ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

BUSINESS ANALYTICS
Master of Science

Machine Learning and Content Analytics –Spring Semester 2022

Professor: H.Papageorgiou, Instructor : G.Perakis
Students: Xheni Cobanaj (f2822117), Nikolaos Mantzakos (f2822107), Dimitris Papageorgiou (f2822109)

02/09/2022

Project: MBTI Personality Detection

Table of Contents

Introduction

The way that a person behaves in a particular situation or under particular conditions is commonly studied and supervised by many areas of research. For instance, the situation and condition may vary but lead to an inference of personality indicator. Currently many personality tests are available and used by recruiters and companies, universities and other fields to access someone's behavior and identifying human personality. Among the most known ones are the Myers-Briggs Type Indicator, the DISC personality test and the Caliper Profile one. In this current era of rise in Artificial Intelligence and Machine Learning many algorithms have been developed and trained to make such predictions in Natural Language Processing and Text classification with a very high level of accuracy.

Our project

The main idea of this project is to classify people into their Myers-Briggs Type Index (MBTI) personality type (output) based on text samples (input) from their posts in social media, employing Machine Learning algorithms such as Logistic Regression and XGBoost, and Deep Learning techniques such as Convolutional Neural Networks and Transformers and evaluating the accuracy of these models and their ability to predict the Myers-Briggs Personality Indicator. In other words, we will work on building models in order to detect a person's personality type in connection to his style of writing. Focusing on the business aspect of the project, we are mainly aiming at two ends. Firstly, to provide a solution that will improve the performance of online marketing, since, identifying personality types will help companies to build more efficient targeted campaigns reducing the advertising costs and increasing their revenues at the same time. Secondly, we want to assess how reliable are these tests in the process of evaluation of a particular candidate's personality by theirs answers in these type of assessments. Generally, we wanted to build a more generalizable system that can incorporate meaning of writing to determine overall personality types.

The MBTI

Personality type detection background

The "Myers-Briggs Type Indicator (MBTI)" is one of the most widely known approaches that spots a person's character, and states that random variation in behavior is accounted for by the way people use judgement and perception. This type of test classifies personality type into sixteen different types as shown in Figure 1 based on 4 key features/factors that categorize human behavior. This classification is based on a group of characteristics that defines the unique feelings, thoughts and behavior of every person [1,3]. The MBTI was created by Isabel Briggs Myers and Katharine Briggs, based on Carl Jung's theory on personality in which is assumed that people are born with preferences for different functions, and that these preferences interact with environmental factors [3]. Nowadays, there is a variety of personality models used to characterize personalities, such as the "Big Five Personality Traits model", "VIA Classification of Character Strengths", "Myers- Briggs Type Indicator (MBTI)" model, and "Jung's Theory of Personality Type models" [1]. Although the concept of personality is considered as useful and powerful, it lacks of preciseness and robustness and for this reason, psychologists are working on developing more empirical and solid models of

personality [2]. However, we choose the "MBTI" personality model for this project since it has been established as a more powerful model, having a broader application range in different disciplines and because of its popularity and the potential for utilizing it in a variety of fields.

Figure 1 The sixteen different personality types (https://en.wikipedia.org/wiki/Myers–Briggs_Type_Indicator)

Personality types

Focusing on the sixteen different types of personality, the dichotomies are Extraversion-Introversion, Sensing-Intuition, Thinking-Feeling, and Judging-Perceiving as we can also see from Figure 2, which they are usually represented by their first letter with "N" standing for Intuition [3]. Analyzing the aforementioned dichotomies, the E-I dichotomy describes how people respond and interact with the world around them with extraverts being action-oriented, enjoying social interaction and the introverts being thought-oriented, enjoying deeper/meaningful social interactions [4]. In addition, S-N dichotomy is looking at how people gather information from the world around them, with sensing people preferring to focus on facts and enjoying hands-on experience, while those who prefer intuition enjoy thinking abstract theories and imagining the future [4]. Following the T-F dichotomy, it focuses on how people make decisions based on the information gathered from their sensing or intuition functions. In specific, thinking personalities prefer to make consistent decisions

based on logical reasoning while feeling personalities are attached to their emotions [4]. In contrast with the E-I, S-N and T-F which are independent of each other, J-P was found to have link with at least one of the previous dichotomies [3]. The J-P scale focuses on how people tend to deal with the outside world with judging personalities prefer structure and robust decisions, while perceiving personalities are more supple and protean [4,5].

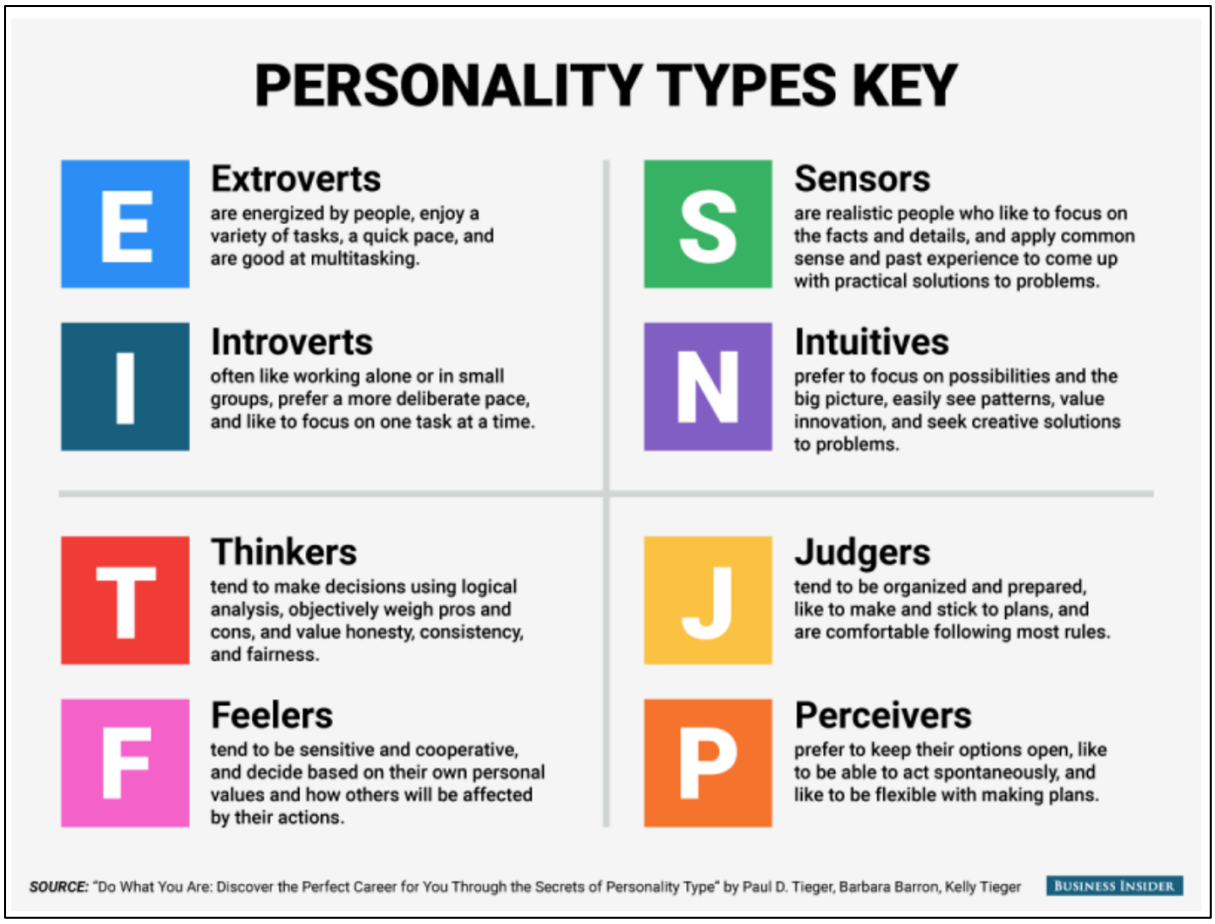

Figure 2 The four dichotomies of each personality

MBTI Applications in Business

Human personality plays an important role in any society. Personality type prediction is nowadays a useful technique that many organizations and educational sectors are using to understand an individual's personality type. The most popular use of MBTI is the MBTI Assessment Test. This assessment is mainly applied on job applicants to help recruiters identify the ideal fit for their company's built-in culture. In specific, after the job applicant finishes this psychometric questionnaire, its' psychological preferences on how he perceive the world and make decisions are measured and thus, a first view and relatively precise in general, is taken. Another application of the MBTI assessment in business is on companies' employees in order for the main personality aspects of them to be identified having as a result the improvement of the way that colleagues communicate with each other and hence, the teamwork. Apart from communication, the MBTI assessment inside companies could also help to eliminate conflicts among colleagues and capitalize on peoples' strengths [6].

Our Vision/Goals

There is no doubt that marketing is a crucial component to the success of any business, regardless the size. Since there is a critical link between psychology and marketing, every marketeer tries to find the characteristics of the target markets focusing mainly on customer's attitude, personality, beliefs and interests which together with the demographic characteristics provide smaller and more focused subsets of the potential customers [3,7]. The development of buyer or customer personas is an important part of every marketing strategy and provides the chance to map out the content marketing plan to each of the target audience's needs [8], due to the fact that the markets are not homogenous. Thus, since Myers Briggs Type Indicator is a universally accepted system for understanding different personalities, identifying personas using MBTI based on text samples in social media, could benefit companies in market segmentation and personality-targeted advertisements (one-to-one marketing) [3].In specific, with this project we aim on providing the answer to the business question of market segmentation through personality classification with MBTI on texts, and as a result, help companies to create targeted campaigns. Except this, companies will improve the communication bridge with the audience, using the personality characteristics that we will provide through the classification process explained in advance. For example, if we identify a customer as Feeling personality, it is more likely to respond to marketing messages that appeal to its' emotions rather than sense of logic. Similarly, if the target audience are Thinkers, a company should try more direct marketing campaigns with a clear and rational tone. A crucial point at this time which encouraged us to work on this project, is that MBTI detection through text is more robust and responds better to reality due to the fact that the "writer" of the text is unaware of the "monitoring" and thus, its behavior is closer to its genuine character/personality.

Data

Data Collection

The actual data source that our personality type dataset came from is Kaggle (https://www.kaggle.com/datasets/datasnaek/mbti-type). It consists of 8675 unique observations (people) and two "attributes". Every person is accompanied by its' Myers-Briggs personality type as a 4-letter code (1st column/type) and a collection of its' last 50 social networking posts (2nd column/posts) coming from "PersonalityCafe" forum. Every user has undertaken a questionnaire concerning the Myers-Briggs personality test prior to joining the forum and later has been interacting in this forum with other users. Each post is separated by "|||".

	type	posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...
1	ENTP	'I'm finding the lack of me in these posts ver...
2	INTP	'Good one _____ https://www.youtube.com/wat...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...
4	ENTJ	'You're fired.|||That's another silly misconce...

Figure 3 First rows of the dataset

Dataset Overview

As we can observe from figure 4 and figure 5, the distribution of the personality types is disproportional compared to roughly uniform distribution for the general population. In specific, the most frequent personality types that the dataset contains are INFP with 1832 (~21%) and INFJ with 1470 (~17%) occurrences. On the other hand, the personality types with the lowest number of occurrences are ESFJ and ESTJ with 42 (~0.5%) and 39 (~0.4%) occurrences respectively.

INFP	1832
INFJ	1470
INTP	1304
INTJ	1091
ENTP	685
ENFP	675
ISTP	337
ISFP	271
ENTJ	231
ISTJ	205
ENFJ	190
ISFJ	166
ESTP	89
ESFP	48
ESFJ	42
ESTJ	39

Figure 4 Number of occurrences of personality types

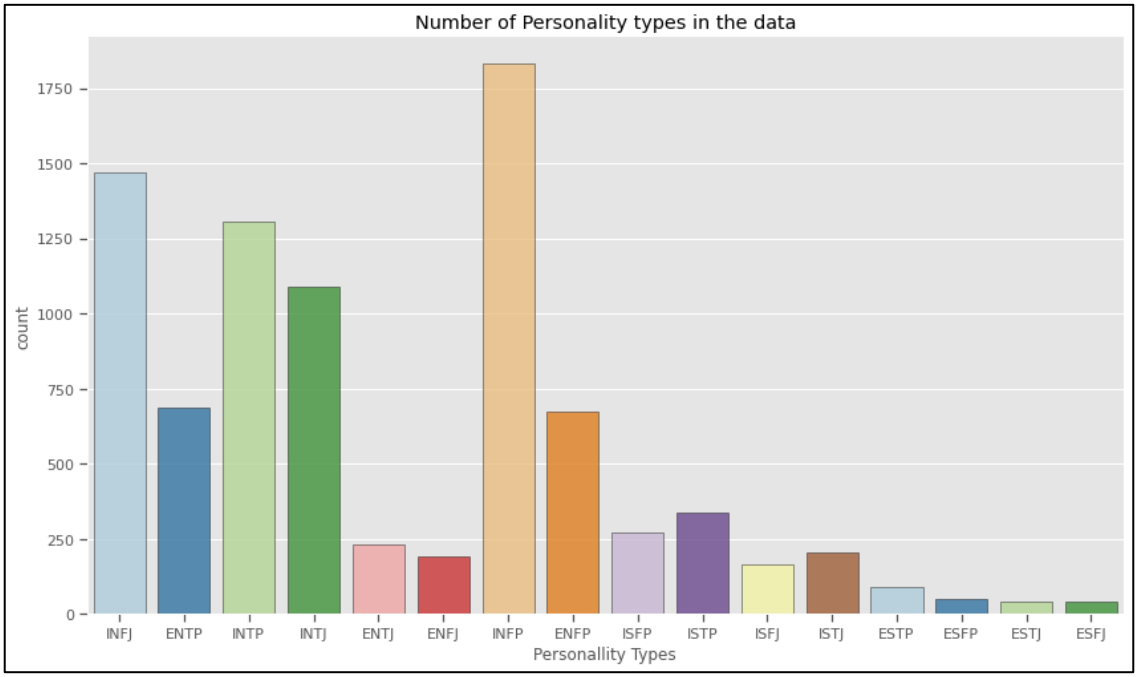

Figure 5 Distribution of personality types

In Figure 6 we can see the number of posts for each personality type in the data. As each user has contributed 50 posts in the "PersonalityCafe" forum. The personality type with the most posts is INFP followed by INFJ. At the end of the barplot, we have the personality types with the least contributions in the forum, which are ESFP,ESFJ,ESTJ.

