Text Similarity

By

Shiva Sanketh Ramagiri Mathad

Dataset and basic analysis of Dataset

The given dataset consists of 7 features and 20347 samples. Each row in the dataset consists of a "Question" and "Sentence" pair as two separate features (columns). There are multiple sentences for each question with the same question being repeated for multiple sentences in

multiple rows. There is another column which contains the “Label” which denotes if the “Sentence” is correct for the corresponding question. There are 2117 unique “questions” and 1039 “positive labels” for sentences indicating that few of the unique questions have no positive label associated with it. There are also questions with multiple correct sentences associated with it.

There are 1039 positive(1) Labels and 19308 Negative(0) Labels representing the data to be skewed. Some of the ways by which skewed data can be handled are by collecting more data, trying different approaches, resampling techniques, Synthetic data generation, using application specific metrics, etc. For this challenge collecting more data, resampling techniques or Synthetic data generation is not apt. Trying different approaches with suitable metrics would be the right way to move forward.

Missing Values

The dataset does not consist of any missing values. All the values in “Label” are binary.

Aim

The aim of the challenge is not definitive but rather open - to derive useful insights from the dataset which consists of Question/Answer pairs.

Feature Engineering - (preprocessing.py):

The dataset is converted to a Pandas dataframe for easier handling of data. Text data from “Question” and “Sentence” columns are considered and converted to lowercase. Later Regular Expression conversion, Stop word filtering and lemmatization are applied.

Another function is defined which excludes all the “Question” and “Answer” pairs which have no single positive label for a given unique “Question”. This means that for a given group of same “Questions”, if there are no positive “Label”, all are excluded. This function is not applied for all the models but for specific models.

Metrics

3 metrics are used to evaluate the models.

1. **Sensitivity score (Recall):** It is the ability of a test to correctly identify those that are true positives.

   Sensitivity = True Positive/(True Positive + False Negative)
2. **Precision score:** It is the ratio of correctly predicted positive observations to the total predicted positive observations

Precision = True Positive/(True Positive + False Positive)

3. **F1 Score:** It is a combination of Precision and Recall

F1 = 2 * Precision * Recall /(Precision + Recall)

Using the right metric depends on the application. As there is no specific application, the above mentioned three metrics can be used.
For Eg:

1. If an application requires us to capture all the positive cases without missing any positive cases, Sensitivity score is better. It is used when the cost of missing a positive is problematic than the cost of including a negative
2. If we want to avoid detecting a case as positive when it's actually negative but vice versa is fine, we go for Precision.
3. F1 Score is best if there is some sort of balance between precision & recall in the system.

Approaches

I follow different approaches which perform different tasks in order to determine an efficient approach for this dataset.

1. Jaccard Similarity considers which members are shared and which are distinct. Jaccard similarity is good for cases where duplication does not matter.
2. Cosine Similarity considers cosine of angle between two vectors. Cosine similarity is good for cases where duplication matters while analyzing text similarity.
3. Word Net consides Semantic similarity between the sentences.
4. Word Mover's Distance considers Semantic and syntactic approaches to get the similarity between the texts

These are four different approaches, these approaches combined with different feature extraction techniques gives us a wide range of direction to select and move towards in order to find the most similar text approach.

Jaccard Similarity - (jaccard_similarity.py)

It gives the ratio of size of intersection to the union of two sets. This is a basic approach to find Text Similarity. It compares members for two sets to see which members are shared and which are distinct. Although it's easy to interpret, it is extremely sensitive to small samples sizes and may give erroneous results.

Cosine Similarity - (cosine_similarity.py)

It calculates similarity by measuring the cosine of angle between two vectors.Cosine similarity is advantageous because even if the two similar documents are far apart by the

Euclidean distance (due to the size of the sentences), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity. To calculate Cosine Similarity, we need to convert sentences into vectors.

Two Feature extraction methods for Cosine Similarity

1. Countvectorizer with ngram = (1,1) + Cosine Similarity
2. Countvectorizer with ngram = (1,2) + Cosine Similarity

Ngram gives the number of words to be considered in a sequence. Ngram_range(1,2) considers both sequences with 1 word as well as sequences with 2 words. Higher numbers of ngram considers more words together and could lead to a better similarity coefficient.

