Daniel Simpson

Malcolm Angelo de Villar

12/11/20

CSC-664-01

Prof.Singh

1.Introduction

* Our aim for this project was to create a set of procedures that could be used on a dataset of Yelp Reviews to accurately classify the sentiment of each review into 5 classes representing the 5 different star ratings available for users to pick when submitting a review on the Yelp website. In the process, we also created a basic sentiment analysis machine learning workflow, that could theoretically be repurposed for use with any labeled database of texts for solving a problem of similar scope or just for educational/demonstrative purposes.
* Being able to solve this seemingly simple but actually quite difficult problem would show that we have a good understanding of the most common and useful tools of sentiment analysis. If we are able to create a well structured sentiment analysis workflow, we could theoretically share it online to serve as a template or demonstration for other machine learning/data analysis beginners like ourselves
* This report will illustrate our ideas on how to solve this problem, review some important and relevant works in sentiment analysis that we referred to during this process, demonstrate the structure and logic of our program, and reveal our proposed best method as well as the reasoning behind it. Finally, we will share what we learned in this process, and reflect on what aspects we could have improved on as well as the shortcomings of our solution.

2.Proposed problem formulation

We will create a testing workflow for each of the most popular text preprocessing methods, feature extractors, and machine learning models for sentiment analysis. After creating that workflow, we will test combinations of preprocessing methods, feature extractors, and machine learning models, while relying on intuition and online research to determine the best combinations to test and refine.

Because of the difficulty of this problem, we may see the best results in pre-processing the data as minimally as possible, and attempting to give the machine learning model a large amount of data through higher counts of ngrams to work with.

Our preferred combination, determined through research and testing utilizes filtering of a custom and corpus specific stopword list, TFIDF Vectorization of unigrams, bigrams, and trigrams from the reviews, and fitting a model utilizing a class-weighted linear support vector with Stochastic Gradient Descent optimization to make our predictions.

3.High-Level description of our idea

Classifying text into 5 different classes of sentiment, is very difficult to do accurately, even more so when the 5 different classes consist of two varying intensities of negative sentiment, two varying intensities of positive sentiment, and one neutral class. For that reason and also because we are newcomers to sentiment analysis, we thought it may be wise to rely on past research to determine the best techniques for a multiclass sentiment analysis. In creating a workflow for testing, we would be able to test these accepted techniques on our own unique problem.

Our theory is that because the distinction in sentiment between the 5 classes would likely be very small, we should aim to give our machine learning model as much data as possible so that it may be able to see latent trends in the data that we would not be able to see. After all, even for many humans, it would be very difficult to distinguish between some unlabeled reviews that may have very similar intensities of sentiment or have very mixed sentiment.

4.Prior work in the area

As an introduction to the subjects of sentiment analysis, we referred to the work of Justin Martineau and Tim Finin in “Delta TFIDF: An Improved Feature Space for Sentiment Analysis”. The main purpose of this work was to show the efficacy of using Delta TFIDF in place of traditional TFIDF in sentiment classification applications. Unlike traditional TFIDF it considers the class of document each term occurs in,which “boosts the importance of words that are unevenly distributed between the positive and negative classes and discounts evenly distributed words”(2). We initially used Delta TFIDF in our own multiclass sentiment analysis as it proved to be more effective than other feature extraction methods, but realized the library which we were using was meant for binary classification. Therefore we abandoned that idea at a later point as we could not analyze/track/understand why it was more effective. This literature introduced us to the efficiency and performance of the support vector machine with a linear kernel for our testing. Finally, it introduced the idea that lemmatization and removing stop words could be harmful, rather than helpful, in text classification problems, citing “Text Categorization with Support Vector Machines. How to Represent Texts in Input Space?” by Edda Leopold and Jӧrg Kindermann.

