TCSS 555 Part B Documentation

Ken Gil Romero

Mike Cresswell

Yicun Zeng

Kevin Flora

1. **Data Analysis**

The data represents hotel reviews in English text from 20 Chicago hotels. The text entry features are Positive or Negative, Deceptive or Truthful, the hotel name, and the text review. Each entry is in a .txt file separated by folders. There are 2 folders positive and negative. Each of them have 2 folders which are deceptive and truthful. Inside them folders separated by 10 folds. Inside these folders is the .txt files. There are 1600 .txt files where 400 truthful positive, 400 deceptive positive, 400 truthful negative, and 400 deceptive negative. The data is balanced and unskewed because half are positive reviews and the other half is negative and half are deceptive reviews and the other half is truthful.

1. **Visualization**

Visualization is comparing statistical numbers from raw csv files and finding potential direction for feature selection and engineering. Also the visualization is used for comparison of processing file, model training and evaluation. To give a distinct comparison for the result we get.

1. **Data Cleaning**

The purpose of cleaning is to organize and streamline the data so the machine learning algorithms can develop the most efficient model. Cleaning is especially important to text classification. Noisey text can inhibit the accuracy of the model. The first step of our data cleaning process was to consolidate the data. In the submitted code, there is a file called DataCleaining.py. In the file there is a function called readTranscripts(). This function loops through all the directories and files in the downloaded zip file data from the problem website. The function reads categorical data from the directory and files names. For each text file a record is created with the text and corresponding categorical data. The output of the function is a pandas dataframe containing a unique identifier, deceptive, hotel, polarity, source and the raw text.

The next step was to clean the raw text data. In the PreProcessing.ipynb file there are a number of functions that perform cleaning for raw text data. Together these functions comprise the steps of our different data cleaning algorithms. The first step is to tokenize the text. Tokenization is the process of splitting the raw text into individual words or strings. This operation is performed in the Tokenize() function. Tokenize() splits the words on whitespace. The function takes a string and returns a list of strings. The next function call is stripWords(). This function uses regular expressions to replace any added whitespace or special characters and converts the token to lower case. The next function is removeSpecialCharacterTokens(). This function loops through every token in the list, if the token is a special character, it is removed. The last step of the cleaning process is removing stop words. Stop words are words that appear frequently and only make training models less efficient. To remove stop words we wrote a function called removeStopWords(). The function takes a list of tokens and iterates through it, removing any token that is equal to a word in the list of stop words. The stop word list is imported from the nltk.coprus library.

There were a couple other data cleaning functions that were experimented with but ultimately didn’t become part of the cleaning algorithms. The first function was checkWords(). This function validated the tokens against the nltk.corpus.words() library, and removed any token that didn’t appear in the list. Ultimately there were too many words not found in the dictionary. The second function was removeNumber(), which removed any numeric values.

The functions that remained in the cleaning algorithms were nested steps within the parents functions processTextWithLemmaization(), processTextWithStemming() and processText(). These three methods were written to output different versions of the cleaned data. One that isn’t lemmatized or stemmed, a lemmatized version and a stemmed version. This gave us some added flexibility when experimenting with model training. This leads us to our next step, pre-processing the data.

1. **Pre-Processing**

Data pre-processing is performed after the data has been organized into a logical structure and cleaned. Pre-processing is also designed to optimize the classification accuracy of our models by manipulating the text to undercover patterns. As well as convert the features into data that is consumable by our models. In our processTextWithStemming() function, we use the process of stemming. To stem a word is to reduce it to its root form by removing any suffixes. For example, the stemmed version of lately is late. This helps our models uncover patterns within the text by revealing common roots among the words. The processTextWithLemmaziation() function uses the process of lemmatization. Lemmatization is similar to stemming, it reduces the word to its root from lemmatization is more accurate than stemming because it performs a full morphological analysis on the text. Stemming could transform the word careful into car. Lemmatization prevents this error.

The final step of pre-processing is transforming our features into data that our models can use. For our Support Vector Machine and Logistic Regression models we must transform our text into vectors and encode our categorical text features. To encode our categorical text features we used a technique called one hot encoding. One hot encoding is the process of converting integer encoded categorical text features into a binary value for each unique integer value. We performed one hot encoding on the hotel, source and polarity features. We performed integer encoding on the class label deceptiveness.

