Pristine Sentence Translation: A New Approach to a Timeless Problem

Meenu Ahluwalia, Brian Coari, and Ben Brock

[1]Master of Science in Data Science
Southern Methodist University
Dallas, Texas USA
{mahluwalia, bcoari, bbrock}@smu.edu

Abstract. Translating text from one language to another is a continuous technological challenge. Although many technologies, such as Google Translate, have used machine learning and neural networks to close the translation gap, there are still many translation problems to be solved. Issues such as multiple word meanings, proper sentence structure, slang, colloquialisms, and determining the literal meaning of words vs contextual intent of those words are areas where we sometimes still see Google Translate struggle. In this paper we explore an original strategy that provides a solution to these translation issues, demonstrate a proof-of-concept of the solution, and examine the feasibility of a large-scale solution. For our translation solution we populated a database with translations of entire sentences from one language to another, instead of the words in a sentence. Since a sentence represents an entire thought instead of an assembly of words, the translation did not suffer from the issues that plague Google Translate. We also used Natural Language Processing (NLP) and predictive modeling in order to find sentences close to the sentence requested, which provides the user examples of common grammatically-correct sentences. With these approaches we were able to translate sentences that seemed impossible using traditional translation methods.

1 Introduction

The ability to easily communicate with people in another language is one of the most powerful and satisfying experiences in life. Technology has come a long way from the discovery of the Rosetta Stone in 1799, which allowed us to translate Egyptian hieroglyphics to ancient Greek in a mere 23 years. In the modern day, tools such as Google translate can be used in real time to convert between languages and allow people to connect from different cultures [6]. The latest iterations of Google Translate even use machine learning and neural nets to parse more than just single words, delivering a more satisfying user experience [7].

As far as we have come, however, the areas where we struggle are still painfully obvious. While Google Translate will usually allow you to find a bathroom and order off a menu, the intricacies and complexities of a normal, native conversation still can cause a non-fluent speaker issues. For example, if an American coworker mentions to a Brazilian coworker about their performance on a project with "You hit one out of the park.", the Brazilian coworker could translate the words, but without familiarity with the context of a baseball game, the Brazilian would be confused and would have to ask for clarification if it was possible. It would be even harder if the Brazilian was reading a book in English with a colloquialism, since there would be no human to ask for help.

If you consider these kinds of issues from a high level they might seem unsolvable. How can you train a translation tool to look at the meaning behind sentences using on the words provided? We think we have a possible answer: an original concept we are calling Pristine Sentence Translations (PSTs). The concept of PST is that instead of translating words or phrases using neural nets and machine learning, we simply store an entire sentence in a database, and we have entire sentences in other languages that represent the meaning of that sentence.

For example, using the example above we would have an entry for the English sentence "You hit one out of the park", and we would have an entry for a Portuguese sentence mapped to that English sentence that says "Você foi ótima" which translates in English to the meaning behind the phrase: "You did great". For another example, in Portuguese there's a sentence "Eu adoro Cafuné" Google Translate does not have a translation for "Cafuné", because it's a complicated word which loosely means "the act of running fingers through hair" . Our program's goal is to return an English translation "I love the feeling of fingers running through my hair" when asked to translate "Eu adoro Cafuné" into English. Using this method there is no sentence or concept we will not be able to translate into another language given enough time and resources.

One main issue with the approach outlined above is that if we do not have an exact match for the sentence, our method return nothing. so if we tried to translate "You really hit one out of the park" from English into Portuguese we would not get any results. We decided to address this concern using Natural Language Processing (NLP) to filter out the noise in a sentence, and then use Predictive Modeling in order to find the sentence "most like" the input sentence. Using this method, "You really hit one out of the park" would ideally map most closely to "You hit one out of the park", and return the same translation: "Você foi ótima". The front-end will indicate that the translation is not for the original input sentence, instead it will indicate that it is "Showing Results for: You hit one out of the park."

Due to the strictly educational and academic nature of the project, we are not attempting to provide a full translation solution. We will limit our translations to English, Portuguese, and Hindi, and we will only provide translations for a few hundred phrases. This will be sufficient to demonstrate the appeal and power of this technique, and we will show how this solution could grow into a complete, living solution using crowdsourcing and time.

Given the nature of PSTs we will need to find a way to determine the closeness of sentences to each other. We will analyze two different methods of determining how close one sentence is to another, Euclidean Distance and Cosine Similarity, and choose the best method based on the metrics it provides for closeness. After we determine the closest match of an input sentence to one we know through our PST database, we can return a curated translation that matches that sentence in the requested language.

By the end of this paper, hopefully we can convince the reader of the potential value of this approach to translations and inspire future work that build on our base proof-of-concept into a more robust translation mechanism.

