Introduction/Background

Given a random joke in text form, how can we classify which categories the joke falls into? This problem is especially intriguing since humour differs greatly from person to person. Allowing a user to see jokes from their preferred category can increase user engagement on a website. Furthermore, with the vast amount of data available these days, our methods for classifying jokes can be expanded to group all kinds of textual data into categories for organization.

Data Collection

We get our Comedy Central dataset from Youtube Data API. Our dataset has corresponding features like joke category, content, author, etc.

Problem definition

The main goal of our project is that given a dataset of jokes, create a program to identify which category a joke falls into and suggest a joke of a specified category to a user. Our project will be based on another project one of our teammates worked on previously, so we will improve it by trying more methods and models.

Methods

We plan on using the following supervised classification methods to go about solving our problem. **Random Forest, Multi-layer Perceptron,** **Logistic Regression, K nearest neighbors, Tree classifier, and Naive Bayes.**

**Logistic regression** is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. In our project we will use Logistic Regression to assess the relationship between a joke's features and its classification. From there, we can hopefully build a model equation that, given a joke's features, will predict a jokes classification.

**K-nearest Neighbors** - KNN is an algorithm that assumes similar things exist in close proximity. A set of already classified data points is given and plotted, and using these points we can classify a point that is not already classified based on its proximity (likeness) to points that are already classified. The number, K, determines the number data points in proximity we use to classify our unlabeled data point. That is, if K = 1, we classify our unlabeled data point to be whatever the nearest data point is classified is.

**Tree Classifier** is a predictive modeling approach that identifies ways to split a data set based on different conditions. It breaks down a data set into smaller subsets, while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. We plan on using this method, to build a classification tree that generates its branches based on the features of the jokes in our data set. This tree will represent a model that can be used to decide a joke’s classification.

**Naive Bayes** is a simple, yet effective and commonly-used, machine learning classifier that utilizes Bayes Theorem to calculate the probability of features occurring in each class. Consider “B” to be the features of a data set and “A” to be the possible classifications of each data point. The equation describes the probability that a data point belongs to a class ,A, given its features B. In our project, we will use Naive Bayes classification methods to build a model that calculates the probability that a joke belongs to a class given its features.

**Random Forest** is an algorithm that consists of a large number of individual decision trees(which was introduced above). Each decision tree contributes one vote to a class prediction and the class with the most votes will be the model’s prediction.

**Multi-layer Perceptron** is a supervised learning algorithm that takes in a number of features and generates an outcome. It can deal with both linear or non-linear function approximator for both classification and regression. It contains multiple layers and each layer weights the input and sends the result to the next layer until we can the final result.

Potential Results

Our expected result is a joke classification system that takes in a joke and outputs one or several labels, such as “animal”, “food”, or “marriage” for this specific joke. Our research method is to use 80% of our dataset as the training set and use the other 20% of them as the validation set, a research method used in a paper also on joke classification (Chakrabarty 233). If our project went well, we would like to take external datasets of jokes for validation, and we would expect a good performance on such datasets too.

Discussion

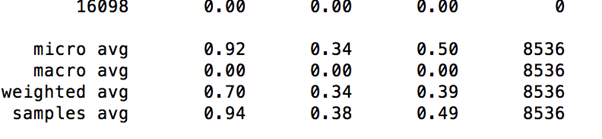
~~At this stage of our project, we have proposed many machine learning methods and will try to see how well they can generate a right label for each joke. Compared to a similar paper that uses Bernoulli Restricted Boltzmann Machine, which is a large-scale multi-layer deep learning model, it is natural to think that our Multi-layer Perceptron model, which is also a deep learning model, may perform a little better given our rather large dataset of 9136 data points (Chakrabarty 232). We cannot wait to see how each model performs.~~

Overall, we attempted a variety of machine learning methods to test how well they can generate a right label for each joke. While a similar paper uses Bernoulli Restricted Boltzmann Machine, which is a large-scale multi-layer deep learning model, we decided to see if we could achieve high accuracy without neural networks (Chakrabarty 232). The highest accuracy we achieved was through a Logistic Regression model with an accuracy of about 92%.

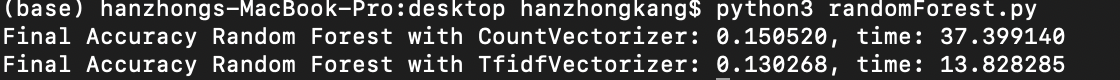
One possible reason that some of our models did not perform very well is that the jokes are composed of a short corpus that only has a small amount of information, so it was difficult to identify a complex set of labels for each joke. In the future, one way to improve our model accuracy would be generalizing the labels into fewer categories. For example, we may generalize the “men/women” and “marriage” into a new label “gender/relationships.” With fewer labels, we expect the overall accuracy to be much better.

