Evan Lapid

Professor Galletti

CS 506

28 October 2024

Midterm Writeup

The final algorithm that was used is a combination of TF-IDF Vectorization and Logistic Regression. Term Frequency-Inverse Document Frequency (TF-IDF) is a metric that is used to measure how relevant a word is in a collection of documents. It converts the given textual data into numerical values by focusing on words that are more common in the document. However, as used in the code, it can reduce the influence of super common and general words. In this case, the vectorizer collects the top 500 most common words and uses the parameter “stop_words = ‘english’”. This parameter eliminates commonly used words that are of little value to the significance of a document. Then, logistic regression uses this output to make classification predictions. The reason that this approach was taken, is that it is a mix between simplicity and effectiveness. Since regression is very well suited for classification tasks, especially with TF-IDF, it is highly beneficial to use for this project.

Initially, my approach was to use Random Forest or XGBoost. However, I found it difficult in both scenarios to configure each of them to this specific scenario. I was getting lower scores with XGBoost and it took way more time to run the code. With XGBoost I was waiting so much longer for my code to run, and it would often execute after an hour with an error or just give me the same score even if I added something. For Random Forest I found that I couldn’t get my accuracy as high as I wanted, and so I figured I would switch to logistic regression. I found this by asking ChatGPT how I might be able to find better results, and it gave me a list of algorithms that might be better to use. Another complication I had even after I switched to this algorithm, was that if I tried to add more complex features or special tricks, it would take extremely long to run my code. In hindsight, and after hearing about some classmates' solutions, I should have stuck with what I had and ran my code overnight. Being new to this as a computer science major, I have not yet dealt with datasets this large and didn't know that it could take as long as it does.

For the final algorithm, some key steps that I took are as follows. The first step was to build a pipeline to streamline the modeling process. The first was the TF-IDF Vectorier which I have already explained. The second step was to create the Logistic Regression classifier. One key parameter that I used was the saga solver, which optimizes for large data sets and improved the speed that my code ran. Since TF-IDF has high-dimensional features this was helpful in running my code efficiently. I also used a regularization parameter which balances bias and variance. Then I used grid search for hyperparameter tuning. For this a GridSearchCV was employed to fine-tune both TF-IDF and Logistic Regression model parameters. GridSearchCV is a method that automates the tuning of hyperparameters by testing each possible combination within specified ranges, ultimately selecting the configuration that achieves the best performance on the validation data. This is good because even minor adjustments can raise accuracy, and nearly all possible configurations are tested. I used two main hyperparameters which were max_features and classifier_c. The max-features parameter controls the number of words that are used. By selecting the right number, we can ensure that the model chooses the most common and informative words in the dataset. This makes sure the model isn't overloaded with insignificant words which can add noise and slow down training, The regularization parameter C helps control complexity and prevent overfitting. The good thing about grid search is it efficiently identifies which C value is the best balance between underfitting and overfitting, creating a model that generalizes well to unseen data.

When running grid search I used a reduced cross-validation fold where cv=2, which is used to speed up the process while still ensuring a representative performance check across different subsets of data. I then designated that we are prioritizing configurations that maximize accuracy. Verbose output was used to track progress and make it easier to look at what was happening in real time, and get estimates on the algorithm. We then apply grid search to the training data, iteratively training and evaluating the model with each parameter set to identify the best-performing configuration. This refines the pipeline, optimizing it for speed and accuracy.

In conclusion, this project allowed me to explore various algorithms and techniques that I could use to develop an efficient and effective algorithm. Having the ability to trial and error allowed me to explore the pros and cons of each. Seeing how each of them were different from one another was interesting because it gives you perspective into how a data scientist thinks. Coming from a computer science background I had difficulty visualizing how these models solved the assignment. For many of my peers in data science, they were familiar with this problem and how to approach it. They knew what to look for and how to think about it. I found this aspect of the project and this class overall extremely helpful and motivating for the rest of the semester. On top of that, the project itself was pretty cool. In most of my classes we make algorithms to solve problems that I would rarely ever use in my regular life. However, this project created a model that is exciting and applicable. Overall, while this project was a big learning curve and difficult experience I enjoyed getting hands-on experience and being able to create an efficient text classification model.