1.1 Existing Tools and Methods

The concept of translating one langauge to another is one of the most prevalent topics in data science, so it was all too easy to find existing tools and methods. This first example we would like to cover is one of the stronger forces in translation today: Google Translate. In 2016 Google started using Neural Nets to perform translations and saw vast reduction in errors. [18]. Even though this method was vastly superior to Google's previous system it is still dependent on the individual words in a sentence, and still subject to the same issues stated above with translations, since it would not handle colloquialisms or wirds that have no direct equivalent in the destination language.

The next method we found was much closer to our intent. The outhor developed a Neural Machine Translation System using Keras, and used a "closeness of match" system to translate between one language and another. [16] This approach is almost perfect for what we need, with the exception of only returning one sentence in the intended language. We would like an approach that allows us to find more than one sentence ID from the destination language. PST will use a similar approach, but aims to find a matching sentence ID for the input sentece, then lookup all sentences that are translations for that sentence in the requested language. More details will be provided in the "Solution Approach" section.

2 Solution Approach

Our approach to solving the issues around language translations takes five main steps. Each step is discrete and can be run and unit-tested independently. The "Full Demo" section will go over what the steps look like and how they relate to each other in a full iteration of the program.

First, we will form a pristine database of known, correct, contextual translations to sentences that focus on translating the intent of sentences rather than the word in that sentence. There is more detail on this step in the "Data Collection" section.

Second, we will take input from a user on the requested translation and the languages involved in the translation. This is covered in more detail in the "Full Demo" section.

Third, we plan to use a sequence to sequence model to find as close a match as possible to the input sentence and its associated translation. This section is still in progress in this iteration of the document, we will use it to verify our approach in step four which meets functional expectations for the project. More details of that can be found in the "Encoder-Decoder Long Short-Term Memory (LSTM) Networks" section.

Fourth, we will measure the closeness of our guessed sentence to our input sentence using Euclidean Distance and Cosine Similarity and return the result of the method that we determine is the best performing indicator, along with alll sentences that relate to the matched sentence to the front end.This is covered in more detail in the "Euclidean Distance and Cosine Similarity" and "Results and Analysis" sections.

Fifth and last, we will display our results to the user through the front end. This is covered in more detail in the "Full Demo" section.

2.1 Database Design

For our database design we wanted to leverage a cloud-based storage system so that we could both reference the same live database, as well as making the database updateable in realtime in case we want to utilize a realtime update strategy in the future. We chose Google Coud Platform (GCP) as a host for our database due to its ease of setup and robust set of no-cost features.

Our GCP database consists of two tables:

1. Sentences: Stores a unique Sentence ID for a combination of text (max 500 characters) and a language key (EN=English, PT=Portuguese, HI=Hindi). There is also a place for a suggested replacement for possible future enhancements, but this is not used in our current version of the program.
2. Translations: A table that stores the Sentence IDs that are translations of each other, linking the sentences together. These Sentence IDs are foreign keys that link to Sentence IDs in the Sentece table

The use of the Translations lookup table allows us to associate any number of sentences to each other, so we can have more than one translation for a sentence, and we can maintain translations to sentences only that match up together. In the fuure we might have sentences in English that only have translations in Hindi, but not Portuguese, for example.

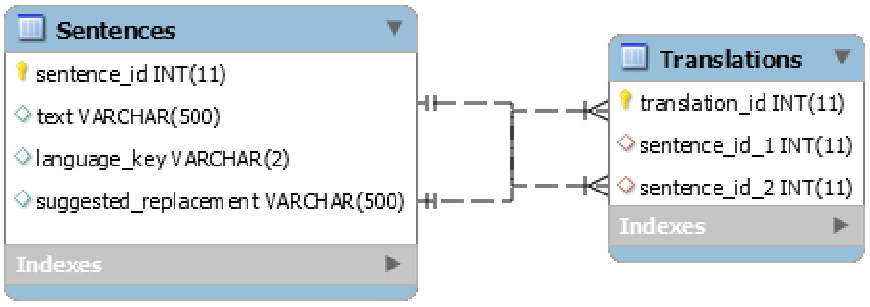


Fig. 1: Database Diagram

One of the problems we wanted to solve early was to be able to test our model offline, in case a network connect occurred we still wanted access to our model. To facilitate this we extracted the data in the database into dataframes through sqlalchemy [15] and exported the data to .pkl files. We then load the .pkl files whenever we need the data for a model, which makes us immune to network interruptions. Examples of the SQL Queries used to extract the data can be found in Appendix A.

