SI 670 Final Project Report: Predicting the Value of Jeopardy Questions

Team: Ermioni Carr, Yashaswini Joshi, & Hsin-Yuan Wu

Introduction

Our goal for this project was to predict the value of jeopardy questions based on each questions' text and answer. We initially planned on predicting a set of five values ($200, $400, $600, $800 and $1000) as these represent one of the typical sets of values seen on a Jeopardy board but due to initial results showing poor accuracy (~20-25%) and time limitations, we decided to instead shift our focus to performing binary classification between just $200 and $1000 as these should be the most "different" in terms of the question difficulty. This is based on the assumption that questions with higher monetary values tend to be more difficult. However, there is likely subjectivity in such an assignment since some individuals could view a question as relatively easy whereas other individuals could view it as difficult depending on their level of knowledge on the topic. Therefore, our overarching goal was to utilize machine learning and natural language processing techniques to engineer features that can serve as proxies for question difficulty. Then, these features can be used to develop and train models to predict the value of previously unseen questions. If such a model could be successfully developed, it could be used to verify or assign values to Jeopardy questions for future shows. In addition, we would gain knowledge regarding what aspects of a question make it more "difficult" and if/how these can be potentially represented as features in a model. This knowledge could also be useful in other domains such as designing questions for standardized tests.

Methods

Data

The dataset we used for our project is a jeopardy dataset available at Reddit [1]. The file contains 216,930 Jeopardy questions, answers and other data. For each question, the following information is available: the question category (e.g., "HISTORY"), the value (e.g., "$200"), the text of the question, the text of the answer, the round (e.g., "Double Jeopardy!"), the show number and the show air date. In order to select our subset of the data for analysis, we used the following criteria: (1) only question in the 'Jeopardy!' round, (2) no answers containing numbers or questions containing hyperlinks (to support our use of NLP techniques), (3) only shows numbers after 4000 (to focus on more recent data), (4) only questions with 5 or more words (in order to have enough text data to support feature extraction) and (5) only questions with a value of $200 or $1000 since we decided to focus on binary classification. Then, with the resulting data, we selected a random subset of 2,000 cases for each value. With this subset, we split our data (stratified by value to ensure equal representation of $200 and $1000 in the training and test set) resulting in 3000 training cases and 1000 testing cases.

Feature Extraction

We extracted a range of features from the dataset focusing on the ‘Question’, ‘Answer’ and ‘Category’ columns. The features we extracted include:

- ‘Question Len’: The number of words in the question computed using the lexicon_count function from the textstat library [2]
- ‘Flesch Reading Ease Score’: The readability of the question computed using the flesch_reading_ease function from the textstat library [2]
- ‘Flesch-Kincaid Grade Level’: The grade level associated with being able to comprehend the question computed using the flesch_kincaid_grade function from textstat [2]
- ‘Longest Word (Question)’: The longest word in the question computed using a custom made function
- ‘Longest Word (Answer)’: The longest word in the answer computed using a custom made function
- ‘Average Answer Word Length (Cleaned)’: The average length of the words in the answer (after cleaning text by lowercasing and removing stopwords)
- ‘Answer Len (Cleaned)’: The number of words in the answer (after cleaning text by lowercasing and removing stopwords)
- ‘QA Similarity’: The cosine similarity between the question and answer computed using pre-trained word embeddings [3][4]
- ‘TA Similarity’: The cosine similarity between the category/topic and answer computed using pre-trained word embeddings [3][4]
- Term Frequency–Inverse Document Frequency (TF-IDF): TF-IDF indicates how important a word or terms are in the document. We used 2-gram and removed the stop words to transform the text.
- Word Mover’s Distance: It uses word2vec embeddings and computes distance, which is the most efficient way to move the distribution of sentence 1 to the distribution of sentence 2 [5]. Thus, we computed the distance between questions and answers and the distance between questions and categories.