To transform our text data into vectors we used the TfidfVectorizer() library. Tf-idf stands for term frequency inverse document frequency. Tf-idf vectorization generates a number representation of a token by comparing the number of times the token appears in a document with the number of documents the words appear in. Together these techniques comprise our pre-processing steps.

1. **Feature Engineering & Selection**

Feature engineering is the process of extracting new features from the existing data. With the hotel review dataset we were able to extract a number of additional features from the raw text data. The goal of feature engineering is to increase the accuracy of our machine learning models by providing them with additional information. In the FeatureEngineeringandSelection.ipynb file, we use a number or different operations to extract meaning metrics for each hotel review. Each line of the code adds a new feature to the dataset. We add a variety of different metrics based on the text for each review. The metrics include word count, character count, sentence count, average sentence count, number of punctuation marks used, etc.

Feature selection is the process of selecting the optimum subset of features to use for training the machine learning models. Feature selection is important because irrelevant features can have a negative impact on the resulting model. They can make the model more complicated to interpret, increase the training time and result in overfitting. We implemented two different methods to help us determine the optimum subset of features. The possible subset of features can be an engineered feature or features extracted from the files and directories. The first method we utilized was SelectKBest() from the sklearn feature selection library. The SelectKBest() object uses a chi-squared statistical test as the scoring function. The output of the function is shown in the co-lab notebook Feature\_EngineeringSelection.ipynb. It’s important to note that the most important feature by a wide margin was source. The second feature selection method we used was the ExtraTreesClassifier() model from sklearn’s ensemble library. Decision tree models have built in feature selection capabilities and these libraries have feature selection analyzation functionality built in. By training the ExtractTreesClassifier() on all the data and features, we can measure their importance. The output is shown in the file. The source feature’s dominance is also extreme using this method as well.

1. **Survey Potential Models**
   1. **KNN**

KNN needs training data, testing data, the k, and a distance metric for it to work. We found that there is no best way to find k,so we will be using some type of trial and error. We are going to run through all k to find the best accuracy. For the distance metric, we found that Euclidean distance is one of the best ways in text mining, so that is what we chose. [13.1] KNN will output the majority of all classes of the k nearest neighbors based on what the distance metric is.

KNN is not too complex to use. It does not have a training period, since all calculations will be when predicting the classes of the test data. [13.3] Its disadvantages are we do not have an easy way to find the optimal k-value for best accuracy. [13.2] Also, when the data is too big, using KNN will be too slow because we need to compute the distance for all the training data. [13.4]

To use KNN, we separate every word count of the preprocessed data, specifically the hotel review part. This will be our new preprocessed data we will use. We will be using the deceptive as each entry’s class. Then we need to select k which is 1 then split the data. Then we can finally use KNN. Each of the testing data’s classes will be the majority of all the training data classes of the 5-nearest neighbors. For better accuracy, we will try to change the distance metric and k by trial-and-error.

Why this will work is that each entry will have one of the 2 classes which are truthful, and deceptive. Using our KNN, a new hotel review will be on those 2 classes. Therefore if its class is in the truthful then it is not spam otherwise it is a spam.

* 1. **Decision Tree**

The definition of a decision tree is a decision support tool that uses a dendritic graph or model of decisions and their possible consequences. We chose a tree-based classifier because of its simple properties, the explicit meaning, and easy transformation to ‘‘if–then’’ rules[13.15].

Decision tree needs appropriate depth for the tree to avoid overfitting and underfitting. For this spam review task, we use frequency of unique words as features to distinguish different reviews. To find the optimal depth of tree k, we need to try different feature sets and different k presets.

The data setting is almost as KNN since each word column is either 0 or 1 indicating that word is shown in column or not, so the “if-else” branch in the tree can be built.

* 1. **SVM**

Support Vector Machines is a supervised learning method. The objective of a support vector machine is to identify a hyperplane in a N-dimensional space that will classify data points into distinct groups. Many hyperplanes can be chosen to separate the data. The SVM will attempt to determine the plane with the maximum distance between data points. This helps ensure minimization of bias when classifying future data points. A support vector is a data point that is closest to the decision hyperplane. These are the data points most difficult to classify. They have a direct impact on the optimal location of the decision hyperplane.

One reason SVMs are so popular is their memory efficiency. Even with high dimensional data, SVMs can remain efficient because they can represent the similarity of the data in a lower dimensional space. Rather than applying the transforms, minimizing computations performed [13.11]. This is referred to as the kernel trick. There are a number of different kernel functions, including polynomial and radial basis function. Another benefit of support vector machines is that it can be used for both classification and regression.