In the future, one way to improve the accuracy would be generalizing the labels into fewer categories, because the jokes are short corpus that may not have enough information for 20 labels.

Results

**Random Forest-** We used sklearn Random Forest Classifier to implement the model. In the data processing part, we applied Countvectorizer function to convert the corpus into integer arrays. The current accuracy is shown in the graph

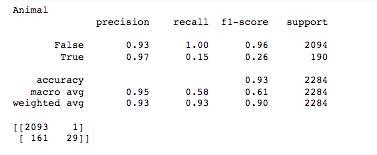
After I did some changes to the original model with Countvectorizer data processing method, the accuracy of the model dropped significantly. With a similarly accuracy score, the model with Tfidf data processing method, has a 13% accuracy. I also keep track of the time each model uses to finish training. The potential reason could be there are usually multiple categories for one prediction, and the result would be considered a “bad prediction” when there is just one label prediction off.



**Multi-layer Perceptron-**(too hard)

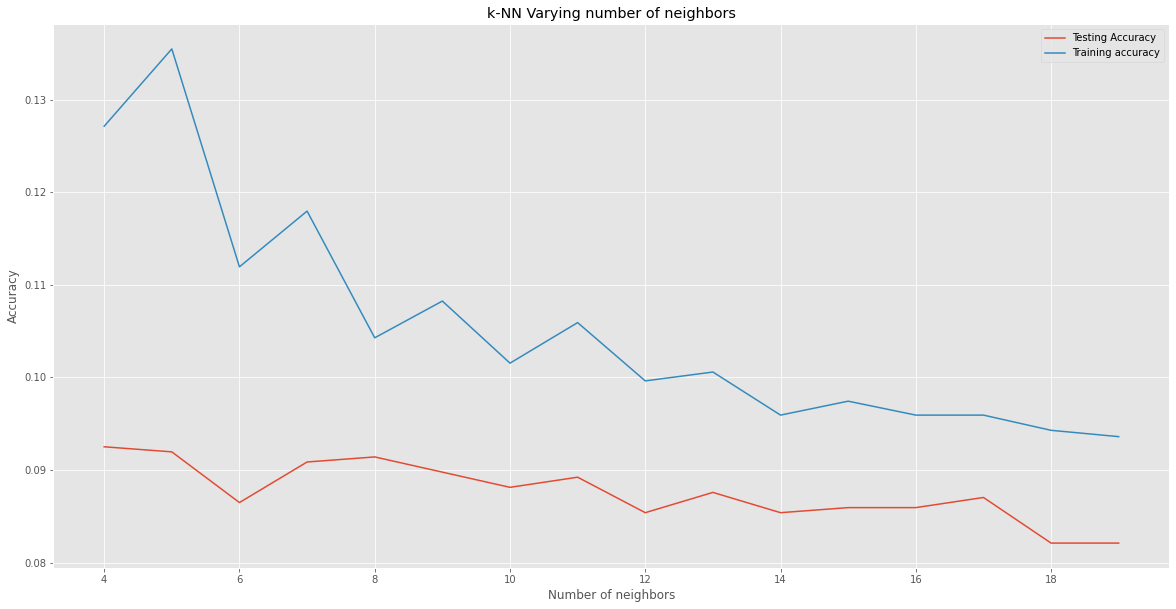
**Logistic Regression-** We used sklearn Logistic Regression to build a model that predicts if a joke belongs to a single category. We split out data 75% training and 25% testing to run our model. We created a new column for each vector classification possible, and put 1 if the joke classified as the category, and 0 if it did not. Then the data was cleaned, by taking content of joke and deleting stop words, unnecessary characters, and putting everything in lowercase letters. We converted the content feature into a matrix of TF-IDF features with tfidfVectorizer (equivalent to countvectorizer) and finally trained the data with Logistic Regression. The overall average accuracy of the model across categories was well over 90%, but we saw accuracies as low as 80%. Our next steps in improving this model are to explore other training testing data ratios, and also look into other means of feature engineering besides TfidfVectorizer.

Example Results for Animal Category



**K nearest neighbors-** For K Nearest Neighbors, we first extracted the joke and flattened categories from the CSV file. Twenty percent of the data was set aside for testing purposes, leaving 80% for training. Since the data was given in a textual format, we used CountVectorizer from SKLearn to tokenize all the words, which mapped each unique word to a unique number.

For the training, we used the built-in KNeighborsClassifier in SKLearn. For the final report, we experimented with a range of neighbors and saw generally diminishing returns with more neighbors.



