# CS 4641 Final Project

Dec. 6, 2019

## Contents



## 1 Introduction

Humor analysis has recently become an important aspect of many NLP (Natural Language Processing) studies. Earlier NLP tasks, such as sentiment analysis, usually have signature words that makes classification trivial in many cases and can be easily distinguished by any human with basic understanding of the language. Jokes, however, requires a deeper understanding of the cultural content and the specifics of a language. Many jokes rely on different meanings of the same word (*lexical ambiguity jokes*) Jokes also mean different things to different people. A joke that can make someone spill out the water in their mouth may make no sense to somebody sitting right beside them in the classroom because of their different backgrounds. Personally, as an international student, I rarely have problems entertaining my classmates with jokes in my home country. However,

despite my moderately well spoken English, thinking about jokes for other native English Speakers is completely another story (because I never thought of one that made people actually laugh and things usually went really awkward). It would be really interesting for me to see whether machine learning algorithms can do better job at distinguishing jokes from other contexts than me in English.

I created a binary classification supervised learning problem to distinguish between jokes and non-jokes in English. The jokes are primarily lexical ambiguity jokes from the Kaggle public dataset Short Jokes by Abhinav Moudgil, which are viewed as positive examples, and non-jokes are from the Wikipedia Sentences dataset by Mike Ortman which includes sentences from an August 2018 Wikipedia dump. Approximately 10000 sentences are selected from both datasets for compatibility with this project. Primary metric used in evaluating performance is accuracy (including visualization such as ROC curves) due to its compatibility to all classifiers implemented in this project. Accuracy is also very suitable as the dataset is very balanced and does not need to consider the biased representation that the accuracy measure gives on a unbalanced dataset.

# 2 Algorithm Description and Hyperparameters

Algorithms implemented within this project include Random Forest (an algorithm of Decision Tree with Bagging), Adaboost (Decision tree with boosting), non-linear kernel SVM and Convolutional Neural Networks (for lexical analysis)[1]. However, the first three algorithms only supports vector inputs. Therefore, the **Tf-idf** vectorizer is used to convert all sentences in the dataset into vectors of relevant frequencies of each word in the corpus before passing the data into these classifiers. Hyper-paramers are different for each learning method and are listed below.

## 2.1 Hyper-parameters for the Tf-idf Vectorizer

[2] **ngram_range**: The Tf-idf vectorizer have the option to encode multiple words at a time. The range of the number of words in one encoding in the Tf-idf vectorizer is specified by the ngram_range parameter. Larger ngram_range parameter would result in phrases and combination of words to be taken into consideration in the hypothesis class but would also create larger vectors and sparser common features between difference sentences, making more features in the vector useless in the test set. The tested value for ngram_range is (1, 1) and (1, 2), for "including only single word" and "both single word and bigram (two words)" respectively.

**stop_words**: Stop words are words that provide little information and commonly appear in almost all contexts. These include words such as "I", "because" etc. Including stop words would remove the influence of such words in the hypothesis class and may include generalization strength of a model.

**tokenizer**: Tokenizers are methods that converts a sentence into a list of words. In this project, I tested the influence of word stemming in the tokenizer, which transforms different forms of a same lemma into the same token. Applying the stemmer would reduce the length of feature vectors and increase the generalizability of the model, but may lose information presented in the differences between forms of the same lemma which may be important under certain circumstances.

### 2.2 Hyper-parameters for the Random Forest Classifier[2]

**max_depth**: the maximum depth of each decision tree in the random forest. Larger depth would allow more complex decision boundaries but may cause severe overfitting while smaller depth would result in less complex decision boundaries but with better generalization.

**n_estimators**: the number of decision trees in the random forest. Higher number of trees would result in more complex decision boundaries and usually increase accuracy but have large impact on the efficiency of training. Tested number of estimators are 10, 50 and 100

### 2.3 Hyper-parameters for the Adaboost Classifier[2]

**n_estimators**: Similar to the same hyper-parameter in the Random Forest Classifier. Due to the nature of boosting algorithms of combining weak learners, I set the values for this hyper-parameter larger than it's counterpart in the Random Forest Classifier. Tested values include 50, 100 and 200. Larger numbers of estimators would result in more complex decision boundaries.

