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 focusing 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. </p>

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 that 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. </p>

As of the current status of this report, we are prototyping using Scikit-Learn Classifiers. These Classifiers require minimal effort to set up with default hyperparameters. They train speedily and provide results in a timely manner, allowing us to adjust our hyperparameters on-the-fly to see if there are any

noticeable differences. It is simple to add additional Classifiers using the Pipeline class. All that is required is the addition of a new import statement for that Classifier and to replace the class 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. </p>

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 (Scikit-Learn documentation). We plan to utilize one or both of these hyperparameters tuning methods in order to expedite the search for optimal hyperparameters for all of the Scikit-Learn Classifiers we will use. As we add additional Classifiers to our codebase, it becomes time-saving to automate parameter tuning as much as possible. </p>

Once we have established which classifiers are most effective at topic classification, 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 overnight training utilizing Nvidia Geforce Titans on the Borg supercomputer to perform stance analysis training on the larger Twitter datasets consisting of 650k+ 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 assuming adequate scaling is possible. </p>

We implement metric visualizations via the use of the Scikit-plot library which is built on the matplotlib library and via 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 suitable ones 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 but not currently implemented. We will provide more details on our data visualizations in the results section. </p>

Implementation:

These sections will describe in detail (perhaps too much detail) our current implementation for SLO TBL topic classification in association with the current state of the codebase. We have decided to keep all debug output statements as “log.debug()” statements that can be shown or hidden by setting “log.basicConfig(level=log.DEBUG)” to the appropriate level and by setting the relevant debug variables ON/OFF. </p>

Our Tweet preprocessor program is separated into 3 individual functions that perform preprocessing specific to the datasets we are utilizing. The first is a Tweet dataset consisting of 229 labeled examples, the second is a Tweet dataset consisting of 31 labeled examples, and the third is a dataset consisting of 658983 unlabeled examples. </p>

The first Tweet dataset we are performing text pre-processing on is 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, respectively. </p>

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”. </p>

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. </p>

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. </p>

As we are incapable of using the Linux/Mac only CMU Tweet Tagger for tokenizing (due to working solely on a Windows OS workstation), our decision was to manually clean the raw Tweets using Python regular expressions and utilize Scikit-Learn to tokenize the Tweets via the CountVectorizer Class. The Natural Language Toolkit was considered as an alternative but ultimately we chose to just use built-in Python functions and Scikit-Learn. A for loop is used to send each Tweet to a preprocessing function that does the following: </p>

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: </p>

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

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. We save the

processed dataframe to a comma-delimited CSV file to be used in training our Scikit-Learn Classifiers. </p>

The second Tweet dataset we are performing text preprocessing on consists of 32 hand-labeled examples provided by Professor VanderLinden. We follow a similar path as above with our first dataset of 229 labeled examples. We noticed that there was a spelling error present in one of the examples where “environmental” was misspelled to “enviromental”, resulting in the erroneous creation of a 4th target label later on when we were training our Classifiers. This was corrected manually by editing the original CSV file before re-preprocessing and saving out to a comma-delimited CSV file. Of note, is that each example only possesses up to two different TBL classifications as opposed to up to three with the first dataset. The Tweet itself was in the same format as the other and thus we could trust that preprocessing, in the same manner, would yield similar processed data. </p>

The third Tweet dataset we are performing text preprocessing on consists of 658983 unlabeled examples with 11 columns of different data including Tweet ID#, language of the Tweet, whether it is a re-Tweet, associated hashtags, associated mining company, Tweet text with mentions, user screen name, user description, Tweet text without mentions (replaced with slo_mention), and Tweet author profile description. While this dataset has technically been previously preprocessed by earlier research on SLO stance classification, we noticed some discrepancies between these processed Tweets and ours. </p>

a) The Tweets still had “#” hashtags, whereas we replaced with slo_hashtag in ours.
b) The Tweets still had punctuation, whereas we removed them in ours.

