Brian Fay, Jarius Thomas, and Stokley Voltmer

Machine Learning
D Term 2022

# U.S. Patent Phrase to Phrase Matching

## Introduction

For this Kaggle competition, we are trying to predict (on a scale of 0.0 to 1.0) how related two phrases are to one another. The description provided these thresholds for similarity between a target phrase and a more general anchor phrase:

1.0 - very close match
.75 - close synonyms
.5 - synonyms which don't have the same meaning
.25 - somewhat unrelated
0.0 - Unrelated

This is a semantic similarity challenge (expand quickly on what this is). It differs from other semantic similarity challenges because the relationships between words, phrases, etc. are dependent on the context they're used in. The relationships can significantly change when the context changes. We are given a train and a test set with multiple anchor phrases, target phrases, and contexts:

i) Train Set - Size: (36,473 x 4)
4 columns include anchor phrase, target phrase, context, and semantic similarity scores for every anchor/target pair.

ii) Test Set - Size: (36 x 3)
3 columns include anchor phrase, target phrase, and context.
Does not contain any score (Y) data.

This problem excited our project team; in the planning and competition selection phase, team members expressed an interest in trying a challenge involving an application of NLP or word vectors.

## Methods

For the shallow model, we begin by converting the phrases to word vectors; to do this we make use of an existing framework, namely the spacy lemmatizer. The spacy lemmatizer is a more powerful version of the typically used stemmer algorithms that remove the ends of words, e.g. examples to example. Unlike normal stemmers, the spacy lemmatizer tags the words with their part of speech and intelligently reduces the words to their root. Additionally, the preprocessing pipeline that we created uses spacy to remove stopwords. After the phrases have been vectorized, they are converted into tf-idf vectors. Tf-idf vectors are vectors that have been augmented by their term frequency inter-document frequency, that is the frequency that a word occurs within a document as well as within the set of documents. By this manner, a document that occurs often within the set will be given less weight.

The context values were converted into oneHot encoded vectors using the scikit-learn OneHotEncoder, this was chosen over ordinal encoding to ensure that when regression or principal component analysis was performed, the values of context would not weight it unfairly (as there is not true distance between the values). After the encoding the DataFrames were transformed into an array for principal component analysis. Principal component analysis was performed using the scikit-learn PCA algorithm.

The similarity for the shallow method was predicted straight from the tf-idf vectors by comparing their cosine similarity. The cosine similarity gives the same type of result as is used in the problem, where 0 denotes orthogonal vectors, and 1 denotes parallel vectors. Cosine similarity for two vectors A and B is calculated with the following equation where n is the length of the vectors A and B.

$$\frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Introduce the Deep NN model, and pivot to how we form/encode the input data in loadData()

Our deep learning model made use of the third party libraries Tensorflow and Keras, as well as a freely available dictionary of 40,000 words called GloVe. Within this system, each word in the X matrix (be it an anchor, target, or context string) is encoded - the word is replaced with an int representing that word’s index in GloVe’s dictionary. After encoding, we used Keras to pad every list of words in x. For example, if there exists an anchor string defined as a list of 5 words, in our system this list becomes [_PAD_, _PAD_, _PAD_, _PAD_, _PAD_, (the five words) ] if the padding length is 10. For our purposes, we set the padding length to 5 to minimize the sparsity in the data set. The descriptions for CPC codes vary in length from 1 word long to over 20 words long, but a padding size of 5 allowed us to train our model on inputs with consistent, uniform size despite this variance.

The deep learning model had a siamese BiLSTM architecture. The same encoding network was used to encode each sequence of word vectors into a 64 node long tensor. The two encoded phrases were then subtracted from each other. The patent category in which the comparison is made was represented as word vectors after the code was converted into its description in words, this was then feed into another encoding network with a similar structure to but different weights than the one used to encode the phrases. The encoded context and the subtracted encoded phrases are concatenated before being fed into the deep neural network. The neep neural network consists of deep layers with dropout layers in between to increase the robustness of our model. The final output was a single value. The value was bound between 0-1 with sigmoid activation, and a gaussian noise layer over the single value prevented overfitting to the incremental training labels. Tanh activation was used for the deep layers so that certain contexts could have a negative impact on the similarity of phrases. Binary cross entropy was used as a loss function when training the model as it is suitable for a probabilistic binary classification problem such as this.

In order to check the performance of our data we had our program print out the 10 most incorrect classifications in the testing data, shown in appendix A. This was helpful in our hyperparameter optimization.

To create our 3-layer NN, we adapted the Deep NN to use only one hidden layer. The second function returns a model made by concatenating all inputs together, instantiating a model compatible with this architecture.