Another introductory piece we referred to a great deal was the article titled “Classification of sentiment reviews using n-gram machine learning approach”of Abinash Tripathy, Ankit Agrawal, and Santanu Kumar Rath. Although this article’s implementation was also a binary classification, its results detailing the effects of using different n-gram methods on prediction performance was very helpful in determining that we would likely have the best performance either utilizing unigrams on their own, or a combination of unigrams, bigrams, and trigrams. Later tests showed that these n-gram counts were suitable to our application as well. Like the Martineau and Finin paper, this paper also introduced us to some classifier models like naive bayes, maximum entropy or logistic regression, and stochastic gradient descent.

After the initial phase of development and testing was done, these initial works proved less useful because they dealt with binary classification. First, the binary classification results were not comparable to results we could expect for our multiclass classification and they failed to address some of the key issues specific to problems of multiclass classification. “Multi-Class Sentiment Analysis on Twitter: Classification Performance and Challenges”, an article written by Mondher Bouazizi and Tomoaki Ohtsuki, of Keio University in Japan, delivered on both of these fronts. First of all, the classification problem presented in the article was an attempt to classify sentiment of tweets into 7 different classes. Although these classes were not linear progressions of the same sentiment like our problem, they still explained the challenges of multi-class classification that were relevant to our problem as well. These included closeness between different sentiments, which we struggled with in differentiating between 3 star and 4 star ratings in particular, context dependency, which we tried to address by decreasing the amount of words in the stopword list, and presence of multiple sentiments in one text, which is extremely common in any type of Yelp review, especially those of the more neutral classes. Although the article did not give us any direct solutions to these issues for us, it did give us a good idea of where performance losses could be coming from. Finally, it gave us a good performance target, which was about 60% accuracy.

5.Description of our work

Setup (Malcolm Angelo De Villar)

At the start of the phase, our team had no idea where to start and what tools we needed to utilize. By figuring out how to start the idea through finding out what other people use, we were able to have a foreground of what to do and what tools best suit our needs. Moreover, we were able to experiment more and find out our own way of approaching the problem.

Data Collection from Yelp Website (Daniel Simpson)

(Note: Although the data collected by our code was not utilized for the final version of the project and it no longer seems to work with the current version of the Yelp website, we will still include this code and the collected data set, as it was part of the process of creating the project.)

First a Pandas data frame is initialized. Then we use the requests library from python, to make a get request for the search results link for restaurants in a chosen city. We then used xpath expressions to retrieve the individual links to each restaurant's review pages and create a list of those links in a loop. In another loop a get request is made for each restaurant link in that list, and an xpath expression is used to retrieve all the review containers on the first page and create a list of those review containers. Finally in a nested loop, two xpath expressions are used to retrieve the text content of the review in the review container as well as the star rating in the review container. 20 reviews are taken from each restaurant’s page and the number of pages iterated through in the main search page can be modified as desired.

Preprocessing of CSV Files (Daniel Simpson)

The CSV file of raw data is read into a Jupyter Notebook, where contractions within each text were expanded and exported into another CSV file as the runtime is quite long. A separate copy of unaltered data is also kept. We created multiple preprocessing functions that can be run on the two sets of data. These consist of removing punctuation, lemmatizing the words in the review, reducing repeat characters used for emphasis, removing stopwords, and part of speech tagging (forming strings of “word/part of speech” from words).

Classification of Data (Daniel Simpson)

Copies of the data applied with different preprocessing techniques ready to undergo the classification steps. The set of reviews with no preprocessing are analyzed for their 100 most common words, which was used as a reference while experimenting with different stopword lists. Various different models like Linear SVC, Logistic Regression, Naive Bayes, and Stochastic Gradient as well the TFIDF and countvectorizer are initialized in one block of code. Parameters for each can be set in the block above, including max\_df and min\_df to ignore very common or very rare n-grams, class\_weight to assign weight to each class, and setting the underlying formula/algorithms for each classifier. The RandomUnderSampler and SMOTE(Synthetic Minority Oversampling technique) oversampler are also initialized. The undersampler and oversampler as well as the models that could be initialized with a random\_state seed are all initialized with a random\_state of 23 to create reproducible results, allowing us to test the effectiveness of various preprocessing techniques as well as parameters of feature extractors, machine learning models, and sampling methods.