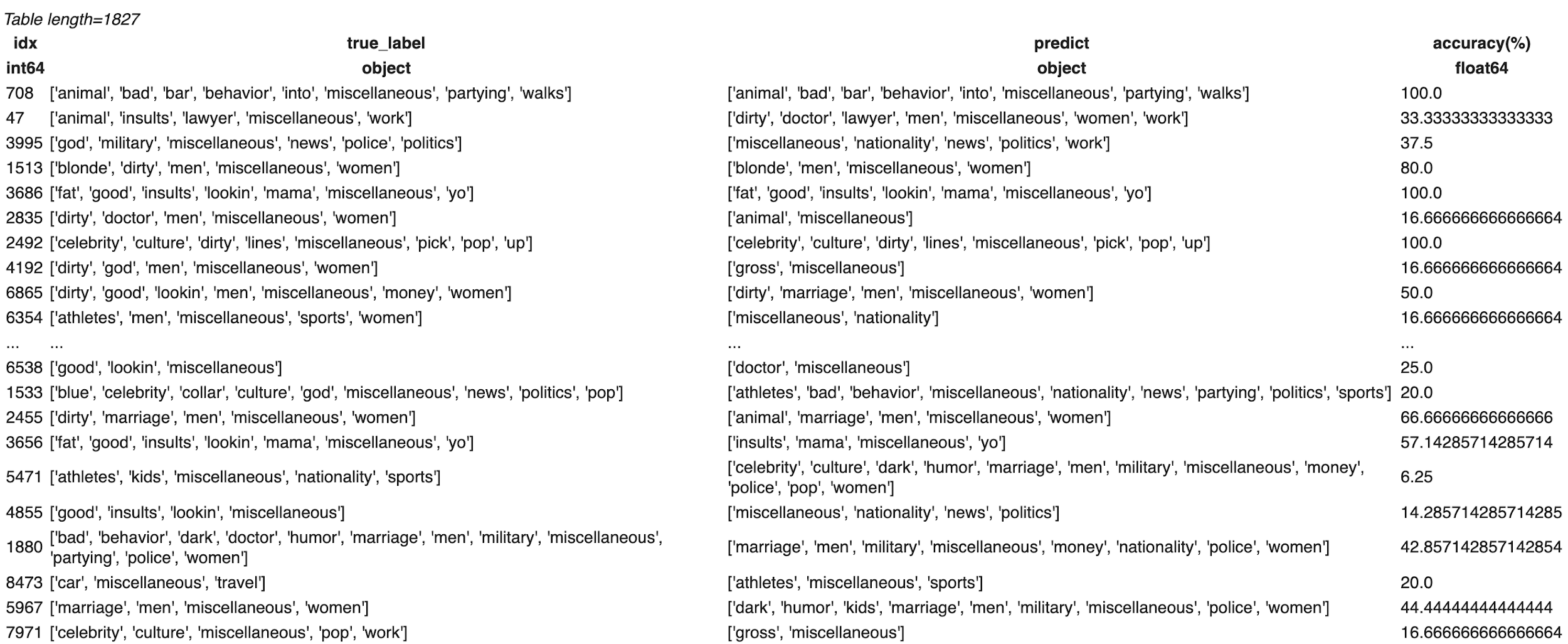
The best accuracy on the training set was 9.141% with 8 neighbors. This is to be expected because the jokes in our data rarely have the same exact categories as another joke, so finding the nearest neighbor will usually be wrong.

We also tried a variety of vectorization methods and found CountVectorizer to be the best, so this was left unchanged. Furthermore, we attempted token extraction to strip jokes of unimportant words which may have skewed the data, but the accuracy was left overall unaffected.

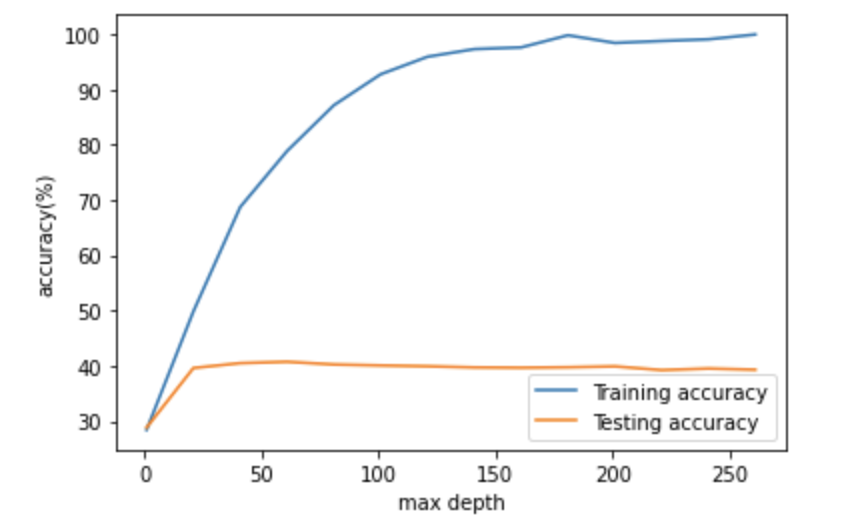
**Tree classifier-** For our Tree Classifier model, we split the data into 80% for training and 20% for testing. We utilized count-vectorizer as a method of converting the corpus into a matrix to use the frequency of occurrence of each tokenized word in each sentence from training data as features of our model.