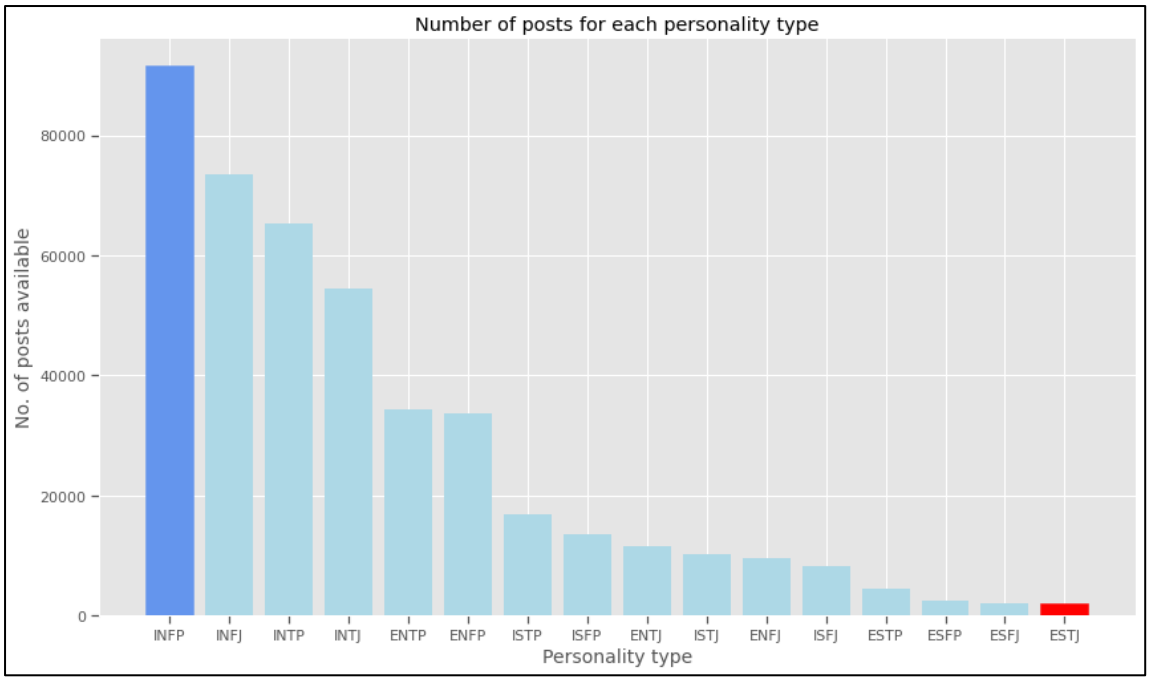

Figure 6 Number of posts for each personality type

In Figure 7, we have an example of the form of a post from an individual in the forum. It is obvious that there are links in the text data which must be removed since is useless for our goal.

'Good one _____ https://www.youtube.com/watch?v=fHiGbolFFGw|||Of course, to which I say I know; that's my blessing and my curse.|||Does being absolutely positive that you and your best friend could be an amazing couple count? If so, than yes. Or it's more I could be madly in love in case I reconciled my feelings (which at...|||No, I didn't; thank you for a link!|||So-called Ti-Si loop (and it can stem from any current topic/obsession) can be deadly. It's like when you're stuck in your own thoughts, and your mind just wanders in circles. Feels truly terrible. ...|||Have you noticed how peculiar vegetation can be? All you have to do is look down at the grass: dozens of different plant species there. And now imagine that hundreds of years later (when/if soil...|||The Smiths - Never Had No One Ever|||I often find myself spotting faces on marble tiles/wood.|||This 5 year-old sentence is an incredibly accurate and beautiful description.|||I haven't visited this website in the last 3 years. So whoever reads this (and maybe even remembers me, which I highly doubt): hi. 700049 700057|||When you sit in your garden until 10:30 PM writing songs, and sing them (together with dozens of crickets) while playing your acoustic guitar.|||This is the most INTP-ish thread I've ever seen.|||I wouldn't be able to look at the painting for the entire life if I knew that I picked it over the human being.|||I was drawing a background for my animation on which I'm working right now - it should have been Mars.. But I felt obligated to make Mark Watneyx92s postcard from it :D If you read the book...|||I started to make comics about turtle Gordon and unicorn Chimes - here you can see two first stories: https://www.tumblr.com/blog/-alexxxandra-|||INTJ Recently I started to post my comics about two friends - turtle Gordon and unicorn Chimes. Before that, I just posted stuff that interested me, but from now on I'll try to include only my works...|||Probably we could work together on a new model - I'm an expert in abrupt explosions of laughter upon various weird stuff. That happens because of peculiar sense of humor - so peculiar that not much...|||Hellooo Nah, you can tou

Figure 7 "Post" example

Data Preprocessing

At this point, in order to manipulate the posts and transform them to a more useful and meaningful state, we will create a function with some applications being described for the next bullet points.

- Lower case transformation

Firstly, we will transform all characters to lower case since upper/lower case words with the same meaning will be represented as two different words in the vector space model and thus, they will make harder the classification process.

- Delimiter removal

After that, delimiters will be removed since they are not adding something meaningful to the text expression for our goal.

- URL's removal

Another application inside the function is the removal of URL's due to the fact that it doesn't contains information as written text in our case.

- Punctuation removal

In addition, as another preprocessing technique, we will remove punctuations from the textual data in order to ensure the equally treatment of each text. In other words, the word "sandwich" and "sandwich:" are treated equally after the process of removal of punctuations.

- Numbers removal

Equally meaningful is the removal of number/digits, since they don't hold any vital information in the text for our purpose.

- Non-words removal

Enriching our function, we will remove all the words with no meaning or existence, called non-words since for some of them, their meaning is controversial. For example, words such as 25-hour, !mportant, h-ell-o are non-words.

- Personality type 4-letter code removal

At this point, it is crucial to remove the 4-letter code representation of personality type in order to the classification process not be biased. In other words, a text containing the MB personality type of the person will lead to misinterpretations since the models will "cheat" by learning to recognize mentions of MBTI by name.

- Big spaces removal

Finally, big spaces will be removed for the posts since there is no useful information in them.

Hence, we have an example of the final manipulated post depicted in Figure 8 with no URL's, no punctuation, no 4-letter codes for MBTI etc.

```
committing suicide the next day rest in peace hello sorry to hear of your distress its only natural for a relations
hip to not be perfection all the time in every moment of existence try to figure the hard times as times of growth
as welcome and stuff game set match prozac wellbrutin at least thirty minutes of moving your legs and i dont mean m
oving them while sitting in your same desk chair weed in moderation maybe try edibles as a healthier alternative ba
sically come up with three items youve determined that each type or whichever types you want to do would more than
likely use given each types cognitive functions and whatnot when left by all things in moderation sims is indeed a
video game and a good one at that note a good one at that is somewhat subjective in that i am not completely promot
ing the death of any given sim dear what were your favorite video games growing up and what are your now current fa
vorite video games cool it appears to be too late sad theres someone out there for everyone wait i thought confiden
ce was a good thing i just cherish the time of solitude bc i revel within my inner world more whereas most other ti
me id be workin just enjoy the me time while you can dont worry people will always be around to yo ladies if youre
into a complimentary personalitywell hey when your main social outlet is xbox live conversations and even then you
verbally fatigue quickly i really dig the part from to banned because this thread requires it of me get high in bac
kyard roast and eat marshmellows in backyard while conversing over something intellectual followed by massages and
kisses banned for too many bs in that sentence how could you think of the b banned for watching movies in the corne
r with the dunces banned because health class clearly taught you nothing about peer pressure banned for a whole hos
```

Figure 8 "Post" example after manipulation

Continuing the preprocessing, we will create a function for producing the four binary target variables (4 dichotomies mentioned in Personality Types section), transforming the task from multiclass to multilabel. In the multilabel problem, each label represents a different classification task and in contrast with the multiclass problem, the tasks are not mutually exclusive. Thus, we will create four new columns named "Extraversion", "Sensing", "Thinking", "Judging" in which, the existence of a dichotomy will be represented by "1" while the non-existence from "0". Thus, for each column we have the first one for Extraversion (E), /Introversion (I), the second category for Sensing (S) /Intuition (N), the third for Thinking (T)/Feeling (F) and the fourth category for Judging (J)/Perceiving (P). For example, if a person has MBTI ESFJ will be represented as 1,1,0,1 in the four columns respectively. In Figure 9 we can observe the final shape of the dataset until this time.

	type	posts	Extraversion	Sensing	Thinking	Judging
0	INFJ	and moments sportscenter not top ten plays pr...	0	0	0	1
1	ENTP	im finding the lack of me in these posts very ...	1	0	1	0
2	INTP	good one of course to which i say i know thats...	0	0	1	0
3	INTJ	dear i enjoyed our conversation the other day ...	0	0	1	1
4	ENTJ	youre fired thats another silly misconception ...	1	0	1	1
...	...	...	...	...	...	...
8670	ISFP	ixfp just because i always think of cats as f...	0	1	0	0
8671	ENFP	soif this thread already exists someplace else...	1	0	0	0
8672	INTP	so many questions when i do these things i wou...	0	0	1	0
8673	INFP	i am very conflicted right now when it comes t...	0	0	0	0
8674	INFP	it has been too long since i have been on pers...	0	0	0	0

Figure 9 Dataset with the four binary target variables

After these, we will check the proportions of the target variables depicted. In Figure 11 and 12 the distribution across type indicators is presented. We can observe that Introverts are 77% while Extroverts are 23%, Intuitives are 86% while Sensors are 14%, Feelers are 54% while Thinkers are 46% and finally, Perceivers are 60% while Judgers are 40%.

```
Introversion (I) / Extraversion (E):      77 %  /  23 %
Intuition (N) / Sensing (S):              86 %  /  14 %
Feeling (F) /Thinking (T) :               54 %  /  46 %
Perceiving (P) /Judging (J) :             60 %  /  40 %
```

Figure 7 Proportion of each personality key feature

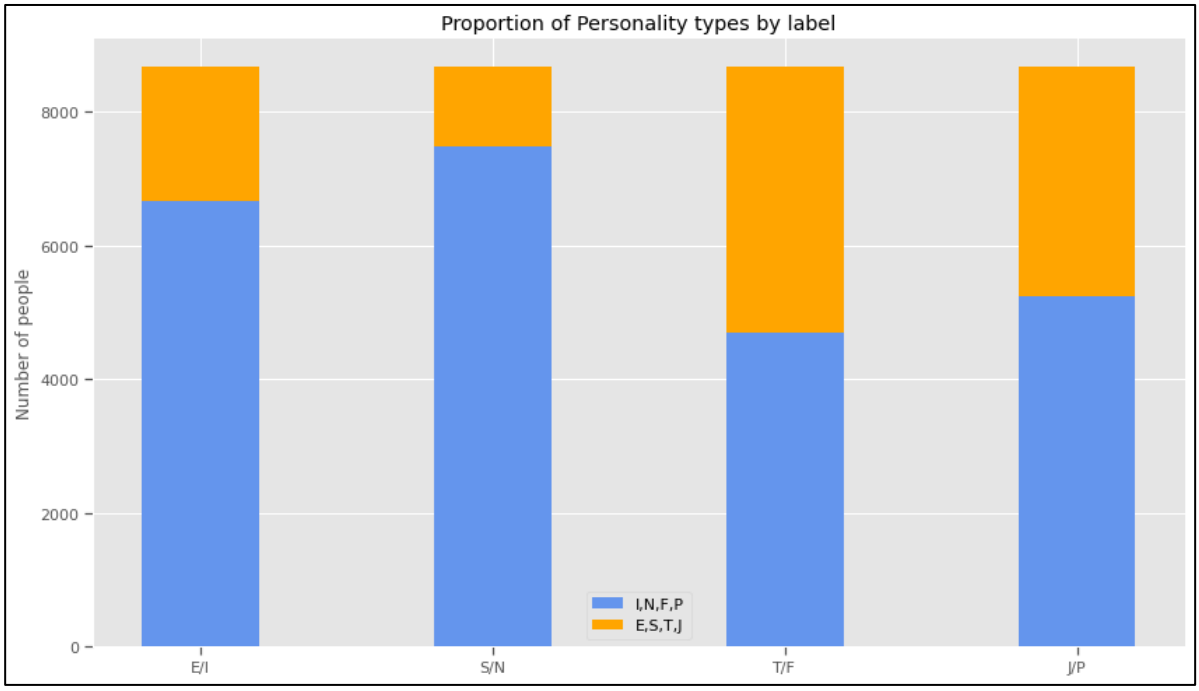

Figure 12 Proportion of each personality key feature plot

Following the construction and the use of the first function described in the above sections, we will create a second function in order to tokenize words and apply the stemming method to the text with all the useless numbers, symbols and words been removed.

- Stop words

Firstly, and before the creation of the function, we will create a custom stop words vocabulary, combining the English stop words from the NLTK package (corpus) with the 100 most frequent stop words in our texts. The collection of words in the vocabulary is depicted in Figure 13.

```
['theirs', 'itself', "hadn't", 'because', 'until', 'further', "you'd", 'yours', 'dont', 'has', 'you', 'any', 'too',
'such', "you'll", "you've", 'her', 'much', 'even', 'these', 'that', "needn't", "hasn't", 'but', 'very', 'now', 'our
selves', 'all', 'feel', 'like', 'nor', "isn't", 'don', 'weren', 'my', "it's", 'did', 've', 'under', 'being', 'throu
gh', 'whom', 'into', 'me', 'they', 'hasn', 'over', 'most', 'herself', 'aren', 'are', 'those', 'our', 'at', 'a', "mi
ghtn't", 'and', 'both', "couldn't", 'myself', 'it', "haven't", 'about', 'doing', 'few', 'ours', 'think', 'own', 'do
esn', 'would', "didn't", 're', 'on', 'where', 'things', 'way', "shan't", 'really', 'mustn', 'with', 'during', 'only
', 'before', 'should', 'll', 'each', 'them', 'shan', 'be', "shouldn't", 'who', 'himself', 'same', 'needn', 'while',
'was', 'been', 'your', 'some', 'people', 'no', 'after', 'below', 'between', 'i', 'am', 'other', 'im', 'up', 'someth
ing', 'not', 'just', 'or', 'she', 'again', 'won', 'against', 'his', 'wouldn', 'also', 'hers', "won't", 'shouldn', "
wouldn't", 'here', 'themselves', "mustn't", 'more', 'by', 'yourself', 'from', 'do', 'above', 'then', 'so', 'haven',
'one', 'love', 'in', 'know', 'ain', 'yourselves', 'didn', 'lot', 'for', 'once', "weren't", 'does', "you're", 'its',
'time', 'isn', 'want', "doesn't", 'ive', 'the', 'off', 'what', "she's", 'of', 'ma', 'this', 'when', "wasn't", 'how'
, 'd', 'couldn', 'there', 'have', 'someone', 'having', 'if', 'youre', 'their', 's', 'is', 'y', 'o', 'we', 'down', "
that'll", 'out', 'than', 'as', "aren't", 'were', 'an', "should've", 'm', 'wasn', 'say', 'get', 'him', 'make', 'well
', "don't", 'type', 'had', 'see', 'to', 'mightn', 'can', 'why', 't', 'always', 'will', 'which', 'hadn', 'good', 'he
']
```

Figure 13 Custom stop word vocabulary

After the creation of the stop word vocabulary, we will create the function that at first will tokenize the words of the texts using Keras word tokenizer. Applying tokenization method, we break natural language text into chunks of information that can be considered as discrete elements. For example, having the sentence "I want a sandwich", after tokenization which is a process of tokenizing or splitting a string, text into a list of tokens, we will have ["I", "want", "a", "sandwich"]. After the word tokenization, using PortStemmer from WordNetLemmatizer, we will apply the stemming method in the words of the posts, in order to normalize the texts and prepare words. In specific, using the stem of the words, we remove the commoner morphological and inflectional suffixes of the words, producing sequences of letters that may not be actual words (e.g., change, changing, changeable → "chang"). At this point, we should underline that the stemming method is applied to words that are not a part of the stop words vocabulary.

