ECG Classification Report

Omenetti Matteo[1] and Srinivasan Ravi[2]

[1]ETH Zurich - momenetti@ethz.ch
[2]ETH Zurich - rsrinivasan@ethz.ch

Summary This report is part of the second project for the course "Machine Learning for Health Care" hosted at ETH Zürich. This project's scope was to build different models for sequential sentence classification of abstracts of randomized controlled trials. All of the notebooks described in this report can be found in "src/" and have been optimized to be used on Google COLAB-Pro to take advantage of GPU training. The notebooks contain graphs that have been omitted due to the length constraints of the report.

Dataset The dataset used is the PubMed 200k RCT dataset (which can be found at the following link: https://github.com/Franck-Dernoncourt/pubmed-rct). The dataset consists of 2.3 million sentences that have been labeled according to the corresponding abstract. The list of labels is: "Background", "Objective", "Method", "Result" and "Conclusion". The data is split as follows: Train, Dev, and Test.

Preprocessing As the original data format is ".txt", we started our work by preprocessing the data. The notebook used for preprocessing for Task1 is "src/Task1_preprocessing.ipynb". The preprocessed sentences are then saved as pandas Dataframe. The steps we took for preprocessing are the following:

1. Filtering out the lines of the empty lines and the tag lines (before each group of sentences an identifier tag is presented, starting with "###");
2. Splitting the label from the sentence;
3. Lower-casing the sentences and tokenizing them using RegexpTokenizer from nltk;
4. Removal of stopwords;
5. Stemming the words using PorterStemmer from nltk;
6. Lemmatizing the words using WordNetLemmatizer from nltk;

Data analysis After preprocessing the dataset we conducted some analysis. We first investigated the length of the abstracts.

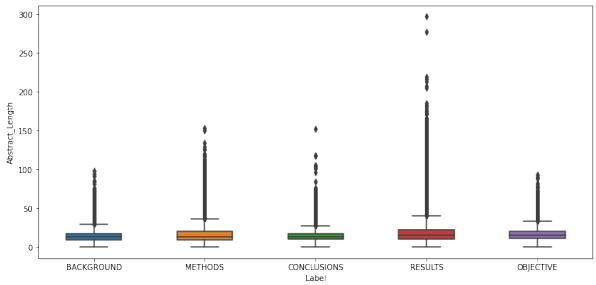

Figure 1: Length of abstracts divided by class

We observed a mean length of 15.8 words, a minimum length of 0, and a maximum length of 297 words. Fig. 1 shows the length of the abstracts divided by class. We can observe that all the types of abstracts share a similar length and there is no major difference between types of abstracts. We also analyzed the distribution of the classes in the dataset which is plotted in Fig. 2. As one can see, the dataset is not balanced, as the classes "Result" and "Method" are over-represented compared to the other classes. This had to be taken into account when building and training the models. We also decided not to process ngrams during the preprocessing as we decided to study ngrams differently depending on the model.

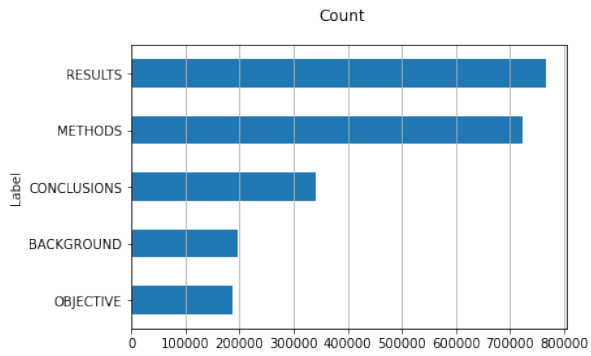

Figure 2: classes distribution in the dataset

TF-IDF TF-IDF (term frequency-inverse document frequency) is a method that allows to determine the importance of a word to a document in a collection of documents.

Vectorization We decided to use TfidfVectorizer from sklearn to create the vectors representing each document (in our case each sentence). To improve performance for the classification step, we decided to exclude from the vocabulary the words that appeared in more than 50% of the sentences (generic words) and in less than 100 sentences (words that are too specific). We also included 2-grams. The size of the vectors we obtained is 45643. Being that the sentences have a mean length of 15.8, the matrix representing the corpus is a very sparse matrix, meaning that most of the cells of the vector carry no information.

Classification To perform the classification using the previously generated matrix we opted for a linear support vector machine (LinearSVC from sklearn). Two factors had to be taken into account for choosing the hyperparameters: first, the sparsity of the vectors which meant that feature selection was necessary. To deal with this we used L1 regularization. Fig. 3 shows the distribution of values of the weights of the feature used by LinearSVC. We can interpret this as feature importance and one can see that only a fraction of the total feature has high importance compared to the rest.