2.2 Data Collection

The data for this project was collected to help people who travel for businsess or pleasure. If someone who can speak English is travelling to India, though most Indians know some English, he will face a few communication issues when conversing with waiter, taxi driver, hotel staff etc. The initial collection of 200 English Sentences are valuable to know when travelling. The sentences were assembled in English from three travel websites. [11,12,13] and translated to Potuguese and Hindi using Google Translate [14]. The translations were verified manually by a human who is proficient in Portuguese and another one in Hindi. If the translation done by Google did not make sense, the translation was modified to make it more sensible. The list of sentences and translations can be found in our public Github Repository:

`https://github.com/coarib/SMU_Masters_PST/blob/master/data/processed/PST_workbook.xlsx`

3 Predictive Modeling

Automatic or machine translation is one of the most challenging AI tasks given the fluidity of human language. Classically, rule-based systems were used for this task, which were replaced in the 1990s with statistical methods. More recently, deep neural network models achieve state-of-the-art results in a field that is aptly named neural machine translation. [16]

Sequence to Sequence (often abbreviated to seq2seq) models are a special class of Recurrent Neural Network architectures typically used (but not restricted) to solve complex Language related problems like Machine Translation, Question Answering, creating Chat-bots, Text Summarization, etc. Our aim is to translate given sentence from one language to another. We will target sentence translations to and from English, Portuguese and Hindi languages only. Use of seq2seq (or Encoder-Decoder) architecture is appropriate in this case as the length of the input sequence does not has the same length as the output data

To summarize our model, the Encoder simply takes the input data, and train on it then it passes the last state of its recurrent layer as an initial state to the first recurrent layer of the decoder part. The Decoder takes the last state of encoder's last recurrent layer and uses it as an initial state to its first recurrent layer , the input of the decoder is the sequences that we want to get. We will use Keras API with Tensorflow backend to build our model.

3.1 Data Preparation

Before we start building the model, we need to clean up the text data (i.e. the sentences). We will remove all punctuation characters, normalize the case to lowercase, normalize all Unicode characters to ASCII and remove any tokens that are not alphabetic. To build the model, we need to map words to Integers. We will use Keras Tokenize class for this. The Tokenizer must be constructed and then fit on either raw text documents or integer encoded text documents. Once fit, the Tokenizer provides four attributes that you can use to understand about your text., viz.,

1. word-counts: A dictionary of words and their counts
2. word-docs: A dictionary of words and how many documents each appeared in.
3. word-index: A dictionary of words and their uniquely assigned integers.
4. document-count:An integer count of the total number of documents that were used to fit the Tokenizer.

We will also compute the vocabulary sizes and the length of maximum sequence for both the languages. We need to encode each input and output sentences to integers and pad them to the maximum phrase length to make all sentences of the same length. This is because we will use word embedding for the input sentence and one hot encoding for the output. In one hot encoding, a document is represented as a sequence of integer values, where each word in the document is represented as a unique integer. One hot encoding is needed because the model will predict the probability of each word in the vocabulary as output.

3.2 Euclidean Distance and Cosine Similarity

For our comparison on closeness of strings we will leverage two methods: Euclidean Distance and Cosine Similarity. Euclidean Distance measures the distance between two points in a space, and therefore the minimum distance is the closest match. Cosine Similarity is the similarity between two vectors in a space, and therefor the maximum similarity is the closest match. In this section we will decribe how these methods calculate their scores, and in the "Results" section we will show the result of the comparison and choose our preferred match.

Euclidean distance, also known as L2 distance or the Pythagorean metric, is a basic measure of distance in a space. given a number of dimensions in a space, the Euclidean Distance calculates the length of a line segment connecting those points. This distance is calculated by going through all of the dimensions, adding up the square of the difference between the points of each value for that dimension, then taking the square root of that sum. This method is called the Pythagorean Formula. [19]. This method is easy to understand, but it seems like it might struggle if the data are not of similar shapes, as is the case with our sentence sample.

Cosine Similarity tries to measure similarity in a fundementally different way from Euclidean Distance. Instead of stepping through and comparing data point-by-point, Cosine Similarity instead steps through each sample and tries to build a directional veector for ther shape of the data, then measures the cosine of the angle between the two vectors. In the case of text analysis the cosine similarity uses the frequency of words in a sentence order to determine a directional vector. Cosine Similarity ignores the words that are not in common between two sentences, and focuses only on words that are in common, which avoids penalizing sentences that are very long and simply focuses on finding the closest possible match. Therefore cosine similarity is very resilient to data that is of different shapes, since the shape of the data would not necessarily affect the direction of the vector, instead the sentences with the closest ratios of like words would be considered a match. [21] This reslience to differently-shaped data is promising for our purposes, and seems to be a good fit for comparing sentences, and we will see in the "Results" section how it performs.