The final morphology of the posts is depicted in the following example in Figure 14.

	type	posts	Extraversion	Sensing	Thinking	Judging
0	INFJ	moment sportscent top ten play prank lifechang...	0	0	0	1
1	ENTP	find lack post alarm sex bore posit often exam...	1	0	1	0
2	INTP	cours that bless curs absolut posit best frien...	0	0	1	0
3	INTJ	dear enjoy convers day esoter gab natur univer...	0	0	1	1
4	ENTJ	fire that anoth silli misconcept approach logi...	1	0	1	1

Figure 14 Finalized morphology of post

- Wordcloud based on each Personality key feature:

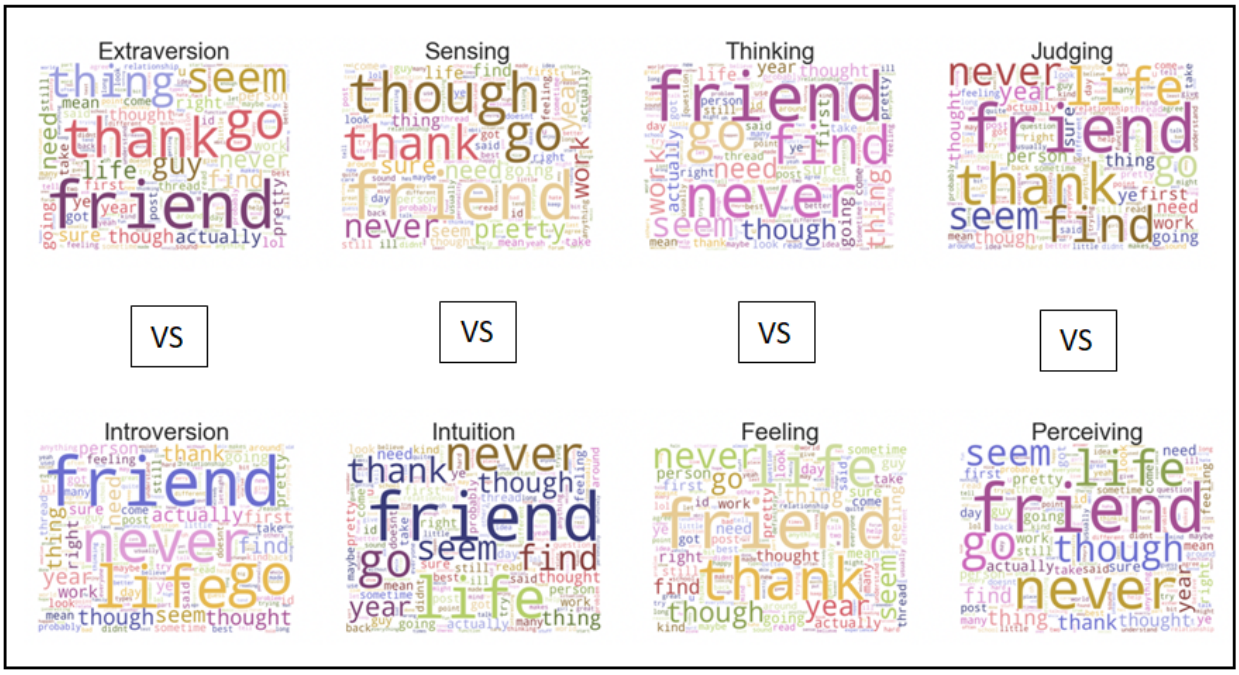

Figure 15 Most used words for each Indicator type

In Figure 15 above, the most common words in the posts according to the personality type dichotomies are presented.

- Wordcloud of all the posts:

Figure 16 Most used words throughout all posts

According to Figure 16 above, the most common words in the posts of all users in Personality Café Forum are the following: people, know, really, think, one, say, thing, time, feel, make, thought, friend, type, really, watch, say, want, find, see, look, work, way, thank.

Data Split

To assess the correctness of the "MBTI" personality classifiers, we will split the dataset into training testing and validation part, using stratified sampling in order to preserve the proportionalities of the initial dataset. Utilizing the "train-test split" function in the "Scikit-learn" package, we segregated 90% of the data for training and 10% for testing. After this split, following the same process we split the training set to two new parts, 90% for training and 10% for validation. Eventually, we end up with three sets:

- Training set for implementing the fit of the models
- Validation set for early stopping the training process when validation loss starts to increase
- Test set, which is the unseen by the model collection of data, used solely for the evaluation of the model that performed the best, efficiency-wise in the validation data and hasn't participated in the training phase.

Methodology

To find the best classifier in order to predict an individual's personality type according to the Myers Briggs Type-Indicator, a selection of Machine Learning and Neural Network methods were used. We used a Logistic Regression as the baseline model. For a more thorough experiment and analysis we applied machine learning classical models, which we tuned, specifically a Logistic Regression and an XGBoost model. Furthermore, in this particular project, going a step further and trying to increase efficiency, we trained four types of Deep Learning Models. The models used were Feed Forward Network (FFN), Multilayer Perceptron (MLP), Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN) and the 'DistilBert' model, which is based on the BERT architecture. Later a comparative analysis of these classifiers was implemented. All the models were trained on the training data and evaluated using the validation set. The test was used only for the predictions of the last model, the Transformer one, which was the one with the lowest validation loss and highest accuracy scores. The evaluation results are comprised of validation accuracy, validation loss, hamming score (1-hamming loss), Exact Match Ratio and ROC curves. It should be noted that, for these models a long process of trial and error as well as background theory was done in order to come down to the choice of layers, number of neurons, dropout rate, kernel size and other parameters. The was that are described in this paper, are those that achieved the best results and therefore we will not mention the others that were also tried.

Traditional Machine Learning Models

The following three machine learning models are trained using the data with preprocessing steps explained in the Data Preprocessing section above with the addition of the TF-IDF approach to the data that will be explained during the first neural network model.

Logistic Regression model

Logistic regression is a classification algorithm, intended for datasets that have a categorical target variable that has two values or classes, using a binomial probability distribution function. Problems of this type are referred to as binary classification problems. The class labels are mapped to 1 for the positive class or outcome and 0 for the negative class or outcome. The fit model predicts the probability that an example belongs to class 1. By default, logistic regression cannot be used for classification tasks that have more than two class labels, and it requires some modifications. Since our data was already modified to multiple binary classification problems, we are going to use the OneVsRest classifier combined with the Logistic Regressor in order to handle this prediction. In the multilabel learning literature, OneVsRest is also known as the binary relevance method.

As a baseline model we implemented a simple Logistic Regression using the default hyperparameters. The evaluation of the model can be seen on table right below.

Evaluation Metrics:

Table 1 Evaluation of Logistic Regression model

	Exact Match Ratio	Hamming Score
Logistic Regression Model	28,93%	72.45 %

Classification Report:

```
Classification Report
              precision    recall  f1-score   support

           0       0.40      0.62      0.49       181
           1       0.28      0.56      0.37       108
           2       0.77      0.84      0.80       359
           3       0.55      0.64      0.59       310

   micro avg       0.54      0.70      0.61       958
   macro avg       0.50      0.66      0.56       958
weighted avg       0.57      0.70      0.63       958
 samples avg       0.46      0.55      0.48       958
```

Figure 17 Classification report of simple Logistic Regression

Hyperparameter tuning with Grid Search

After the implementation of the baseline Logistic Regression model, we move on and build a function to tune the hyperparameters of the Logistic Regression in order to increase the accuracy of the model. The hyperparameters searched were "C" and "penalty" among the values depicted below.

```
alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']
```

- Penalty (or regularization) intends to reduce model generalization error and is meant to disincentivize and regulate overfitting. Technique discourages learning a more complex model, to avoid the risk of overfitting. The choices are: {'l1', 'l2', 'elasticnet', 'none'}.

- C (or regularization strength) must be a positive float. Regularization strength works with the penalty to regulate overfitting. Smaller values specify stronger regularization and high value tells the model to give high weight to the training data.

Evaluation Metrics:

Table 2 Evaluation of Logistic Regression model with tuning

	Exact Match Ratio	Hamming Score
Logistic Regression with tuning	38,92%	77.57 %

Classification Report:

```
Classification Report
              precision    recall  f1-score   support

           0       0.48      0.49      0.48       181
           1       0.45      0.38      0.41       108
           2       0.81      0.80      0.80       359
           3       0.59      0.58      0.58       310

   micro avg       0.64      0.62      0.63       958
   macro avg       0.58      0.56      0.57       958
weighted avg       0.63      0.62      0.63       958
 samples avg       0.49      0.50      0.47       958
```

Figure 18 Classification report of Logistic Regression with tuning

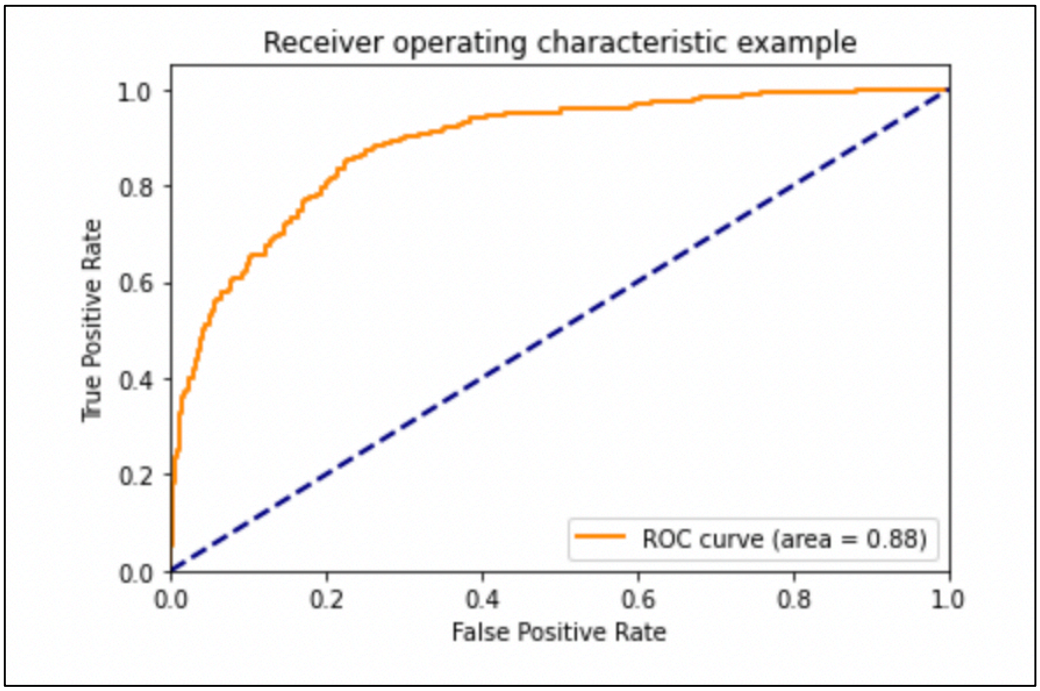

Figure 19 ROC curve of Logistic Regression model with tuning

XGBoost Classifier model

XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision-tree-based ensemble Machine Learning algorithm. It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems. In prediction problems involving unstructured data such as text in our case, artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium data, decision tree-based algorithms are considered best-in-class right now. Therefore, below we built an XGBoost algorithm with tuning.

Hyperparameter tuning with Grid Search

The hyperparameters searched were “max_depth”, ‘learning_rate” and “n_estimators” among the values depicted below.

```
max_depth=[3,6,8]
learning_rate=[0.1, 0.2, 0.3]
n_estimators=[100, 250,300]
```

- max_depth is the maximum depth of a tree. It is used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample. Increasing this value will make the model more complex and more likely to overfit.
- learning_rate: is the step size shrinkage used in update to prevent overfitting. After each boosting step, we can directly get the weights of new features, and learning rate shrinks the feature weights to make the boosting process more conservative.
- n_estimators: represents the number of trees in the forest. Usually, the higher the number of trees the better to learn the data.

Evaluation Metrics:

Table 3 Evaluation of XGBoost model with tuning

	Exact Match Ratio	Hamming Score
XGBoost model with tuning	36,87%	77.47 %

Classification Report:

```
Classification Report
              precision    recall  f1-score   support

           0       0.59      0.18      0.27       181
           1       0.57      0.04      0.07       108
           2       0.77      0.77      0.77       359
           3       0.61      0.42      0.49       310

   micro avg       0.70      0.46      0.56       958
   macro avg       0.64      0.35      0.40       958
weighted avg       0.66      0.46      0.51       958
 samples avg       0.45      0.37      0.39       958
```

Figure 20 Classification report of XGBoost model with tuning

Deep Learning methods

All the neural network models mentioned below share these characteristics.

- Target Variable Encoding

Since categorical data are unrecognizable by neural network models, they require numerical ones in order to be understood by the models. We used “encoding” technique to make our categorical data legible to the models. Therefore, each set of key features from the target categories is transformed to an array of 0s and 1s like in Figure 21.

```
array([[0., 0., 1., 0.],
       [0., 0., 1., 1.],
       [0., 0., 1., 0.],
       ...,
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [1., 0., 1., 0.]], dtype=float32)
```

Figure 21 Target variable Encoding

- Loss function

Since we are dealing with a multilabel (four outputs) classification problem the loss function used will be binary cross entropy. Binary cross entropy is the negative average of the log of corrected predicted probabilities. Basically, it compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value.

- Optimizer

As an optimizer we used the Adam optimizer with learning rate of 0.001. The Adam optimizer is an adaptive learning rate method. It can be considered as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

- Metrics

For metrics we used accuracy and binary accuracy. The accuracy metric calculates how often predictions equal labels. Whereas binary accuracy calculates how often predictions match binary labels.

- Callbacks

Lastly, in order to avoid overfitting our model and to save checkpoints, we included callbacks to each model using 'EarlyStopping', 'ModelCheckpoint' and 'ReduceLROnPlateau'. Early stopping allows us to monitor, using the validation loss metric and stop model training when it stops improving. We specified min_delta to 0,1, which means the minimum amount of improvement we expect in every epoch is 0,1. Patience parameter was set to **5**, which means the number of epochs to wait before stopping the training is 5 epochs. Furthermore, 'ModelCheckpoint' callback was added to save the model regularly during training. This callback monitors the training and saves model checkpoints at regular intervals, based on the metrics. Lastly, 'ReduceLROnPlateau' callback is used to change the learning rate when validation loss has stopped improving. This callback reduces the learning based on the validation loss.

MLP Models

A multilayer perceptron (MLP) is a fully connected class of feed forward artificial neural network (ANN). An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Each new layer is a set of nonlinear functions of a weighted sum of all outputs (fully connected) from the prior one. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. We tested four different models, a simple Feed Forward model using Bag of words approach, an MLP with trainable word embeddings, an MLP model with GLOVE embeddings and an MLP model using BERT embeddings.

