Humor Detection

Andreas Lentini

1. Abstract:

Humor Detection can be used for a numerous number of text-based systems so developing an automated detection system is a good way to strengthen NLP systems. Objective: develop a humour detection system to determine humourous text in a dataset. Methods: Splitting up a dataset into training and testing data to test engineered features with supervised learning classifiers (K-Nearest Neighbors, Decision Trees, Naïve Bayes Methods, and SGD) to determine the strength of the features to develop strong humour detection. Results: The highest resulting test classifier was a KNN classifier with n=7 neighbors which returned a 0.73 (73%) confidence value with 13 features with individual confidence scores ranging from 0.41-0.62 which all performed above the minimum goal for individual confidence scores of 0.40.

2. Introduction:

Natural Language Processing (NLP) allows for the automatic evaluation of text in computer processing in modern technologies. NLP allows us to explore how a computer could be able to recognize and detect humour within text. Automated humour detection improves the value and strength of virtual assistants, chatbots and other automated text-based response systems. These systems should be able to differentiate jokes from facts to properly provide answers to whatever queries that have been made. This paper will explore key details in the “ColBERT: Using BERT Sentence Embedding for Computational Humor” dataset to explore automation of humour detection through supervised feature engineering.

3. Methods:

All feature construction and classification are completed using python 3.8 [10]

3.1.1 Data:

This study is using the “ColBERT: Using BERT Sentence Embedding for Computational Humor” dataset which is comprised of two hundred thousand short texts (100k Humor and 100k Non humorous) optimized for training purposes of a Humor Detection Machine Learning algorithm. [9]

3.1.2 Data Processing:

The dataset was already well organized and appropriate for use in the training, so no additional pre-processing was required for this model.

3.2 Feature Construction

3.2.1 Common Text Features

This dataset strongly emphasizes feature engineering because the data is completely void of readily available features that can strengthen training models. The first feature examined for this dataset was the sentence structure itself. Most jokes set up questions as short answers or singular punchlines. For example, What do you call a <word?>… <blank!>. (What does the dentist of the year get? a little plaque) Where numerous examples of <set up> <question mark> <pay off> are found, the amount of punctuation such as <?> and<!> helps to define the data in the dataset as humourous text. Also, certain phrases define a build up toward humour. Phrases such as “What do you get…” or “Why does…” are common setup questions for humourous texts. The next feature looked at common question starters such as “who, what, when, where, why, how, do, you” since those words often are used in tandem to start questions in joke formats. Finally, this report studied common text and punctuation features which occurred frequently in the jokes but not enough to be standalone features. The ellipsis, semi-colons, and colons were compiled into one feature. Figure 6.3 shows the breakdown of these searches.

3.2.2 Sentence structure

The breakdown of sentence structure itself was examined. Features of data composition were examined, not the actual words in the sentences. The first two features examined the length of the sentences and the average length of the words in the sentences. Jokes tend to be short and concise in their structure. The shorter the sentence and shorter the words in the sentence the more likely it is to be found funny. The number of words that repeat in a sentence was also examined. Sentences using simpler vocabulary tend to repeat descriptive words. This is a common characteristic of written text. A feature was then created to see how many unique words existed within the sentence.

The next set of features studied were the nouns, verbs, pronouns and adverbs in the sentences [11] using a spaCy function. Jokes typically lead with a noun and the punchline ends with a noun and supporting verbs to make the sentences flow. For example: “Why did the pigeon get thrown in jail? For staging a coo”. In this joke there are 3 nouns and 2 verbs that the code will look for.

The next more complex feature examined is bigrams. Bigrams use common word pairings to predict the appropriate follow up word to the first word in the pair. This is an important feature because the predictable pattern of sentences and key phrases in jokes can be found and can be used to predict those patterns in the future. Bigrams counts the number of predictable second-pairing words and returns that value for future use.

3.2.3 Sentiment Analysis

Finally, a NLTK (Natural Language Tool Kit) [12] was used to find the feature of sentiment analysis to breakdown the sentences**.** Sentiment analysis is a tool that looks at the words in the sentence to determine the relative emotion of the sentence. Sentiment analysis looks for words that it deems “positive” and “negative” and determines confidence values for the sentence which can be used for the features to find if sentiment analysis can be used to predict humour from non-humour data.