3.3 Encoder-Decoder Long Short-Term Memory (LSTM) Networks

A typical seq2seq model consists of 2 major components

1. Encoder
2. Decoder

Both these components are essentailly two different Recurrent Neural Network models combined into one giant network.

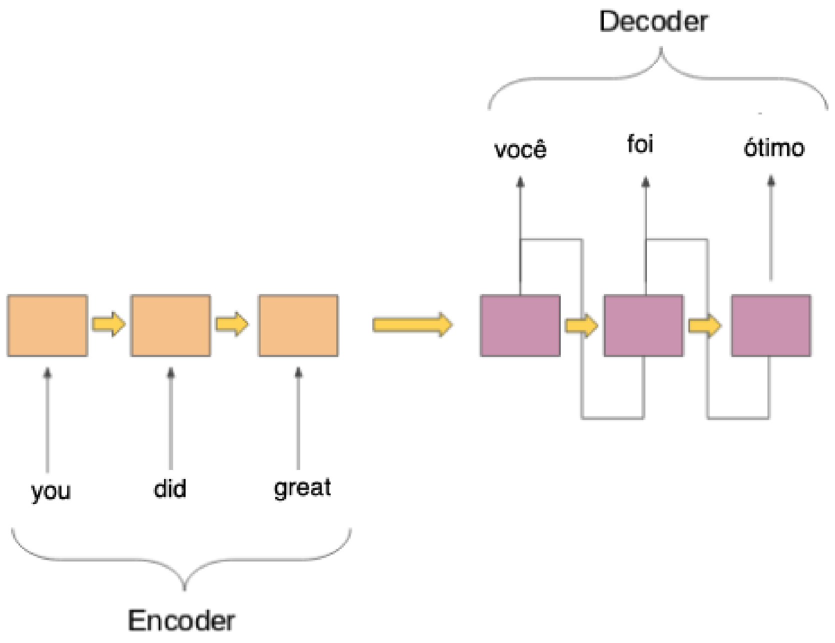


Fig. 2: Sequence to Sequence Modell
[4]

We will explain the Encoder and Decoder model in more detail.

Let's say we are trying to convert the following sentence from English to Portuguese.

Input sentence (English) - i have lost my passport

Output sentence (Portuguese) - eu perdi meu passaporte

A sentence can be seen as a sequence of words or characters. We will split the sentence by words. So, for the above example in English, there are 5 words which are fed to the encoder as shown in the figure below. The input is referred to as X and Xi is the input sequence at time step i. So we have the following input. X1 = i, X2 = have, X3 = lost, X4=my, X5 = passport. Each Xi is mapped to a fixed-length vector using the built-in embedding layer of Keras API.

The LSTM will read this sentence word by word in 5 time steps as shown in the figure.

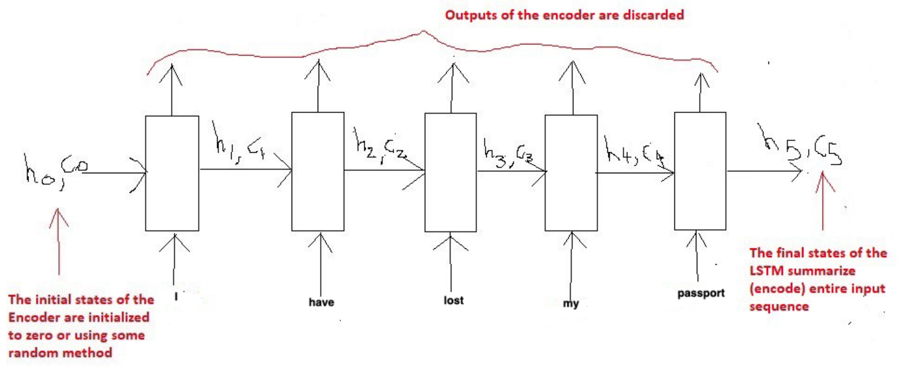

Outputs of the encoder are discarded
h0,c0
h1,c1
h2,c2
h3,c3
h4,c4
h5,c5
I
have
lost
my
passport
The initial states of the Encoder are initialized to zero or using some random method
The final states of the LSTM summarize (encode) entire input sequence

Fig. 3: Encoder LSTM
[5]

hi and ci in the figure above represent the internal state, viz., the hidden state and the cell state of the Encoder. In simple terms, they remember what LSTM has read till now. For example, h3, c3 vectors will remember that the network has read “I have lost” till now. Basically its the summary of information till time step 3 which is stored in the vectors h3 and c3 (thus called the states at time step 3). So, h5,c5 will contain the summary of the entire sentence. These states coming out of the last time step are also called as the “Thought vectors” as they summarize the entire sequence in a vector form. We initialize h0,c0 to zero as the model has not started to read the input.