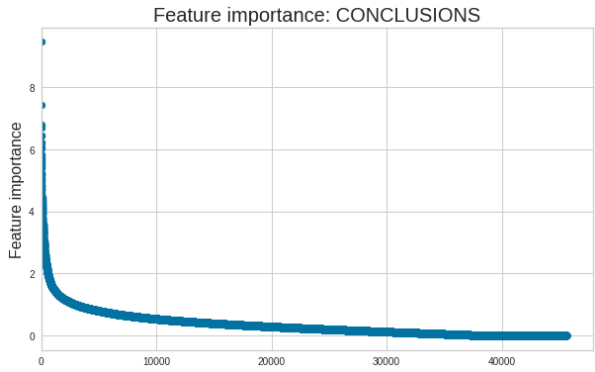

Figure 3: Absolute value of feature weights for the class "CONCLUSIONS"

Second, as we saw in the previous section the dataset is imbalanced meaning that some classes are over-represented. To solve this problem it was sufficient to set the parameter "class_weight='balanced'". The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data. The model achieved an F1 score of 0.729 and an accuracy score of 0.800.

Interpreting the model Thanks to the way LinearSVC works, it is possible to extract some information from the model: by plotting the highest absolute value feature weights it is possible to see which features are selected by the model as the most important features: Fig. 4 shows the top 30 weights by absolute values. Looking at the label of these features it is easy to see that a lot of the labels refer to p-values, which we would expect to find in the abstracts referring to the results. This also confirms the importance of 2-grams in the classification problem, as most of the most important features are actually 2-grams.

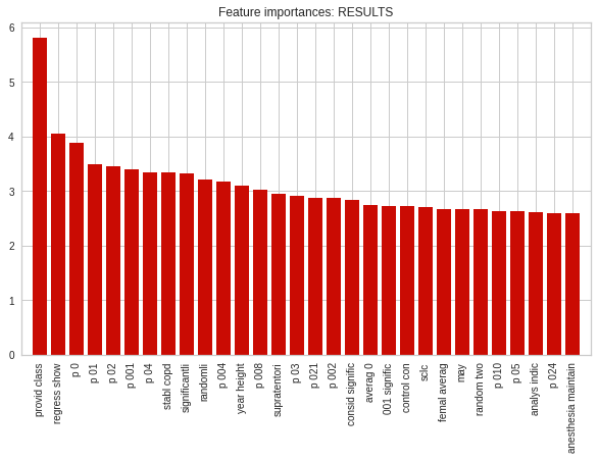

Figure 4: Top 30 most important features for the class "RESULTS"

Other models When trying other models, the same problems mentioned above apply: a high number of features and an imbalanced dataset. We tried to build a deep learning model using Keras to perform the classification. To handle the class imbalance problem we used imblearn SMOTE (Synthetic Minority Over-sampling Technique) to create a balanced dataset to feed to the deep learning network. We then tried to use the same Feed Forward Network used in the next task (Word2Vec). The matrix generated by TfidfVectorizer is a sparse.csr_matrix from scikit (this allows to handle the matrix despite the size). The problem we encountered is that Keras does not natively sup-

port this format as an input and we were not able to find a way to make it work apart from converting the sparse matrix to a NumPy array (which would cause Colab to crash due to memory saturation). We also tried to reduce the number of dimensions using TruncatedSVD from sklearn (similar to PCA for sparse.csr_matrix) but we would still need too many features to maintain a good percentage of variance. Thus, we decided to abandon this approach in favor of the LinearSVC.

Word Embedding The scope of task 2 was to train a word embedding model such as W2V or FastText. W2V was chosen and the implementation of the gensim library was used. We found out that training the W2V model using also bigram was beneficial for the final classification models, therefore we used the gensim library to detect bi-grams in our dataset. Immediately we had to face a decision in the choice of hyperparameters, the length of each embedding vector. First, we chose this length to be 300, which means that every word in our dataset was represented using a vector with 300 entries. We made this choice because our vocabulary was large and we thought that with such large vectors our models had more freedom to work with. We were wrong, even with a size of 200, the models we used in this task achieved almost the same results, therefore using vectors of size 300 was not justifiable from a performance standpoint.