## Results

Unfortunately, the vectorization and the principle component analysis were not very successful. Through TF-IDF we were only able to achieve a percent correct of 16.79, and our two principal components were only able to account for around 2 percent of the variance each. Although, while these results are unfortunate, they are not at all unexpected. One problem with this very basic implementation of tf-idf is that it is not actually able to deal with context; while the lemmatizer theoretically will clean the phrases based on their context(e.g., their part of speech), in practice the phrases are too small to be properly lemmatized. This leads to the same words being turned into different words; additionally, because the phrases are so small, it may not be the case that lemmatizing is a good idea in the first place. It is possible that because the phrases are so small, every bit of context matters, and the lemmatization smooths everything out too much. This smoothing action would go some way to explaining why we have much higher levels of similarity than are witnessed in the training set, as seen in figure 7.

## Table of Results

| | | |
|---|---|---|
| TF-IDF | 0.186 MSE | 16.79 PC |
| 3-layer NN | 0.0650 MSE | 20.56PC |
| DNN | 0.0422 MSE | 22.74 PC |

## Conclusion

Within each phrase, the team used a dictionary to replace the list of words with a Tensor encoding of integers referring to the words' indices in the dictionary. This worked well for the most part, but encoding certain words within some of the context strings became more complex than anticipated. Some words in these context strings were not recognized by our vocabulary dictionary, many of which were chemistry terms (i.e. words like "Carbocyclic", "Rizatriptan", etc). We set these to unknown, and we believe these unknown entries are negatively affecting the accuracy shown.

For our purposes the deep neural network model worked best. Keras and TensorFlow proved highly effective for processing and optimizing our hyperparameters, though on some machines the processing can take upwards of two to three minutes.

## References

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Appendix

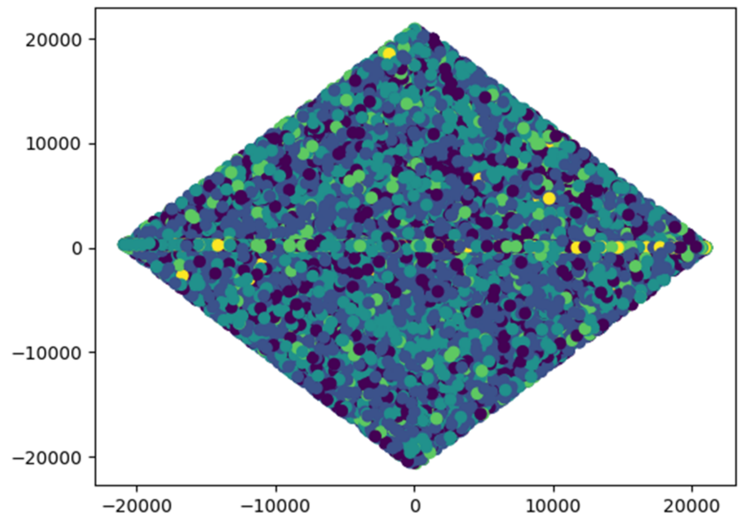

Image 1. PCA of the data with the fields ordinally encoded.

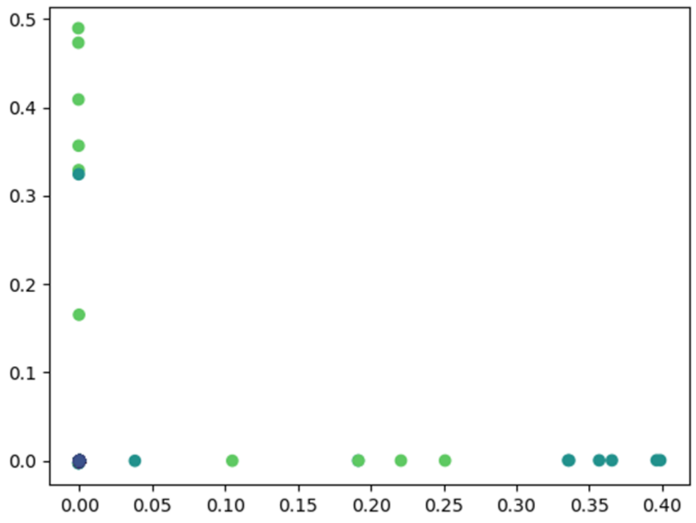

Image 2. Kernel PCA of the data with the fields ordinally encoded. The kernel is 'rbf', number of components is 2, gamma is 10 and alpha is 0.1.