Yi is the output of the LSTM at each step. We discard the outputs of the encoder and only preserve the internal states as the model has nothing to output unless it has read the entire English sentence.

Next, we define the Decoder. Unlike the Encoder LSTM which has the same role to play in both the training phase as well as in the inference phase, the Decoder LSTM has a slightly different role to play in both of these phases. Recall that given the input sentence ”i have lost my passport”, the goal of the decoder is to output ”eu perdi meu passaporte”.

The intial states (h0,c0) of the Decoder are set to the final states of the Encoder. This intuitively means that the decoder is trained to start generating the output sequence depending on the information encoded by the encoder.

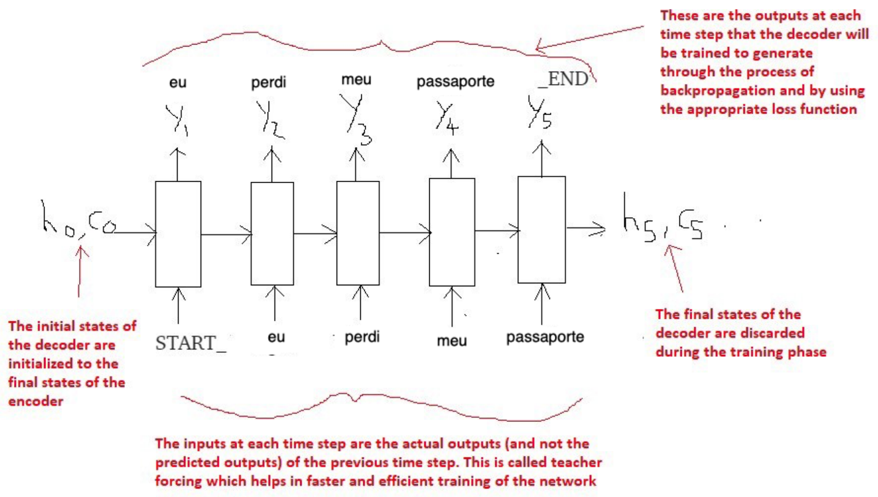

These are the outputs at each time step that the decoder will be trained to generate through the process of backpropagation and by using the appropriate loss function
eu
perdi
meu
passaporte
_END
Y_1
Y_2
Y_3
Y_4
Y_5
h_0, c_0
h_5, c_5
START_
eu
perdi
meu
passaporte
The initial states of the decoder are initialized to the final states of the encoder
The final states of the decoder are discarded during the training phase
The inputs at each time step are the actual outputs (and not the predicted outputs) of the previous time step. This is called teacher forcing which helps in faster and efficient training of the network

Fig. 4: Decoder LSTM
[6]

3.4 Building the Neural Translation Model

We will split our dataset into train and test set for model training and evaluation, respectively. Our seq2seq model is defined as the following

1. For the encoder, we will use an embedding layer and an LSTM layer
2. For the decoder, we will use another LSTM layer followed by a dense layer

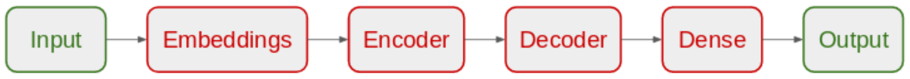
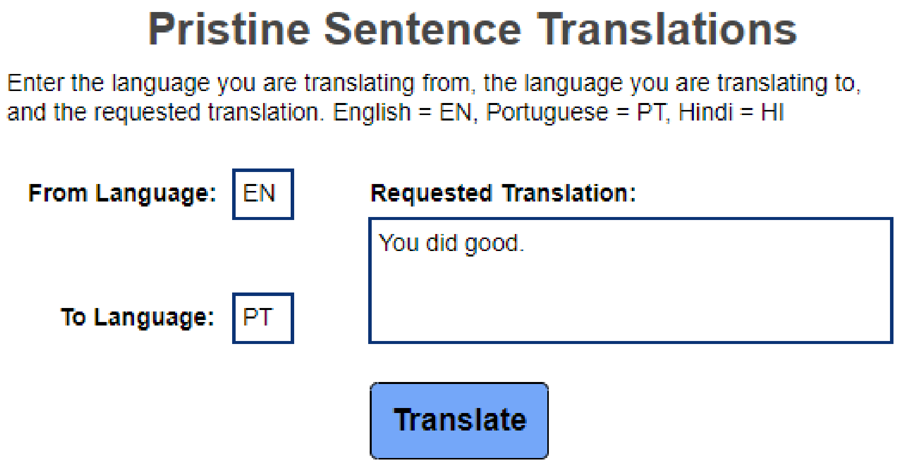


Fig. 5: Model Architechture
[7]

4 Full Demo

The front end for the Pristine Language Translation (PST) site is a simple proof-of-concept page with three boxes for input: The language and the text the user is translating from, and the language code the user is translating to. The user must fill in all three boxes with valid input before hitting the "Translate" button:


Pristine Sentence Translations

Enter the language you are translating from, the language you are translating to, and the requested translation. English = EN, Portuguese = PT, Hindi = HI

From Language: EN

To Language: PT

Requested Translation: You did good.