Word2Vec Having now a trained W2V model, we started playing with its embedding to see if the model was able to grasp the semantic meaning of the words in the dataset. The results were satisfactory. When asking the model for the most similar words to HIV we get the following results (in descending order of similarity): 'HIV infection', 'virus HIV', 'HIV positive', 'HIV aid', 'human immunodeficiency'. When asking the model to tell us which word between doctor, patient and sick was the least pertinent, the word doctor was chosen. The model, therefore, understands in some way that patients are sick and not doctors. Then the model was asked to tell which word is to PrEP as HIV is to immunodeficiency and the output was 'HIV prevention', a word that makes perfect sense since PrEP is a drug that prevents people from getting infected with HIV in the case of an exposure with an HIV positive person. Finally, the 10 most similar words to "Doctor" were plotted along with 8 random words in two dimensions using PCA. As Fig. 5 shows the 8 random words are further away in the graph compared to the 10 most similar words. This is exactly what an embedding model tries to achieve, to have similar words physically close together in the learned embedding space.

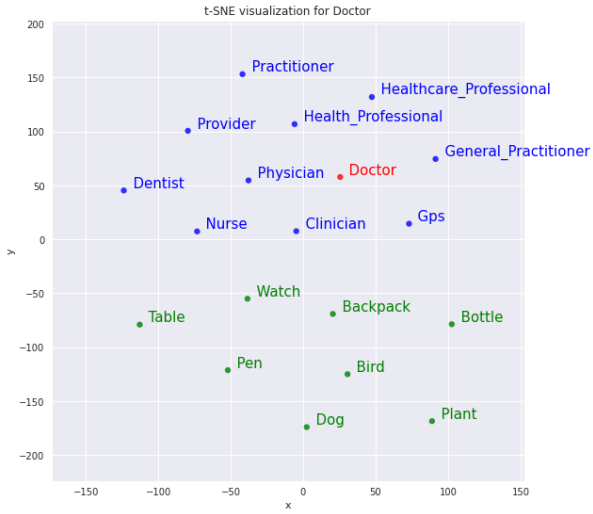

Figure 5: The 10 most similar words to "Doctor" and 8 random words plotted in two dimensions using PCA

Models To train the models, for each training sample its sentence embedding was obtained by averaging each word vector in the sentence. Since the W2V model was trained only on the training dataset, when computing the sentence embeddings for test and dev some words were out of vocabulary and our code had to take this possibility into account using a try-catch block. The W2V model was trained only on the training corpus because if this model was ever deployed in a real-world context we wouldn't know which words were to be used and some could be out of vocabulary. Therefore training the W2V model on the entire dataset was perceived somehow as cheating, making the performance of our classification model on the test set not representative of its real-world performance. Two models were used for this task. A support vector machine, given its excellent performance in task 1 and a multi-layer perceptron.

Support Vector Machine Surprisingly, the SVC was very slow in the training process and we didn't succeed in training with the entire dataset. To make it converge in a reasonable amount of time, we had to cut the size of the training set by half. This model even with just half of the dataset was able to achieve a 0.73% accuracy. We are confident that this model could have performed much better, but due to the long training time, we weren't able to use the entire dataset and perform an adequate hyper-parameter tuning.

Multi Layer Perceptron The second model we experimented with is a standard MLP. Different kinds of architectures were used, bigger and smaller. Not huge differences were noticed in the choice of architecture. We finally settled in with the one you can find in Fig. 7 which was able to provide the best ratio performance/size. The MLP model was able to achieve a good performance (f1 score : 0.722, accuracy score: 0.801) in a reasonable amount of time thanks to GPU acceleration. The confusion matrix of this model can be found in Fig. 6

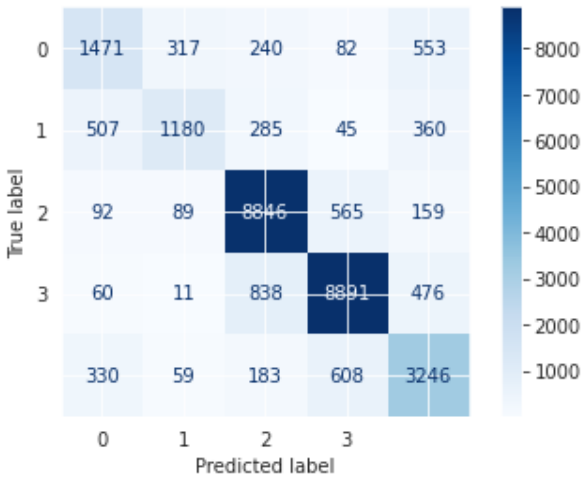

Figure 6: The confusion matrix of the MLP model

BERT The scope of task 3 was to finetune a pre-trained BERT model on the provided dataset. We used the pre-trained models of the Huggingface library. Since the training procedure of a BERT model is more complicated (masking arrays) the provided per model tokenization algorithm had to be used. To use the Autotokenizer, the max_length hyperparameter had to be chosen. This parameter truncates the phrase after the specified number of words. We experimented with different choices, but given that the average sample length (without any preprocessing) is 26 (median 23) we decided to go with a max_length of 30.