1. TF-IDF Feed Forward Neural Network model

The first model is a Feed Forward model where data moves only forward from the input layer. This is the simplest model yet is quite efficient in text classification like in our case. The data to be fed in the FFN model for the training phase are 7061 rows, and the validation set consists of 781 rows. The steps for this model are presented right above

- TF-IDF Process

Before implementing the FFN model, some preprocess on the three sets was necessary (training, validation, test). We had to convert the text data (attribute 'posts') to numerical features, so that they can be processed by the ML models. The approach followed for the first model was TF-IDF which stands for Term Frequency and Inverse Document Frequency, a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. TF means the probability of occurrence of a word in a sentence and finds the most frequently occurring words in a document corpus. IDF is given by the formula log(N/n), where N is the total number of documents in our corpus, and n is the number of times each word appears across the entire corpus. So for each word in each post we will consider the product of (TF and IDF). This method creates one multidimensional vector of 30000 dimensions represented by a sparse matrix in python for each observation (posts total),

- Parameters

The next step before building the model, was to set the model parameters. We defined the following parameters:

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (30)
- batch_size: the batch size of the data that will be fed to the model while training (32)
- max_words: the input layer will consist of (30000)
- dropout_rate: how many neurons the model will shut down at every epoch (0.4)

- Architecture

After defining the parameters of the FFN model, we build the architecture of the model. The architecture consists of a sequential model with an input layer (equal to 30000 features), two hidden layers (of 64 neurons each) using activation function "relu", followed by one regularization layer each (Dropout layer), and an output layer (of 4 neurons) with one sigmoid activation function each, as the problem is multilabel classification with four binary outputs. The plot of the architecture is presented in Figure 22 below.

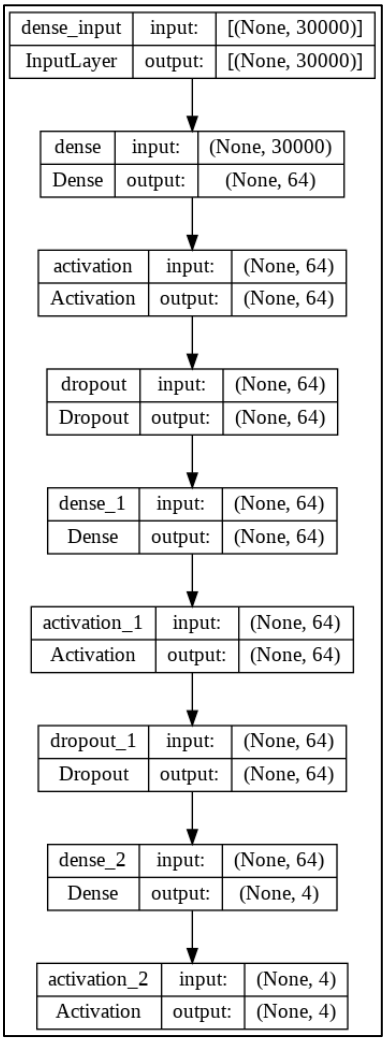

Figure 22 FFN model architecture

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam, loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 8 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 4 Evaluation of FFN model

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
Feed Forward TF-IDF Model	0,46	46.12 %	77.62 %

Loss and Accuracy plots:

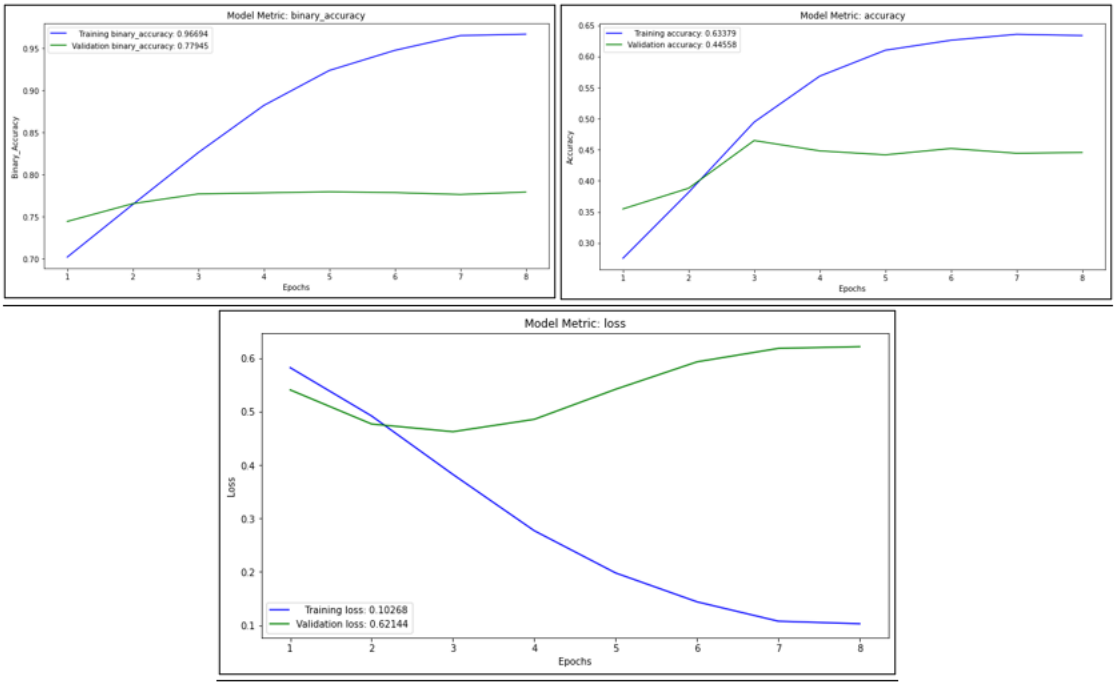

Figure 23 Loss & Accuracy plots FFN model

As we can see by the loss plot of Figure 23, validation and training loss are not synced and validation loss keeps increasing throughout the epochs, indicating that the model overfits. This is justifiable, as using EarlyStopping the model stops training at the eighth epoch.

2. MLP model with trainable embeddings

- Word Embeddings

A word embedding is a class of approaches for representing words and documents using a dense vector representation. It is an improvement over more traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. This embedding can map semantically similar words. It does not consider the text as a human language but maps the structure of sets of words used in the corpus. They aim to map words into a geometric space which is called an embedding space. In addition to these, a word embedding can be learned as part of a deep learning model. This can be a slower approach, but tailors the model to a specific training dataset. The second MLP model we implemented does exactly that.

- Preparation process

In order to build the model an embedding layer requires that the input data be integer encoded so that each word is represented by a unique integer. This data preparation step was performed with the word embeddings mentioned above.

Another preprocessing step was to create another column in the data, representing the length of each user's post. Then, looking at the distribution of column "posts_length" in Figure 24 we decided since it is left skewed to use the median of the distribution, which is 540 words, as the maximum length of each observation. Then, to make sure that all observations are equal we used the pad_sequences function. This way when an observation exceeds the number of max_len, then it will truncate the sentences. Also, if an observation is less that the max_len variable, it is filled with the value zero. Finally, the embedding layer is initialized in the model with random weights and will learn an embedding for all the 30000 words that were set for the training dataset.

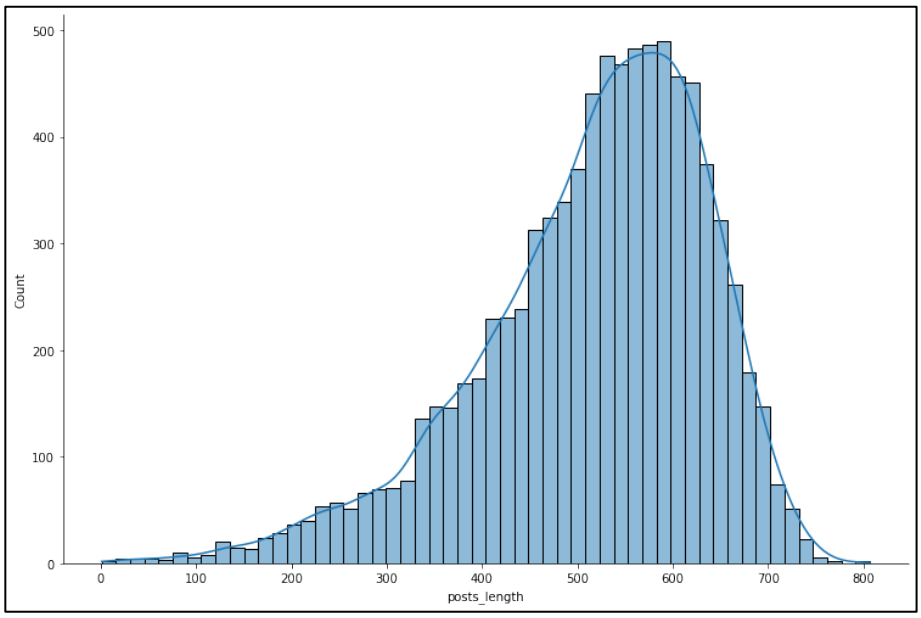

Figure 24 Distribution of posts_length

Therefore, when building the model we have the Embedding layer as the first hidden layer of the network with 3 arguments:

- input_dim: This is the size of the vocabulary in the text data. In our case, specified by the max_words variable. After finding the unique words of the 'posts' column we used the 70% of them and set max words equal to 30000 words.

- output_dim: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. In our case specified by the embed_dim variable and set to 300.

- input_length: This is the length of input sequences, as you would define for any input layer of a Keras model. In our case specified by the max_len variable, defined for the padding process and set to 280.

Before building the model, we will also have to set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- max_words: the size of the vocabulary (30000)
- max_len: the length of each sentence (540)
- nb_epochs: the number of epochs that the model will train for (50)
- batch_size: the batch size of the data that will be fed to the model while training (32)
- embed_dim : the output layer will consist of a 300- dimensional tensor to represent each word
- dropout_rate: how many neurons the model will shut down at every epoch (0.4)

- Architecture

After defining the parameters of the MLP model, we build the architecture of the model. The architecture consists of a sequential model with an Embedding layer (with the arguments input_dim, output_dim and input_length as mentioned above). The output of the embedding layer is a 2D output matrix therefore in order to connect it to a Dense layer we must first convert it to a 1D vector using a Flatten layer. Later, follow two hidden layers (of 128 and 64 neurons each), using activation function "relu", followed by one regularization layer each (Dropout layer), and an output layer (of 4 neurons) with one sigmoid activation function each). The plot of the architecture is presented in Figure 25 below.

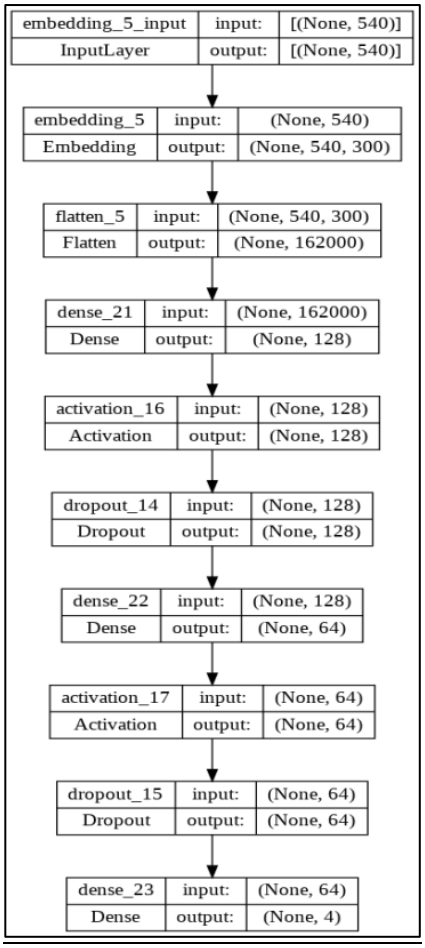

Figure 25 MLP model with trainable embeddings

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam, loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 9 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 5 Evaluation of MLP model with trainable embeddings

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
MLP model with embeddings	0.544	34.31 %	72.24 %

Loss and Accuracy plots:

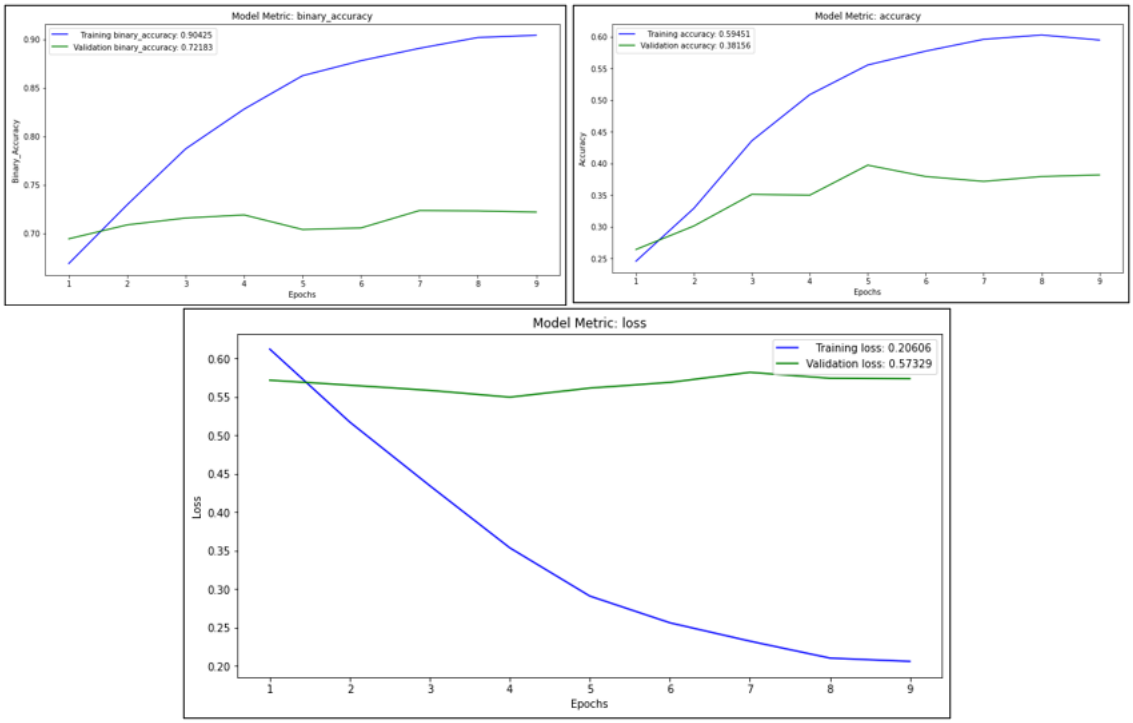

Figure 26 Loss & Accuracy plots MLP model with trainable embeddings

As we can see by the loss plot of Figure 26, validation and training loss are not synced and validation loss begins at a rather high level keeping a stable path throughout the epochs. This plot indicates a case of overfitting. The validation loss is greater than the training loss, perhaps the model is overfitting, and cannot generalize on new data. In particular, the model performs well on training data but poorly on the new data in the validation set.