Fig. 6: Language Input

After the user has entered the data and hit the ”Translate” button the information is passed to our pyton NLP processing. First our model finds the closest linguistic match to the sentence entered to a sentence for which the translation is known. Then we have a separate pos-processing step to calculate how close of a match the input sentence is to the matched sentence, and we retrieve that translations of that sentence for the requested language:

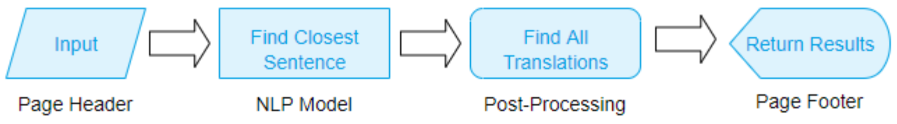
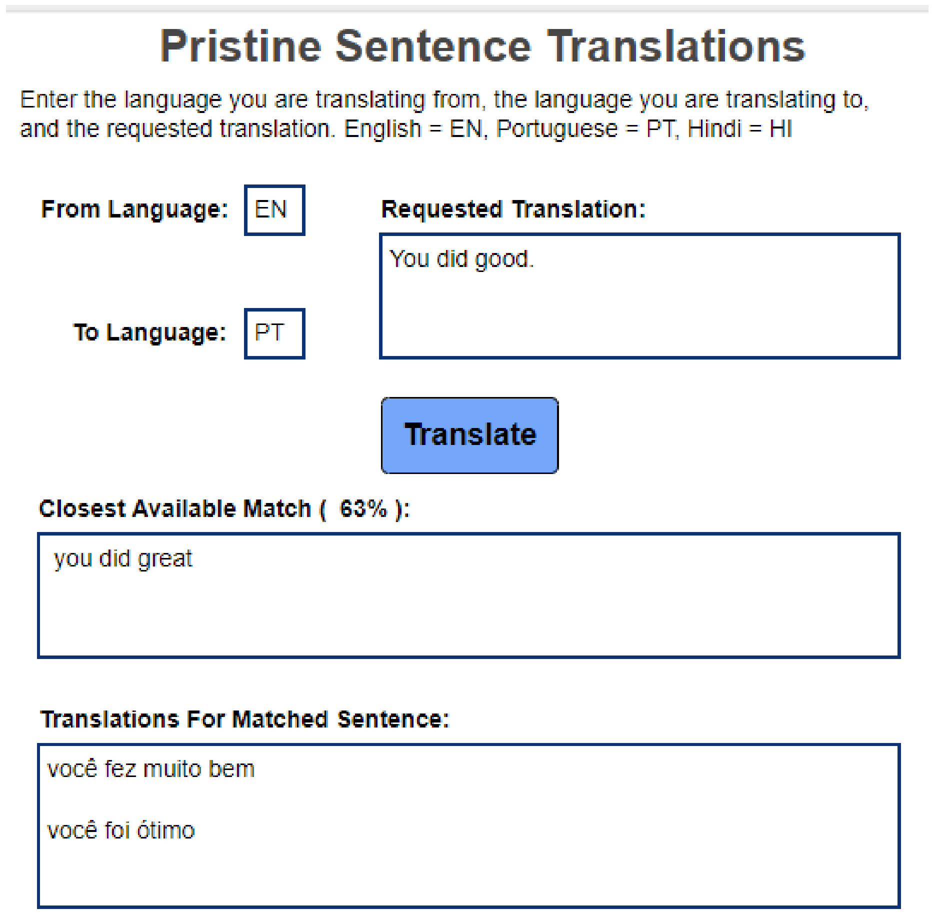


Fig. 7: Process Map

The results are then reurned to the screen for display. Note that there can be more than one valid translation for an input sentence, provided our database has the mappings:


Pristine Sentence Translations

Enter the language you are translating from, the language you are translating to, and the requested translation. English = EN, Portuguese = PT, Hindi = HI

From Language: EN

To Language: PT

Requested Translation: You did good.