We used scikit-learn Decision Tree Classifier to build a classification tree. However, the overall testing accuracy we got from this model was about 39.36% while the overall training accuracy was about 99.95%.

Result for testing:



The graph below represents the accuracy against the max depth set for Decision Tree Classifier.



Though the testing accuracy for our model is low, we realized adding depth after about 20 does little to improve the accuracy. We will work on pruning to reduce the likelihood of overfitting as much as possible for the final project report. Another possible cause of the low accuracy in this model is due to the method of converting corpus to a matrix. We are considering using a pre-trained model, such as GloVe, to utilize our data more effectively in the future.

**Naive Bayes-** For our Naive Bayes Model, we used two different ways of classification (classifying all labels together using a multinomial Naive Bayes wrapped in a one-vs-rest classifier and classifying one label at a time using a multinomial Naive Bayes) and two methods for vectorization (count-vectorizer and tfidf-vectorizer) on the dataset that is splitted 80/20 for training and testing.

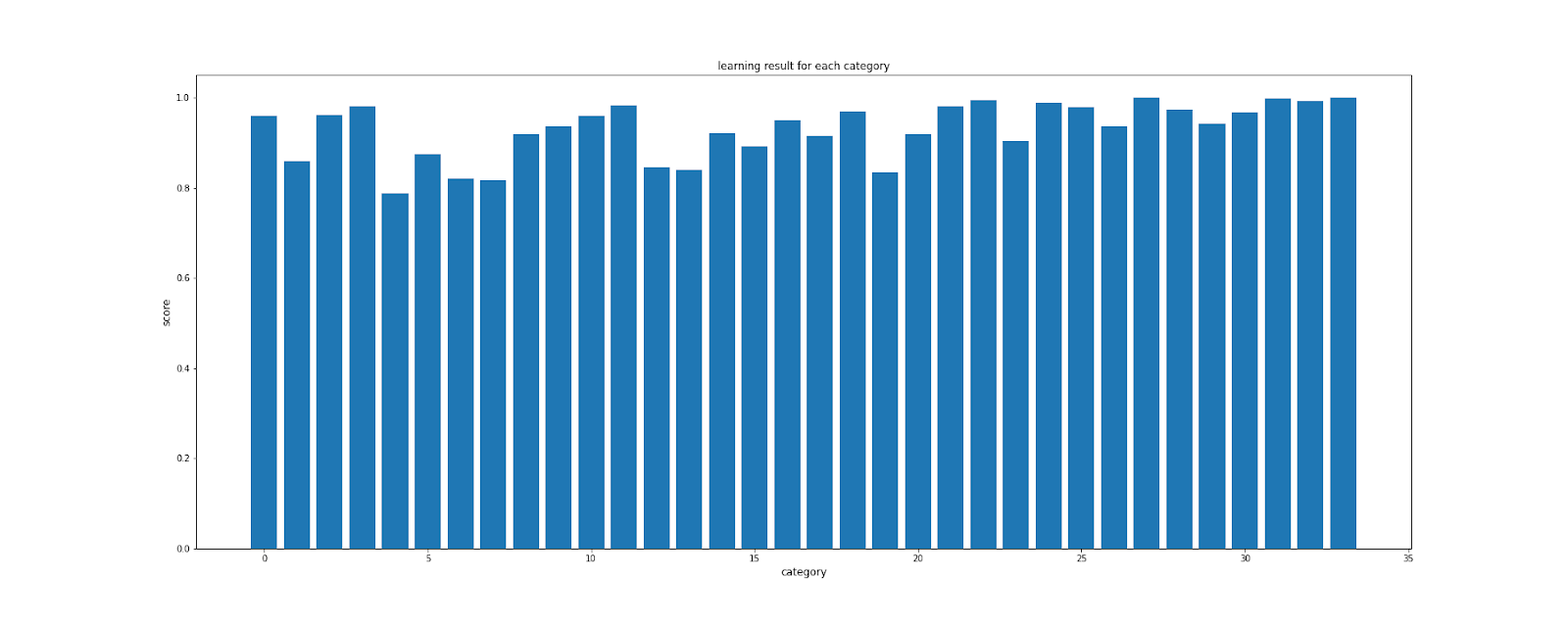
For data pre-processing, we extracted the joke content column as a list of jokes, and processed it with either CountVectorizer or TfidfVectorizer by sklearn, and we call it X. We also extracted the flattened\_labels column and transformed it to a N x C matrix of occurrence, where N is the number of datapoints and C is the total number of labels, and we call it Y. If an element Mn,c in this matrix equals to 1, that means the nth datapoint has the label c, and 0 otherwise.

