Classification of Interview Excerpts from National Public Radio

Sethu K Boopathy Jegathambal, Howard Huang, Jonathan Campbell
(Kaggle Groups: J Campbell, Sethu, howardh. Group with final results: J Campbell.)

October 22, 2015

1 Introduction

Every day, vast amounts of text articles are being published online. As a result, it is becoming increasingly difficult to find texts that are relevant to a person's interests. There are also myriad commercial applications for automated text classification. For instance, classifying news articles according to the content from an online news website or suggesting a news article for a particular user. Text classification also establishes its domain in audio repository classification and video classification based on text transcripts and subtitles of videos respectively [6].

This project will make an attempt at classifying excerpts from NPR interviews into one of four categories: author, music, movies, or interview. We will be exploring three techniques for feature extraction (bag of words, TF-IDF, and mutual information), and testing the resulting data on three different classifiers: Naive Bayes network, decision trees, and support vector machines.

2 Related Work

Text classification is a popular topic in the field of machine learning and is used in many different applications. Work has been done to classify text using many algorithms, including Support Vector Machines [5], Naive Bayes [7], and Decision Trees [10]. Apart from algorithm choice, feature design and selection, as well as data pre-processing, are important factors to consider in any classification problem, with much research going into this area [4].

3 Data pre-processing methods

Each interview excerpt is parsed to shape the text into a format more suitable for feature selection. Text capitalization and some punctuation is removed, including end of sentence markers, dashes, and slashes.

More involved approaches to data pre-processing were also used. Such approaches included removal of all punctuation, removal of stop words, and stemming of words. Stop words were removed using the stop word list provided by the Natural Language Toolkit (NLTK) [2]. Stop words, defined as the most common words in a language (e.g. 'and', 'the', 'or', etc. for English), are used for sentence structure but do not convey any significance in a document [9]. For stemming, we used the stemmer provided by the aforementioned NLTK library [2]. Stemming a word gives its root, removing most suffixes (e.g. plurals or verb endings), which allows for more accurate counts of words in a document [7].

4 Feature design / selection methods

Several different attempts were made at choosing features. We looked at the bag of words algorithm, tf-idf values, and mutual information.

One choice of features was to use the bag of words values. The bag of words algorithm provides a simple count integer whose value is the amount of a specific word in a specific interview excerpt, over all words in the document corpus. We used the scikit-learn Python machine learning library's CountVectorizer method [8] to find the count for every word, then selected the 3000 most common words and turned them into binary features to be passed to our classification algorithms.

We also looked into using term frequency-inverse document frequency (tf-idf) weights for the classification. The tf-idf value signifies the importance of a particular word to a document corpus. The value will increase based on the frequency it appears in a document, and decrease by the frequency it appears in the corpus [9].

Basic idea of tf-idf is shown below

$$\mathrm{tf}(t,d) = 0.5 + 0.5\frac{f(t,d)}{\max\{f(t,d)\}} \tag{1}$$

Where t denotes a word, d denotes a document

$$\text{idf} = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right) \tag{2}$$

Where t denotes a word, d denotes a document, D denotes the corpus and N denotes total number of documents in corpus

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, d) \tag{3}$$

Our third feature selection was based on mutual information, which is a measure of how well a single feature performs in predicting the output class. Each possible word in the corpus is ranked based on their mutual information measure, and the top k were taken as features. In the text classification case, the equation for computing mutual information for a given term t and class c can be written out as follows:

$$I(t, c) = \sum_{i,j \in \{0,1\}} \frac{N_{ij}}{N} \log_2 \frac{NN_{ij}}{N_{i.}N_{.j}} \tag{4}$$

where N_{ij} is a count of documents whose properties are described by its indices i and j. $i = 0$ if the documents counted in N_{ij} contain the term t, and $i = 1$ otherwise. $j = 1$ means N_{ij} counts documents in class c, and $j = 0$ counts documents not in class c. [7]

5 Algorithm selection

5.1 Naive Bayes

The Naive Bayes algorithm has several advantages despite its simplicity. Naive Bayes has been widely used in the field of machine learning particularly for text classification. Naive Bayes is a generative model which makes the assumption that the features are independent (i.e. assumes the X_i are conditionally independent given C).