Closest Available Match (63%): you did great

Translations For Matched Sentence:

você fez muito bem

você foi ótimo

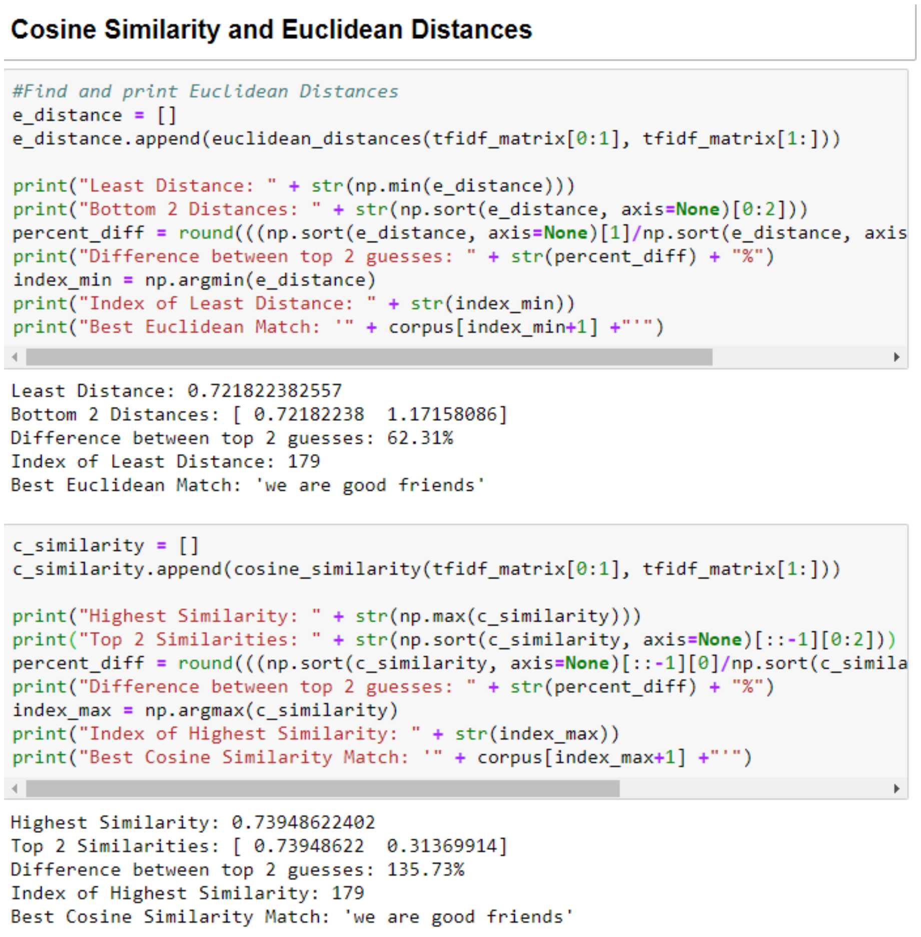
Fig. 8: Display Translations

If the user wants to try another sentence the user can change any of the three inputs above and hit the ”Translate” button again to repeat the process.

4.1 Results and Analysis

After we assembled our database of translations and built our PKL files our only outstanding task was to pick a method for determining how close sentences are. We had two contenders: Euclidean Distances and Cosine Similarities. We needed a way to determine, given our dataset, which method stood the best chance of matching an input sentence to the sentence we thought it should match to. We decided to try slightly altering some of our pristine sentences in our database and runnign them through both comparisons, then analyze how well each method performed.

To start, we altered our known sentence ”’we are good friends” to ”we are best friends”, and ran it through the Euclidean Distance and Cosine Similarity comparisons. Here is the output:



Cosine Similarity and Euclidean Distances

```
#Find and print Euclidean Distances
e_distance = []
e_distance.append(euclidean_distances(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Least Distance: " + str(np.min(e_distance)))
print("Bottom 2 Distances: " + str(np.sort(e_distance, axis=None)[0:2]))
percent_diff = round(((np.sort(e_distance, axis=None)[1]/np.sort(e_distance, axis
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_min = np.argmin(e_distance)
print("Index of Least Distance: " + str(index_min))
print("Best Euclidean Match: '" + corpus[index_min+1] +"'")
```