When we used our first way of classification, which is feeding OneVsRestClassifier (by sklearn) with X and Y, the accuracy score was pretty bad, whether X is vectorized by CountVectorizer or TfidfVectorizer.

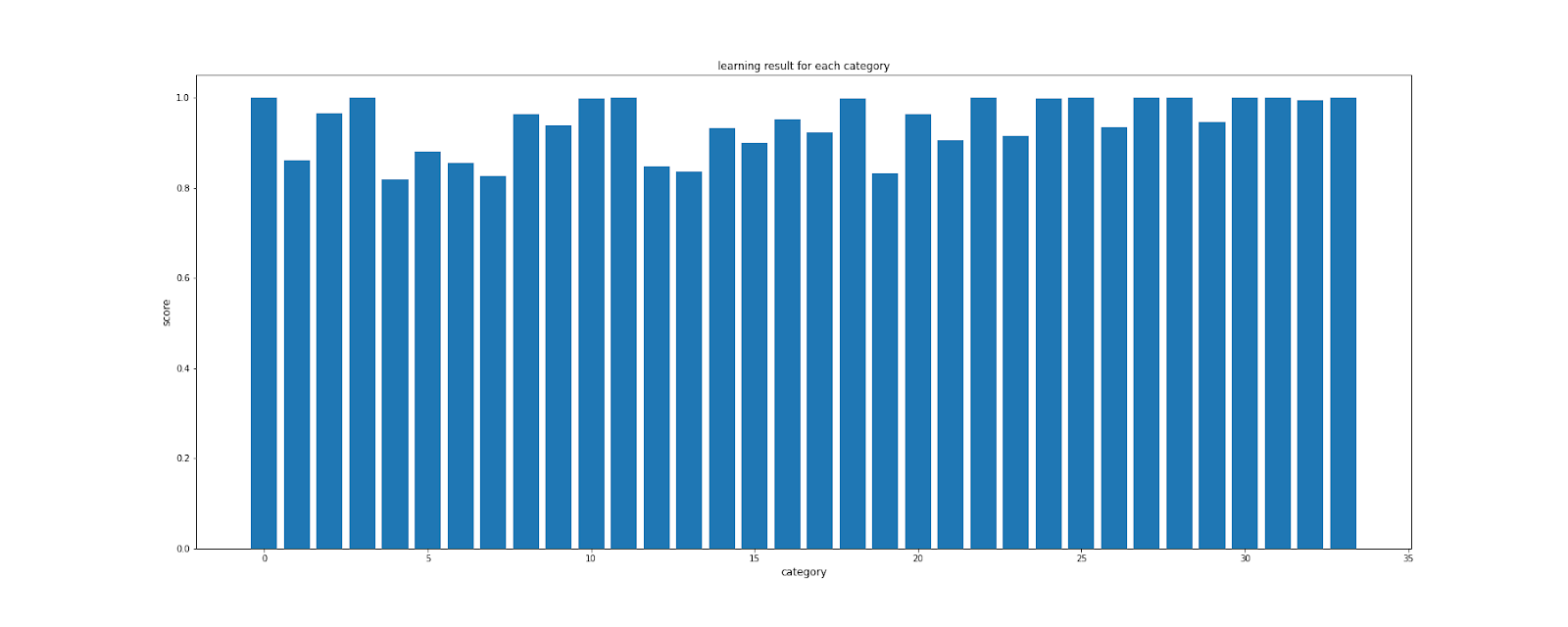
|  | Count vectorizer | Tfidf vectorizer |
| --- | --- | --- |
| Accuracy | **0.050082918739635156** | **0.022222222222222223** |

When we use our second way, which is feeding MultinomialNBClassifier (by sklearn) with X and one column of Y (one label at a time), the score of accuracy was pretty good for each label:

Count Vectorized X score:



Tfidf Vectorized X score:



As we can see, for each category the score goes up to at least 0.8 and sometimes close to 1.0, while Tfidf Vectorized X generally has better results versus Count Vectorized X.