**learning_rate**: The learning rate during each iteration. Setting a large value may cause the algorithm to converge quicker at the risk of diverging from the minimum loss and overfitting. Setting a small value would usually lead to smoother decent along the loss function but converges slower and is more likely to be stuck in local minima. Changing the learning rate does not change the hypothesis class. Tested values include 0.1, 0.5, 1.0 and 5.0 [3]

### 2.4 Hyper-parameters for the SVM classifier[2]

**C**: The regularization term, which in SVM is the weight assigned to miss-classifications. Larger regularization term would result in less misclassifications in the training set and may result in overfitting and smaller regularization term would allow more misclassifications in the training set and may be more generalizable. Does not affect the hypothesis class. Tested values include 0.1, 0.5, 1.0, and 5.0.

**kernel**: The kernel function to use in the SVM. Different kernels would result in different types of decision boundaries in the hypothesis class. Tested values include 'rbf', 'poly', and 'sigmoid'.

**gamma**: parameter in the gaussian kernel. Increasing gamma would result in more complex decision boundaries and may result in overfitting.

### 2.5 Convolutional Neural Network of Lexical Analysis[1]

The CNN is different from the previous models in that it does not strictly require a fixed-size vector input. In our CNN, we first embed each word into its own corresponding trainable vector and combine all word vectors in a sentence into a matrix. The convolutional and pooling layers would then use filters to extract features in this sentence that would then be passed into a regular neural network what was a multilayer perceptron.

**embedding_dim**: The number of dimensions of each word embedding. Larger dimension would result in more features related to each word but may result in overfitting with a small number of samples. Tested values include 64 and 128

**num_filters**: number of convolutional filters of each filter size. Larger number of filters results in more comprehensive feature extraction and more complex decision boundaries but may also cause overfitting. Tested valued include 64 and 128.

Due to the complexness and low efficiency of neural networks, it is impossible to test more combinations of hyper-parameters for this project. Therefore, only a small number of cross validation and hyper-parameter combination test is performed for tuning.

# 3 Tuning

A lot of tools are available for tuning hyper-parameters. I mainly chose to use the GridSearchCV function for RandomForest, AdaBoost, and SVM using a 5-fold cross validation with different hyper-parameters combinations and manually tuned the CNN using 2-fold cross validation.

## 3.1 Random Forest with Tf-idf

A total of 96 hyper-parameter combinations (all possible combinations of hyper-parameters mentioned in the previous section) have been used for tuning with 5-fold cross validations. This totals to 480 tuned models. The tuning took 14.2 minutes with the highest accuracy of $0.966 \pm 0.004$. As it is unnecessary to provide the result of all tuned models, the following table demonstrates the hyper-parameter combinations with the 10 highest accuracy.

| Performance of Hyper-parameter Combinations in Random Forest Using Cross Validation | | | | | |
|---|---|---|---|---|---|
| ngram_range | stop_words | tokenizer | max_depth | n_estimators | accuracy |
| $(1, 2)$ | No | Stemmed | Unlimited | 100 | $0.966 \pm 0.004$ |
| $(1, 1)$ | No | Stemmed | Unlimited | 100 | $0.966 \pm 0.002$ |
| $(1, 1)$ | No | Stemmed | Unlimited | 50 | $0.965 \pm 0.004$ |
| $(1, 1)$ | No | Regular | Unlimited | 100 | $0.965 \pm 0.002$ |
| $(1, 2)$ | No | Stemmed | Unlimited | 50 | $0.964 \pm 0.003$ |
| $(1, 1)$ | No | Regular | Unlimited | 50 | $0.963 \pm 0.004$ |
| $(1, 2)$ | No | Regular | Unlimited | 100 | $0.963 \pm 0.003$ |
| $(1, 1)$ | No | Stemmed | Unlimited | 10 | $0.960 \pm 0.003$ |
| $(1, 1)$ | No | Regular | Unlimited | 10 | $0.953 \pm 0.004$ |

From this table we can see that stop words are not very useful when using Random Forest on jokes and limiting the maximum depth does not increase generalizability in our case. Note that hyper-parameter combinations with the same accuracy in the table are ranked by smaller differences in their accuracy values but is not displayed by the output of the program. The most accurate parameter set during tuning will be applied later to the larger test set.

