SHIVA KUMAR KANNEBOINA

MLND - Capstone Project Report

Title: Detecting Online Fake Reviews (Opinion Spam)

Date: 25/11/2017

Definition

Introduction:

Almost everyone of us in this modern era depend on the online reviews before purchasing any product or booking any service via online. These online reviews provide valuable information about the products, services to consumers. They can promote their business or harm the brand of a product or service by writing fake reviews. In order to damage the reputation of the competing organization, some companies promote the entities by providing false negative opinions. When deciding on whether to go for that product/service, these reviews/opinions plays a huge role and act as major source of information on the overall product, service or the organization which is providing that product/service. This kind of illegal activities that try to mislead the reader or consumer by giving undeserving positive opinions or giving the false negative reviews to damage the reputation is also called as 'opinion spam'. Opinion spam has many forms: Ex., Fake/Bogus Reviews, Fake Comments, Fake Blogs, Fake Social Media Posts, Deceptions, Deceptive Messages, etc..

Due to drastic change in the technology in online market recent years, some companies are paying to people to write fake reviews on their product or service so that their reputation in the market will be high. Buying/selling fake reviews has become a profitable business and a major threat at the same time. This problem of detecting fake reviews was first addressed by a group of researchers at Cornell University, these guys have designed an algorithm by analysing the language used in legitimate and phony write-ups, their work details are published here. Here in this problem I am trying to design a similar model with my approach to detect fake reviews on hotels.

Problem Statement:

The main goal of this project is to find out if a particular review is fake or not by effectively using the Classification techniques available in Machine Learning space(such as RandomForestClassifier with GridSearchCV to start with, SVM, NaiveBayes Classifier, etc..) and find the best algorithm that accurately predicts by effectively training on large datasets available here. The algorithm is expected to

learn from the large dataset and predict the future reviews. The major part of the problem is to find out a best algorithm that acts well on this dataset.

Metrics:

The performance of the classifier is measured using the evaluation metric 'accuracy' which says that number of examples classified correctly over the total number of examples.

$$accuracy = \#correctly\ predicted/\#total\ test\ sets$$

Who will calculate this evaluation(i.e. accuracy) metric for various classification algorithms like RandomForestClassifier, SVM, NaiveBayes, etc.. and find the best performing model for predictions.

Analysis

Data Exploration:

Dataset/Corpus I want to use in this project can be downloaded from here. This corpus consists of truthful and deceptive hotel reviews of 20 Chicago hotels.

This corpus contains:

- ➔ 400 truthful positive reviews from TripAdvisor (described in [1])
- ➔ 400 deceptive positive reviews from Mechanical Turk (described in [1])
- ➔ 400 truthful negative reviews from Expedia, Hotels.com, Orbitz, Priceline, TripAdvisor and Yelp (described in [2])
- ➔ 400 deceptive negative reviews from Mechanical Turk (described in [2])

Each of the above datasets consist of 20 reviews for each of the 20 most popular Chicago hotels. The files are named according to the following conventions:

- Directories prefixed with fold correspond to a single fold from the cross-validation experiments reported in [1] and [2].
- Files are named according to the format %c_%h_%i.txt, where:
    - **%c** denotes the class: truthful or deceptive
    - **%h** denotes the hotel: [Ex: affinia: Affinia Chicago (now MileNorth, A Chicago Hotel)]
    - **%i** serves as a counter to make the filename unique

Benchmark:

In a test on 800 reviews of Chicago hotels, Cornell researchers have developed a computer software that's pretty good at detecting deceptive reviews with almost ~90% accuracy. This will be the aspirational target for this problem that I want to achieve.

Solution Statement:

To solve this problem I will use range of classification algorithms available in the Machine Learning(such as Random Forest classifier with GridSearchCV to start with, SVM, NaiveBayes Classifier, etc..) and compare the metrics like accuracy_score for each algorithm and pick a model that gives best results and use that algorithm for prediction. The solution is expected to use some NLP techniques for data pre-processing.