The significant difference in accuracy score between one-vs-rest methodology and one-label-at-a-time methodology may indicate that accuracy score may not be the best way to measure the success of multilabel naive bayes classification, because one-vs-rest is essentially predicting labels one by one and append the truth value of labels (0 or 1) together. Such difference may be caused by the accuracy score mechanism, where one fault in a row in the predicted label-occurrence matrix means inaccuracy for the whole row, while all of the values predicted right are ignored.

Because of the low accuracy, we also used other metrics to evaluate the model. The micro f1-score of the multilabel classification using tf-idf vectorizer is 0.5166, which indicates that the model does predict labels well enough. Judging from the relatively high accuracy for individual labels, the f1-score is more representative of the model’s performance than the harsh accuracy.

Conclusions**:**

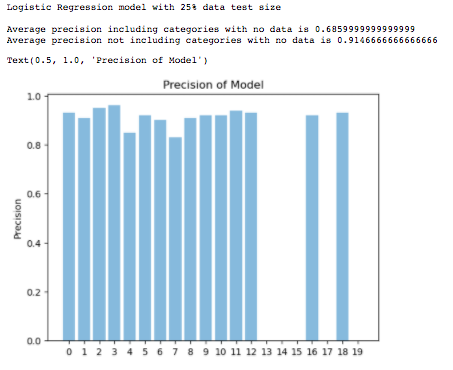
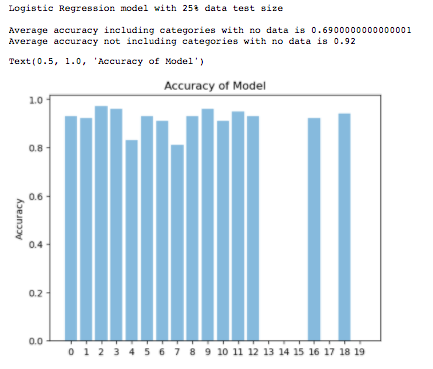
**Ishan:** Overall, we have decided that K-Nearest Neighbors is not a well suited classification model for this problem. With an accuracy rate of about 9%, the model is inadequate at categorizing jokes correctly. However, this is expected since the category a joke falls into is heavily dependent on the theme of the joke and the string of words in a certain order. Very rarely will 2 jokes match up exactly in terms of the categories they fall under. Since KNN attempts to find the nearest neighbors, it will try to match the joke to the most similar joke. This similar joke may not have the exact same categories, though, so the joke will be marked as incorrectly categorized.

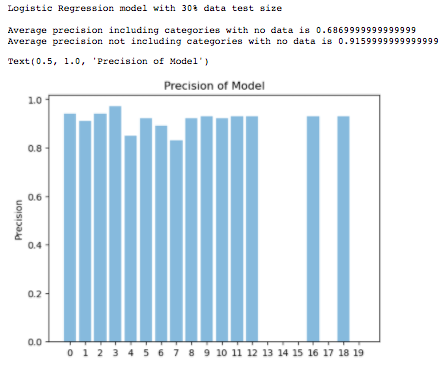
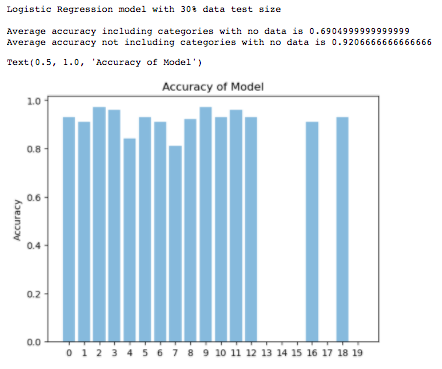
A different classification method that may work with KNN would be to first train based on a single category with a boolean value of whether the joke belongs to the category or not. Training such a model would take weeks with our data set, however, so we decided not to follow through with this route for KNN. Furthermore, we found much better success with some of the other models we tried.