There are a couple of drawbacks when implementing support vector machines. Due to the implementation of the algorithm, SVMs can become inefficient on large datasets. Another drawback is the lack of insight into the classification model. There is no probabilistic explanation for classification. This can also make it difficult to fine tune the model’s hyperparameters.

* 1. **Logistic Regression(Kevin)**

The logistic regression (LOGR) model “(or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick.” [13.16] That is to say that the “the goal of [LOGR] is to train from the probability of variable Y being 0 or 1 given x.” [13.16]

Just like a linear regression (LINR) model, a LOGR model computes a weighted sum of the input features (plus a bias term), but instead of outputting the result directly like the [LINR] model does, it outputs the logistic of this result.” [13.9]

In the design of our spam detection system, one of the models we employed is the LOGR learning algorithm with a supervised learning design. More specifically, our configuration is designed to execute, as mentioned, as a supervised learning model because it learns on labeled spam email data with the goal of solving a multiple regression problem by using multiple data features to make a prediction. Further, our model configuration will execute a univariate regression since our spam detection system needs only predict a single value for each distinct email, i.e. spam or ham.

Lastly, our LOGR model will use a batch learning type because there will not be a continuous flow of data entering the system and so there will be no need to adjust or change data rapidly. [13.16] An advantage of this configuration is that we can front load labeled data and train the algorithm before deployment and produce a more accurate result.

1. **Train-Test Split**

From sklearn.model\_selection we import train\_test\_split to split our training data and our testing data. The test size would be 0.3. For KNN and Decision trees, we will be using our preprocessed data word counts.

1. **Experiment settings**
   1. **Train data, test data**

We will be splitting our data by 70-30 where 70% will be in training and 30% will be in testing. This means 1120 are in our training and 480 are in our testing. We will also be using 10-fold cross validation. So, 9 groups of 160 are for training and 1 group of 160 is for testing.

* 1. **Metrics**

To calculate the accuracy of the 4 models, from sklearn.metrics we imported accuracy\_score when we used the 30-70 split data and from sklearn.model\_selection we imported cross\_val\_score for the k-fold cross validation using 10 folds then used its mean.

* 1. **Train time (Visualization of running from end to end, Yicun)**
     1. **KNN**

KNN doesn’t have a training time. [13.3] All calculations are in the predicting phase. For 1 entry of the testing data, the average time was 1.17 seconds from 100 executions when using the Euclidean distance with k as 5. However, 1 of the distance metric that yielded a better accuracy which is the Bray Curtis Distance had the average time of 0.88 seconds from 100 executions with k as 15.

* + 1. **Decision Tree**

Decision tree training time is also around 1 second for 100 executions since the training set is relatively small.

* + 1. **SVM**

The training time for our Support Vector Machine classification model was 1.5 milliseconds. The model used the tf-idf vectorized text and the source feature to train. This model was using a linear kernel. Experiments with different features and kernels yielded different results. Using a linear kernel and classifying only the tf-idf vectorized text, the training time took 1.45 seconds. Using the radial basis kernel function to classify the tf-idf vectorized text and nine engineered features from the text, yielded a training time of 483 milliseconds.

* + 1. **Logistic Regression (Kelvin)**

The LOGR model was trained with and without the presence of the source feature Mechanical Turk (MTurk) category. The MTurk category includes 400 deceptive negative reviews and enables the LOGR learning algorithm to more rapidly identify deceptive reviews in the data set. And as a result, the LOGR algorithm can more rapidly identify truthful reviews by the absence of the MTurk source feature. The dominance of the weight of the MTurk source feature reduces training time which can be seen in the results.

The LOGR classification learning model training times:

1. Without MTurk Training Time = 0.56386344 Seconds
2. With MTurk Training Time = 0.00554361 Seconds
   1. **Hardware used for training**

We will be using Google Colab to train all the models.

1. **Evaluation results on train and test set**
   1. **KNN**

KNN was 53.54% accurate for when splitting the data by 30-70 where 30 is the testing data, and it was 56.62% for 10-cross fold validation for using the Euclidean distance. For using the Bray Curtis distance it got 78.13% when splitting the data by 30-70 where 30 is the testing data, and it was 74.06% for 10-cross fold validation.