WordNet based similarity - (wordnet_similarity.py)

Using the WordNet database and NLTK module to find synonyms, antonyms, etc of words and a similarity value is determined based on the most similar word and this is averaged over the entire sentence. This is a knowledge based measure of similarity. WordNet focuses on Semantic similarity between the sentences.

Word Mover's Distance (w2v_wordmoverdistance.py)

It calculates the distance between two documents as an optimal transport problem between the embedded words. It targets both Semantic and syntactic approaches to get the similarity between the texts. Word2Vec is used in order to process Word Mover's Distance.

Two Feature extraction methods for Word2Vec

1. CBOW (Common Bag of Words): The current word is predicted from a window of surrounding words
2. Skip gram (SG): The current word is used to predict the surrounding window of context words

Evaluation and Performance:

For evaluation purposes, a random label is generated. As the data is imbalanced, random labels cannot contain a random number of positives and negatives. I have adopted a method in which the random labels consist of the same number of positives as the actual labels (1039) but spread across in a random fashion. This method is not the best method as it still derives some knowledge of the labels from the original dataset. But this method can be used as a significant amount of randomness still exists.

Two approaches are followed through the implementation.

1. One is to run all the algorithms on the entire dataset.

2. Another approach which I have adopted is to eliminate all the groups of questions which have absolutely no correct sentence (This is usually associated with “exc_unrelated” in the code).

Summary of Performances:

Performance on Complete Dataset:
CV - CountVectorizer
W2V - Word2Vec
CBOW - Common Bag of Words
SG - Skip Gram

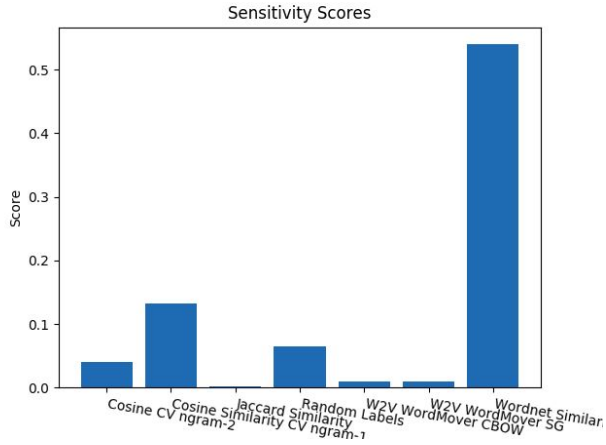
Method	Cosine CV (1,1)	Cosine CV (1,2)	Jaccard Similarity	Random	W2V WordMover - CBOW	W2V WordMover - SG	WordNet
Sensitivity	0.132	0.040	0.0019	0.065	0.0096	0.0096	0.539
Precision	0.17	0.181	0.08	0.065	0.072	0.072	0.108
F1	0.150	0.066	0.003	0.065	0.016	0.016	0.180

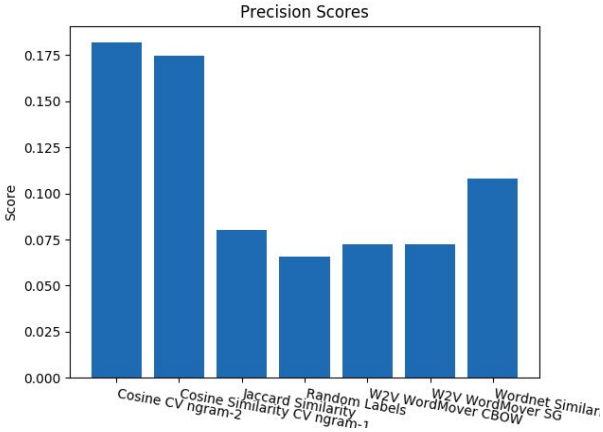
Performance on dataset excluding complete irrelevant question groups:

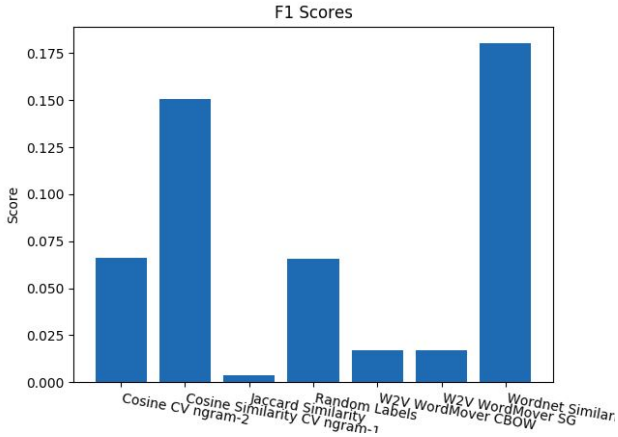
Method	Cosine CV (1,1)	Cosine CV (1,2)	Jaccard Similarity	Random	W2V WordMover - CBOW	W2V WordMover - SG	WordNet
Sensitivity	0.132	0.0404	0.0019	0.142	0.0038	0.0038	0.5399
Precision	0.312	0.291	0.117	0.142	0.108	0.108	0.215
F1	0.186	0.071	0.0037	0.142	0.0074	0.0074	0.308

Bar Graph Plots comparing performances:

Performance on Complete Dataset:

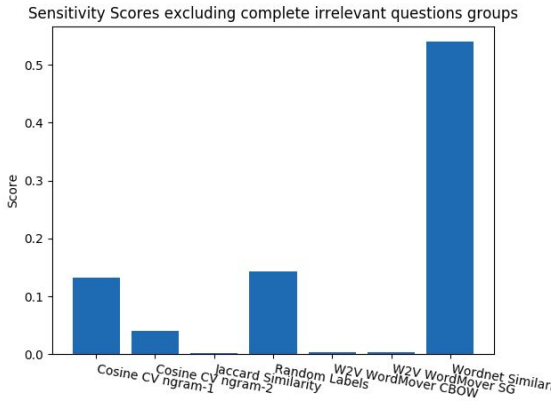


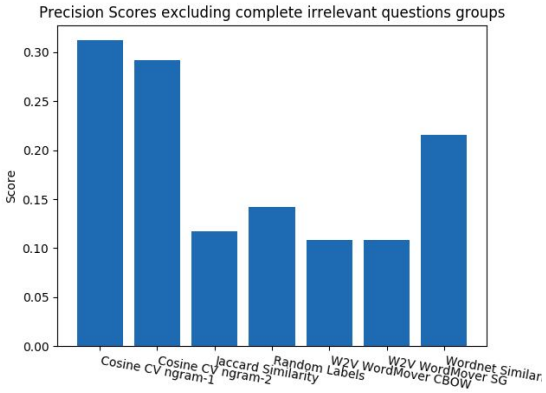


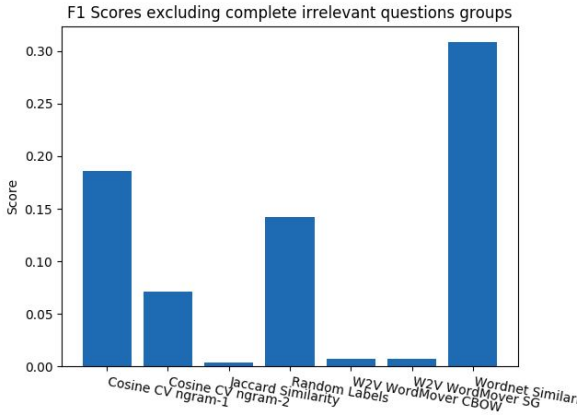


We can observe from the above performance comparisons that **Word Net** gives a better Sensitivity Score, **Cosine CountVectorizer(CV) with ngrams_range (1,2)** gives a better Precision Score and **Word Net** gives a better F1 Score. Both Cosine Similarities are pretty close in Precision score.

Performance on dataset excluding complete irrelevant question groups:







We can observe from the above performance comparisons that **Word Net** gives a better Sensitivity Score, **Cosine CountVectorizer(CV) with ngrams_range (1,1)** gives a better Precision Score and **Word Net** gives a better F1 Score. Both Cosine Similarities are pretty close in Precision score.

Conclusion and Future Scope

The approach we have arrived from the above observation concludes that **Word Net** gives a better Sensitivity Score, **Cosine CountVectorizer(CV) with ngrams_range (1,2)** gives a better Precision Score and **WordNet** gives a better F1 Score. This is based on the entire dataset calculations.