Another block of code was used to initialize a pipeline containing one of those vectorizers and models. That pipeline was passed into our cross validation function where data was split into 5 sets of training data and test data. The StratifiedKFold class was used to split the data while preserving the amount of samples in each class. The vectorizer processes the review texts and star ratings in each subsample of data, and the model passed in is fit with that data. If the undersampler or oversampler is passed in, the vectorized data is over or under sampled before fitting to the model. A prediction is made for each of the 5 folds of test data, and the prediction statistics for each fold as well as the overall macro averages across the 5 folds is printed out.

Creating a workflow that could be used to test multiple classifiers, vectorizers and preprocessing techniques allowed us to verify the efficacy of techniques presented in the various articles we read like n-grams or reveal that certain techniques would not work well for our problem, in the case of the proposal of the use of the Random Forest Classifier in “Multi-Class Sentiment Analysis on Twitter: Classification Performance and Challenges”. It also allowed us to possibly present this as a supplemental learning tool for beginners in data science, as it allows the demonstration of some of the core principles/techniques of sentiment analysis. Although obviously it would have to be paired with independent research as well, as it isn’t intended to explain how and why each component works.

Best results were obtained with these steps taken:

* Contractions in word expanded
  + Allowed contractions that may contain stop words in contracted form to be removed or not removed in case of important words like ‘not’
* Removed words in stopword list: “in”, “of”, “at”,”a” “the”, “and”, plus removing ngrams present in at least 65% of documents
  + At first, this seemed counter intuitive to our intention of giving large amount of data, but this value of 65% is still relatively conservative based on my research
* TFIDF Vectorization of unigrams, bigrams, and trigrams
  + Determined to be best balance of giving enough data to model, but avoiding adding more noise
    - Considering grams longer than trigrams started decreasing performance
* Applying weighted vector to Linear SVM model with SGD(Stochastic Gradient Descent) optimization
  + Stochastic gradient descent: Efficient way to estimate slope and intercept for best fit line/hyperplane suitable for larger datasets
    - Also gave performance boost in our case, may also be down to underlying implementation of Linear SVM
* Setting class\_weight parameter of TFIDF vectorizer to ‘balanced’
  + Automatically assigns weight values to classes (star ratings in this case) inversely proportional to their frequency
  + Penalizes mistakes/rewards correctness in classification of classes according to that weight
  + Necessary to balance bias of data set towards higher ratings
* Early stopping enabled for SGD Classifier
  + Precaution to stop overfitting of training data set

All coding was done in a python notebook through Google Colab, a cloud based python notebook editor and execution environment that allows real time collaboration.

6.Experimental evaluations

**Comparison of Preprocessing Techniques**

All of these tests were conducted using the preferred set of parameters/methods/techniques listed above