## 3.2 Ada Boost with Tf-idf

A total of 96 hyper-parameter combinations (all possible combinations of hyper-parameters mentioned in the previous section) have been used for tuning with 5-fold cross validations. This totals to 480 tuned models. The tuning took 37.8 minutes with the highest accuracy of $0.966 \pm 0.004$. The following table demonstrates the hyper-parameter combinations with the 10 highest accuracy.

| Performance of Hyper-parameter Combinations in AdaBoost Using Cross Validation | | | | | |
|---|---|---|---|---|---|
| ngram_range | stop_words | tokenizer | learning_rate | n_estimators | accuracy |
| $(1, 2)$ | No | Stemmed | 0.5 | 200 | $0.959 \pm 0.005$ |
| $(1, 1)$ | No | Stemmed | 0.5 | 200 | $0.958 \pm 0.004$ |
| $(1, 1)$ | No | Stemmed | 1 | 200 | $0.958 \pm 0.006$ |
| $(1, 2)$ | No | Stemmed | 1 | 200 | $0.958 \pm 0.004$ |
| $(1, 1)$ | No | Regular | 0.5 | 200 | $0.958 \pm 0.004$ |
| $(1, 1)$ | No | Stemmed | 0.5 | 100 | $0.957 \pm 0.001$ |
| $(1, 2)$ | No | Regular | 0.5 | 200 | $0.956 \pm 0.003$ |
| $(1, 1)$ | No | Regular | 1 | 200 | $0.955 \pm 0.004$ |
| $(1, 2)$ | No | Stemmed | 1 | 100 | $0.955 \pm 0.004$ |
| $(1, 2)$ | No | Stemmed | 0.5 | 100 | $0.953 \pm 0.005$ |

We can see that stop words are still not very helpful in the AdaBoost case, and in general AdaBoost performs slightly worse than the Random Forest classifier during tuning.

### 3.3 SVM with Tf-idf

A total of 96 hyper-parameter combinations (all possible combinations of hyper-parameters mentioned in the previous section) have been used for tuning with 5-fold cross validations. This totals to 480 tuned models. The tuning took 76.1 minutes with the highest accuracy of $0.971 \pm 0.003$. The following table demonstrates the hyper-parameter combinations with the 10 highest accuracy.

| Performance of Hyper-parameter Combinations in AdaBoost Using Cross Validation | | | | | | |
|---|---|---|---|---|---|---|
| ngram_range | stop_words | tokenizer | C | gamma | kernel | accuracy |
| $(1, 1)$ | No | Stemmed | 5 | scale | rbf | $0.971 \pm 0.003$ |
| $(1, 1)$ | No | Regular | 5 | scale | rbf | $0.971 \pm 0.003$ |
| $(1, 1)$ | No | Stemmed | 1 | scale | rbf | $0.970 \pm 0.004$ |
| $(1, 1)$ | No | Regular | 1 | scale | rbf | $0.968 \pm 0.003$ |
| $(1, 1)$ | No | Stemmed | 5 | scale | poly | $0.955 \pm 0.003$ |
| $(1, 1)$ | No | Stemmed | 1 | scale | poly | $0.953 \pm 0.007$ |
| $(1, 1)$ | Yes | Regular | 5 | scale | rbf | $0.951 \pm 0.006$ |
| $(1, 1)$ | No | Regular | 1 | scale | poly | $0.951 \pm 0.005$ |
| $(1, 1)$ | Yes | Stemmed | 1 | scale | rbf | $0.949 \pm 0.007$ |

It's worth noting that the gamma hyper-parameter is essential for tuning the SVM. All other values of the 'gamma' hyper-parameter in the tuning process other than 'scale' lead to a model that only outputs the (slightly) majority class. It also seems that the RBF kernel is more suitable for this particular task than the polynomial kernel.

