Prasatha, «Effects of Distance Measure Choice on KNN Classifier Performance - A Review,» p. 39, 2019.

[4] N. S. Chauhan, «Decision Tree Algorithm,» 2022. [En ligne]. Available: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html.

[5] E. García-Gonzalo, «Hard-Rock Stability Analysis for Span Design in Entry-Type Excavations with Learning Classifiers,» p. 19, 2016.

[6] S. H. K. a. K. PANISKAKI, «Text analysis for email multi label classification,» Gothenburg, Sweden, 2020.

[7] Yann LeCun et Yoshua Bengio, «Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks,» 1995.

[8] S. R. Medina, «Multi-Label Text Classification with Transfer Learning for Policy Documents,» Uppsala, Sweden, 2019.

[9] Razvan Pascanu, , Tomas Mikolov et Yoshua Bengio, «On the difficulty of training Recurrent Neural Networks,» p. 12, 2013.

[10] K. Ryszard Tadeusiewcz "Sieci neuronowe", «Principles of training multi-layer neural network using backpropagation,» 1992. [En ligne]. Available: https://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html.

[11] Mark Hughes, Irene Li, Spyros Kotoulas et Toyotaro Suzumura, «Medical Text Classification Convolutional Neural Networks,» p. 5, 2016.

[12] Rozilawati Dollah, Chew Yi Sheng et Norhawaniah Zakaria, «Deep Learning Classification of Biomedical Text,» p. 6, 2019.

[13] Liang-Chih Yu, Jin Wang et K. Robert Lai, «"Refining Word Embeddings For Sentiment Analysis",» pp. 534-539, 2017.

[14] I. P. e. al., «Continious Space Word Vectors Obtained by Applying Word2Vec to Abstracts of Biomedical Articles,» pp. 1-4, 2014.

[15] Z. Ma, «"Improved Convolutional Neural Network for Biomedical Text Classification",» p. 7, 2021.

[16] Xinying Song,, Alex Salcianu, Yang Song et Dave Dopso, «Fast WordPiece Tokenization,» p. 15, 2020.

[17] T. George , B. Georgios , M. Prodromos , P. Ioannis et Z. Matthias , «An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition,» 2015. [En ligne]. Available: http://participants-area.bioasq.org/datasets/.

[18] Jacob Devlin, Ming-Wei Chang et Kenton Lee, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,» p. 16, 2018.

[19] Jinhyuk Lee , Wonjin Yoon, Sungdong Kim et Donghyeon, «BioBERT: a pre-trained biomedical language representation model for biomedical text mining,» p. 7, 2019.

[20] Yinhan Liu, Myle Ott et Naman Goyal, «RoBERTa: A Robustly Optimized BERT Pretraining Approach,» p. 13, 2019.

[21] Zhilin Yang, Zihang Dai, Yiming Yang et Jaime Carbon, «XLNet: Generalized Autoregressive Pretraining for Language Understanding, Available:https://arxiv.org/pdf/1906.08237.pdf,» p. 18, 2020.

[22] Ashish Vaswani, Noam Shazeer et Niki Parmar, «Attention Is All You Need,» p. 15, 2017.

[23] Joseph Lemley, Shabab Bazrafkan et Peter Corcoran, «Transfer Learning of Temporal Information for Driver Action,» p. 7, 2017.