3. MLP model with GLOVE embeddings

In the third MLP model instead of training our own embeddings we will use the pretrained GloVe embeddings. GloVe stands for Global Vectors for word representation. It is an unsupervised learning algorithm developed by researchers at Stanford University aiming to generate word embeddings by aggregating global word co-occurrence matrices from a given corpus. The basic idea behind the GloVe word embedding is to derive the relationship between the words from statistics. Unlike the occurrence matrix, the co-occurrence matrix tells you how often a particular word pair occurs together. Each value in the co-occurrence matrix represents a pair of words occurring together.

Before building the model, we will set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (50)
- batch_size: the batch size of the data that will be fed to the model while training (64)
- embedding_dim: the embedding vector size will consist of a 300-dimensional tensor
- dropout_rate: how many neurons the model will shut down at every epoch (0.4)

- Architecture

After defining the parameters of the MLP model, we build the architecture of the model. The architecture consists of a sequential model with the embedding layer (with an input dim equal to 30000 words, output_dim of a 300-dimensional tensor, weights from the pre-trained embedding for the words in our training dataset and input_length equal to 540). Then follows a flatten layer, two dense layers (of 128 neurons each) with a 'relu' activation function each, followed by one regularization layer each (Dropout layer), and an output layer (of 4 neurons) with one sigmoid activation function each). The plot of the architecture is presented in Figure 27 below.

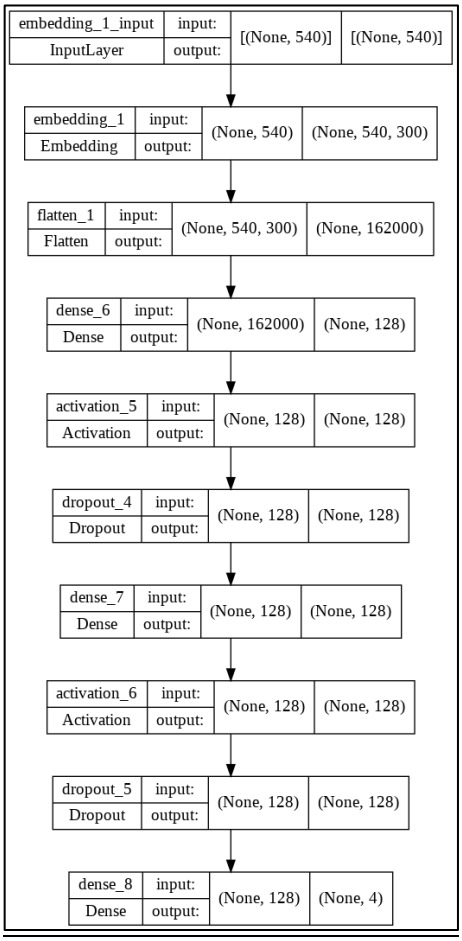

Figure 27 MLP model with GLOVE embeddings

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam (with a Learning rate of 0,001), loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 13 epochs and after evaluation, gave the following results.

Evaluation Metrics:

Table 6 Evaluation of MLP model with GLOVE embeddings

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
MLP model with GLOVE embeddings	0.574	27.78 %	69.39 %

Loss and Accuracy plots:

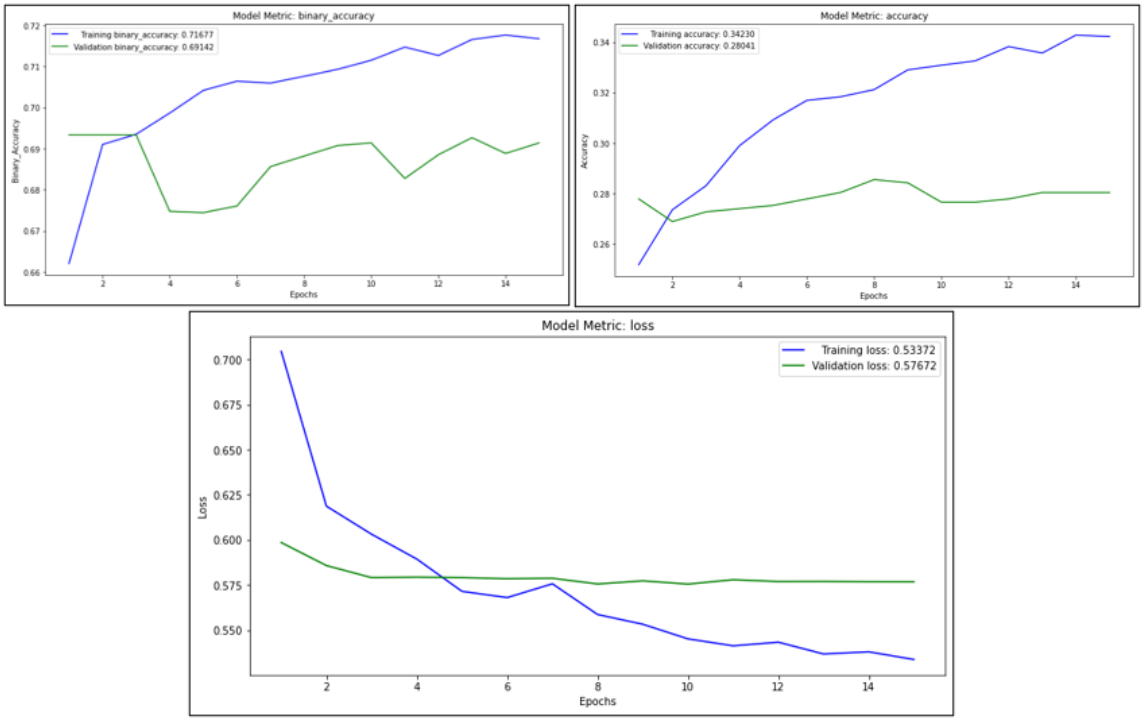

Figure 28 Loss & Accuracy plots MLP model with GLOVE embeddings

As we can see by the loss plot of Figure 28, validation and training loss are not synced but they follow the same path, and don't deviate that much. Validation loss has a stable progression throughout the epochs, indicating that the model doesn't overfit. This is an indication that the model with GLOVE embeddings is more efficient than the one built with trainable embeddings from the data.

4. MLP model with BERT embeddings

- Preparation Process

For this task we followed a specific procedure in order to use BERT transformer model to generate an average embedding for each observation (user). At first, we split the posts for each user. Each row now contains only one post while previously contained all the posts form a user. So we got a dataset with 422844 rows. After that we downloaded the BERT preprocessor and encoder from Tensorflow hub. Bert preprocessor carries out all the text cleaning preprocess needed to the text and transforms the given text sample into three tensors. One containing the input word ids and with padding or truncation if needed, one with the token type ids and one with the attention mask. Bert encoder transforms the input tensors taken from the preprocessor into 768-dimensional word embeddings or sentence embeddings depending on the task you want to solve. In our case we want sentence embeddings as we want one embedding for each post of each user. Hence, we built a model that takes as an input a text and giving as an output the bert embedding for this sentence. Next, we built a function using this model which for each user and for each one of his posts produces an embedding and at the end averages the embeddings into one. So at the end of the day, we have a 8675 x 733 data frame containing an embedding for each observation (user) based on the average of all his post's embeddings and the corresponding labels. The goal of this process was shrunk as much information as possible for each user into one vector. Finally, we split the dataset into train, validation and then we passed the train dataset through an MLP model the details of the model shown below:

Before building the model, we will set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (30)
- batch_size: the batch size of the data that will be fed to the model while training (32)
- max_features : the input layer will consist of a (768) dimensional-matrix
- dropout_rate: how many neurons the model will shut down at every epoch (0.4)

- Architecture

After defining the parameters of the MLP model with BERT embeddings, we build the architecture of the model. The architecture consists of a sequential model with one dense layer of 512 neurons and an input shape equal to the 768-dimensional matrix, followed by the self-regularized activation function “mish”, then comes one regularization layer (Dropout layer), and an output layer (of 4 neurons) with one sigmoid activation function each). The plot of the architecture is presented in Figure 29 below.

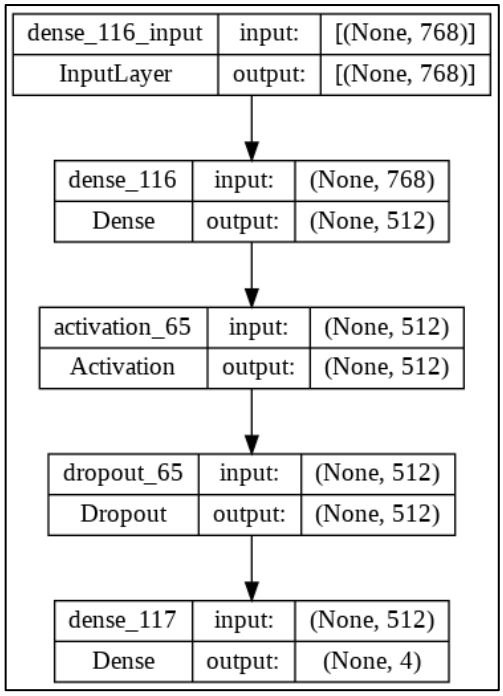

Figure 29 MLP model with BERT average embeddings

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam (with a Learning rate of 0,001), loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 16 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 7 Evaluation of MLP model with BERT average embeddings

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
MLP model with BERT average embeddings	0.464	42.51 %	78.45 %

Loss and Accuracy plots:

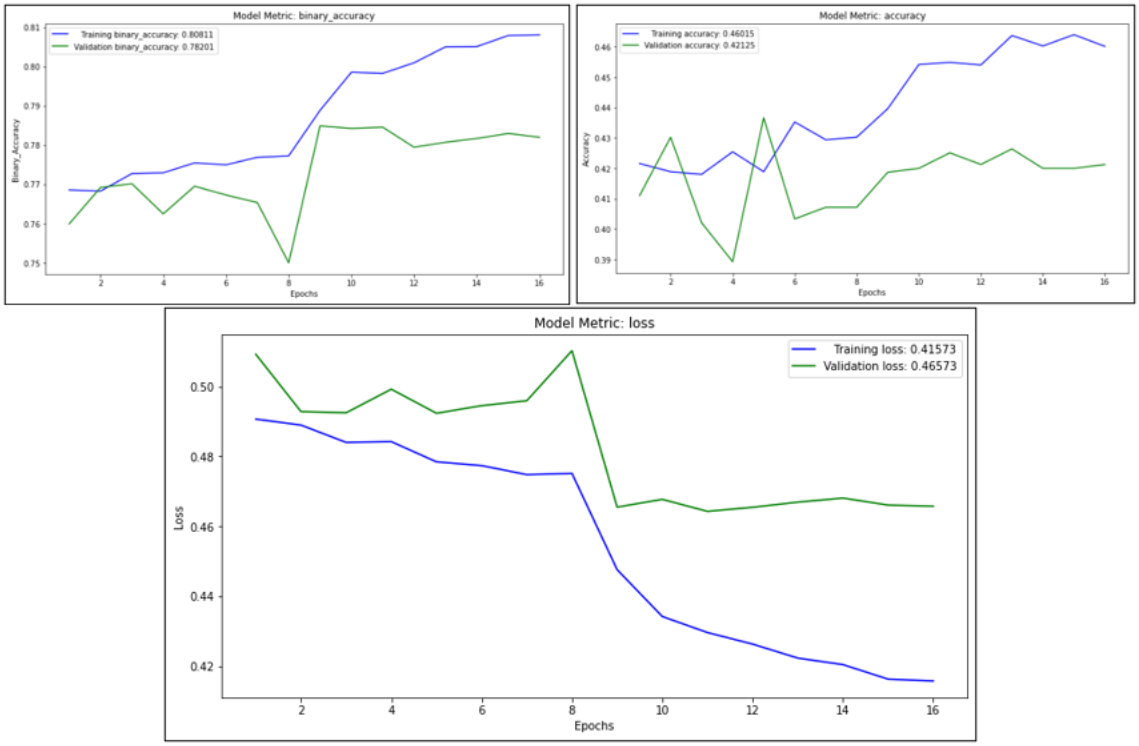

Figure 30 Loss & Accuracy plots of MLP model with BERT average embeddings

In the loss plot in Figure 30 we can see that there are no fluctuations until the 9th epoch, where there is a big drop and from this point on the validation loss is stable and then starts to increase. On the other hand, training loss decreases unexpectedly. This is a clear sign of overfitting as the difference between the losses increases as the epochs evolve.

Preparatory steps for the rest of the models

After the MLP models we decided to test three other types of Neural Networks, RNN, CNN models. These models require a different preprocessing on the data as opposed to the architectures that we saw so far. All models were tried initially with the stopwords and after with the removal of them, as a result the performance was more robust when keeping them in the posts. Therefore, for the models that follow we will keep the stopwords and will not implement stemming as we did on the Machine Learning Models, the Bag-of-Words or the MLP ones. We are doing these because, we concluded that more complex models such as the LSTM or CNN ones benefit from stopwords in the case of text classification in order to capture the semantic meaning of a sentence.

Another important step that needs to be clarified is the choice of max_len for the input observations in the models that will follow. As we know neural networks models and especially models like RNN or CNN's which are more complex require more evolved handling of data. Training models takes a lot of time and with less tokens to be trained, the training time should decrease. Therefore, we decided to create another column named "selected_tokens". This specific attribute contains the first six words of each post of the 50 that

each user has. That way we managed to keep an essence of every post while trying to reduce the vocabulary size and have a smaller length variance and subsequently smaller distribution. After having plotted the distribution, we can clearly see the difference in range on Figure 26 as compared to Figure 31. Again, the distribution is left skewed, and we decided to use the median, in this case equal to 280 as the max_len of the sentences for each user. Moreover, ss all neural networks need to have the inputs that are in similar size we will again use the pad_sequences function. The only difference this time is that we specify the padding argument to "pre". That way if an observation is less that the max_len variable, it is filled with the value zero in the beginning of the sentence. This is because post-padding seems to add noise to what has been learned from the sequence through time, and there aren't more timesteps for the RNN to recover from this noise. With pre-padding, however, the RNN is better able to adjust to the added noise of zeros at the beginning as it learns from the sequence through time.