Conditional probability can be estimated by:

$$P(X|C_o) = \prod P(X_i|C_o) \tag{5}$$

where C_o is the event that the data is in class o, X is the data, and $X_i \in X$. From Bayes' Rule, we know that

$$P(C_o|X) = \frac{P(X|C_o)P(C_o)}{P(X)} \tag{6}$$

From equations 5 and 6, we get the conditional probability of a target class for a particular feature vector. This is shown below,

$$P(C_o|X) = P(C_o) \prod \frac{P(X_i|C_o)}{P(X)} \tag{7}$$

Further expanding P(X),

$$P(X) = P(C_o) \prod P(X_i|C_o) + P(C'_o) \prod P(X_i|C'_o) \tag{8}$$

Where, $X = \{X_1, X_2, \ldots, X_n\}$, n is the number of features, C_o is particular target class.

For classification of classes we used multinomial naive bayes with Laplacian smoothing, We estimated $P(X_i = v|C = O)$, $P(X_i = v|C = O')$, $P(C = O)$ and $P(C = O')$ from the training data. This method implemented for training is called is called One Vs. All or One Vs. Rest Strategy [3]. For target class prediction, these values are substituted in equation 7.

$$\text{Predicted class} = \arg\max_c \{P(C = c|X)\} \tag{9}$$

5.2 Decision trees

The selection of the decision tree algorithm came mainly from our choice of features. The use of binary features meant that training a decision tree is simple, and allows for more flexibility in our model (lower bias) than SVMs, without running into issues with high dimensionality as is the case with K-nearest neighbour.

We implemented the decision tree algorithm in Python using a 'build-down' approach, starting with the root node and recursively creating the sub-trees. Entropy and information gain were implemented using the regular formulas. To make sure the algorithm was working properly, we created several test cases with small datasets.

5.3 Support Vector Machines (SVMs)

Support vector machines, currently an attractive area of interest in the machine learning community [7], were chosen for our third algorithm because they provide several qualities which are useful in the field of text classification. Support vector machines are ideal for use with large feature spaces and sparse document instances, and require little tuning [5]. They also have a short running time, which is convenient for testing and tweaking purposes. Further, SVMs have been shown to outperform other algorithms including Naive Bayes in several text classification scenarios [1].

We used the implementation of SVM from the scikit-learn library for our analysis [8] (in particular, the stochastic gradient descent classifier with hinge loss, or SGDClassifier).

6 Optimization

One of the methods for preventing overfitting with decision trees is to build the tree all the way down to the roots, then prune until the error on the validation set ceases to decrease. This approach was not taken here because computation time was too long. Instead, the tree was simply limited to a depth of 5 (an arbitrary choice). Performance on the test set would likely be improved if we could grow the tree out some more.

7 Parameter selection method

The SVM algorithm uses an automatically decreasing learning rate, so no tuning was needed. We do use a rate of 0.005 as the multiplicative rate for the regularization term. This value controls the degree to which the penalty (regularizer) affects the minimization function. The value of 0.005 was found to give the least error through conducting tests evaluating accuracy of the algorithm on the validation set for values in the range of 0.01 to 0.0001. For the regularizer itself, we use the default L2 (squared euclidean norm). There is no manual adjustment of learning rate since the stochastic gradient descent implementation reduces it automatically.

The number of features (words) to use for mutual information was set to 1000, after tests were performed that showed it to give the lowest error compared with smaller values. Higher values gave a very small decrease in error, with much larger computation cost.

8 Testing and Validation

For initial testing, we split the Kaggle training datafile into a training and validation set (80%/20%) and randomize the order of the instances. For all tests, we use the same seed for the random number generation, to ensure accurate comparisons between algorithms.

In general, we found that the decision tree performed the best overall, followed by SVM and Naive Bayes. As seen in Figure 1, Decision Trees using the bag of words as features performed better than any other algorithm and feature combination.

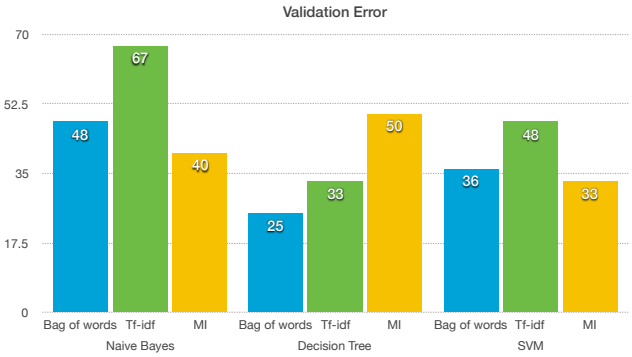

Figure 1: Validation error of each classifier and feature set pair.