Using the right model from this depends upon the application. As mentioned above

1. If an application requires us to capture all the positive cases without missing any positive cases, Sensitivity score is better. We can chose WordNet for these kind of tasks
2. If we want to avoid detecting a case as positive when it's actually negative but vice versa is fine, we go choose **Cosine CountVectorizer(CV)** with higher ngrams for this.

3. F1 Score is best if there is some sort of balance between precision & recall in the system. We can choose **WordNet.**

These are not the final methods that we should conclude upon. We know that WordNet performs better with sensitivity score and F1. Word Net considers semantic similarity. Word Net is a knowledge based approach. We can work with other similar algorithms which focus on semantic similarity between texts such as BERT, word embeddings such as GloVe etc.
When we have to consider Precision, Cosine similarity performed better. So we can concentrate more on this variation such as using different ngrams for Cosine Similarity. We can also work with different word embeddings such as Word2Vec, TF-IDF with Cosine Similarity.

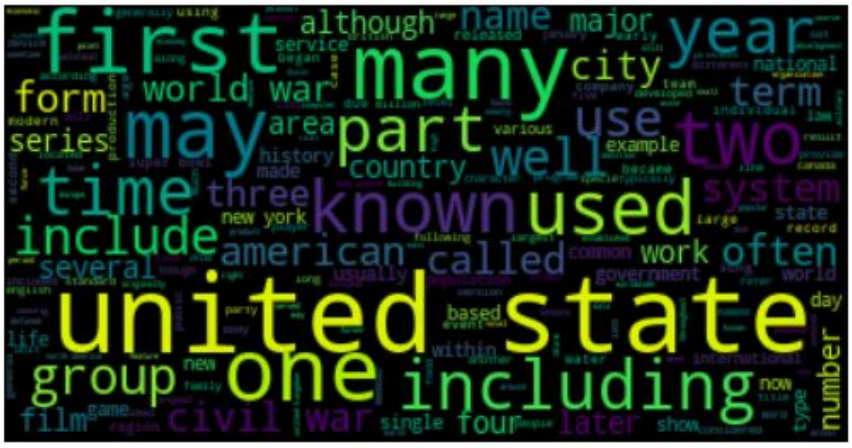
If I had some more time, I would have taken the direction mentioned above such as BERT, GloVe, InferSent or Cosine similarity with different Vectorization techniques to arrive at a more efficient algorithm. Along with this, I would have also trained a RNN based Supervised learning model as RNNs perform better on sequential data.

Few of the other areas to concentrate on:

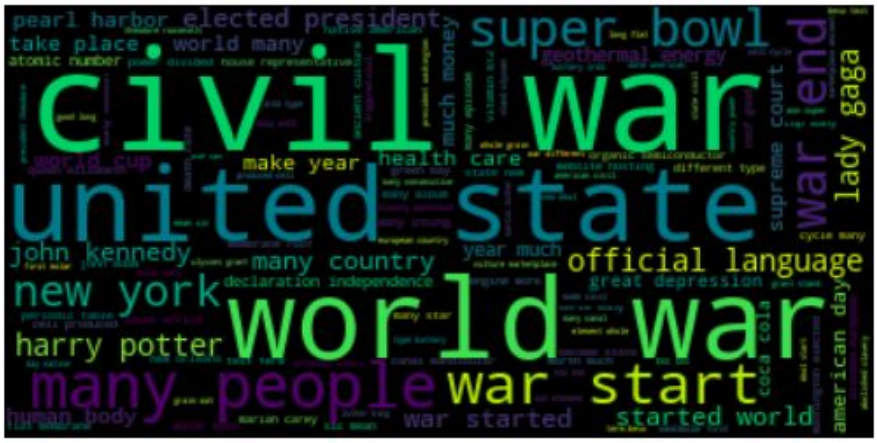
1. To find the dis-similar sentence for a query and finding patterns in these kinds of false responses.
2. Combining two correct answers to provide a more accurate answer.
3. Extracting partial correct answers if they are present in incorrect answers.
4. Extracting partial incorrect answers if they are present in correct answers.

Some Cool Visualizations

Word Cloud on Entire Question/Answer pair.



Word Cloud on Questions.



Word Cloud on Answers.