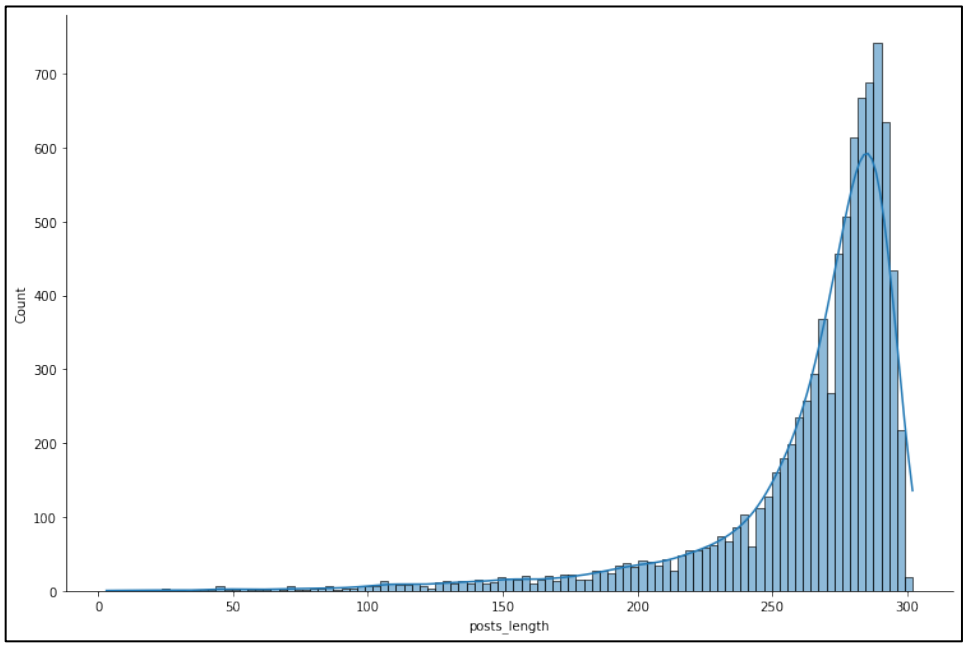

Figure 31 Distribution of posts length

RNN Models

Recurrent neural networks, also known as RNNs is a special type of an artificial neural network adapted to work for time series data or data that involves sequences. They allow previous outputs to be used as inputs while having hidden states. They consist of nodes in different layers of the neural network which are compressed to form a single layer of recurrent neural networks. RNNs have the concept of 'memory' that helps them store the states or information of previous inputs to generate the next output of the sequence. an RNN layer uses a for loop to iterate over the timesteps of a sequence, while maintaining an internal state that encodes information about the timesteps it has seen so far. We are going to implement two types of RNN's, the GRU architecture and the LSTM architecture both using the pretrained GloVe embeddings.

1. Bidirectional GRU model

Gated Recurrent Units belong to the Recurrent Neural Networks, designed to handle the vanishing gradient problem. They have a reset and update gate. These gates determine which information is to be retained for future predictions. Next, we are going to build a Bidirectional GRU, which is a sequence processing model that consists of two GRU's one taking the input in a forward direction, and the other in a backwards direction. It is a bidirectional recurrent neural network with only the input and forget gates.

Before building the model, we will set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (50)
- batch_size: the batch size of the data that will be fed to the model while training (64)

- Architecture

After defining the parameters of the GRU model, we build the architecture of the model. The architecture consists of a sequential model with the embedding layer (with an input dim equal to 30000 words, output_dim of a 300-dimensional tensor, weights from the pre-trained embedding for the words in our training dataset and input_length equal to 280). Then follows a Bidirectional GRU layer with 256 neurons in total and one dense layer of 300 neurons, equal to the output dimension. Lastly, we have an output layer (of 4 neurons) with one sigmoid activation function each. The plot of the architecture is presented in Figure 32 below.

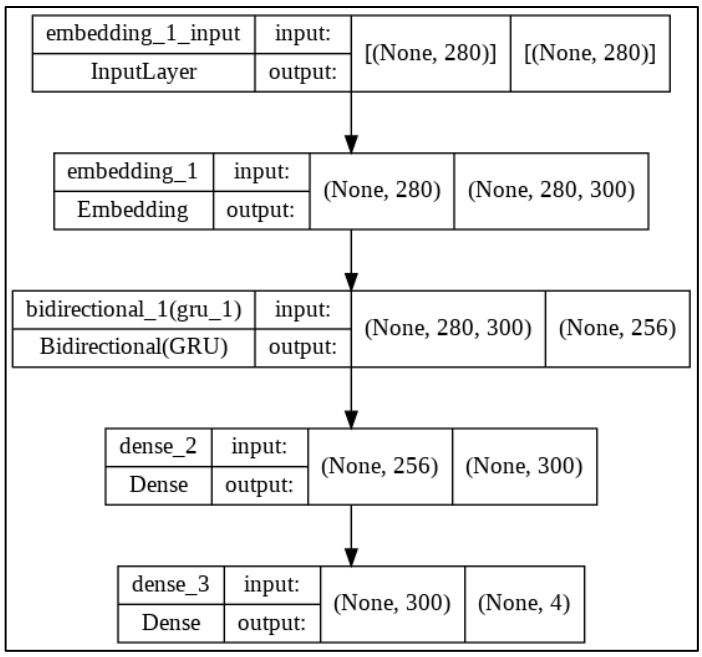

Figure 32 GRU model

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam with learning rate equal to 0.0001, loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 27 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 8 Evaluation of GRU model

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
GRU model	0.547	35.21 %	73.62 %

Loss and Accuracy plots:

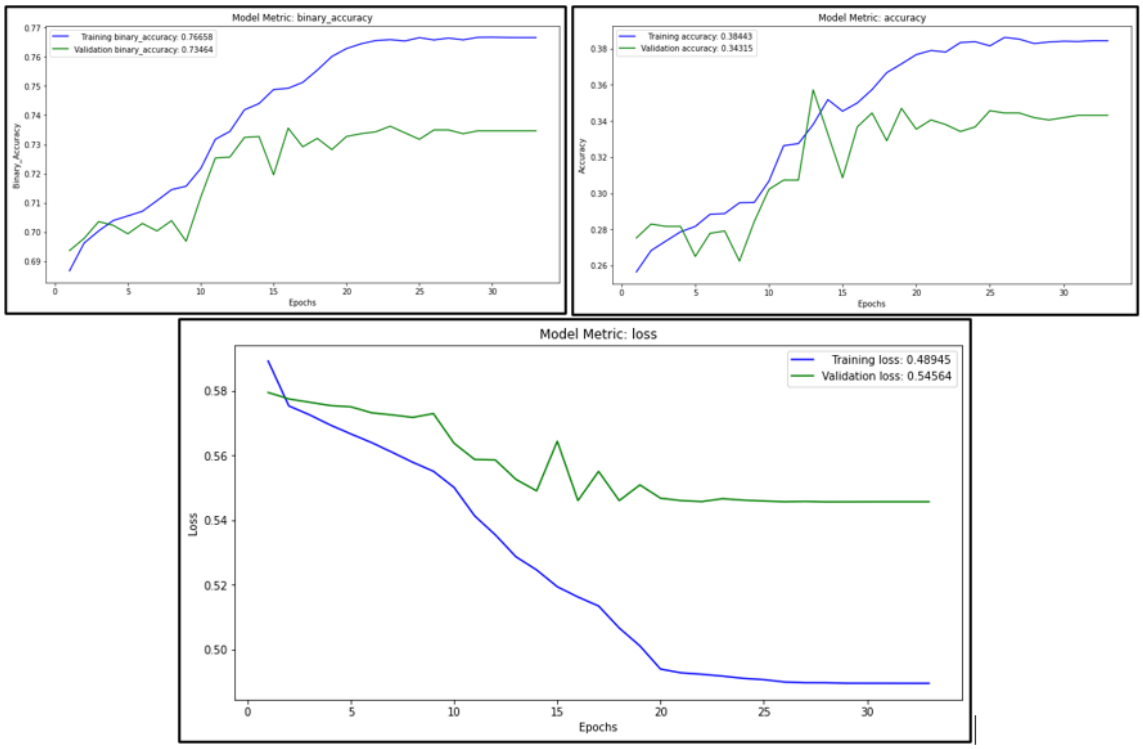

Figure 33 Loss & Accuracy plots of GRU model

As we can see by the loss plot of Figure 33, validation and training loss are not synced and validation loss tends to experience a few fluctuations around a specific range and then starts to decline after the 20th epoch.

2. Bidirectional LSTM model

LSTMs are a special kind of RNN, capable of learning long-term dependencies. LSTM itself can find the hidden layer of each cell and is designed to store previous cell information. This method is used by classifying long-term data by storing it in memory cells. Instead of having a single neural network layer, four interacting layers are communicating extraordinarily. were also designed to address the vanishing gradient problem in RNNs. This allows the model to remember information for longer periods and consequently understand the context better. These features are ideal for NLP problems such as this one since the context of words in a sentence and sentences in a paragraph are important.

Before building the model, we will set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (50)
- batch_size: the batch size of the data that will be fed to the model while training (128)
- dropout_rate: how many neurons the model will shut down at every epoch (0.5)

- Activation Function

In the specific model, we used a self regularized non-monotonic neural activation function named “Mish”, as the activation function of the Dense layer described right after. Mish is a novel smooth and non-monotonic neural activation function which can be defined as: $(x) = x \cdot tanh(\varsigma(x))$ (1), where, $\varsigma(x) = \ln(1 + e\hat{}x)$ is the softplus activation function. According to recent papers and novel research, Mish tends to match or improve the performance of neural network architectures compared to other activation function such as ReLU and Leaky and will therefore be used in the MLP model with BERT embeddings and the Bidirectional LSTM one. We tested both this function and a ReLU one on our data and concluded that the Mish function had a higher accuracy score.

Before proceeding to the architecture of the model, it is crucial to define what a Bidirectional LSTM layer is. Bidirectional LSTM (BiLSTM) is a recurrent neural network used primarily on natural language processing. Unlike standard LSTM, the input flows in both directions, and it’s capable of utilizing information from both sides. It’s also a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence. This is an advantage and can produce even better results since every component of an input sequence has information from both the past and present. For this reason, BiLSTM can produce a more meaningful output, combining LSTM layers from both directions.

- Architecture

After defining the parameters of the LSTM model, we build the architecture of the model. The architecture consists of a sequential model with the embedding layer (with an input dim equal to 30000 words, output_dim of a 300-dimensional tensor, weights from the pre-trained embedding for the words in our training dataset and input_length equal to 280). Then follows a Bidirectional LSTM layer with 128 neurons in total followed by a regularization layer (Dropout). Then, we added a dense layer of 8 neurons with the self_regularized activation function 'mish'. Lastly, we have an output layer (of 4 neurons) with one sigmoid activation function each. The plot of the architecture is presented in Figure 34 below.

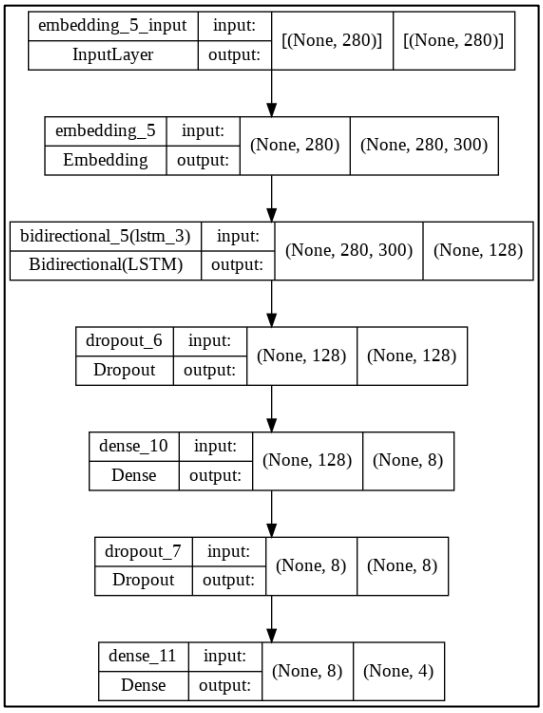

Figure 34 LSTM model architecture

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam with learning rate equal to 0.0001, loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 28 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 9 Evaluation of LSTM model

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
LSTM model	0.544	34.57 %	74.26 %

Loss and Accuracy plots:

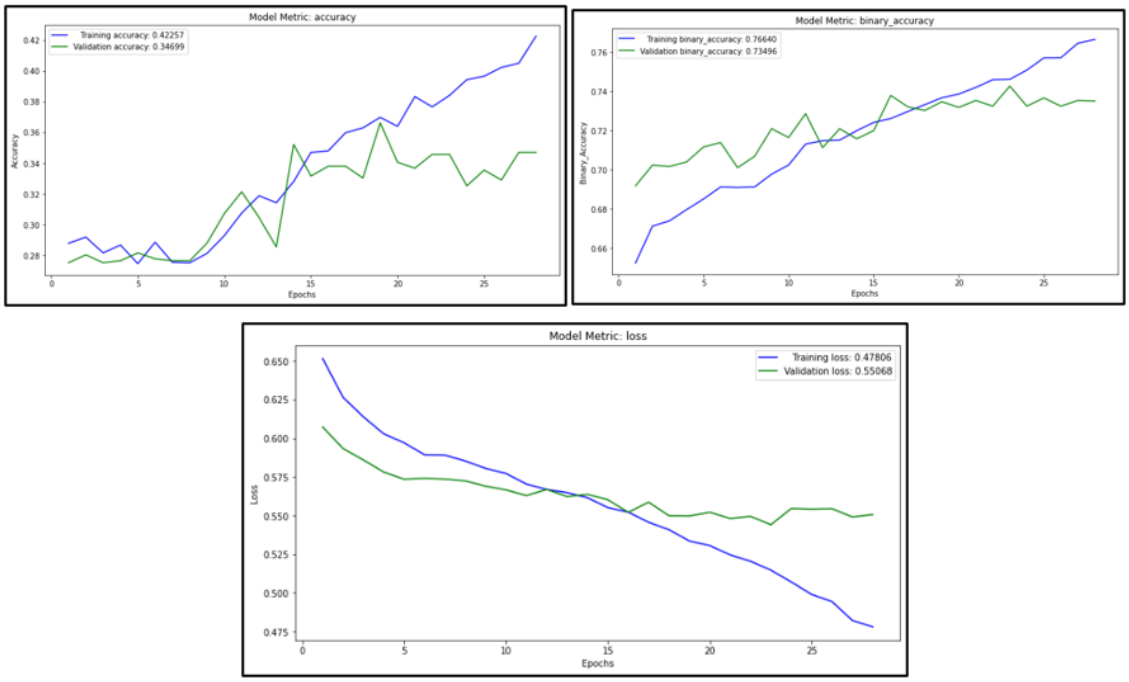

Figure 35 Loss & Accuracy plots of LSTM model

As we can see by the loss plot of Figure 35, validation and training loss are not synced and validation loss is initially smaller than the training loss an indication of underfit of the model. After the 15th epoch, we can observe that validation loss tends to progress and even increase with some up and downs until epoch 28th when it stops because of EarlyStopping.

CNN Models

A convolutional neural network, or CNN, is a deep learning neural network designed for processing structured arrays of data such as images. These models are widely used in computer vision for many visual applications such as image classification and have also found success in natural language processing for text classification. A convolutional neural network is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer. CNN models contain many convolutional layers stacked on top of each other, each one capable of recognizing more sophisticated shapes. To use a convolutional neural network for text classification, the input sentence is tokenized and then converted into an array of word vector embeddings using the pretrained GLOVE embeddings. It is then passed through a convolutional neural network. Because sentence lengths can vary, but the size of the input must be fixed, we used padding technique to make all sentences shorter than the maximum size equal by padding with the value zero.

Before proceeding to the model, we will give a few definitions regarding the parameters and architecture of a Convolutional Neural Network.

- Convolution1D

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs.

- Pooling

Pooling works by placing a matrix, for example a 2 by 2 matrix, on the feature map and performing a certain operation. In average pooling, the average of the numbers falling within that matrix is computed. The result is known as a pooled feature map. Pooling ensures that the network can detect features irrespective of their location. It also ensures that the size of the data passed to the CNN is reduced further. The difference between pooling and global pooling is that pooling is applied over features in each window independently, while global pooling performs over the whole input. For text like in our problem, global average pooling will be used to get a single vector representing the whole text.