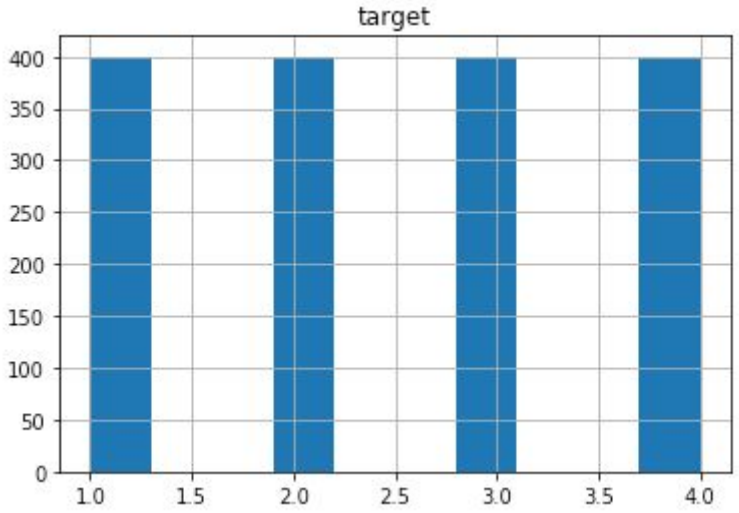
Methodology/Approach:

Data Preprocessing:

The first step before we use the data that's downloaded is to understand and pre-process it to eliminate any noise if exists and prepare it in a way that's accepted by classifier. In the pre-processing phase the below steps are performed:

- Once you've downloaded the corpus/data, extract the compressed file and navigate to the root folder 'op_spam_v1.4'. This directory contains two subfolders namely 'nagative_polarity', 'positive_polarity'. Recursively traverse to all the directories inside ''nagative_polarity'' and move all '.txt' files into 'negative_polarity'. And do the same for 'positive_polarioty' directory. Now that all the positive reviews moved into one folder and negative reviews moved into another folder. You may use the below linux/unix based commands for doing this.
    - cd **negative_polarity**
    - find . -type f -print0 | xargs -0 -I file mv file .
    - rm -rf truthful_from_Web
    - rm -rf deceptive_from_MTurk
    - cd **positive_polarity**
    - find . -type f -print0 | xargs -0 -I file mv file .
    - rm -rf deceptive_from_MTurk
    - rm -rf truthful_from_TripAdvisor

- Next step is to create the dataframes for holding required data with fields named target_class, actual_class, review. Create two separate dataframes each for holding positive reviews and negative reviews. Python method named ‘pre_processor’ will do this job of creating dataframes.
- Introduce a new variable called ‘target’ with the value calculated using logic explained in code and assign it to dataframe. This variable is of our interest to know which class the review belongs to.
- The below is the histogram of the data distributed based on the field ‘target’.



- This doesn’t complete the preprocessing of data, we have to do some more processing(Like tokenization, vectorization, etc..) to eliminate the noise as much as possible so that our algorithms gives us more accurate results.
- **Tokenization**:
    - **Word Tokenization:** This step extracts the tokens from the data using python method extract_tokens.
    - It converts each of the review into lowercase and append each character of the reviews into a list.
    - **Lemmatization:** And then we tag parts of speech(POSTag) of each word in the list and lemmatize the words. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.
    - **Stop Words:** Here in the tokenization step, we will eliminate the list of predefined words from stop_words corpus if any exists in our reviews.
- **Semantics:**

- To deal with the semantics of each review, I've used the gensim library to create a bag of words and to make the corpus more condensed.
- Created a new column 'tokenized_review' and then filtered the words which have occurred in less than 2 reviews and have occurred more than 0.8 time(no_below=2, no_above=0.8)
- Then vectorizing all reviews, we created the bag of words model.

Implementation & Results:

Test/Train Data Split:

I used the cross_validation technique provided by scikit-learn's model_selection module and splitted it into test and training data. Set the parameter test_size to 0.25 which is the default value.

Implementation of Models & Model Selection:

Implementation of the Solution is nothing but trying out and analysing various classification algorithms and select one algorithm which is performing better on our dataset. I've used the three classification algorithms available in SK-Learn's library. They are:

➔ RandomForestClassifier with GridSearchCV
➔ SVM(Support Vector Machine) Classifier
➔ Naive Bayes Classifier

The observation and results of using each of the above mentioned algorithms is recorded and discussed in below sections. Pros and Cons of each algorithm also discussed here in following pages of this report.