Preprocessing

Some of the preprocessing methods we experimented with include MinMaxScaler and SimpleImputer. MinMaxScaler was applied to a subset of the features to ensure they were all on approximately the same scale with values from 0 to 1. Therefore, it was not necessary to apply the scalar to columns such as ‘QA Similarity’ and ‘TA Similarity’ that were naturally on this scale but we applied it to most of the other columns such as ‘Longest Word (Answer)’ and ‘Flesch Reading Ease Score’ that all had very different distributions of values. This was achieved using a ColumnTransformer. Next, we also experimented with a SimpleImputer with a mean strategy to account for missing values (primarily in ‘QA Similarity’ and ‘TA Similarity’ columns). Both of these preprocessing steps were fit first to the training data and then applied to the testing data to prevent data leakage.

Models

Originally we planned on focusing the majority of our attention on deep learning models such as CNN and LSTM but as we ended up focusing more on engineering our own features rather than performing textual analysis, we decided to shift our focus to classic machine learning algorithms. With this in mind we experimented with the following algorithms: Linear SVC, Naive Bayes MLP and Random Forest. We tried these models using two main sets of features:

Feature set 1: Using Word Mover's Distance as features, we computed the distance between questions and answers and the distance between questions and categories. Also, we included the average tf-idf value as a feature. With these features, we ran the Naive Bayes with default parameters, RandomForest models (GridSearchCV to optimize value of n_estimators, max_depth and max_features with 3-fold cross-validation) and SVC (GridSearchCV for values of kernel and regularization parameter C).

Feature set 2: Following our feature extraction, a subset of features was selected using the initial results from Random Forest showing the feature importances. This resulted in a final set of 7 features which appeared to be the most predictive: 'Flesch Reading Ease Score', 'Flesch-Kincaid Grade Level', 'Longest Word (Question)', 'Longest Word (Answer)', 'Average Answer Word Length (Cleaned)', 'QA Similarity', and 'TA Similarity'. Using these features, we applied four models: Linear SVC (default hyperparameters), Naive Bayes (default hyperparameters), MLP (GridSearchCV to optimize the value of alpha (the regularization term) using 3-fold cross-validation) and Random Forest (GridSearchCV to optimize value of n_estimators, max_depth and max_features (to help address initial overfitting) using 3-fold cross-validation).

Lastly, we also tried a deep learning model, LSTM. First we trained a tokenizer on all the text. The tokenizer creates a dictionary mapping words to an index, aka tokenizer.word_index. Then it converted the questions (which are strings of text) into a list of lists of integers, each representing the index of a word in the word_index. Then we padded each "list of list" into a single numpy array. To do this, we use the pad_sequences function, and set a maximum length (50 is reasonable since most questions will be at most 20 words), after which any word is cut off. With this we achieved the accuracy of 50%, which is the same as the baseline model we have chosen. For this reason, we chose to focus the majority of our efforts on the classic ML algorithms.

Code

With this report, we have included a few code files detailed below:

- SI_670_tfidf_wmdistance.ipynb: a Jupyter Notebook, running models on Colab, for the feature set 1 with Word Mover's Distance and tf-idf.
- SI 670 Final Project-FeatureSet1.ipynb: a Jupyter Notebook detailing the approaches taken for feature extraction and model building for feature set 2.

- SI 670 Final Project-LSTM.ipynb: a Jupyter Notebook for LSTM model building for feature set 2.

Evaluation and Analysis

As we split our data prior to model development/training, our testing set remained untouched and therefore served as an unbiased estimate of the performance of our models. To measure performance, we used F1 score, precision, recall and accuracy. As our baseline, we chose 50% as this represents how a majority dummy classifier would perform since our data has a 50-50 split between the two value classes (i.e., it would predict all of the test cases as either $200 or $1000). Below is a table comparing the performance of the models on this task:

Feature set 1:

Model 1: Naive Bayes (Default Hyperparameters)
Training Accuracy=53%, Test Accuracy = 54%

NB	precision	recall	f1-score
$200	0.54	0.54	0.54
$1000	0.54	0.53	0.54

Model 2: Random Forest (n_estimators=500, max_features=2, max_depth=3)
Training Accuracy=57%, Test Accuracy = 52%