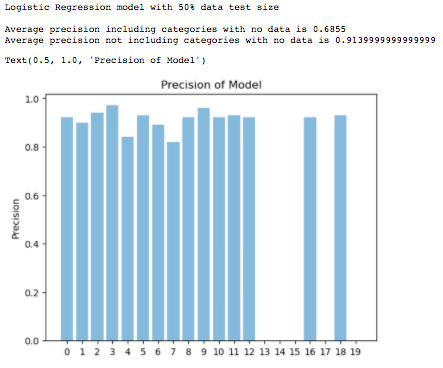
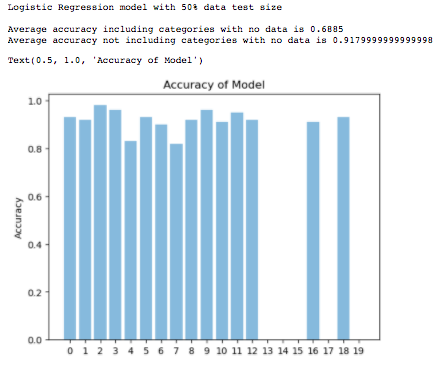
**Malaikatu:** The success of our logistic regression model was evaluated based on accuracy (ratio of correctly categorized jokes and total amount of jokes categorized) and precision (the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives). We changed the size of the training data from 25% to 30% to 50% to see if it would affect the precision or accuracy. With an increase of the training data size, we saw a very small decrease in both precision and accuracy that we can assume is attributed to overfitting. The decrease is extremely slight.

As we ran our model, we saw that some categories did not appear in our testing data. We also changed the size of our testing data to see if a larger testing size would incorporate those missing categories, but increasing the size of our training data did not help this at all.

Overall, the model was successful with an average accuracy of about 92% and an average precision of about 91.5 %. The graphs below illustrate the different joke categories on the x axis (they’re represented by numbers because I couldn’t figure out how to label the x axis without all the words being jumbled) and precision and accuracy on the y axis.

****

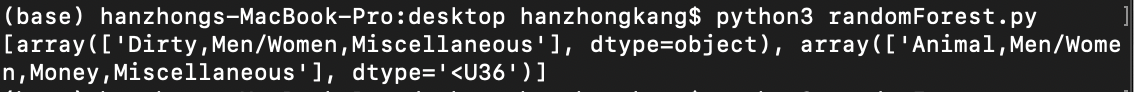
****

****

**Ivan:**

To summarize, we converted the corpus into a tf-idf matrix with the same data cleaning methods as Countvectorizer(removing stopping word and irrelevant character). We also implemented Pipeline from sklearn which works as an integration package of data processing and machine learning modeling. Therefore, we can run a certain type of data processing method and ML model by calling only one object. The results of the two different data processing methods with the Random Forest model is 15% accuracy for the Countvectorizer method and 13% accuracy for the Tf-Idf method.

And the image demonstrates a new joke content input and the predicted categories the model generated. We can see some mismatches of the predicted categories and its original input. The reason behind this is that the data we have is still not comprehensive enough and not large enough to train a sophisticated model due to the complicated nature of semantics.



**Jason:**

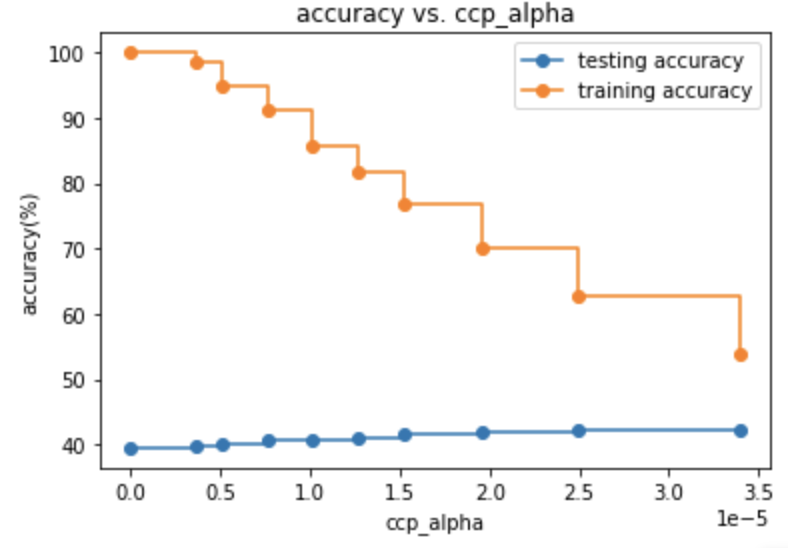
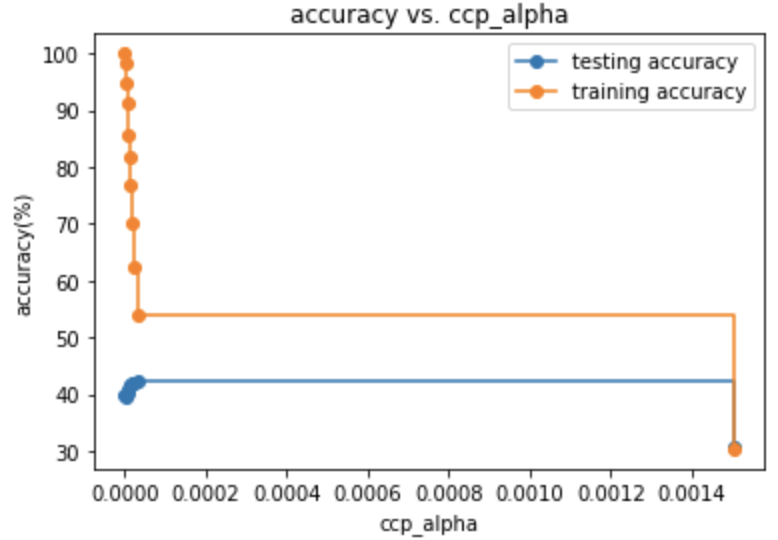
The results of the multinomial Naive Bayes model indicates that this model may not be the best model for joke classification. Although the accuracy for each label is relatively high and mostly above 90%, the overall accuracy is merely 2.33%, while the f1-score is 0.52, which is also not high enough for the model to be an optimal model. A possible explanation would be that the dataset has many labels, so it would be rather difficult to predict the labels perfectly right.