* 1. **Decision Trees**

Decision tree got accuracy around 68% for 70-30 train test split and an average 55% accuracy for 10-folds cross validation.

* 1. **Support Vector Machine**

By selecting the source and tf-idf vectorized text features and using a linear kernel, our Support Vector Machine was able to classify both the test and training data with 100% accuracy. The cross validation average, with a K equal to 10, was also 100% accurate. When selecting only on the tf-idf vectorized text data, the model classified the test data with 87.9% accuracy, the training data with 96.8% accuracy and yielded a cross validation average of 86.7%.

* 1. **Logistic Regression (Accuracy scores)**

The LOGR model, as previously mentioned, was trained with and without the presence of the source feature Mechanical Turk (MTurk) category. The MTurk category includes 400 deceptive negative reviews and enables the LOGR learning algorithm to more rapidly identify deceptive reviews in the data set.

According to Cardie and Hancock, “The rising influence of user-generated online reviews (Cone, 2011) has led to growing incentive for businesses to solicit and manufacture DECEPTIVE OPINION SPAM—fictitious reviews that have been deliberately written to sound authentic and deceive the reader. […] Based on this dataset, we find that standard n-gram text categorization techniques can detect negative deceptive opinion spam with performance far surpassing that of human judges.” [13.17]

The prediction accuracy results of our LOGR algorithm appear to positively correlate with presence of the source feature MTurk category, aligning with the findings of Cardie and Hancock.

Without MTurk:

Test Accuracy = 70.83333333333334

Training Accuracy = 70.0

Cross Validation Mean = 69.75

With MTurk:

Average Accuracy = 100.0

Average Training Time = 100.0

Cross Validation Mean = 1.0

1. **Comparison of machine learning models**

Based on Section 9, SVM have the highest accuracy score of 86.7

|  |  |  |
| --- | --- | --- |
| Model | Train test split accuracy | 10 folds cv accuracy |
| KNN | 78.13% | 74.06% |
| Decision Tree | 68.125% | 55% |
| SVM(support vector machine) | 87.9% | 86.7% |
| Logistic Regression | 70.0% | 69.75% |

1. **Findings and conclusion**

The support vector machine model was able to classify the dataset with 100% accuracy. This was due to the importance of the source feature. The source, which was parsed from the directory name of the data file folder, will determine whether or not the hotel review is spam. Every review from the Mechanical Turk is deceptive. The truthful reviews are divided up between TripAdvisor, Expedia, Orbitz etc. The support vector machine is able to create a decision hyperplane using a linear kernel that classifies all of the reviews from Mechanical Turk as deceptive, and reviews from all other sources as positive. However with future data, we don’t necessarily want source to be the only determining factor. Therefore we also implemented support vector machines and classified the spam using only the text data. We were curious how well the model would do without the source feature, and only using the processed text. A support vector machine using a linear model and a TF-IDF vectorizer optimized for our dataset, we were able to classify the test data with 87.9% accuracy, the training data with 96.8% accuracy.

KNN was not as good as SVM with an approximately 30% difference, so at the way we used KNN, it is not the best to use it for classification. Since KNN used the preprocessed word count as its dataset, there may be words that should have carried more weight for better classification accuracy. For example, words that are used often for deceptive reviews or non-deceptive should have more weight. So, to do that, we should update the preprocessed word count data set by multiplying the word count of the words that are used often for deceptive reviews or non-deceptive with some number X. X will be how often the word was used. X should be adjusted by trial-and-error to find a better classification accuracy.

Decision-tree performed similarly like KNN with 10 folds cross validation accuracy around 55% percent when it used the euclidean distance metric, the feature selection is based on the distinct words between truthful review and deceptive review, so when data is limited, the difference between truthful and deceptive reviews may not be large enough to indicate the result. That can explain when we use train test split and train data is relatively large(70% of data), accuracy got improved to 68%.

Logistic regression performed marginally better on average Train test split accuracy than KNN and only marginally better than Decision Tree, and worse than SVM. And the same general pattern follows with the 10 folds cv accuracy. SVM appears to have performed the best for this SPAM classification learning problem. And the underlying reason why SVM performed better may have to do with the nature of SPAM classification. SPAM classification may not be an entirely linear classification problem. SPAM detection is a multiple univariate regression, and in our case, batch supervised learning problem. The multiple features are labeled by humans, and as such will not represent an entirely linear classification system. LOGR appears to be more suited to problems with potentially better signal to noise ratios and more completely linear or consistent feature selection labeling. While LOGR may be more able to identify the significance of data set features better than Decision Tree, it appears SVM is more able to handle nonlinear problems and outliers, as may be indicated by our results.