BERT base uncased The first model we experimented with in task 3 is the Huggingface BERT base uncased. With its ~15millions downloads, this model is the most common BERT model provided by the Huggingface library. BERT base uncased is a transformers model pretrained on a large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labeling them in any way with an automatic process to generate inputs and labels from those texts. By fine-tuning only the head of the model, we were to achieve an f1-score of 0.800 and an accuracy of 0.861. This model already outperforms all the models and approaches used in tasks 1 and 2.

BioClinicalBERT Then we proceeded in looking through the Huggingface documentation to see if there was a BERT model that was trained on BioMedical data. We found emilyalsentzer/BioClinicalBERT. The model hyperparameters were first initialized with BioBERT a publicly available BERT model trained with BioBERT-Base v1.0, PubMed 200K, and PMC 270K. Finally, all the weights of the model were fine-tuned on all the notes from MIMIC III, a database containing electronic health records from ICU patients at the Beth Israel Hospital in Boston, MA. Once again, we fine-tuned the head of the model with the given dataset. The final model was able to achieve an f1 score of 0.808 and an accuracy of 0.867. Overall, this was our best-performing model. The increase in performance is not drastic compared to the "standard" BERT model mentioned above, but given that the training cost is the same as well as the number of total parameters (~110M) even this slight performance improvement is not to be ignored.

References

Alsentzer, E., Murphy, J. R., Boag, W., Weng, W.-H., Jin, D., Naumann, T., & McDermott, M. B. A. (2019, June). Publicly Available Clinical BERT Embeddings. *arXiv:1904.03323 [cs]*. Retrieved 2022-04-23, from http://arxiv.org/abs/1904.03323 (arXiv: 1904.03323)

Table 1: Summary of results - Accuracy and F1-Score

	Accuracy	F1-Score
Task 1 SVC	0.800	0.729
Task 2 SVC	0.730	0.682
Task 2 MLP	0.801	0.722
Task 3 BERT	0.861	0.800
Task 3 Clinical BERT	0.867	0.808

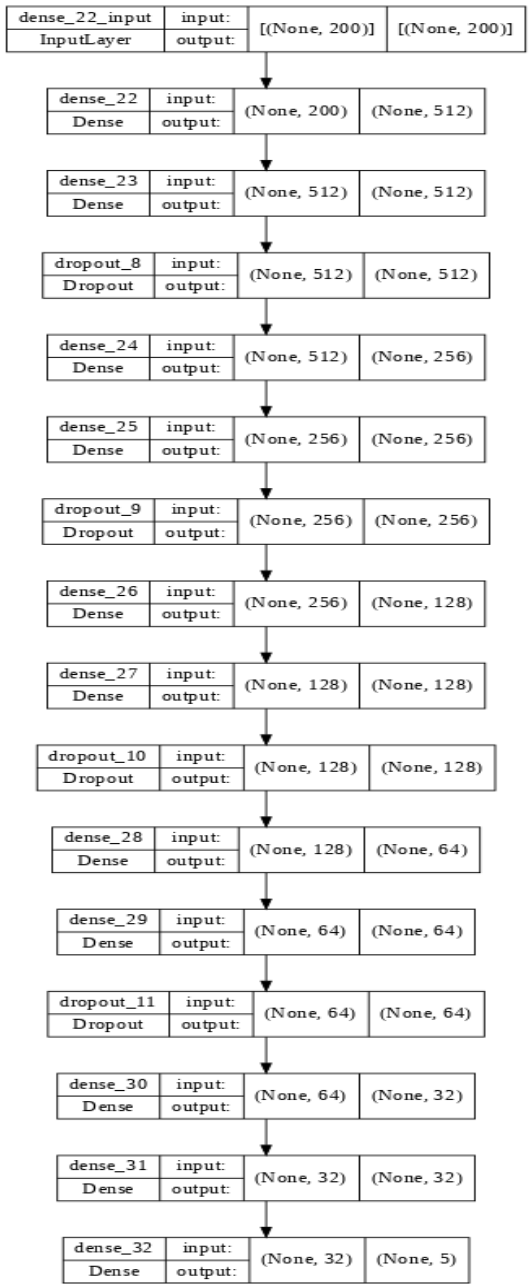

Figure 7: MLP architecture used in Task 2. Total number of parameters 627,813