- Kernel size

Kernel size is the number of input elements (tokens) a convolution looks at each step. For text, typical values are 2-5. In our case, we will be using a kernel of size five.

1. CNN model

Before building the model, we will set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (50)
- batch_size: the batch size of the data that will be fed to the model while training (128)
- kernel_size: the kernel size for the convolution1d layer is (5)
- nof_filters: the number of filters for the convulution1d layer is (32)
- hidden_dims: the number of hidden neurons is (300)
- dropout_rate: how many neurons the model will shut down at every epoch (0.5)

- Architecture

After defining the parameters of the CNN model, we build the architecture of the model. The architecture consists of a sequential model with the embedding layer in which we will find the embeddings of 30000 words, into a 300-dimensional embedding and the input we can take in is defined as the maximum length, which is 280. Then, we add the convolutional layer and global-average-pooling layer. Finally, we add a dense layer of 300 neurons followed by a regularization layer (Dropout). The last Dense has four as a parameter because we are doing a binary classification with four outputs and so we need four output nodes in our vector with a sigmoid activation function each. The plot of the architecture is presented in Figure 36 below.

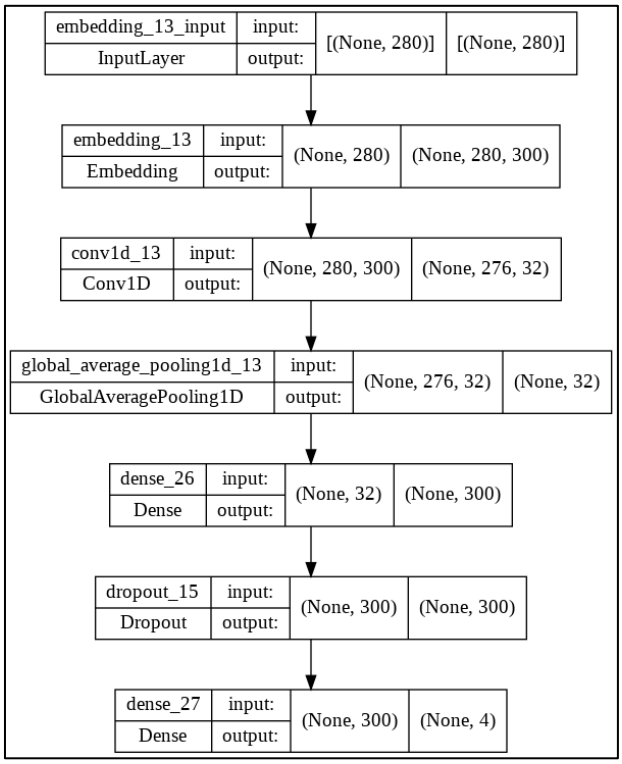

Figure 36 CNN model architecture

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam with learning rate equal to 0.0001, loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 50 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 10 Evaluation of CNN model

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
CNN model	0.509	36.74 %	75.19 %

Loss and Accuracy plots:

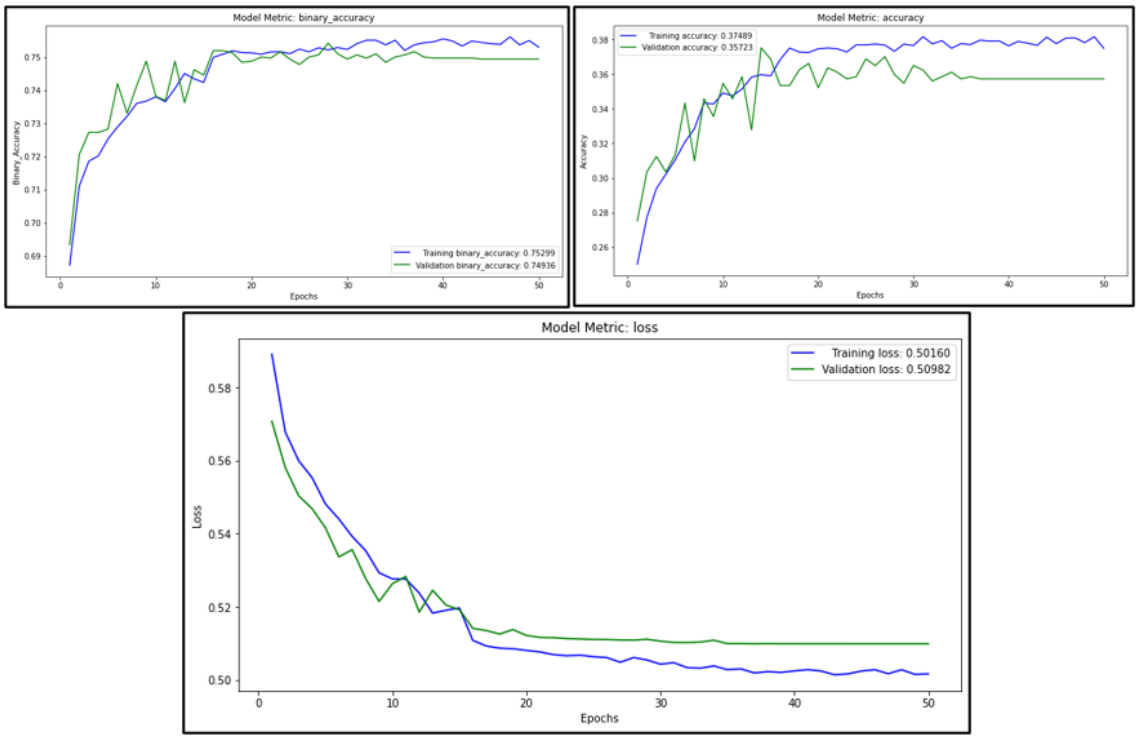

Figure 37 Loss & Accuracy plots of CNN model

As we can see by the loss plot of Figure 37, validation and training loss are synced and validation loss tends to progress in the same way with training loss. This model seems to perform very well since both losses have a decreasing path and since it trains for all epochs that it was set. This model doesn't seem to overfit from the data.

2. CNN-LSTM model

Before building the model, we will set the model parameters. We defined the following parameters:

- Parameters

- nb_classes: the number of classes for the output labels is (4)
- nb_epochs: the number of epochs that the model will train for (50)
- batch_size: the batch size of the data that will be fed to the model while training (256)
- kernel_size: the kernel size for the convolution1d layer is (3)
- nof_filters: the number of filters for the convulution1d layer is (128)
- hidden_dims: the number of hidden neurons is (300)
- dropout_rate: how many neurons the model will shut down at every epoch (0.25 and 0.4)

- Architecture

After defining the parameters of the model, we build the architecture. The architecture consists of a sequential model. The convolution network will be made of the following:

An embedding layer that turns the data into dense vectors of fixed size. A `Conv1D` with 128 units with the `relu` activation function. a `MaxPooling1D` layer with pool size 20. This layer downsamples the input by taking the maximum value from the region of the input which is covered by the filter/kernel. Then follows a regularization layer. Next, we have an LSTM layer of 64 neurons followed by another regularization layer. A flatten layer to flatten the input without affecting batch size. Then follows a `Dense` layer with 32 units for the fully connected layer followed by a regularization layer. Finally, we have the output layer of four neurons, with the sigmoid activation function because this is a multi-output binary problem. The plot of the architecture is presented in Figure 38 below.

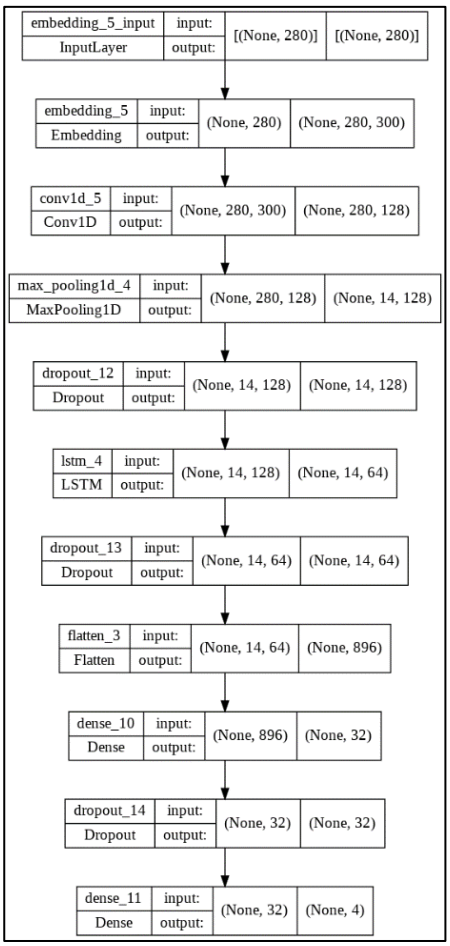

Figure 38 CNN-LSTM model architecture

Lastly, we compiled the model using the parameters that were mentioned above (optimizer: adam with learning rate equal to 0.0001, loss: binary cross entropy, metrics: accuracy and binary accuracy) and then we fit the model to the training data while evaluating on the validation data. The model trained for 14 epochs and after evaluation, gave the following results:

Evaluation Metrics:

Table 11 Evaluation of CNN-LSTM model

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
CNN-LSTM model	0.5246	34.31 %	74.23 %

Loss and Accuracy plots:

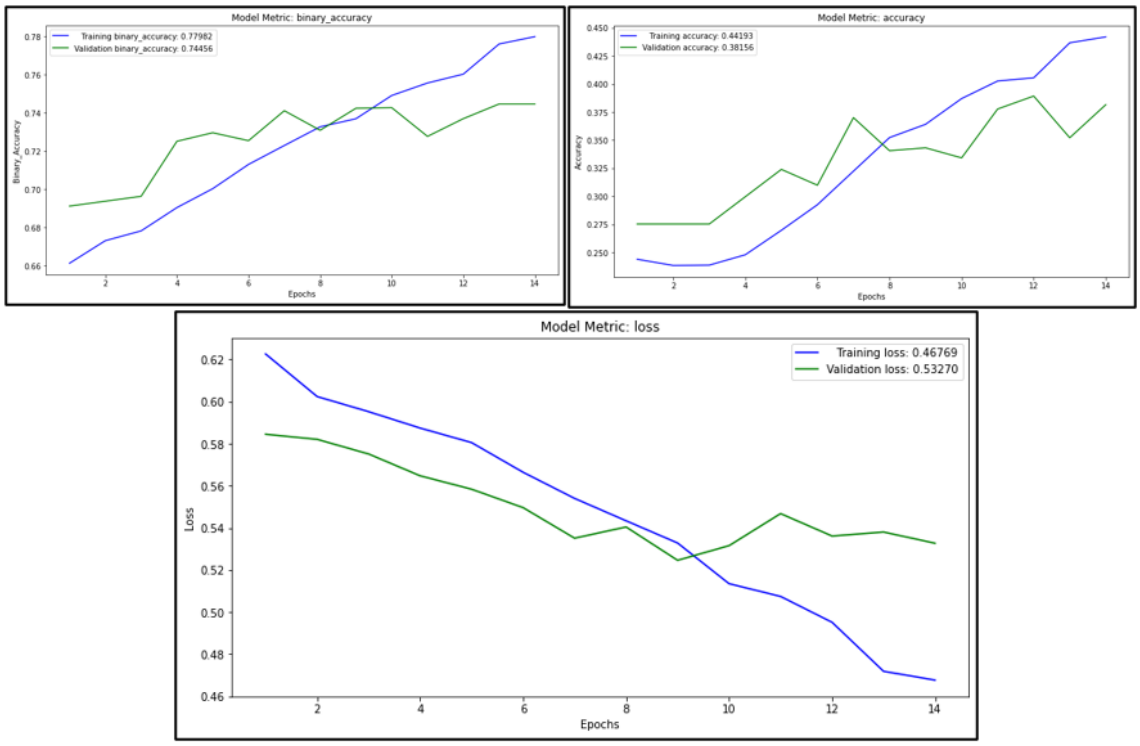

Figure 39 Loss & Accuracy plots CNN-LSTM model

As we can see by the loss plot of Figure 39, validation and training loss are again synced and validation loss tends to decrease until the 10th epoch where validation loss starts to increase, and the model stops training to avoid overfitting in the 14th epoch.

Distil Bert- Transformers pretrained model

Transformers Fine Tuning

For our last model we choose to use a pretrained transformer from hugging face expecting better results from it. The model we choose was “DistilBert” for sequence classification (multilabel text classification in our case). The “DistilBERT” model was proposed is smaller, faster, cheaper, lighter than BERT it is a distilled version of BERT, it has 40% less parameters than BERT-base-uncased, runs 60% faster while preserving over 95% of BERT’s performances as measured on the GLUE language understanding benchmark. “DistilBERT” doesn’t have token_type_ids, you don’t need to indicate which token belongs to which segment. The specific model of our case is DistilBert Model transformer with a sequence classification on top (a linear layer on top of the pooled output) that will be trained in our data for a small number of epochs in order to get the best out of it in our data.

- Preparation Process

First step in the preparation process for the Transformers model was to download the Transformers library in order to have access to the pretrained models. After that we downloaded the “DistilBert” encoder to properly handle and tokenize the text samples. Then we split the dataset into train validation and test dataset and fitted the tokenizer to each one of them. “DistilBert” takes as inputs tensors of length of 512 tokens, so the examples with greater length had to be truncated. As for the padding process, we choose to go with “dynamic padding” as it is more efficient than the standard. “Dynamic padding” fills the sequences to the length of the longest sequence in every batch that goes through the model.

Then we trained the model for 4 epochs with the results shown below:

Table 12 Evaluation of Transformers model "DistilBert"

Method	Binary Cross Entropy	Exact Match Ratio	Hamming Score
Transformers model ‘DistilBert’	0.39	57.49%	84.12 %

Results & Model Selection

Table 13 Comparison of evaluation metrics for all methods

Method	Binary Cross Entropy	Exact Match Ratio	Binary Accuracy
Logistic Regression model	-	38.92%	77.57%
XGBoost model	-	36.87%	77.47%
FFN model with TF-IDF	0.45	43.02%	78.74%
MLP model with trainable embeddings	0.54	34.31%	72.54%
MLP model with GLOVE embeddings	0.57	27.91%	69.30%
MLP model with BERT embeddings	0.46	42.51%	78.45%
BI-GRU model	0.54	35.21%	73.62%
BI-LSTM model	0.54	34.57%	74.26%
CNN model	0.50	36.74%	75.19%
CRNN model	0.52	34.31%	74.23%
Transformers model 'DistilBert'	0.39	57.49%	84.12%

As we can see from Table 13, we have a summary of the evaluation metrics for all models implemented in this assignment. Having taken into consideration the loss plots of all the models as well as the above table, we have resulted that the obvious winner, the one that performed the best based on validation loss (0.26) and the hamming score (84.12%) achieved is the pretrained Transformers model "DistilBert". Therefore, the following section that will include the testing of our model on the unseen data, as well as testing on a random text sample, is performed on the "DistilBert" model. It is worth mentioning that the Multilayer Perceptron method with BERT embeddings and the Feed Forward Neural Network method with the TF-IDF approach, also performed quite well.