Consequently, we decided to run the entire set through our custom preprocessor in order to normalize the Tweets to be consistent with the two other datasets. Python’s timer class records that it took approximately 11412.2 seconds to process the entire dataset of 658,983 Tweets. This was done

overnight and the results were again saved to a comma-delimited CSV file. We will make predictions using this dataset 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. The CMU Tweet Tagger was used to pre-process these Tweets. </p>

Please refer to “slo_tbl_preprocessor.py” for the codebase. </p>

Our “slo_topic_classification_v2-0.py” program implements Scikit-Learn Classifier training, prediction, and hyperparameter tuning via the Pipeline and GridSearchCV classes. We import our processed datasets, re-index and shuffle the data, and generate a Pandas dataframe for each. We then concatenate the individual datasets together into one cohesive dataframe and again re-index to ensure our range starts from 0. The total number of useable labeled examples is at 277, each with a single TBL topic classification of economic, environmental, or social. </p>

The input feature set was created using the “Tweet” column and a target label set created using the “SLO” column. We chose to refactor our code for this into a separate function so that we can run multiple iterations for training our Classifiers on randomized Tweet and target label test and training sets each iteration. 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 for each. </p>

With the training, test, and prediction sets properly prepared, we utilized Scikit-Learn’s Pipeline class to set up various Classifiers. Each Classifier is contained in its own module and provided log output is set to “debug” or lower, will display accuracy metrics and a classification report summary. The summary includes statistics on precision, recall, f1-score, as well as the micro, macro, and weighted averages for each. A for loop is used to generate metrics over N iterations and a mean accuracy metric is provided. The trained Classifier is also passed to our “make_predictions” method afterward which attempts topic

classification using the large unlabeled 658,983 Tweet processed dataset. The currently implemented Classifiers include: </p>

a) Multinomial Naïve Bayes' Classifier.
b) Stochastic Gradient Descent (SGD) Classifier.
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 many of the Classifiers capable of multi-class single-label topic classification in the Scikit-Learn toolkit. 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. </p>

For each Scikit-Learn Classifier Pipeline, we implement a CountVectorizer(), TfidfTransformer(), and the relevant Classifier Class(). The following 2 sections describe in some detail the reasons we utilize these three classes: </p>

~~The target label train and test sets were 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 use require and support only numerical data. (Note: this is done automatically by the Pipeline() class, I think)~~

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. Scikit-Learn 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 994 in comparison to 185 examples for the training set and 92 examples for the test set. </p>

The Scikit-Learn TfidfTransformer class was 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 (Scikit-Learn documentation). Normalization of this kind provides better input feature vectors for training our model. </p>

Each Classifier is also paired with a Grid Search function utilizing Scikit-Learn's GridSearchCV() class that provides automated parameter tuning. The grid search requires the setup of a classifier (which we did via Pipeline) and the specification of a dictionary storing all the keys (parameters) and values (parameter values) to tune with. The dictionary is passed as an argument to the GridSearchCV() class along with the Classifier Pipeline. We also passed along optional arguments specifying it should run in parallel using all available cores and perform 5-fold cross-validation splitting. This class provides an exhaustive search of all possibilities, meaning it tries all possible combinations of the parameters and associated values you provide it with. Hence, the time to find optimal parameters using our grid searches varied drastically from a few minutes to a few hours (Scikit-Learn documentation). </p>

As mentioned previously, we utilize a large 658,983 Tweet dataset to make predictions using each of our trained Classifiers. The prediction set, so to speak, is prepared in its own function. We drop all columns except the “tweet_t” column containing the processed Tweet to create an input feature dataframe. Our prediction function is then called by each Classifier’s module, passing in the Classifier itself. The Classifier makes predictions on all Tweets and we use counter variables to calculate what percentage of Tweets were classified as economic, social, or environmental among the entire dataset. </p>

Results:

Our data visualization is centered around the use of the Scikit-plot library that is built on top of matplotlib. This is a simplified and user-friendly plotting library that requires little setup to generate data visualizations. Unlike with standard matplotlib, there is no need for extensive manual coding to create even a simple plot. Scikit-plot is integrated with Scikit-Learn and includes function calls that automatically generate a variety of plots. It contains a Metrics, Estimators, Clusterer, and Decomposition Module. The Metrics module supports confusion matrices, ROC curves, KS Statistics, Precision-Recall curves, Silhouette analysis, Calibration plots, Cumulative Gains curves, and Lift curves. The Estimators module supports Learning curves and Feature importance. The Clusterer module supports Elbow curves. The Decomposition module supports PCA Component Explained Variances and PCA 2-D Projection.