3.3 Classification Techniques

Key features were studied to see if they could work individually as well as together to see how strong of a prediction tool could be created. The features were tested using the 1)K-Nearest Neighbors classifier[1], 2)Random Forest[2], 3)Decision Tree[3], 4) stochastic gradient descent[4], 5)Passive Aggressive[5], 6)multinomial Naïve Bayes[6] and 7)Bernoulli Naïve Bayes[7]. These tests returned confidence scores which were used to determine the strength of the features in the training methods.

3.4 Classification Explanations

3.4.1 KNN

The K-Nearest Neighbor classifier works on the thought process of looking at where data falls around the other features. It determines how likely data is to fit into each category depending on the k closest neighbors in the training samples to a new sample in the feature. This means that if we have a large set of features, it will check each feature and see how many neighbors each feature has and determine how likely it is to be humor or non-humor data based on where it lands in comparison to the training data. [1]

3.4.2 Random Forest

The Random Forest classifier is an estimator that fits several decision tree classifiers on subsamples of the dataset and uses averaging to improve the accuracy of the classification. This classifier randomly permutes at each split, so the best split varies per training attempt. This can lead to very large branches and lots of data consumption and complexity in the size of the tree. [2]

3.4.3 Decision Tree

Decision tree classifiers use the non-parametric methods to create a model that predicts the value of a target by learning simple decision rules inferred by the data [3].

3.4.4 Stochastic Gradient Decent

Stochastic Gradient Decent estimates the loss gradient of each sample at a time and the model is updated along the way with decreasing schedule strength. This estimator implements regularized linear models with stochastic gradient descent learning: the loss is estimated for each sample at a time and the model is updated along the way with a decreasing strength schedule. The regulizer is a penalty added to the loss function that shrinks model parameters towards the zero-vector using either the squared Euclidean norm, the absolute norm or a combination of both.[4]

3.4.5 Passive Aggressive

Passive Aggressive classifiers work by checking if the precondition is correct and if it is then it keeps the model and doesn’t make any changes (Passive). Then if the precondition is incorrect, it makes changes to the model (Aggressive). [5]

3.4.6 Naïve Bayes

Finally, the two Naïve Bayes classifiers work similarly, multinomial Naïve Bayes classifier works stronger on multinomial data and the Bernoulli works better on Boolean and binary features. The naïve bayes theorem states that given class variable y and dependent feature vector x1 through xn with the following classification rule simplified to:

Which can be used to estimate p(y) the relative frequency of class y in the training set and p(xi|y)[8].

3.4.6.1 Multinomial Naïve Bayes

This method implements the naïve bayes algorithm for multinomial distributed data which makes it an effective tool for text classification. This distributes parametrized vectors θy = (θy1,…, θyn) for each y where n is the number of features and θyi is the probability P(xi|y\_ of features appearing in the sample belonging to class y[6].

3.4.6.1 Bernoulli Naïve Bayes

This method implements the naïve bayes training classification algorithm for data that is distributed according to the multivariate Bernoulli distribution. There may be multiple features but each one is assumed to be a binary value[7].

3.5 Data Splitting

The data is divided into a training set and a testing set, as for good practice. This is to create a way to train the data and test the data with unique untouched unbiased data. This is created to ensure that the model is trained on larger amount of good data and then tested to return an accuracy score on the smaller set of data from the dataset. This train/test split is a key step for using the classifiers. For this dataset a 70/30% split was used resulting in 140,000 lines of data in the training set and 60,000 lines of data for the tests. This is a standard train/test split ratio for the purposes of this program and works well for this testing.

4 Results:

4.1 Classification Selection:

As stated in section 3.3 multiple different classification methods were tested to determine which of the selected methods returned the highest test score. See figure 6.1 for results. Most of the examples used had a success rate of (+/- 0.05) with one major outlier in the passive aggressive method of 0.28. The rest of the classifiers would have worked and achieved higher than the goal of 0.60 but were too time consuming for the size of the dataset, taking anywhere between 10 minutes for the NB classifiers up to 3 hours for the SGD classifier. The results for the SGD classifier could have been higher with more iterations than the default one thousand but across 200,000 lines of data it would have taken too long to be viable. Thus, the KNN is the best result for the task at hand since it still returns high and much faster (within 3 minutes).