RF	precision	recall	f1-score
$200	0.51	0.77	0.62
$1000	0.54	0.28	0.37

Model 3: SVC (C=0.1, kernel=rbf)
Training Accuracy=54%, Test Accuracy = 52%

SVC	precision	recall	f1-score
$200	0.52	0.75	0.61
$1000	0.54	0.30	0.38

Feature set 2:

Model 1: Naive Bayes (Default Hyperparameters)
Training Accuracy=56.6%, Test Accuracy = 55%

NB	precision	recall	f1-score
$200	0.54	0.63	0.58
$1000	0.56	0.47	0.51

Model 2: Random Forest (n_estimators=100, max_features=2, max_depth=2)
Training Accuracy = 58.6%, **Testing Accuracy=58%**

RF	precision	recall	f1-score
$200	0.58	0.61	0.59
$1000	0.59	0.55	0.57

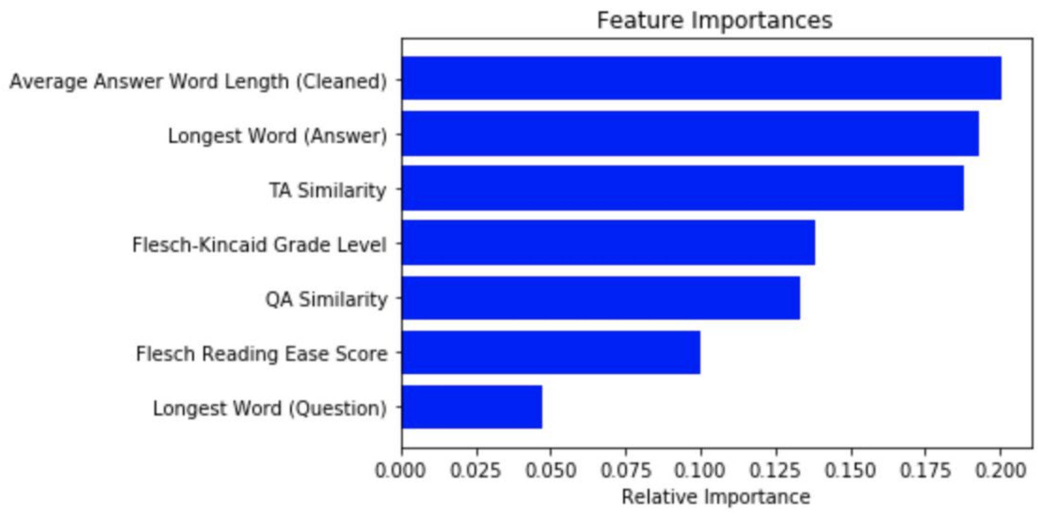
Model 3: Linear SVC (default hyperparameters)
Training Accuracy = 57.1%, Test Accuracy=56.4%

SVC	precision	recall	f1-score
$200	0.56	0.58	0.57
$1000	0.57	0.55	0.56

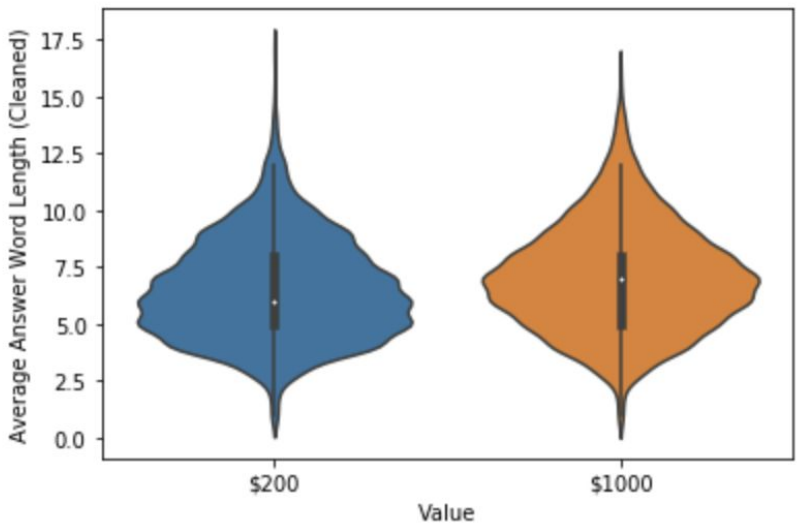
Model 4: MLP (alpha=1e-05)
Training Accuracy = 58.1%, Testing Accuracy=57%