With regards to feature selection, bag of words was best, followed by mutual information, and TF-IDF metrics, as again seen in Figure 1.

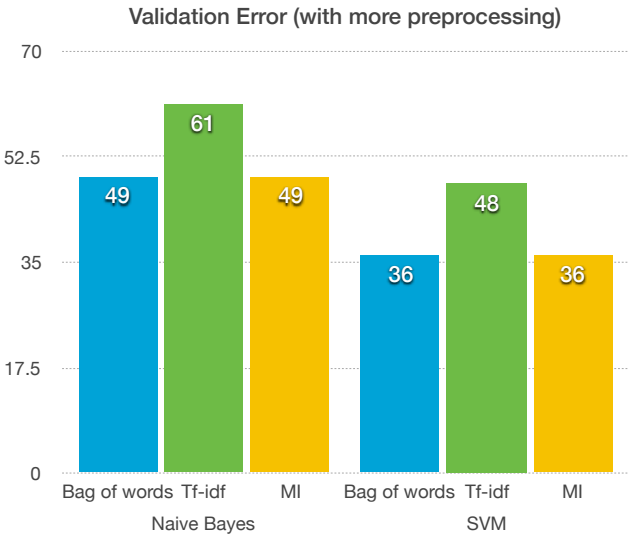

Figure 2: Validation error of each classifier and feature set pair when using more data pre-processing.

Figure 2 shows validation error when more preprocessing has been done on the data (including removal of punctuation and stop words, and stemming of remaining words). Results are similar to Figure 1 except for mutual information, which shows slightly higher error.

Figure 3 shows the final algorithm accuracy on the test set obtained through Kaggle submission. Both Naive Bayes and SVM algorithms were tested using mutual information as features. SVM performed better than the former, bearing out the results demonstrated in the validation errors (Figure 1).

Figure 5 shows the performance of the Naive Bayes classifier on the mutual information dataset with a

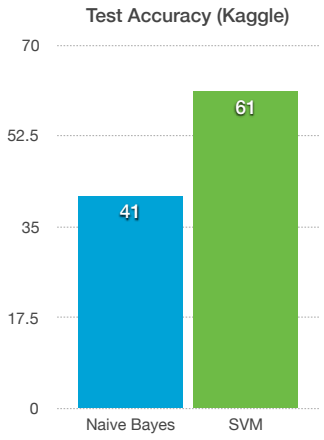

Figure 3: Accuracy on test set (using mutual information) on Kaggle submission.

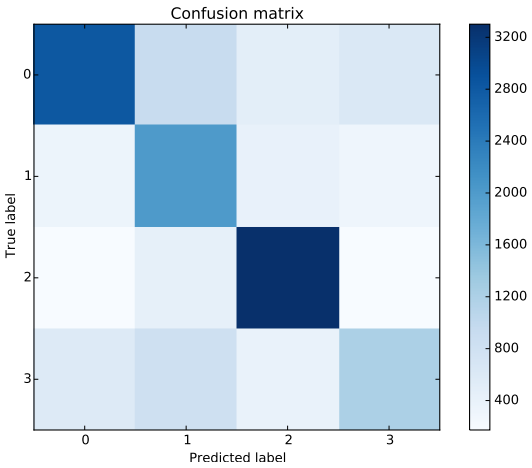

Figure 4: Confusion matrix of Naive Bayes classifier on validation data

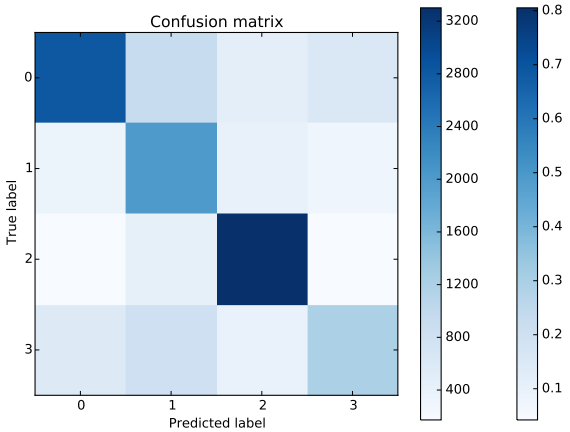

Figure 5: Normalised confusion matrix of Naive Bayes classifier on validation data

validation data split of 20%.