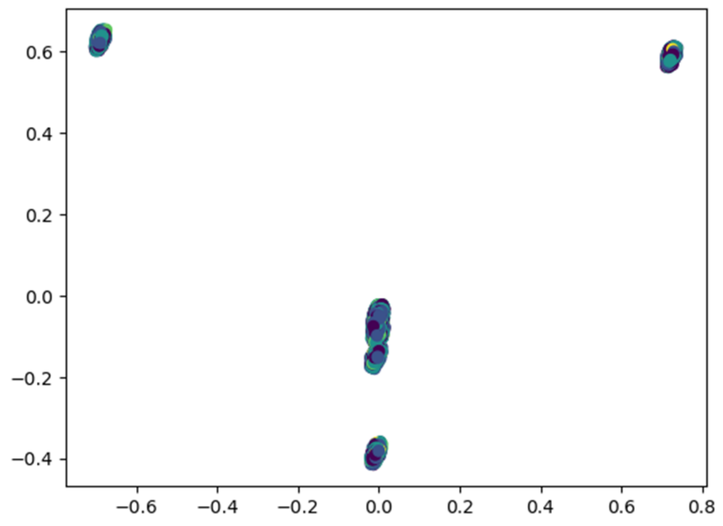

Image 3. Basic PCA of the vectorized data.

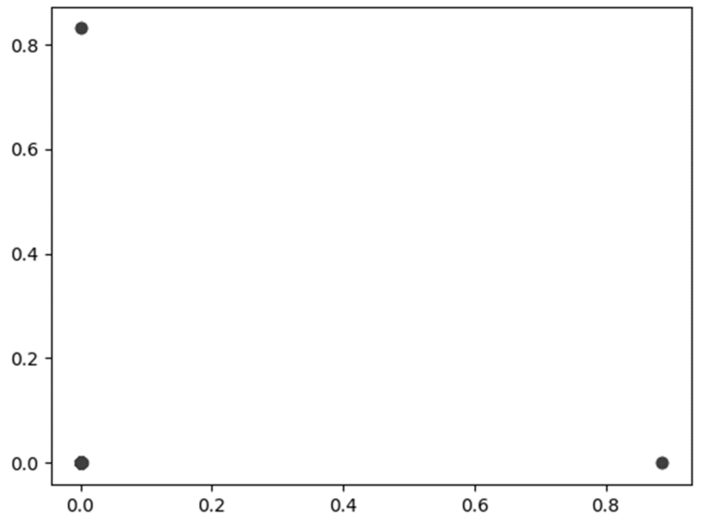

Image 4. Kernel PCA of the vectorized data using the same parameters as those used in Image 2.

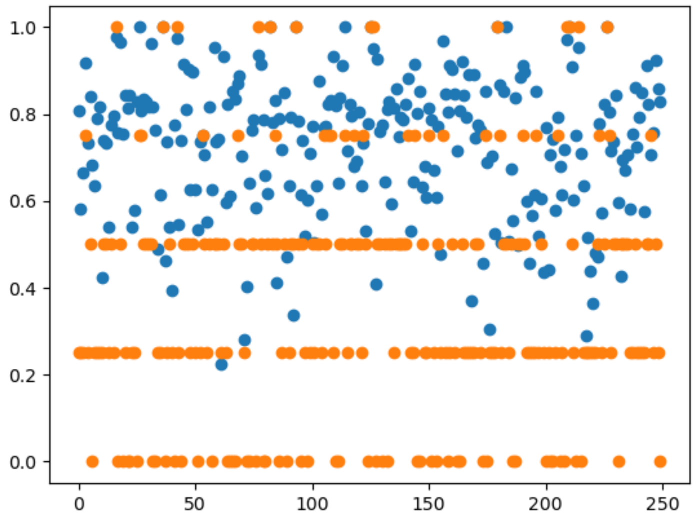

Image 5. Results of calculating the cosine similarity using the vectorized phrases; the orange values are the true values and the blue values are the cosine similarity.

