Joseph Jinn

Project Report

Vision:

The general purpose of the project is to perform Social License to Operate Triple-Bottom-Line topic classification on Twitter data associated with various mining companies. Social License to Operate indicates the ongoing acceptance of a company or industry's standard business practices and operating procedures by its employees, stakeholders, and the general public (Investopedia). Triple Bottom Line is a framework or theory that recommends that companies commit to focus on social and environmental concerns just as they do on profits (Investopedia). We will use supervised machine learning algorithms to perform multi-class single-label classification Tweets to predict whether their topic of discussion corresponds to social, environmental, or economic concerns.

Background:

Our work is a revival and continuation of the work initially done at the Commonwealth Scientific and Industrial Research Organization (CSIRO) by (insert name here) on TBL topic classification. We are not directly referencing that research but instead basing our initial data pre-processing on the anonymous ACL submission titled "Classifying Stance Using Profile Text". We are however using the exact same labeled training dataset that was used in the prior research for TBL topic classification on SLO for mining companies. Our work will also involve use of the datasets available on Calvin College's Borg Supercomputer and will be uploaded to the Calvin-CS / slo-classifiers GitHub Repository. This project will be a prelude to continued research on topic, stance, and sentiment analysis utilizing machine learning for Social License to Operate of mining companies in connection with Professor VanderLinden's "Machine Learning for Social Media" research project.

As of the current status of this report, we are currently rapid prototyping using Scikit-Learn machine learning classifiers. These classifiers require minimal effort to initially setup with default hyperparameters. They train speedily and provide results in a timely manner, allowing us to adjust our hyper-parameters on-the-fly to see if there are any noticeable differences. It is also quite simple to add additional Classifiers as the Pipeline class allows literal copy/paste of a code template. All that is required is the addition of a new import statement for that Classifier and to replace the name of the old Classifier and its corresponding parameters with the new one. This design feature is one of the reasons we chose to utilize Scikit-Learn; that and it was recommended by Professor VanderLinden as the starting point.

Of note is that Scikit-Learn provides automated parameter tuning via the Grid Search and Random Search classes. Grid search methodically builds and evaluates a model for each combination of algorithm parameters specified in a grid. Random search methodically builds and evaluates a model for each combination of algorithm parameters sampled from a random distribution for a fixed number of

iterations. We plan to utilize one or both of these parameter tuning methods in order to expedite the search for optimal hyperparameters for all of the Scikit-Learn Classifiers we are prototyping with. As we add additional Classifiers to our codebase, it becomes time-saving to automate parameter tuning as much as possible.

Once we have established which classifiers have the most potential to provide favorable metrics, we may migrate towards Keras and Tensorflow for GPU support and more versatility. Scikit-Learn does not provide GPU support for its machine learning algorithms. This does not matter at the moment as we are working with two very small datasets which in total only provide us with 330 samples. That and GPU support will primarily benefit deep neural networks while we are also using non-NN algorithms. However, if we wish to crowdsource TBL classification on significantly larger Twitter datasets and work with those, then GPU support will become necessary. We have heard it requires approximately 24 hours utilizing one Nvidia Geforce Titan on the Borg supercomputer to perform stance analysis training on the larger Twitter datasets consisting of 500k+ examples. It would be expedient to parallelize this process utilizing all 4 Nvidia Geforce Titans on Borg to cut the training time down to a quarter.

We plan to implement metric visualizations via the use of the matplotlib library and SciView in Pycharm. The Scikit-learn online documentation has a section on “Classification of text documents using sparse features” that can hopefully be modified to suit our purposes. Their codebase constructs a bar plot comparing a variety of Classifiers side-by-side visualizing the accuracy score, training time, and test time. As we are also training multiple Classifiers in the hopes of finding a suitable one(s) to further explore in the Keras and Tensorflow API, this type of visualization would be very useful. Individual charts detailing a metric summarization of the micro/macro average, weighted average and associated precision, recall, f1-score, and support values are also planned.

Implementation:

These sections will describe in detail (perhaps too much detail) our current implementation for SLO TBL topic classification in Python in association with the current state of the codebase. We have decided to keep all debug output statements in the meantime as this is far from the final system that will be implemented.

We are performing text pre-processing on the training dataset that consists of 229 Tweet examples. Not all of them are labeled with a TBL topic classification and those are dropped from consideration. The data is shuffled randomly upon importation to ensure there is no biased structure to the import order. We do so by utilizing Numpy’s “random. permutation” function. Then, a Pandas dataframe is constructed to store the dataset. Custom column names are added for clarity of purpose

as none originally exist. The “Tweet” column stores the Tweet, “SLO1” stores the first assigned topic label, “SLO2” and “SLO3” do the same.