Naive Bayes is easy to implement and it can be used as a preliminary algorithm for estimating the performance of the dataset.

9 Discussion

TF-IDF, being a measure of the importance of a certain word, may not be as useful for this task as it has a tendency to overfit. For example, in a conversation about a specific musician, the name of that artist may come up several times, but never (or rarely) in other conversations about music. The presence of this feature can tell us with high certainty that the conversation is in the music category, but it would not generalize well to unseen data. If we instead treated all conversations of one category as a document, TF-IDF could potentially yield results similar to mutual information.

Mutual information is measured independently of all other variables. As a result, it is possible to obtain a set of highly correlated features, which would incur a computation time and memory cost, but provide little to no benefit in terms of performance. For example, here is a subset of the highest scoring words selected as features through the use of MI: music, song, sing, band, play. On their own, they are very good indicators that we are looking at a piece about music, but combined, they provide little extra information. This could be rectified if mutual information was used in conjunction with something like Pearson's chi-squared test to get rid of (or perhaps combine) highly correlated features. In theory, mutual information should also be able to remove stop words without domain knowledge (e.g. through the use of a stop-words dictionary), as these words should be present in most documents independently of topic. However, stop words were still found in the features selected this way (e.g. "are", "they", "have", and "not" are all part of the top 50 in mutual information). Looking at the data, these stop words seem to appear less in short dialogues. We also find the word "hi" with a high mutual information for the "author" category. The presence of these small words could be attributed to randomness, causing a certain stop word to appear more often in one class of documents over another. Conversely, it may also be that a certain demographic is more likely to engage in conversations without the use of "and", or that another is more likely to greet each other with "hi". However, our analysis of the data seems to point to the former.

It is possible that the SVM algorithm performed worse than decision trees because of the choice to

use a linear kernel, making the assumption that the text dataset is linearly separable. This assumption may not be true, and it is possible that such a bias could negatively affect the performance of the SVM. However, in most cases, text classification problems have been shown to be linearly separable [5].

One mistake we made in the gathering of results was to include the validation data when extracting features from the dataset. Therefore, the validation error is inaccurate, since it has influenced the feature set (not the actual algorithm training, however). However, test set results, as shown in Figure 3, have no such problem and can be taken as accurate representations of the algorithm accuracy.

10 Statement of Originality

We hereby state that all the work presented in this report is that of the authors.

11 Statement of Contributions

Sethu - Code for stemming, bit of feature extraction and Naive Bayes code. Also contributed in writing the report.

Howard - Contributed in writing code for decision tree, feature extraction and Mutual information. Also contributed in writing the report.

Jonathan - Wrote the data pre-processing for bag of words and TF-IDF. Wrote part of the decision tree algorithm. Wrote the relevant sections in the report. Conducted tests for SVM algorithm and some Naive Bayes tests using the default scikit-learn library.

References

[1] A. Basu, C. Watters, and M. Shepherd. "Support Vector Machines for Text Categorization". In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 4 - Volume 4*. HICSS '03. IEEE Computer Society, 2003, pp. 103.3–.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] P. Jackson and I. Moulinier. *Natural Language Processing for Online Applications: Text retrieval, extraction and categorization. Second revised edition*. Natural Language Processing. John Benjamins Publishing Company, 2007.

[5] Thorsten Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features". English. In: *Machine Learning: ECML-98*. Ed. by Claire Nédellec and Céline Rouveirol. Vol. 1398. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, pp. 137–142.

[6] Polyxeni Katsiouli, Vassileios Tsetsos, and Stathes Hadjiefthymiades. "Semantic Video Classification Based on Subtitles and Domain Terminologies." In: *KAMC*. 2007.

[7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Anand Rajaraman and Jeffrey David Ullman. "Data Mining". In: *Mining of Massive Datasets*. Cambridge Books Online. Cambridge University Press, 2011, pp. 1–17.

[10] H. F. Witschel. "Using decision trees and text mining techniques for extending taxonomies". In: *Proceedings of the Workshop on Learning and Extending Lexical Ontologies by using Machine Learning (OntoML)*. 2005.