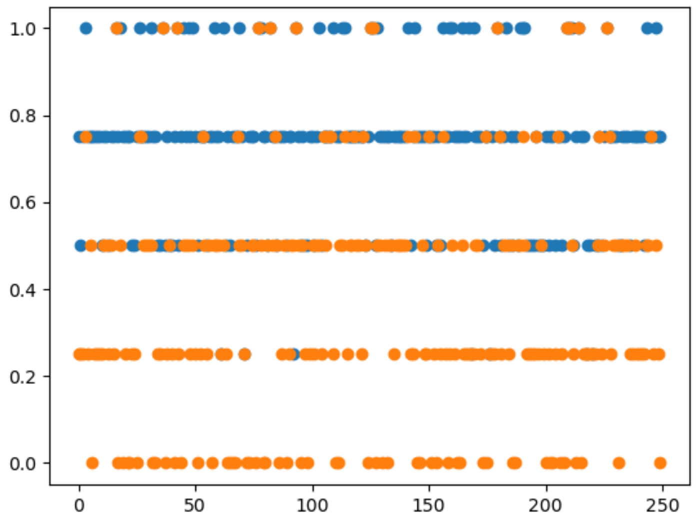

Image 6. Predicted values from cosine similarity are in blue and actual values are in orange; the cosine similarity values from Image 5 have been rounded to the nearest category.

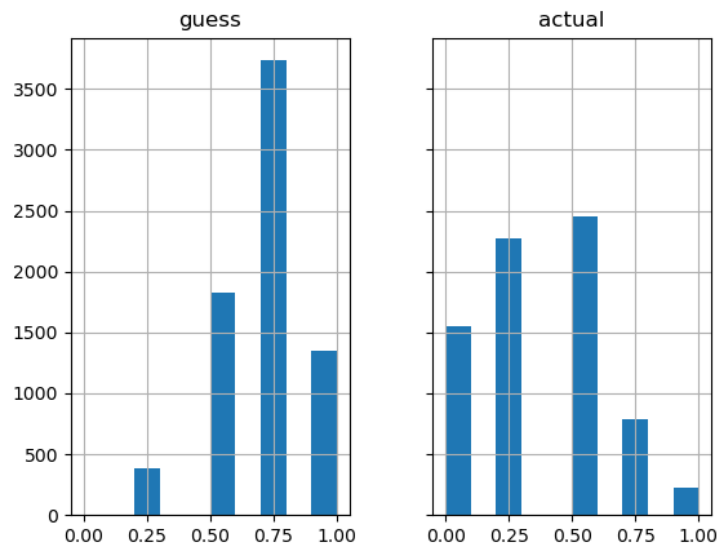

Image 7. Histogram of the occurrences of the guessed values and the actual values, of particular note is that with the cosine similarity we never predicted a zero value.

anchor data:

| Anchor | Target | Context | Prediction | Real | Delta |
|---|---|---|---|---|---|
| ['' '' '' '' 'verifiable'] | ['' '' '' 'verified' 'move'] | ['' '' '' 'measuring' 'testing'] | 0.96906471 | 0 | 0.96906471 |
| ['' '' '' 'expandable' 'intraluminal' ] | ['' '' '' 'expand' 'dna'] | ['medical' 'or' 'veterinary' 'science' 'hygiene'] | 0.96081781 | 0 | 0.96081781 |
| ['' '' '' 'tap' 'portion'] | ['' '' '' 'faucet' 'tap']* | ['machine' 'tools' 'metal-working' 'not' 'otherwise'] | 0.83255786 | 0 | 0.83255786 |
| ['' '' '' '' 'ack'] | ['' '' 'indications' 'of' 'cystoscopy'] | ['' '' '' '' 'horology'] | 0.82515627 | 0 | 0.82515627 |
| ['' '' '' 'encapsulated' 'pigment'] | ['' '' '' 'pre' 'registration'] ] | ['' 'paper-making' 'production' 'of' 'cellulose'] | 0.79478014 | 0 | 0.79478014 |
| ['' '' '' 'vertical' 'chute'] | ['' '' 'block' 'movement' 'lock'] | ['conveying' 'packing' 'storing' 'handling' 'thin'] | 0.7815116 | 0 | 0.7815116 |
| ['' '' '' 'acoustooptic' 'modulator'] | ['' '' '' 'polarizing' 'film'] | ['' '' 'basic' 'electric' 'elements'] | 0.7760998 | 0 | 0.7760998 |
| ['' '' 'moisture' 'proof' 'film'] | ['' '' '' 'modulator' 'enzyme'] | ['electric' 'techniques' 'not' 'otherwise' 'provided'] | 0.76225746 | 0 | 0.76225746 |
| ['' '' '' 'friction' 'lock'] | ['' '' '' 'chute' 'drain'] | ['land' 'vehicles' 'for' 'travelling' 'otherwise'] | 0.99086541 | 0.25 | 0.74086541 |
| ['' '' '' 'pre' 'trip']] | ['' '' '' 'encapsulated' 'nail'] | ['' '' 'basic' 'electric' 'elements'] | 0.7404263 | 0 | 0.7404263 |

Table A - the 10 least accurate predictions from our Deep Neural Network after training