### 3.4 CNN

Due to the complexity of the implementation and the relatively lower efficiency of CNN, only 4 hyper-parameter combinations are tested with a 2-fold cross validation. The test results are demonstrated in the table below.

| Performance of Hyper-parameter Combinations in CNN Using Cross Validation | | |
|---|---|---|
| embedding_dim | num_filters | accuracy |
| 256 | 128 | $0.963 \pm 0.003$ |
| 128 | 128 | $0.962 \pm 0.001$ |
| 128 | 256 | $0.962 \pm 0.001$ |
| 256 | 256 | $0.960 \pm 0.001$ |

It seems that the parameters tuned in this experiment have little impact on the performance of the accuracy during the evaluation stage.

### 3.5 Final Hyper-parameters

Based on results during the tuning stage, the following hyper-parameters are selected for each classifier.

Random Forest with Tf-idf: ngram_range $= (1, 2)$ stop_words=*None* tokenizer=tokenizer_porter max_depth=*None* n_estimators=100

Adaboost with Tf-idf: ngram_range $= (1, 2)$ stop_words=*None* tokenizer=tokenizer_porter learning_rate=0.5 n_estimators=200

SVM with Tf-idf: ngram_range $= (1, 1)$ stop_words=*None* tokenizer=tokenizer_porter C=5 gamma='scale' kernel='rbf'

## 4 Performance

A new dataset with 10000 jokes and non-jokes are handed to all classifiers. Each classifier is trained on all tuning data and is ran 5 times to ensure the performance results are statistically significant. The accuracies and running time are listed as follows.

| Final Large Dataset Performance | | |
|---|---|---|
| Classifier Name | Accuracy | Training and Evaluation Time in Seconds |
| Random Forest | $0.959 \pm 0.001$ | $43.6 \pm 3.17$ |
| Ada Boost | $0.954 \pm 0$ | $30.5 \pm 1.52$ |
| SVM | $0.971 \pm 0$ | $108.2 \pm 2.81$ |
| CNN | $0.966 \pm 0.002$ | $1261 \pm 7.56$ |

From the above figure we can conclude that, for non-CNN models, there's a significant tradeoff between accuracy and running time. The SVM classifier is the most generalizable among these classifiers to new data with tuned parameters with the current training set. The CNN does not achieve the same performance as SVM despite taking significantly longer time. To evaluate how training size affect accuracy on these four classifiers, I applied these classifiers to a truncated training set of only 1000 training samples. The results are listed as follows:

| Final Large dataset performance | | |
|---|---|---|
| Classifier Name | Accuracy | Training and Evaluation Time in Seconds |
| Random Forest | $0.922 \pm 0.007$ | $8.37 \pm 1.25$ |
| Ada Boost | $0.902 \pm 0.027$ | $8.56 \pm 0.15$ |
| SVM | $0.947 \pm 0.03$ | $9.14 \pm 0.09$ |
| CNN | $0.858 \pm 0.10$ | $75 \pm 1.10$ |

# 5 Conclusion

Jokes are significantly different under different cultures and regions. It is therefore impossible to use one single model to judge whether a sentence is a joke universally. Therefore, gathering new jokes in different areas and training models based on them. However, it is not feasible to gather about 10000 jokes everywhere, which makes small training set performance to be really important. SVM performs the best both in the larger and smaller dataset. Moreover, the fall in accuracy resulting in a smaller training set is also not as large as other classifiers and behaves relatively stably across multiple runs (as shown by low margin of error). However, SVM requires a very long tuning process and may need to be retuned for new kinds of jokes for maximum performance.

Also it seems that CNN is not very suitable for this particular problem in both accuracy and performance. It also performs significantly worse with less training data.

Despite performance issues during tuning and training, I would still conclude that SVM with the RBF kernel and Tf-idf word vectorizer is the most suitable classifier for joke classification problems.The accuracy gain from SVM is much more potent than it's relative incompetence in efficiency.

# 6 Acknowledgments

The following external resources are used in order to complete this project.
1. *Implementing a CNN for Text Classification in Tensorflow* code, written by Denny Britz and is used for classifying jokes using CNN in this project. `https://github.com/dennybritz/cnn-text-classificatio`
2. The *scikit-learn* library, used for the RandomForest Classifier, SVM Classifier, the Tf-idf Vectorizer and the AdaBoost Classifier.
3. *Shrinkage parameter in Adaboost?* from the Stackexchange community. `https://stats.stackexchange.com/q`