4.2 Feature selection:

Those findings were then used to see how strong the created features were, and it was found that the features ranged from 0.41-0.62. The strongest features being the question mark check and the sentiment analysis. Since none of the features had confidence values lower than 0.4 which was determined to be the minimum value for use. It was decided to use all available features ending up with 13 total features. See figure 6.2 for details.

4.3 Final Results on Test data:

For the final test results a 70/30 split on the full dataset was used. The features were trained on 140000 sets of data and evaluated the classifiers on 60 000 lines of data. The KNN classifier of n=7 returned a 0.73 test score. The highest feature test score was the <?> with a 0.62 and the sentiment analysis with a 0.61 with the lowest scores being the < **:** >, < **;** > and noun counts with a 0.41 test score. See figure 6.2.

5. Discussion:

A predictive model was developed to help classify humor. An automated humor detection software program using the “ColBERT: Using BERT Sentence Embedding for Computational Humor” dataset. This dataset was challenging to work with because there were numerous ways to breakdown the data. As part of the trial and error for developing features, multiples of features were used that returned low or even zero test scores. These attempts had to be cast aside or a workaround had to be developed.

5.1 Feature Engineering:

The core of this project was the construction and testing of computational features. Computational features that had scores of zero or low scores were scrapped because the values were too small to justify in the study. The data set was further examined to find any notable lines of data. The breakdowns in figure 6.3 show the data sorting in an excel spreadsheet. There are a few obvious features were found from this, “?” had 53% positive and 5% negative results and common words like “what” “do” and “you” had around 30% positive and 10% negative, see figure 6.3 for further breakdowns. Those are large parts of the dataset. 58% of the dataset is 116 000, for only looking at the “?” that is a good portion of the data and is a good feature. The next set of features explored were features that looked at every line of data and returned some set of values for all the data. Counting how many words were in each sentence, the average length of the words in the sentences, how often words repeat in each sentence were the features of the physical layout of the dataset that were chosen to be studied. Generally common patterns in the lengths and structure of each sentence were studied to see if there was any patterns present. Then the words in the sentence themselves were studied, to see how many nouns and verbs appeared, how many bigrams existed and whether the emotion of the sentence had any impact on humor detection. The sentiment analysis was another factor receiving a lot of testing. Since it returns 4 values for positive, negative, neutral and a combination set of emotions all possibilities were looked at to see if they should have been kept separate or whether they should have been combined. Several tests were conducted, and it was found that in the overall classification there was no difference between running separate or combined tests so to save time and space they were run altogether. All this experimentation took time to see if each of the values even merited being included in the testing. Figure 6.2 shows that each of the features were at least half decent at worst and made it well worth the effort and time testing each small variation of the features to get good results. More features could have been created to get the result from 0.73 up to .8 or higher but for the purpose of this assignment the features tested and used were enough**.**

5.2 Classification:

Classification was also difficult because determining the best classifiers was very dependent on the data. The obvious starting point was using the KNN to get started on the feature engineering since the KNN is super flexible and can be used effectively for almost all the preliminary tests for the dataset and features. A random forest test for a classifier was used but the default parameters were too large and took up too much memory. Even with evaluating the random forest classifier with an estimator of 1, there is still not enough memory allocation; it would take about 10GB of data. The random forest classifier will unfortunately not work for this dataset. The next problematic classifier was the SGD classifier. This classifier had a chance to be effective and returned a high score however it took multiple hours to process to return the same score as the KNN classifiers. After achieving final scores, the SGD classifier was tested again with a max iteration of 100 and 10 000 and resulted in 0.70 and 0.72 respectively. Letting the SGD with max iteration of 10 000 takes extra time to process and the result not being higher, made this classifier useless because the results did not justify the time. This classifier is still useful, but it takes too long to process for not high enough results. The classifiers was a situation where I tried classifiers to see what the results would be and the results reflected the data**.** The two naïve bayes classifiers both tested higher than 0.6. The multinomial NB worked well hitting around the 0.7 threshold for the data which is where most of the classifiers landed. However, the Bernoulli wasn’t the most effective because the features didn’t all reflect the processing methods of the Bernoulli classifier. Most of the features were binary so they did work well with the Bernoulli classifier but a few of the features returned with values that were decimal approximations. These results probably dragged the results of this classifier down to the 0.68.