MLP	precision	recall	f1-score
$200	0.56	0.62	0.59
$1000	0.58	0.52	0.55

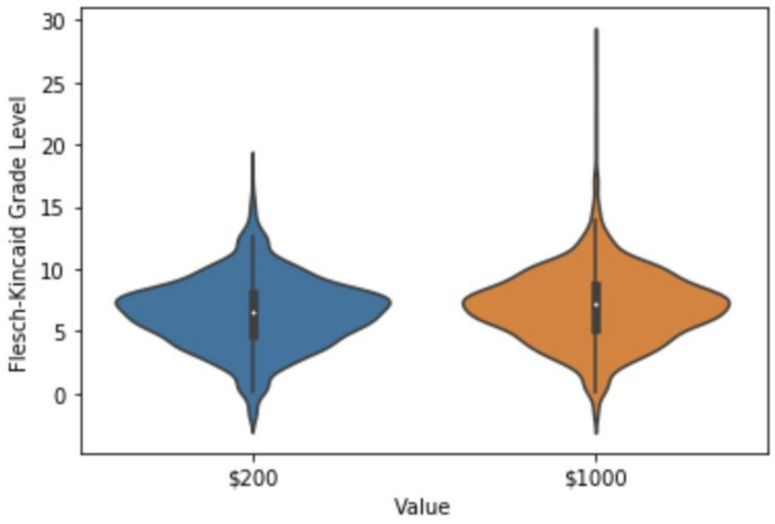
We see that all of the models surpass our baseline accuracy of 50% but the highest accuracy we were able to achieve was for model 2 for feature set 2, the tuned Random Forest model with an accuracy of 58%. Below is the feature importance plot for this model:



Looking at the graph, we see that the most predictive feature in the model was the 'Average Answer Word Length (Cleaned)". Below is a violin plot of the feature comparing questions of value $200 to $1000.



Looking at the violin plot, the distribution of values for $200 and $1000 are relatively similar except that the greatest width of the violin for $200 is slightly lower than that of the $1000 violin indicating that questions with values of $200 tend to have answers with slightly shorter words, on average. This is supported by the mean values (6.53 for $200, 7.06 for $1000). We hypothesize that we see this difference since longer words are likely to be more complex, on average. We see that the next most predictive features are the 'Longest Word (Answer)' (likely for similar reasons as discussed above for the 'Average Answer Word Length (Cleaned)'), 'TA Similarity' (we hypothesize that the category/topic may act as a primer so if the answer is more similar to the topic, it may be less difficult) and 'Flesch-Kincaid Grade Level'. The distribution of 'Flesch-Kincaid Grade Level' is shown below:



We see that the distribution of values for 'Flesch-Kincaid Grade Level' is similar but the tail of the distribution for $1000 is much longer than that for $200. This indicates that there are some $1000 value questions that may use very complex language that would require a higher level of education to comprehend.

Related work

In order to provide necessary context to our proposed project of predicting values of Jeopardy questions, it was worth investigating previous research to identify the pros and cons of each approach. Below are some examples of the existing research on similar problems.

In the paper 'Question Difficulty Prediction for READING Problems in Standard Tests', the authors designed a CNN-based architecture to extract sentence representations for the questions. They then utilized an attention strategy to qualify the difficulty contribution of each sentence to questions. Considering the incomparability of question difficulties in different tests, they proposed a test-dependent pairwise strategy for training TACNN and generating the difficulty prediction value [5]. In our approach we also tried to find the difficulty level/ Reading Ease score to predict the values of Jeopardy questions.

