IML Hackathon 2020

Group Members: Guy Lutsker, Maya Harari, Omer Plotnik, Hadas Nahshon

June 11, 2020

Task Chosen: Github repositories classification

Description of our work: Our Team Github Repository

"Github repositories classification" is a supervised multi-classification learning problem. We chose this problem because it was a really cool challenge and we had to no clue how to even start.

1 Data Partitioning

The repository file sizes range from 5,000 code lines to 100,000 code lines. We started out by seperating our data into 80% train and 20% test. We put aside the test data and did not use it until our final run. We also decided to work with random_state set to 25 to keep our different algorithms consistent.

2 Exploratory Data Analysis

We began by trying to get an idea of the word distribution within each project. We created histograms of word count vectors.

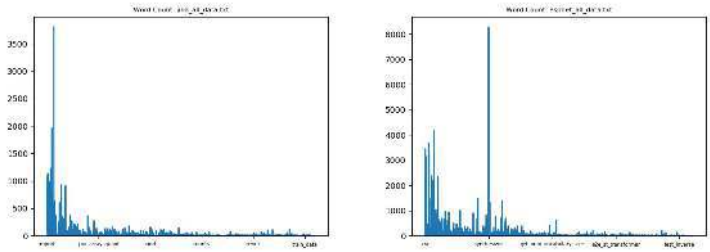

Figure 1:

TF-IDF is a term comprised of two parts:

TF term-frequency describes the frequency of each word in the document (by normalizing word count).

IDF inverse-document-frequency describes the specificity of each word by taking into account how rare a given word is in a specific document, as compared to by the rest.

TF-IDF multiplies a word's tf by its idf, which gives a number that signifies its importance within a document.

We decided to work with the concept of bag of words and tf-idf with n-grams of size 1 only . N-grams of size > 1 seemed more relevant to natural language, whereas in code pairs of words do not neccessarily provide more information beyond syntax rules. (In code, classic n-grams such as "San Francisco" would appear as one word: "san_francisco".)

3 Model Selection

We used the Multinomial Naive Bayes classifier as our baseline algorithm. We began by trying multiple classifiers and multiple sample sizes. We wanted to get an idea of accuracy rates and see which classifiers were completely irrelevant (line plot). Once we had a general idea, we started testing our models on different parameter values (boxplot). Even after trying different parameters, LinearSVC reached the best results. We ran the model using different k values for k-fold Cross Validation in order to see how it would affect our models and to make sure our folds were representative. For example, both $k = 10$ and $k = 5$ gave the same accuracy but one gave the best results for $\lambda = 2$ and the other for $\lambda = 1$.

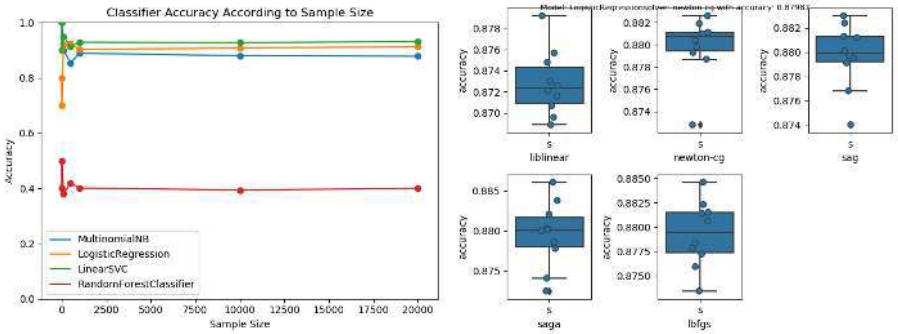

Figure 2:

We also tried to run boosting but the runtime was too long and the performance was terrible. We suspect that this was caused by the fact that the base learner we provided it with was too complicated. The simplest learner, DecisionTree, had terrible results pre and post boosting. .Because we reached satisfying results with LinearSVC and because RandomForestClassifier performed badly during our model selection process (with various depths), we decided to choose LinearSVC as our final mode. We zeroed in on the the successful λs and tried a range between 1 and 2.

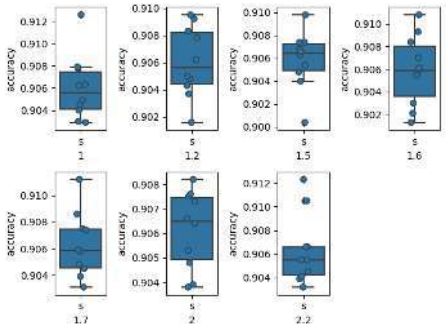

Figure 3:
Model: LinearSVC, C: 1.7 with accuracy: 0.9062699999999999. (plot per $s = C$)

4 Model Evaluation and Generalization Error Estimate

We ran LinearSVC with $C = 1.6$ on our untouched test data and reached an accuracy rate of 0.914.