References and Appendix:

[1] Scikit Learn Developers. Sklearn neighbors KNeighbors Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

[2] Scikit Learn Developers. Sklearn Ensemble Random Forest Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

[3] Scikit Learn Developers. Sklearn Tree Decision Tree Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

[4] Scikit Learn Developers. Sklearn Linear Model Passive Aggressive Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html#examples-using-sklearn-linear-model-passiveaggressiveclassifier>

[5] Scikit Learn Developers. Sklearn Linear Model SGD Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier>

[6] Scikit Learn Developers. Sklearn Naïve Bayes BernoulliNB Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html>

[7] Scikit Learn Developers. Sklearn Naïve Bayes MultinomialNB Classifier. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB>

[8] Scikit Learn Developers. Sklearn Naïve Bayes. scikit learn 2007-2023. <https://scikit-learn.org/stable/modules/naive_bayes.html>

[9] Annamoradnejad, I., &amp; Zoghi, G. (2020). Colbert: Using Bert Sentence embedding in parallel neural networks for computational humor. Github. https://github.com/Moradnejad/ColBERT-Using-BERT-Sentence-Embedding-for-Humor-Detection

[10] Python Core Team. Python: A dynamic, open-source programming language. Python Software

Foundation, 2019.

[11] Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.

[12] Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. " O&#x27; Reilly Media, Inc."

[13] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., &amp; Duchesnay, É. (2011). Scikit-Learn: Machine learning in Python. Journal of Machine Learning Research. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

|  |  |
| --- | --- |
| Test Classifier | Test Score |
| K-Nearest Neighbor (2) | 0.63 |
| K-Nearest Neighbor (3) | 0.70 |
| K-Nearest Neighbor (5) | 0.72 |
| K-Nearest Neighbor (7) | 0.73 |
| K-Nearest Neighbor (9) | 0.73 |
| Decision Tree | 0.66 |
| Multinomial NB | 0.69 |
| Bernoulli NB | 0.68 |
| Passive Aggressive | 0.28 |
| SGD | 0.72 |

Figure 6.1 Classifier Test Scores

|  |  |
| --- | --- |
| Test Feature | Feature Score (KNN n=7) |
| F1 (?) | 0.62 |
| F2 (;) | 0.41 |
| F3 (:) | 0.41 |
| F4 (...) | 0.46 |
| F5 (!) | 0.45 |
| F6 (common words) | 0.58 |
| F7 (nouns) | 0.41 |
| F8 (verbs) | 0.43 |
| F9 (word count) | 0.56 |
| F10 (avg word length) | 0.57 |
| F11 (unique words) | 0.55 |
| F12 (bigrams) | 0.53 |
| F13 (sentiment analysis) | 0.61 |

Figure 6.2 KNN Feature Scores

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Examined Value | TRUE | FALSE | TOTAL | PERCENT TRUE | PERCENT FALSE |
|  | 100000 | 100000 | 200000 | 50 | 50 |
| punctuation |  |  |  |  |  |
| ? | 53051 | 5255 | 58306 | 53.051% | 5.255% |
| ... | 12725 | 644 | 13369 | 12.725% | 0.644% |
| / | 1000 | 439 | 1439 | 1% | 0.439% |
| ; | 526 | 169 | 695 | 0.526% | 0.169% |
| : | 5461 | 19079 | 24540 | 5.461% | 19.079% |
| ! | 10867 | 8075 | 18942 | 10.867% | 8.075% |
| { | 10 | 1 | 11 | 0.01% | 0.001% |
| [ | 2 | 1 | 3 | 0.002% | 0.001% |
| ( | 1359 | 9377 | 10736 | 1.359% | 9.377% |
| . | 66536 | 6052 | 72588 | 66.536% | 6.052% |
| , | 16751 | 18209 | 34960 | 16.751% | 18.209% |
| Common Words |  |  |  |  |  |
| what | 28538 | 3897 | 32435 | 28.538% | 3.897% |
| who | 5237 | 2380 | 7617 | 5.237% | 2.38% |
| when | 6733 | 1516 | 8249 | 6.733% | 1.516% |
| why | 11875 | 3061 | 14936 | 11.875% | 3.061% |
| how | 8707 | 7098 | 15805 | 8.707% | 7.098% |
| do | 33848 | 14080 | 47928 | 33.848% | 14.08% |
| you | 30099 | 11880 | 41979 | 30.099% | 11.88% |
| get | 8077 | 4000 | 12077 | 8.077% | 4% |

Figure 6.3 Excel Dataset Content Breakdown for Initial Classifiers