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comparison of Preprocessing Techniques on Unaltered Data Set | | | | | | | | |
|  | Unaltered | Nopunct | Manual Stopwords | Trimmed | Lemmatized | Max\_df = 0.65 | | Tagged |
| ACC. | 55.23 ± 0.63% | 55.16 ± 0.74% | 55.21 ± 1.00% | 55.09 ± 0.75% | 54.95 ± 0.82% | 55.34 ± 0.77% | | 52.97 ± 1.43% |
| PREC | 51.35 ± 1.21% | 51.11 ± 1.37% | 51.07 ± 1.06% | 50.88 ± 1.26% | 50.93 ± 0.93% | 51.39 ± 1.38% | | 48.28 ± 1.35% |
| REC. | 51.66 ± 0.99% | 47.89 ± 0.79 % | 51.59 ± 0.42% | 51.17 ± 1.06% | 51.20 ± 0.71% | 51.78 ± 1.21% | | 50.07 ± 1.33% |
| F1 | 51.32 ± 0.93% | 45.19 ± 0.58 % | 51.16 ± 0.73% | 50.89 ± 1.10% | 50.88 ± 0.76% | 51.43 ± 1.20% | | 48.76 ± 1.47% |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Comparison of Preprocessing Techniques on Data Set with Expanded Contractions | | | | | | | |
|  | Expanded | Nopunct | Manual Stopword List | Trimmed | Lemmatized | Max\_df = .65 | Tagged |
| ACCURACY | 54.93 ± 0.66% | 55.01 ± 0.69% | 55.41 ± 0.90% | 55.00 ± 0.70% | 55.56 ± 1.20% | 55.25 ± 0.81% | 53.09 ± 1.07% |
| PRECISION | 50.97 ± 0.79% | 51.13 ± 0.91% | 51.40 ± 0.78% | 51.07 ± 0.85% | 51.81 ± 1.11% | 51.36 ± 0.85% | 48.78 ± 1.05% |
| RECALL | 51.49 ± 0.96% | 51.61 ± 1.09% | 52.03 ± 0.92% | 51.60 ± 1.16% | 52.18 ± 1.36% | 51.85 ± 0.92% | 50.47 ± 1.83% |
| F1 SCORE | 51.08 ± 0.83% | 51.17 ± 0.90% | 51.52 ± 0.88% | 51.11 ± 0.89% | 51.76 ± 1.18% | 51.43 ± 0.91% | 49.01 ± 1.28% |

Observations: Overall, preprocessing of the data did not have a large effect on performance at all, but if it did, it would drop performance slightly. Tagging words with parts of speech was an interesting technique that is used at times in sentiment analysis, but did not prove effective at all for our data set and dropped performance drastically. Also lemmatization and stopwording hurt performance when used together despite helping performance when used individually. The only techniques that we included were trimming n grams present in 65% of documents and removing stopwords.

**Comparison of Balancing Techniques**

Counts of each class of review for reference:

1 Star Ratings : 749

2 Star Ratings : 927

3 Star Ratings : 1461

4 Star Ratings : 3526

5 Star Ratings : 3337

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metrics of Balancing Techniques with Preferred Methods/Techniques Used (p.5-6) | | | | |
|  | Near Miss | SMOTE | No Balancing Techniques | Class\_weight= ‘balanced’ |
| ACCURACY | 51.35 ± 0.27% | 55.71 ± 0.81% | 54.95 ± 0.63% | 55.49 ± 0.59% |
| PRECISION | 46.99 ± 0.32% | 51.96 ± 1.30% | 53.98 ± 1.13% | 51.61 ± 0.55% |
| RECALL | 50.38 ± 0.79% | 50.96 ± 0.94% | 46.55 ± 0.86% | 52.09 ± 0.66% |
| F1 SCORE | 48.04 ± 0.25% | 51.02 ± 1.02% | 48.27 ± 0.76% | 51.64 ± 0.53% |

Comparison of Confusion Matrices

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No Balancing Techniques Applied | | | | |
| 54.1% | 13.2% | 6.3 % | 12.8 % | 13.6 % |
| 17.6% | 21.6 % | 25.6 % | 27.4 % | 7.9 % |
| 2.8 % | 5.8 % | 25.5 % | 56.9 % | 9.0 % |
| 0.7% | 0.4% | 4.4 % | 66.3 % | 28.2 % |
| 0.5% | 0.2% | 0.8% | 33.2 % | 65.3 % |

Observations: Effective at predicting 1 star, 4 star, and 5 star reviews. Unacceptable performance for 2 and 3 star classification likely as a result of training data having much more samples of 4 star ratings, and 5 star ratings. In general, skew of classification is pointed towards the right side, especially when compared to our chosen method of class balancing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SMOTE | | | | |
| 60.6 % | 19.6 % | 5.2 % | 4.7 % | 9.9 % |
| 23.0 % | 35.0 % | 22.7 % | 13.2 % | 6.3 % |
| 4.9 % | 12.7 % | 33.3 % | 39.3 % | 9.9 % |
| 1.5 % | 2.5 % | 7.3 % | 55.4 % | 33.3 % |
| 1.5 % | 0.6% | 2.0 % | 25.4% | 70.5 % |