1. **Individual Contribution**
   1. Ken Romero: Data Analysis, preprocessing the preprocessed data for KNN and Decision Tree, KNN(Survey, train time, code), Decision Trees (Survey), Train-Test Split, Train data, test data, Metrics, Evaluation results on train and test set, Hardware used for training data
   2. Mike Cresswell: feature engineering, feature selection, data cleaning, data pre-preprocessing, support vector machines (survey, model/code, tf-idf tuning, model kernel tuning, train time, evaluation, comparison and findings), logistic regression (model, code)
   3. Yicun Zeng: Visualization(Data analysis, Result evaluation), Decision Tree(model training, evaluation, comparison and finding)
   4. Kevin Flora: Analysis and comparison of models, Logistic Regression, Feature Engineering and Selection
2. **References**
   1. Balasubramanian, Valarmathi, Srinivasa Gupta Nagarajan, and Palanisamy Veerappagoundar. "Mahalanobis distance-the ultimate measure for sentiment analysis." *Int. Arab J. Inf. Technol.* 13.2 (2016): 252-257.
   2. Band, Amey. “How to Find the Optimal Value of K in KNN?” *Medium*, Towards Data Science, 23 May 2020, towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb.
   3. Kumar, Naresh. *Advantages and Disadvantages of KNN Algorithm in Machine Learning*, theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html.
   4. -, Genesis, et al. “Pros and Cons of K-Nearest Neighbors.” *From The GENESIS*, 25 Sept. 2018, www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/.
   5. *Stemming and lemmatization*. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. [Accessed: 17-Nov-2020].
   6. U. Verma, “Text Preprocessing for NLP (Natural Language Processing),Beginners to master,” *Medium*, 26-Feb-2020. [Online]. Available:https://medium.com/analytics-vidhya/text-preprocessing-for-nlp-natural-language-processing-beginners-to-master-fd82dfecf95. [Accessed: 17-Nov-2020].
   7. A. Akashkumar17, “Classifying data using Support Vector Machines(SVMs) in R,” *GeeksforGeeks*, 28-Aug-2018. [Online]. Available: https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/. [Accessed: 17-Nov-2020].
   8. S. Developers, “1.4. Support Vector Machines¶,” *scikit*, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed: 17-Nov-2020].
   9. A. Geron, *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. Sebastopol, CA: O'Reilly, 2017.
   10. S. Patel, “Chapter 2 : SVM (Support Vector Machine) - Theory,” *Medium*, 04-May-2017. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. [Accessed: 13-Dec-2020].
   11. G. Zhang, “What is the kernel trick? Why is it important?,” *Medium*, 11-Nov-2018. [Online]. Available: https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d. [Accessed: 13-Dec-2020].
   12. A. Wrg, “How I improved my text classification model with feature engineering,” *Medium*, 04-Nov-2020. [Online]. Available: https://towardsdatascience.com/how-i-improved-my-text-classification-model-with-feature-engineering-98fbe6c13ef3. [Accessed: 13-Dec-2020].
   13. R. Shaikh, “Feature Selection Techniques in Machine Learning with Python,” *Medium*, 28-Oct-2018. [Online]. Available: https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e. [Accessed: 13-Dec-2020].
   14. M. D. Pietro, “Text Analysis & Feature Engineering with NLP,” *Medium*, 26-Nov-2020. [Online]. Available: https://towardsdatascience.com/text-analysis-feature-engineering-with-nlp-502d6ea9225d. [Accessed: 13-Dec-2020].
   15. Yudong Zhang, Shuihua Wang, Preetha Phillips, and Genlin Ji. 2014. Binary PSO with mutation operator for feature selection using decision tree applied to spam detection. <i>Know.-Based Syst.</i> 64, 1 (July 2014), 22–31. DOI:<https://doi.org/10.1016/j.knosys.2014.03.015>
   16. Wikipedia contributors. (2020, November 5). Logistic regression. In *Wikipedia, The Free Encyclopedia*. Retrieved 19:23, November 18, 2020, from<https://en.wikipedia.org/w/index.php?title=Logistic_regression&oldid=987269378>
   17. 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.