For our visualizations, we use the Metrics module to depict confusion matrices, ROC curves, Precision-Recall curves. We use the Estimators module to depict Learning curves. These are implemented in a generalized function that accepts as parameters the Scikit-Learn Classifier, the Classifier’s predictions data structure, the Classifier’s predictions probabilities data structure, and the string name of the Classifier. We then make the appropriate function calls using Scikit-plot to visualize the above curves and plots. For the LinearSVC Classifier and KNeighbors Classifier, we created specialized functions to visualize the resulting data as they did not have the necessary data to support the 4 types of visualizations in the generalized function.

Grid Search was the essential component to obtaining the best possible results with our limited training and test sets consisting of a total of 277 labeled examples. With default hyperparameters and manual hyperparameter tuning, our accuracy metrics were generally abysmally low and inconsistent. The inconsistency was due in part to initially not running 1000 iterations and then taking a mean of the accuracy metric to find a consistent percentile. We were originally just training on one iteration for each Classifier using a randomized shuffling of the feature and target sets and splitting into training and test sets for each. Utilizing the suggested optimal parameters from exhaustive grid search, we were able to raise our accuracy metrics for each Classifier to around 50%. The lowest was the Multi-Layer Perceptron Classifier at 0.490, Stochastic Gradient Descent Classifier at 0.492, and Multinomial Bayes Classifier at 0.493, approximately. The highest was the Support Vector Classification Classifier at 0.535 and the Decision Tree Classifier at 0.532. The remainder fell somewhere in between. Time permitting, we may attempt to tweak these hyperparameters manually to see if we can improve our metrics further. </p>

Our prediction results for each Classifier indicates that they will not generalize well to new Tweets. At least, not the processed Tweets we are utilizing. “Social” was the favored classification for our trained models, with the Stochastic Gradient Descent Classifier predicting all the Tweets as social in nature (100%). On the flip side, the Decision Tree Classifier was the most balanced in identifying 40% as social, 54% as environmental, and 6% as economic. The rest of the trained models overwhelmingly predicted almost all Tweets as “social”. This seems to indicate that we are improperly utilizing machine learning methodologies in our codebase, almost all the Tweets are actually “social” in nature in that dataset, or we simply do not have enough relevant Twitter data in order to train decent models for TBL topic classification, let alone any deep neural networks. </p>

Please refer to showcase(demo).ipynb and showcase(demo2).ipynb for mean accuracies, metric summaries, data visualizations, and prediction results. Showcase(demo).ipynb is isolated to the essential code to run any single Classifier, in this case, the Multinomial Naïve Bayes Classifier. Refer to this Jupyter Notebook file for a condensed version of our implementation as our full implementation in slo_topic_classification_v2-0.py contains the combined code for all Classifiers. Showcase(demo2).ipynb contains all the entire codebase in slo_topic_classification_v2-0.py with all Classifier training and predictions enabled in the “main” driver function and outputs the same items as Showcase(Demo).ipynb, only for all the Classifiers we currently have implemented.

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. We plan to do further research into how to process our data in order to improve upon our machine learning models. Referencing the Calvin-CS/slo-classifiers GitHub Repository should assist in this as we currently have a limited understanding of the entire scope and implementation of the research project in its current state. Given time constraints, we have still have made decent headway in learning the basics of Pandas/Numpy dataframe manipulation and the Scikit-Learn API. </p>

Please refer to the README.md in the "Project" root directory for additional information on the contents of the GitHub repository project files. </p>

We are unable to directly discuss our research in comparison with similar works as it was an internal project at CSIRO and the individual in charge of that project has since left CSIRO. Professor VanderLinden is unsure whether he can retrieve the relevant materials from that earlier research conducted some years ago. We are however referencing the prior summer's stance classification research material in the Calvin-CS/slo-classifiers GitHub repository as inspiration for our own work. Specifically, we have perused the Calvin-CS/slo-classifiers/stance/data subdirectory and looked over the tweet_preprocessor.py, settings.py, and autocoding_processor.py files. The tweet_preprocessor.py file utilizes the CMU Tweet Tagger to tokenize Tweets and references the settings.py for Python regular expressions for Tweet preprocessing. In hindsight, we should have taken a closer look at the file structure for these two as our own could be cleaner and more efficient.