Observations: 2 star classification good, 3 star classification much improved over baseline. However 4 star classification performance suffers as a result, and 3 star classification performance pales in comparison to the next two balancing techniques. Model generally still classifies ratings given for the reviews as higher than they actually are.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Undersampled | | | | |
| 62.2 % | 24.8 % | 5.1 % | 3.9 % | 4.0 % |
| 22.4% | 38.7% | 22.7 % | 11.0 % | 5.2 % |
| 6.8 % | 16.7 % | 43.0% | 23.5 % | 10.0 % |
| 3.9 % | 4.5 % | 16.1 % | 41.5 % | 34.0 % |
| 3.9 % | 2.2 % | 5.6 % | 21.8 % | 66.5 % |

Observations: Overall, using undersampling produced the most balanced prediction distribution. As you can see in the 2 star ratings and 3 star ratings, wrong predictions are nearly equally distributed to the rating class on either side. Because the number of TFIDF vectors of minority classes now match the majority classes, positive skew is greatly diminished. The overall metrics are poorer than our other techniques though.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Balanced Class Weight | | | | |
| 61.4 % | 19.1 % | 7.7 % | 4.4 % | 7.3 % |
| 20.8 % | 34.6 % | 30.1 % | 9.6 % | 4.9 % |
| 4.9 % | 12.0 % | 43.5 % | 32.2 % | 7.3 % |
| 1.8 % | 2.8 % | 12.3 % | 52.2 % | 30.9 % |
| 1.8 % | 1.0 % | 3.5 % | 25.0 % | 68.7 % |

Observations: Effective at predicting 1 star, 4 star, and 5 star reviews. Acceptable performance for 2 star and 3 star ratings. Slight skew of predictions towards the right side(if we look at the 3 star rating classification), but predictions are generally distributed in the way that one would logically expect. We chose this technique because even though undersampling gave more balanced prediction distribution, the difference is not drastic enough to make up for its poorer overall performance. Also, when the prediction of our model is wrong, it doesn’t specifically matter what class the review was misclassified as, if it is still wrong. For other applications or problems this may not be the case if a prediction one way or the other could have larger ramifications. For example, misclassifying a disease as more severe or less severe, in that case you may want the model to err on the side of caution and skew towards more severe. Finally, this should be an adequately representative set of any set of Yelp reviews because of its size and randomness, so the slight positive skew should not be an issue.

7.Conclusions

We learned that performance metrics do not tell the whole story in terms of performance. If we had simply gone with the steps that initially produced higher numerical statistics, our model would have been more accurate for classification of some classes, but would have had unacceptable performance for 3-star classification. Looking at the confusion matrices helped us determine which methods were helping prediction performance for every class reach a level of acceptable effectiveness. Having our program predict 3 star reviews as 4 star reviews much more often than making the correct prediction, was less desirable to us than having a program with slightly lesser prediction performance overall but having more balanced predictive ability, and that was the outcome of using the undersampler. Although we fortunately found the best performance and relatively balanced classification by using the class\_weight parameter of the classifier.

We also learned that there is no cure all for an imbalanced data set. When adding synthetic samples to our data set with SMOTE, the bias towards 4 star and 5 star reviews, although improved slightly, still produced unfavorable results for us as it would consistently incorrectly classify 3 star reviews as 4 star reviews. Using the class\_weight parameter of the classifier model improved results in that regard, but then the classifier would exhibit the same behavior with 2 star reviews being misclassified as 3 star reviews. Undersampling performed better as it gave relatively balanced performance, although it did show a similar but less severe behavior in misclassifying 4 star reviews as 5 star reviews. No method of balancing the data set is perfect, and even balancing the classes perfectly may not produce balanced performance, especially in our case as deciding between a 2 star and 3 star, or 4 star and 5 star rating may be a relatively arbitrary decision for human beings. But the machine learning model will detect these latent trends (through matching sentiment with other reviews through empirical measures) and may produce unexpectedly unbalanced output as a result.