Random Forest Classifier:

Based on my research on the various classifiers, I found out that random forest model is a good choice to start building our model. Random Forest is just like decision trees, very little. It is desirable to pick Random Forest when pre-processing of the data needs to be performed. The data doesn't need to be normalized and the approach is resilient to outliers if any. If we have many input variables/features, Random Forest Classifier automatically targets the most useful feature/variables. This is achieved by the randomness introduced by the random forest model builder in the dataset selection and in the variable selection delivers considerable robustness to noise, outliers, and overfitting, when compared to a single tree classifier. The model builder doesn't tend to overfit to training data, since there are many trees built and hence there are two levels of randomness. And each tree is independent model. Apart from the above mentioned advantages, random forest

classifier has also got some cons. It is not easy to visually interpret the random forest model. Parameters like tree depth and number of trees should be set although the default parameters may work fine.

In our problem we trained it with the pre-processed data and the random forest classifier and it gave us the accuracy score about ~70%. This is not even close to what I want to achieve. It still needs to be improved.

Accuracy:

- Accuracy of RandomForestClassifier with GridSearchCV on training data is :0.6925
- Accuracy of RandomForestClassifier with GridSearchCV on test data is : 0.69

SVM Classifier:

Support Vector Machine is a supervised learning algorithm which can be used both for regression and classification. In most of the cases it is used for solving classification problems. It handles the high dimensional data very well. This algorithm outputs an hyperplane that separates the data into two classes. It handles high dimensional data well. And performs well with nonlinear boundary depending on kernel used. But, it is susceptible to issues like training issues, sometimes overfitting depending on the kernel being used. There are different types of kernels namely: linear, polinomial, rbf, and sigmoid.

I trained our SVC classifier and gave the pre-processed data as the input. ‘rbf’ kernel gave us the best results among others. It gave us the better results than what we achieved in Random Forest Classifier. This gives us some confidence that we are going in correct direction to solve the problem. But, this is also not close to what we want to achieve.

Accuracy:

- Accuracy of SVM on Training sets :0.745
- Accuracy of SVM on test sets is : 0.7625

Naive Bayes Classifier:

Naive bayes classification algorithm is supervised learning algorithm used in solving classification problems. It is based on the Baye’s probability theorem. It is majorly used in text-classification problems where the training data is highly dimensional computation becoming slower. This model is known for it’s effectiveness and fastness in computations. Various applications of Naive Bayes’s classifier include text classification, Spam Filtering, Sentiment Analysis, Recommender systems, etc.. Naive Bayes algorithm involves simply counting the number of objects with specific

classes or features. But, it majorly relies on independent assumption and might not perform good if that assumption is not met.

I trained this model with our pre-processed data and observed that it gave the better results than both Random Forest Classifier and SVM Classifier. Its accuracy score is around ~83% which is some what closer what we want to achieve.

Accuracy:

- ('Accuracy score: ', '0.825')

Conclusion

To conclude the summary of my solution, I observed that Naive Bayes Classifier gave me ~83% of accuracy. Whereas Random Forest Classifier and SVM Classifier gave ~71% and ~77% of accuracies respectively. Out of three classifiers I tried out, Naive Bayes classifier gave me considerably better results with accuracy of ~83%. So, I pick this model as the final model to be used for my Opinion Spam problem. Though the benchmark model has ~90% of accuracy, it could be achieved in our case as well provided more training to the model and thereby improving the overall performance of the model.

Softwares/Packages used:

- Python, Jupyter for python notebook, anaconda
- Pandas: For creating dataframes and data manipulations.
- OS: As we need to read text files and read reviews from it and work with those reviews.
- NLTK: For POS-Tagging, Tokenization, lemmatization, preventing stop_words, and other processing.
- GENSIM: For implementing bag of words, creating corpus, to convert the corpus to sparse form
- SKLEARN: For GridSearchCV, SVC, RandomForestClassifier, train_test_split

References/Research Links:

1. M. Ott, Y. Choi, C. Cardie, and J.T. Hancock. 2011. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies.

2. M. Ott, C. Cardie, and J.T. Hancock. 2013. Negative Deceptive Opinion Spam. In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.
3. http://myleott.com/op_spam/
4. http://news.cornell.edu/stories/2011/07/cornell-computers-spot-opinion-spam-online-reviews
5. http://www.ijerd.com/paper/vol12-issue4/Version-1/A1240108.pdf
6. http://www.vladsandulescu.com/opinion-spam-detection-literature-review/
7. http://courses.washington.edu/ling575/SPR2014/slides/OpinionSpam.pdf
8. https://arxiv.org/pdf/1107.4557.pdf
9. http://scikit-learn.org
10. http://blog.hackerearth.com/introduction-naive-bayes-algorithm-codes-python-r
11. https://machinelearningmastery.com/support-vector-machines-for-machine-learning/
12. http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/