Pandas provide a “dropna()” method by which we drop all rows without at least 2 non-NaN values. This indicates that the example lacks any TBL classification labels and can be safely discarded. We use Boolean indexing via bitwise operations, the “.notna()” method, to construct a mask by which we isolate those examples with only a single TBL classification. These examples are placed in a new dataframe and afterward, we drop the SLO2 and SLO3 columns as they are obviously just NaN values. This procedure is effective as a preliminary analysis of the CSV file indicates that all labeled examples definitely have a label in the “SLO1” column. Our objective is to construct a dataframe consisting of a column storing the raw Tweet and another column storing a single topic classification. We rename this new dataframe to columns “Tweet” and “SLO”.

Next, we construct another mask to isolate all examples with multiple SLO TBL classifications and apply the mask to construct a new dataframe containing only those examples. We then perform a “drop()” operation on the new dataframe to construct 3 separate dataframes. The first from dropping SLO2 and SLO3, second dropping SLO1 and SLO3, and third dropping SLO1 and SLO2. This inefficient but workable solution effectively create duplicates of all examples with multiple SLO TBL classifications with just a single label per example. We then name the columns “Tweet” and “SLO”. This is done so that our machine learning model can take into consideration those examples that can be classified as multiple topics.

The multiple separate dataframes constructed from the above operations are then concatenated back together as a single whole Pandas dataframe. Any rows with a NaN value in any column are then dropped via “dropna()” to effectively remove all examples with multiple topic classifications that might have had a topic in SLO2 but not SLO3 or vice versa. Last, we drop all duplicated examples possessing the same TBL classification values in the “SLO” column. We do this as the initial imported dataset sometimes contained duplicate labels for the same example. We surmise this is because multiple people were manually hand-tagging the Tweets and sometimes they were in agreement.

Using the “shape()” method call, our final training dataframe contains a total of 245 Tweets with a single TBL topic classification label. It should be noted that as of our current implementation the second TBL labeled dataset provided by Professor VanderLinden is not currently in use. There are 31 additional Tweets and we plan to include these in the future to help alleviate our issue of a small training and test dataset. We are also using a large Twitter dataset that has already been pre-processed and tokenized as the set we will make predictions on in order to test the generalization of our model(s) to new data. This set does not contain any target labels and thus we cannot use part of it to supplement our small training and test sets. There are a total of 658983 Tweets included. The CMU Tweet Tagger was used to pre-process the text but unfortunately, this is not a feasible option for us as we are working solely on Windows OS workstation(s).

As we are incapable of using the Linux/Mac only CMU Tweet Tagger for pre-processing, our decision was to manually clean the raw Tweet using Python regular expressions and other libraries. The Natural Language Toolkit was considered as an alternative but ultimately we chose to just use built-in Python libraries and functions. A for loop is used to send each Tweet to a preprocessing function that does the following:

a) Removes "RT" tags indicating retweets.
b) Removes URL. (e.ge. https//…) and replace with slo_url.
c) Removes Tweet mentions (e.g. @mention) and replaces with slo_mention.
d) Removes Tweet hashtags (e.g. #hashtag) and replaces with slo_hashtag.
e) Removes all punctuation from the Tweet.

We also down-case all text from upper to lower case letters. On our TODO list is to implement regular expressions or other methods in order to:

a) Shrink character elongations (e.g. "yeees" → "yes")
b) Remove non-English tweets
c) Remove non-company associated Tweets.
d) Remove year and time.

For our current two datasets, the yet-to-be-implemented preprocessing features do not seem to be an issue as the preliminary analysis indicates those elements are not present or have already been considered.

The next step was the input feature creation using the "Tweet" column and a target label set using the "SLO" column. Scikit-Learn included a handy function "train_test_split()" which allowed us to easily split our input feature and target labels into a training and test set. The target label train and test sets were then encoded using the Scikit-Learn LabelEncoder class. This converted our categorical labels of "economic", "environmental", and "social", into associated integer values of 0, 1, and 2, respectively. A necessary step as most machine learning algorithms we are interested in prototyping with require and support only numerical data.

The Scikit-Learn CountVectorizer class was used to convert the processed Tweet training and test set into feature vectors with binary values of 0 and 1. Documentation indicates that the class converts a collection of text documents to a matrix of token counts and produces a sparse representation of the counts. As we did not provide an a-priori dictionary and analyzer for feature selection, the total number of features is equal to the vocabulary size of the analyzed data. Hence, we

have a very high dimensionality in our feature vectors compared to our small number of samples. This effectively creates the bag-of-words that we used to represent our categorical Tweet data. The occurrences of each word are stored in the feature vector. Console output shows that we are dealing with a vocabulary size of 809 in comparison to 164 examples for the training set and 81 examples for the test set.

The Scikit-Learn TfidfTransformer class was then used to convert the vectorized categorical Tweet data into term-frequency * inverse document-frequency. The purpose of this is to scale down the impact of tokens that occur very frequently and are therefore empirically less informative than features that occur in a small fraction of the training set. Term frequencies, in general, are better than raw occurrences as larger corpuses will have higher average word occurrence values than smaller corpuses. So, normalization of this kind provides better input feature vectors for training our model.