In the paper 'Predicting the Difficulty of Multiple-Choice Close Questions for Computer-Adaptive Testing', the authors explored a machine learning approach to predict the difficulty of a question from automatically extractable features. With the use of the SVM (Support Vector Machine) learning algorithm and 27 features, they achieved over 70% accuracy in a two-way classification task [6]. Additionally, in the paper 'DiffQue: Estimating Relative Difficulty of Questions in Community Question Answering Services', the authors used network structure to predict the difficulty level of the question [7]. Researchers also tried different extraction methods, like Extraction Using a Proximity Metric, SVM-Based Extraction, and their proposed finite state transducers (FSTs) in a question-answering (QA) system [8].

From the above approaches, we can see that there is no significant, obvious solution that we can apply to predict values for Jeopardy questions. Hence, we had to find libraries and approaches that would help us in estimating the difficulty level of the questions to predict their values. In our search, we came across different methods, one of which was by identifying the distance between key words of the questions and the answer using Word Mover's distance [9]. We also used cosine similarity to find the distance, achieving the same level of accuracy.

There was also an approach by Kristiyan Vachev where he identified key words from question text that would decide which words, or short phrases, are good enough to become answers [10]. Lastly, we also came across another article where they used BERT to predict the subjective ratings of question and answer pairs [11]. They used BERT wordpiece tokenizer to generate (sub)word tokens then averaged the token vectors by a neural network pooling layer and used feed forward layers as the classifier. However, we decided to instead use a typical non-neural network-based solution where we used TF-IDF and word-embedding to get the token based vector representations and then used different ML models as classifiers.

Discussion and Conclusion

Overall, we saw that all of our models had poor performance on this task. We hypothesize that this is related to the nature of the task: assigning monetary values to questions is subjective. However, we were at least able to gain some knowledge regarding what features may be (or may not be) predictive of the difficulty, or value, of a question/answer pair. As the Random Forest model performed the best on this task, it leaves us with nice interpretable results as we can visualize the importance of each feature in the model and generate hypotheses to explain our findings. We saw that some features that add predictive value to this task include features related to the length of words in the answer, the similarity between the topic/category and the answer as well as the similarity between the question and the answer, and readability measures such as the Flesch-Kincaid Grade Level and Flesch Reading Ease Score. It is also interesting to note what did not perform well at our task. We saw that deep learning models such as LSTM performed very poorly on this task compared to classic ML algorithms. This is a good lesson that deep learning models are not better suited for every task and for some tasks, engineering predictive features is most important. Potential future work could include working with Jeopardy personnel to understand their process for assigning values to question/answer pairs in the hopes of being able to generate features that mimic their process for value assignment.

References

1. https://www.reddit.com/r/datasets/comments/1uyd0t/200000_jeopardy_questions_in_a_json_file/
2. https://pypi.org/project/textstat/
3. https://github.com/eyaler/word2vec-slim
4. https://stackoverflow.com/questions/22129943/how-to-calculate-the-sentence-similarity-using-word2vec-model-of-gensim-with-pyt
5. Huang, Zhenya, et al. "Question Difficulty Prediction for READING Problems in Standard Tests." *AAAI*. 2017.
6. Hoshino, Ayako, and Hiroshi Nakagawa. "Predicting the difficulty of multiple-choice close questions for computer-adaptive testing." *Natural Language Processing and its Applications* (2010): 279.
7. Thukral, Deepak, et al. "DiffQue: Estimating Relative Difficulty of Questions in Community Question Answering Services." *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.4 (2019): 1-27.
8. Hiyakumoto, Laurie, Lucian Vlad Lita, and Eric Nyberg. "Multi-Strategy Information Extraction for Question Answering." *FLAIRS Conference*. 2005.
9. https://radimrehurek.com/gensim/auto_examples/tutorials/run_wmd.html
10. https://towardsdatascience.com/word-movers-distance-for-text-similarity-7492aeca71b0
11. https://github.com/KristiyanVachev/Question-Generation
12. https://www.sun-analytics.nl/posts/2020-08-04-accurately-labeling-subjective-question-answer-content-using-bert/