Medical Text Classification

Kevin Chuang, 012480052

Overview

For this project, I implemented a predictive model, specifically a k-nearest neighbor classifier, that can determine, given a medical text abstract, which of 5 classes (medical conditions/diseases) it falls in. The 5 classes are **digestive system diseases**, **cardiovascular diseases**, **neoplasms**, **nervous system diseases**, and **general pathological conditions**. This report will detail the techniques and reasoning behind the choices I made in text processing and model development.

Exploratory Data Analysis (EDA)

Data exploration is used to understand the data. We are given a training set and a test set. Training set contains a column for labels (1-5), and another column for the medical abstract (text data), while the test set contains just the medical abstracts with no labels. The classes are imbalanced, with label 5 being over represented and label 2 & 3 being under represented. Because of the class imbalance, accuracy is not a sufficient metric, and thus F1 score is used to measure performance. There were no null values or redundant rows. However, there were **duplicate abstracts** that belonged to multiple classes for both the training and testing set. This could be because of inconsistencies in the data, or it could mean that a given medical abstract could belong to multiple categories of medical conditions. If the latter is true, then this can be categorized as a multi-label text classification problem, where multiple labels can belong to a single instance. Using data visualizations and the **tf-idf scores** (more on this below), I was able visualize the terms that are most important given the labels to get a rough idea of what class label is what specific medical condition. With some research, my educated guess is that label 1 is neoplasms, label 2 is digestive system diseases, label 3 is nervous system diseases, label 4 is cardiovascular system diseases, and label 5 is general pathological diseases.

Data Preprocessing (Feature Engineering)

I utilize the **Bag of Words** approach for processing the text. Essentially, the BOW approach breaks up the documents into individual words and their counts within the corpus.

In terms of implementation, I use a **tokenizer** to break each medical abstract into a list of cleaned words. I clean the words by removing all punctuation and numbers, lowercasing, and stemming each word. Stemming is the process of converting words to the stem (root) of the word, and the goal is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form (e.g. *used* is same as *use*). I experimented with lemmatization (**WordNetLemmatizer**) and different implementations of stemmers (Porter, Snowball, Lancaster) and found the **Porter Stemmer** to produce the best results. Porter is the least aggressive of the stemmers, and since the data contains a lot of complex medical terminology, it makes sense to keep more of the words in their entirety.

I use a **term frequency inverse document frequency vectorizer** from sci-kit learn (**TfidfVectorizer**) to convert the collection of abstracts into numerical feature vectors (i.e. vectorize). The **TfidfVectorizer** will normalize (Euclidean / L2 norm) and weight with diminishing importance words that occur in the majority of samples / documents, producing a CSR sparse matrix. Term frequency inverse document frequency tries to highlight words that are more interesting, such as words in a single document rather than the whole corpus of documents. Since the medical abstracts contain a lot of multi-word expressions (e.g. *left anterior descending coronary artery*), I utilize the n-grams approach. The reasoning for this is that n-grams (with n > 1) will keep local positioning of important words. After experimenting with many different numbers of grams (1-grams to 4-grams), I found that the optimal results are achieved using both **1-grams and 2-grams**. In addition, I remove all the stop words (e.g. *the, is, in, during, etc.*), which are commonly used words that do not give any information and are irrelevant. I use nltk and sklearn's stop words corpus, which comes out to around **378 stop words** ignored. In addition, I remove all words that only exist in 5 or less documents (**min_df**), since, similar to stop words, they do not provide much more information. Finally, I fit the vectorizer on the combination of the training and test medical abstracts.

k-NN Model Implementation, Evaluation, and Tuning

My implementation of k-nearest neighbor utilizes **cosine distance metric** to measure distances between instances and does a majority vote based on the label with the most neighbors given K nearest neighbors. If there is a stalemate, I use the instances of the class labels that tied, and calculate the inverse squared distance score of those instances to determine the final prediction label. A higher inverse squared distance score correlates with a higher similarity (inverse distance squared is proportional to similarity).

I implemented **k-folds cross-validation** and a function to calculate **F1 score** to estimate the generalization performance of my model based on F1 score. The training data is split into k equal-sized folds (with different distributions of labels), and the model is trained on k-1 folds, and evaluated on a holdout test set. This cross-validation occurs for k iterations, until the model has seen all the data, and each k subsample has been used for both training and validation. The average of the k-folds scores is used to give an estimate of the generalization performance of my model. Because each k fold has a different distribution of class labels, this technique provides a good generalization of performance on the test set, where the distribution of class labels is unknown. In addition, the k-folds cross-validation is used for tuning my model's hyper-parameter K, which is the number of nearest neighbors to consider when deciding what class to classify an instance. I implemented a **grid search function** to loop through a range of values for K in order to find the optimal K. The optimal 10-Fold CV score on the training set for my model came out to around **0.623411** with **K = 43 nearest neighbors**.

Rank & F1-score

My current rank on CLP public leaderboard is **2nd** with a F1-score of **0.7858**.

Running the code

I utilize Python 3 for this project. The script can be run by calling it with python in a terminal (i.e. **python3 medical_text_classification.py**). The code assumes that **train.dat, test.dat,** and **format.dat** are in the same directory as the script. Also, it assumes that you have the libraries installed: numpy, scipy, pandas, scikit-learn, and nltk. I have commented out the evaluation (10 Fold CV) to save time for running the script. The script will produce a text file called **submission.txt** with the prediction results of the test set.