It is at this point in the code base that we also import a very large Tweet dataset consisting of some 600k+ Tweets that are unlabeled to be used as the input feature for making predictions and seeing how well our model generalizes to new data. These Tweets have already been preprocessed and tokenized by the CMU Tweet Tagger. We simply have to run the entire dataset into a Pandas dataframe, isolate the "tweet_t" column that contains the Tweet, and use the CountVectorizer and TfidfTransformer class to normalize from categorical to numerical data. The details are similar to what is described above. For the future, we plan to do further post-processing on these Tweets in order to minimize the discrepancies between how we pre-processed and post-processed our training and test datasets and how it was done on this Twitter dataset. Two things we have noticed is that those Tweets still seem to contain hashtag items and some punctuation. These should be removed as we removed both in our training and test sets. The predictive ability of our trained model may otherwise be compromised when using these Tweets.

With the training, test, and generalization set properly prepared, we utilized Scikit-Learn's Pipeline class in order to set up various Classifiers. These currently include:

a) Multinomial Naïve Bayes'
b) Stochastic Gradient Descent (SGD)
c) Support Vector Machine – Support Vector Classifier.
d) Support Vector Machine – Linear Support Vector Classifier.
e) Nearest Neighbor KNeighbors Classifier.
f) Decision Tree Classifier.
g) Multi-layer Perceptron Neural Network Classifier.
h) Logistic Regression Classifier.

These are all Classifiers capable of multi-class single-label topic classification. As such, we have decided to implement as many as we can to see which one will be the most performant and worthy of further consideration in the Keras and Tensorflow API, provided those API's support or can be made to support that Classifier.

Results:

As we have just begun initial implementation of our machine learning system, most of these classifiers have been using default hyperparameters and thus our results have been pretty dismal, at best. The highest accuracy metric obtained was almost 56% with the lowest dipping in the 20th percentile. It is our plan to use parameter tuning via Grid Search or Random Search to assist in finding hyperparameters that will improve our metrics. As of the moment, the predictive ability of our Scikit-Learn trained models is less useful than flipping a coin.

Of particular concern to us is performing the proper and necessary pre-processing and post-processing of the Twitter data into useable sparse feature vectors. Regretfully, we will need to obtain the assistance of other researchers with a Linux/Mac workstation and the proper set up in order to use the CMU Tweet Tagger on the labeled TBL datasets. Otherwise, we can only find other alternatives.

It is also within our planned schedule to implement matplotlib visualizations of our metric summaries to display the results of training our models and their predictive abilities in generalizing to new data. As of the current writing of this report, this is where are at in our research efforts. Please refer to the code modules included in this Jupyter Notebook for further details.

Placeholder – discuss comparison with similar works.

Works Referenced

1) "1. Supervised Learning¶." *Scikit*, scikit-learn.org/stable/supervised_learning.html#supervised-learning.
2) "A Gentle Introduction to the Bag-of-Words Model." *Machine Learning Mastery*, 12 Mar. 2019, machinelearningmastery.com/gentle-introduction-bag-words-model/.
3) "Classification of Text Documents Using Sparse Features¶." *Scikit*, scikit-learn.org/stable/auto_examples/text/plot_document_classification_20newsgroups.html#sphx-glr-auto-examples-text-plot-document-classification-20newsgroups-py.
4) "Introduction to Machine Learning | Machine Learning Crash Course | Google Developers." *Google*, Google, developers.google.com/machine-learning/crash-course/ml-intro.
5) "How to Tune Algorithm Parameters with Scikit-Learn." *Machine Learning Mastery*, 1 Nov. 2018, machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/.
6) Kenton, Will. "How Can There Be Three Bottom Lines?" *Investopedia*, Investopedia, 9 Apr. 2019, www.investopedia.com/terms/t/triple-bottom-line.asp.
7) Littman, Justin. "Where to Get Twitter Data for Academic Research." *Social Feed Manager*, 14 Sept. 2017, gwu-libraries.github.io/sfm-ui/posts/2017-09-14-twitter-data.
8) Mohammad, Saif, et al. "SemEval-2016 Task 6: Detecting Stance in Tweets." *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, 2016, doi:10.18653/v1/s16-1003.
9) "Multiclass Classification." *Wikipedia*, Wikimedia Foundation, 18 Apr. 2019, en.wikipedia.org/wiki/Multiclass_classification.
10) "Symbolic Reasoning (Symbolic AI) and Machine Learning." *Skymind*, skymind.ai/wiki/symbolic-reasoning.

11) Walker, Leslie. “Learn Tweeting Slang: A Twitter Dictionary.” *Lifewire*, Lifewire, 8 Nov. 2017, www.lifewire.com/twitter-slang-and-key-terms-explained-2655399.

12) “What Is the Social License?” *The Social License To Operate*, socialicense.com/definition.html.

13) “Working With Text Data¶.” *Scikit*, scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html.