**Hatsune:**

The best accuracy we have achieved with Decision Tree is 42.40%.

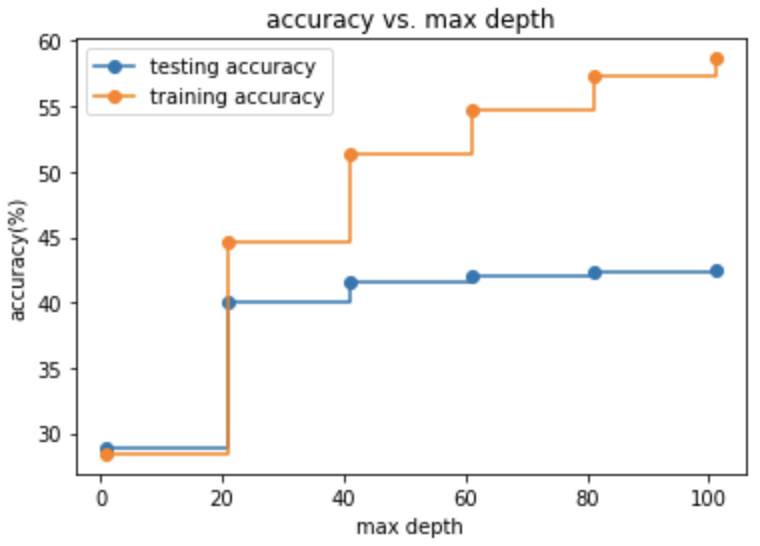
We worked on post pruning to reduce the likelihood of overfitting by finding the optimal value of cost-complexity pruning alpha after we learned the model accuracy stopped growing at some point of depth of a tree in the midterm report. However, it improved the accuracy only by 3.04% from the one before pruning.

In order to determine the optimal value of cost-complexity pruning alpha, we tested the model accuracy with a set of effective alphas in a pruning path of minimal cost-complexity pruning. The below plots show how accuracy of this model varies against different values of cost-complexity pruning alpha.

****

As can be seen from the plot on the right, the optimal value of cost-complexity alpha should fall somewhere in between 0.00002 and 0.000035. After testing a couple of values of cost-complexity alpha, we have chosen 0.0000284 as the value of cost-complexity alpha in our model. This model achieved the accuracy of 42.40% with ccp\_alpha = 0.0000284. It improved only by 3.04% from 39.36%, which is the original testing accuracy before pruning.

We also tried visualizing how the accuracy varies against different maximum depths of a decision tree given that ccp\_alpha = 0.0000284.



As can be seen from the above plot, the improvement of the accuracy significantly slows down after the maximum depth of 20. In order to achieve higher testing accuracy, we decided not to restrict the maximum depth in our model.

In conclusion, we did not achieve an ideal result with this model. One of the possible causes is the unsatisfactory number of observations in contrast with the number of features used in this model. For training, we used 7308 observations, which is equivalent to 80% of the whole data. However, we used the number of occurrences of each word as the features in this model, so the total number of features ended up being 16099, and the number of features exceeded the number of observations substantially. Another possible cause of failures in this model is underutilization of features. We used a set of words which appeared in observations in the features of this model, but each word obviously has a different level of significance. Having treated all words equally supposedly led to the low accuracy of this model.

References:

Chakrabarty, Navoneel & Rana, Srinibas & Chowdhury, Siddhartha & Maitra, Ronit. (2019). RBM Based Joke Recommendation System and Joke Reader Segmentation. 10.1007/978-3-030-34872-4\_26.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. doi:10.18653/v1/n16-1174

Yoon Kim. 2014. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.