Adding more preprocessing techniques did not improve performance and the only preprocessing that caused an appreciable difference in performance was removing stopwords and very frequent ngrams. This is consistent with most of the articles we read dealing with these kinds of classification problems. The general idea that I gathered is that for difficult problems like this one, it may yield better results to leave the data alone as much as possible. This is especially true in regards to things like lemmatization which have a performance cost, although this was not as vital of a factor for our application. We decided to follow the law of Occam’s Razor, and did not further pursue or complicate processing techniques that did not have a positive impact on performance in their simplest form. We did test various combinations of processing techniques that intuitively seemed like they would complement each other, and although we did see some better results with those combinations, none beat out the minimal approach we used for best performance. Minimal processing of data also helps us avoid overfitting in combination with the 5 fold cross validation and early stopping.

Although we did see good results with the Delta TFIDF vectorizer, we decided against using it for our report, because we wanted to avoid using a method that we could not verify the functionality of. If we had more time, we may have been able to implement a way for us to utilize the Delta TFIDF Vectorizer for a multiclass problem, and verify that it was functioning as intended. We also could have fully investigated/troubleshooted why relatively innocuous preprocessing techniques like removing punctuation or trimming repeat characters did not help performance at all, in our testing.

We regret that we could not gather our own data set with adequate size, as the premade datasets we found on the internet were already cleaned of special characters. It would have been interesting to see how much of a difference that initial cleaning would make to performance. Also it would have been interesting to see how a custom stoplist of menu items would have affected performance, although that would have required gathering reviews of a limited number of restaurants, and that would require using our own code to do so. We also tried to test that theory by using the min\_df parameter to remove unique n grams, which food names would definitely have been, and saw a performance drop although that did not discriminate towards food words specifically. However, the dataset of random restaurants should be a better sample to train our model on regardless, because it should have more consistent performance on reviews from any restaurant as a result of that randomness.

One more thing I have to mention is that although our best result came with the use of the data set in which the contractions are expanded, that process is very expensive as it took at least a couple of minutes to run through the 10,000 review data set, and would likely be impractical for larger data sets. The performance gain was probably not worth it, although maybe with further tuning it could prove more useful. I would have stuck with the results from the normal data set but I had already manually calculated some metrics and did not have time to redo them unfortunately. Also the result with only lemmatization on the expanded data set is very slightly better than the result with our proposed best method, but again I realized this late, as I was redoing some testing, and did not have time to compute and input the metrics for it.

Finally I wish that my teammates had done any work at all on the project, as I (Daniel Simpson) did all the research and development, and that is why I was the only one who did the midterm and final presentations as well. Em only contributed to the initial powerpoint presentation, and Malcolm as well. Malcolm also wrote the “Setup” part under “Description of our Work” in this document which is why his name is included on this report, but all that actually entailed was him sending a link to our group chat to a Yelp rating prediction experiment he had found. Other than that, this whole report was written by me. I imagine our project could have been so much better if they had contributed.

8.References:

Martineau, Justin, and Tim Finin. "Delta tfidf: An improved feature space for sentiment analysis." *UMBC Student Collection* (2009).

Bouazizi, Mondher, and Tomoaki Ohtsuki. "A pattern-based approach for multi-class sentiment analysis in Twitter." *IEEE Access* 5 (2017): 20617-20639.

Tripathy, Abinash, Ankit Agrawal, and Santanu Kumar Rath. "Classification of sentiment reviews using n-gram machine learning approach." *Expert Systems with Applications* 57 (2016): 117-126.

Yelp Dataset Downloaded From: https://www.kaggle.com/omkarsabnis/yelp-reviews-dataset