Model Assessment

Prediction on Test Dataset

Table 14 Evaluation of Transformers model "DistilBert" on test data

Method	Exact Match Ratio	Hamming Score
Transformers model 'DistilBert'	60%	84.53%

```
Introversion / Extraversion Accuracy : 0.87
Sensing / Intuition Accuracy : 0.91
Thinking / Feeling Accuracy : 0.81
Judging / Perceiving Accuracy : 0.79
```

Figure 40 Accuracy of each label of "DistilBert" model on test data

From Table 14, we can observe the performance of the model on the unseen part of the data. As expected, the model performs very well, as it did on the validation data. To be more precise, the exact match ratio achieved is 60%, and the hamming score is equal to 84.53%. Also, in Figure 40 we can see the accuracy of each label achieved in the test data. As we can see the accuracy is above 85% for the first two labels. For the last two we have an accuracy of 81% for the "Thinking/ Feeling" dichotomy, and for the "Judging / Perceiving" our model captures it with an accuracy of 79%. The accuracies of these two labels (Judging / Perceiving and Thinking/ Feeling) are perhaps closer to reality and what our model achieves, since these two are the more balanced labels in our data according to Figure 12.

Confusion Matrix:

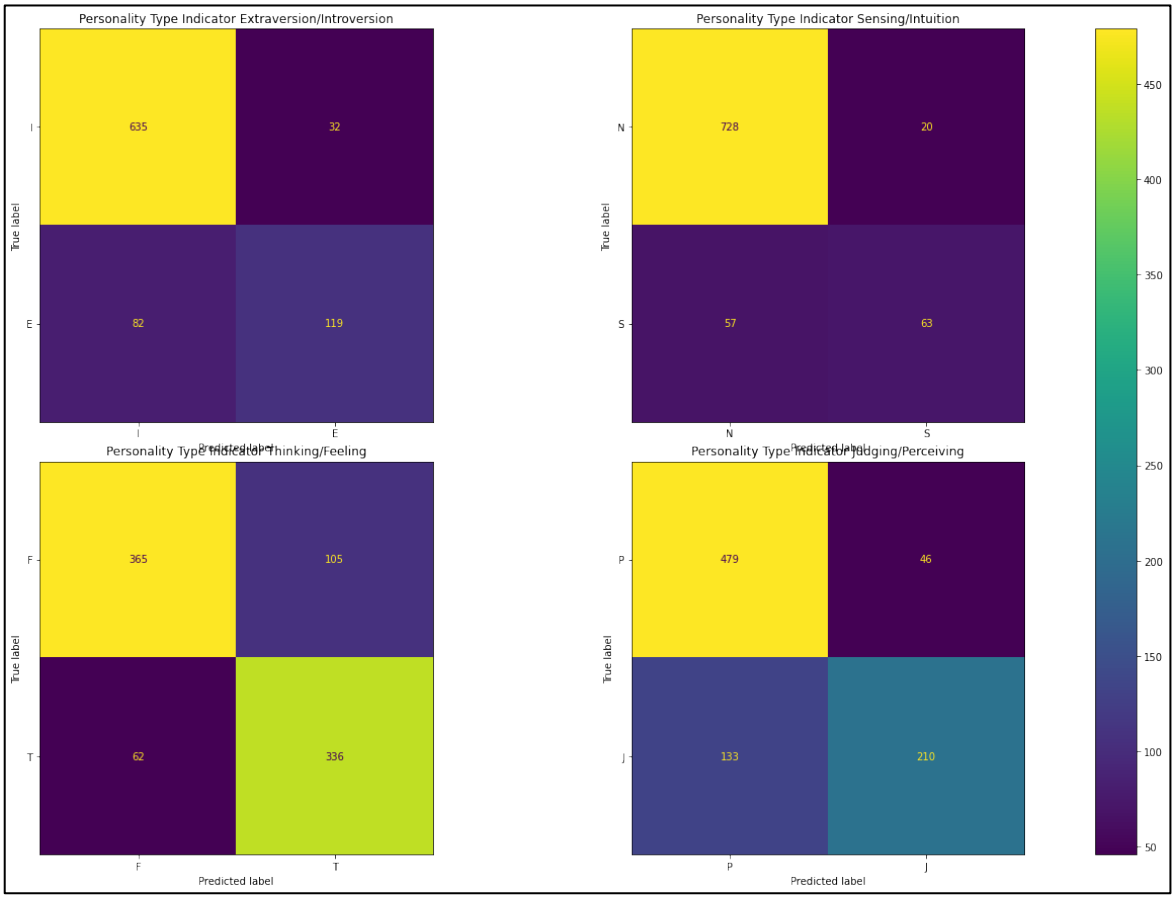

Figure 41 Confusion Matrix of "DistilBert" model

In Figure 41 the confusion matrixes is depicted for all labels of this classification problem. A confusion matrix represents the different combinations of Actual VS Predicted values. At this point we will give a short definition of the metrics that it defines.

- TP: True Positive: The values which were actually positive and were predicted positive.
- FP: False Positive: The values which were actually negative but falsely predicted as positive.
- FN: False Negative: The values which were actually positive but falsely predicted as negative.
- TN: True Negative: The values which were actually negative and were predicted negative.

As we can observe for each label we have:

- Introversion/ Extraversion

I. TP: True Positive: 119
II. FP: False Positive: 82
III. FN: False Negative: 32
IV. TN: True Negative: 632

- Sensing / Intuition

I. TP: True Positive: 63
II. FP: False Positive: 57
III. FN: False Negative: 20
IV. TN: True Negative: 728

- Thinking / Feeling

I. TP: True Positive: 336
II. FP: False Positive: 62
III. FN: False Negative: 105
IV. TN: True Negative: 365

- Judging / Perceiving

I. TP: True Positive: 210
II. FP: False Positive: 133
III. FN: False Negative: 46
IV. TN: True Negative: 479

Classification Report:

In order to analyze the classification report, we will go through a few definitions for the metrics it holds.

- Accuracy: It is calculated by dividing the total number of correct predictions by all the predictions.
- Recall: The recall is the measure to check correctly positive predicted outcomes out of the total number of positive outcomes.
- Precision: Precision checks how many outcomes are actually positive outcomes out of the total positively predicted outcomes.
- F-beta score: F beta score is the harmonic mean of Precision and Recall and it captures the contribution of both of them.

	precision	recall	f1-score	support
Introversion	0.89	0.95	0.92	667
Extraversion	0.79	0.59	0.68	201
accuracy			0.87	868
macro avg	0.84	0.77	0.80	868
weighted avg	0.86	0.87	0.86	868

	precision	recall	f1-score	support
Sensing	0.93	0.97	0.95	748
Intuition	0.76	0.53	0.62	120
accuracy			0.91	868
macro avg	0.84	0.75	0.79	868
weighted avg	0.90	0.91	0.90	868

	precision	recall	f1-score	support
Thinking	0.85	0.78	0.81	470
Feeling	0.76	0.84	0.80	398
accuracy			0.81	868
macro avg	0.81	0.81	0.81	868
weighted avg	0.81	0.81	0.81	868

	precision	recall	f1-score	support
Judging	0.78	0.91	0.84	525
Perceiving	0.82	0.61	0.70	343
accuracy			0.79	868
macro avg	0.80	0.76	0.77	868
weighted avg	0.80	0.79	0.79	868

Figure 42 Classification report of "DistilBert" model

From Figure 42, we can observe that for all metrics we can obtain more balanced results that predict equally well each personality indicator, for the last two labels which are the more balanced ones compared to the first two. Aside from the imbalanced structure in some levels, in general our model achieves satisfying results for all metrics ranging from 53% to 97%.

Prediction in the real world

The last part of this project, in order to test how our model performs in the real world, we decided to have a member of our team undertake the MBTI personality prediction test that can be found here and see how much our model scored, compared to the results from the test. The individual was labeled as the personality type “ISFJ”, aka “the Protector”, according to Myers-Briggs. Subsequently, we fed our model with a cover letter written by this individual and used it to see how it would predict his personality type. The output of the “DistilBert” model for this person is shown in Figure 43 below.

```
[[{'label': 'Extraversion', 'score': 0.183},
  {'label': 'Sensing', 'score': 0.169},
  {'label': 'Thinking', 'score': 0.867},
  {'label': 'Judging', 'score': 0.681}]]
```

Figure 43 Output of prediction for random sample

As we can see the model classified him as “INTJ”. Having said that, this method accurately predicted two out of the four output labels. This is a mediocre result but is quite satisfying if we consider that these models were trained only from a small dataset from posts of people that are quite involved with this personality test, hence a little biased. After all, our data are quite imbalanced and not all classes are equally represented.

Future Work

For the implementation of this project, we used a variety of Neural Network methods in order to maximize the accuracy of the predictions and see if we can find a strong relationship between someone’s writing style and his personality type according to Myers-Briggs test. However, for the training of the models, only a small dataset was found online from a specific forum (Personality Café). In the future, in order to improve results, we would like to take advantage and explore other social networks that may provide useful data. Also, we would like to utilize other Transformer models and see if the predictive results can be improved.

Conclusion

In conclusion, this project provides a solution for predicting personality, utilizing social network data with machine learning and deep learning algorithms that we applied. Beginning with the data preprocessing, we applied multiple tasks to convert the texts in a way that will hold meaningful information and in order to be treated equally, applying lowercase transformation, removing non-words, punctuations, the 4-letter code of MBTI etc. In advance, we proceeded on transforming the problem from multilabel to multiclass and on combining a custom stop word vocabulary with the NLTK package English stop words, in order to be used on the tokenization and stemming process. Then, we segregated the data for training and testing and implemented a series of Machine and Deep Learning algorithms to obtain the best accuracy results. From this project, we resulted on the Transformers model ‘DistilBert’ as the one which had the highest accuracy score in comparison to the rest, for predicting personality based on the “MBTI” personality indicator. We think that this project, obtains results that could be useful for companies when choosing suitable employees, as an individual’s personality and mindset are of utmost importance, and can be crucial for the selection of a position, an establishment of a group and other activities.

Members/Roles

Regarding the responsibilities of each member, throughout these months, we cooperated as a team to find a topic, collect the dataset and do the preparation and cleaning of it. After that we split the models based on a category type for each member. Therefore, Matzakos Nikolaos was responsible for the background research of our project idea and the construction of the business idea, having also a supportive role in the technical part of the project focusing on the RNN algorithms. Continuing to the second member of the team, Papagewrgiou Dimitrios led on the technical research, methodology and implementation of the Transformers Model as

well as the MLP with BERT embeddings one. The third member of the team, Tzeni Tsobanai oversaw technical responsibilities, focusing on the implementation of the machine learning algorithms as well as the CNN and MLP ones. Finally, for the report and the final presentation, all three members of the team worked jointly to present a fully featured result of the aforementioned procedure.

Time Plan

Tasks	Late May	June	July	August
Background theory and finding of topic	✓			
Data collection and creation of time plan		✓		
Data Preparation & Cleaning		✓		
Building of models			✓	✓
Evaluation of Models				✓
Report Writing				✓
Presentation				✓

Notes

The whole project was done on the cloud using Google Colab on GPU and the Python Language. In our Github, six notebooks can be found, one for the Exploratory Data Analysis and Cleaning and the other five hold each type of method. It should be noted that the EDA notebook, the CNN, RNN and Transformers one only need a few minutes up to twenty to run. However, the traditional machine learning approach due to the hyperparameter tuning takes several hours, along with the section “MLP with BERT embeddings” on the MLP notebook.

Bibliography

[1] Sakdipat Ontoum, Jonathan H. Chan, Computer Science Program, School of Information Technology †Innovative Cognitive Computing (IC2) Research Center King Mongkut's University of Technology Thonburi Bangmod, Thung-Khru, Bangkok, Thailand (2022)

Personality Type Based on Myers-Briggs Type Indicator with Text Posting Style by using Traditional and Deep Learning

[2] Hernandez Rayne, Knight Ian Scott, Department of Computer Science Standford University (2017)

Predicting Myers-Briggs Type Indicator with Text Classification

[3] Connor Keith | Southeastern University - Lakeland

USING CONSUMER PERSONALITY TO DIRECT THE CREATION OF MORE EFFECTIVE ADVERTISEMENTS

[4] How the Myers-Briggs Type Indicator Works

https://www.verywellmind.com/the-myers-briggs-type-indicator-2795583

[5] MBTI

https://www.imperial.ac.uk/media/imperial-college/administration-and-support-services/staff-development/public/impex/MBTI.pdf

[6] Benefits of MBTI In the Workplace

https://cmg-agency.com/benefits-of-mbti-in-the-workplace/

[7] How we use Customer Psychographics in Online Marketing

https://www.uniquewebdevelopment.com/blog/using-customer-psychographics-in-online-marketing/

[8] An introduction to MBTI

https://contentoo.com/blog/an-introduction-to-mbti/

[9] Myers-Briggs Type Indicator

https://en.wikipedia.org/wiki/Myers–Briggs_Type_Indicator

[10] A Guide to TensorFlow Callbacks

https://blog.paperspace.com/tensorflow-callbacks/

[11] Use Early Stopping to Halt the Training of Neural Networks At the Right Time

https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

[12] Understanding TF-ID: A Simple Introduction

https://monkeylearn.com/blog/what-is-tf-idf/

[13] A detailed case study on Multi-Label Classification with Machine Learning algorithms
and predicting movie tags based on plot summaries

https://medium.com/@saugata.paul1010/a-detailed-case-study-on-multi-label-classification-with-machine-learning-algorithms-and-72031742c9aa

[14] Recurrent Neural Networks (RNN) with Keras

https://www.tensorflow.org/guide/keras/rnn

[15] GloVe: Global Vectors for Word Representation

https://nlp.stanford.edu/projects/glove/

[16] Intuitive Guide to Understanding GloVe Embeddings

https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

[17] Multinomial Logistic Regression With Python

https://machinelearningmastery.com/multinomial-logistic-regression-with-python/

[18] One-vs-Rest and One-vs-One for Multi-Class Classification

https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

[19] Metrics for Multi-Label Classification

https://medium.com/analytics-vidhya/metrics-for-multi-label-classification-49cc5aeba1c3

[20] Metrics for Multilabel Classification

https://mmuratarat.github.io/2020-01-25/multilabel_classification_metrics

[21] Python for NLP: Multi-label Text Classification with Keras

https://stackabuse.com/python-for-nlp-multi-label-text-classification-with-keras/

[22] Multiclass & Multilabel Classification with XGBoost

https://gabrielziegler3.medium.com/multiclass-multilabel-classification-with-xgboost-66195e4d9f2d

[23] Python – XGBoost for multilabel classification

https://itecnote.com/tecnote/python-xgboost-for-multilabel-classification/

[24] XGBoost Algorithm: Long May She Reign!

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

[25] Bi-directional RNN & Basics of LSTM and GRU

https://medium.com/analytics-vidhya/bi-directional-rnn-basics-of-lstm-and-gru-e114aa4779bb

[26] Mish: A Self Regularized Non-Monotonic Neural Activation Function

https://arxiv.org/vc/arxiv/papers/1908/1908.08681v2.pdf