It is our plan to implement our own autocoding_processor.py file in the future in order to obtain a much larger Tweet dataset that we can procedurally label as one of the TBL topic classifications via some heuristic algorithm. As suggested by Professor VanderLinden, we will use the large 650k+ Tweet dataset used for SLO stance classification in the previous summer's research. We will locate the original Tweet

via the Tweet ID field in the CSV file, analyze the context of the Tweet in relation to what it is responding to, analyze the Tweet author's profile page, and attempt to find key phrases, hashtags, etc., that indicate a near certainty of a TBL classification of Environmental, Social, or Economic. By creating patterns to match against using those key phrases, hashtags, etc., we will create our own auto-coder capable of extracting Tweets that match our defined patterns. This will hopefully provide enough data in order to train a Keras deep neural network and obtain accuracy metrics in the 60%+.

Unfortunately, we opted not to attend Senior Project Presentations by senior students for CS-295 otherwise there would likely be content that is relevant as a comparison to our own work. We did, however, ask Professor Brenda VanderLinden for access to the video recordings on the SLO Topic Modeling and SLO Monitoring presentations by Derek Fisher, Roy Adams, and Brent Ritzema. Provided we have sufficient time to update this report, we will include some details concerning those presentations.

Implications:

Social License to Operate is a dynamic licensing that requires maintenance as the opinions of the local population and relevant stakeholders could change on a daily basis. Attempting to keep up-to-date on the opinions of a large group would require quite the effort if conducted via polls, surveys, questionnaires, or other more traditional methods. It could be more cost-effective and time-effective for an organization or other entity involved in large-scale business ventures to simply hire a group of researchers to utilize machine learning algorithms to construct a model that can predict the acceptability of their current business operations.

Of course, we are then substituting a mathematical model for the actual voice, beliefs, opinions, and personality of human beings. By plugging Tweets into a trained model, we obtain categorical data on the stance, sentiment, and topic classification of those Tweets. From there, we can extrapolate via

some heuristic algorithm and determine the level of acceptability of their current operations. Provided the model is relatively accurate, this should be a decent enough metric by which to make future decisions concerning the direction of the project.

But, is that preferable and desirable over actual interaction with the local population and your stakeholders? Our world is already digitalized to some extremes and even most forms of daily communication is via digital media, not so much face-to-face. Can we imagine a future hypothetical scenario in which we choose even our marriage partners by plugging in the relevant dataset and training on a machine learning model? Perhaps SLO research is but one step in this direction? It would certainly be easier to do that than to go through possible divorces, child custody lawsuits, and other sticky situations. Why blunder blindly when we can determine how likely our chances are of a successful relationship with a person by utilizing artificial intelligence? It could be a slippery slope as to how much of an impact machine learning and A.I. in general should have in our lives. Then again, Google Image Search utilizes machine learning and most of us can't imagine living without the Google Search Engine these days.

Placeholder – continue and revise discussions of implications.

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) “A Gentle Introduction to k-Fold Cross-Validation.” *Machine Learning Mastery*, 21 May 2018, machinelearningmastery.com/k-fold-cross-validation/.

4) “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.

5) “Introduction to Machine Learning | Machine Learning Crash Course | Google Developers.” *Google*, Google, developers.google.com/machine-learning/crash-course/ml-intro.

6) “How to Tune Algorithm Parameters with Scikit-Learn.” *Machine Learning Mastery*, 1 Nov. 2018, machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/.

7) Kenton, Will. “How Can There Be Three Bottom Lines?” *Investopedia*, Investopedia, 9 Apr. 2019, www.investopedia.com/terms/t/triple-bottom-line.asp.

8) 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.

9) 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.

10) “Multiclass Classification.” *Wikipedia*, Wikimedia Foundation, 18 Apr. 2019, en.wikipedia.org/wiki/Multiclass_classification.

11) “Symbolic Reasoning (Symbolic AI) and Machine Learning.” *Skymind*, skymind.ai/wiki/symbolic-reasoning.

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

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

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

15) “Welcome to Scikit-Plot's Documentation!¶.” *Scikit*, scikit-plot.readthedocs.io/en/stable/.