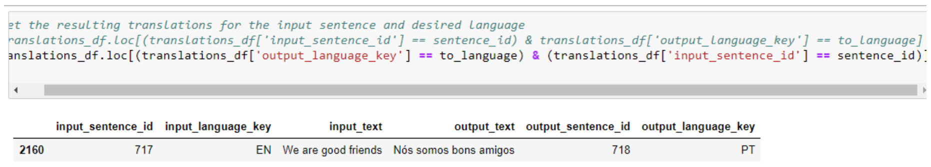
```
Least Distance: 0.721822382557
Bottom 2 Distances: [ 0.72182238  1.17158086]
Difference between top 2 guesses: 62.31%
Index of Least Distance: 179
Best Euclidean Match: 'we are good friends'
```

```
c_similarity = []
c_similarity.append(cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Highest Similarity: " + str(np.max(c_similarity)))
print("Top 2 Similarities: " + str(np.sort(c_similarity, axis=None)[::-1][0:2]))
percent_diff = round(((np.sort(c_similarity, axis=None)[::-1][0]/np.sort(c_simila
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_max = np.argmax(c_similarity)
print("Index of Highest Similarity: " + str(index_max))
print("Best Cosine Similarity Match: '" + corpus[index_max+1] +"'")
```

```
Highest Similarity: 0.73948622402
Top 2 Similarities: [ 0.73948622  0.31369914]
Difference between top 2 guesses: 135.73%
Index of Highest Similarity: 179
Best Cosine Similarity Match: 'we are good friends'
```

Fig. 9: Sentence Comparison 1

You can see that both methods were effective at finding the result we anticipated: ”We are good friends.” However, if you look at how close the second-best guess was in each comparison you can see that the cosine similarity method seemed much better at finding that result. The next-closest Euclidean Distance was only 62.31% more than the best guess, but for Cosine Similarity the best guess was 135.73% better than the next-best guess. Even though both performed admirably in this circumstance, it seems like cosine similarity was able to more clearly diffferentiate the sentence we want from the sentence we don't. As an example of the end-to-end process, here's the translation of our found pristine sentence to the Portuguese ”Nós somos bons amigos.”:

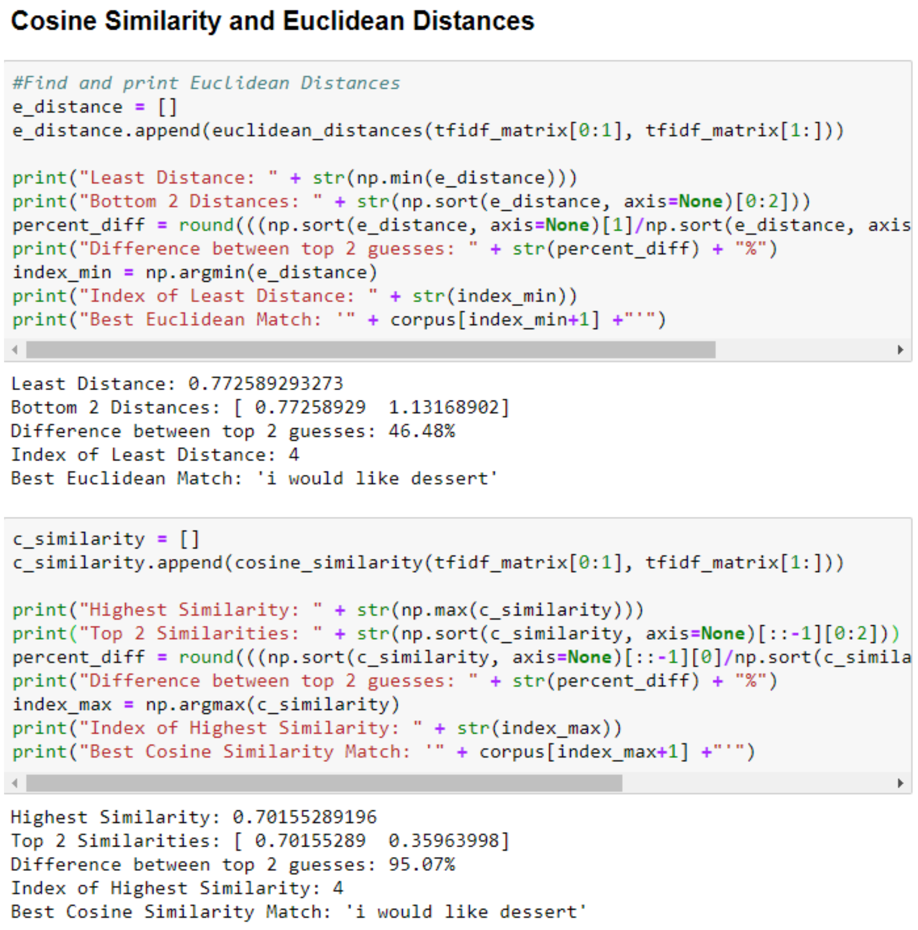


```
et the resulting translations for the input sentence and desired language
ranslations_df.loc[(translations_df['input_sentence_id'] == sentence_id) & translations_df['output_language_key'] == to_language]
anslations_df.loc[(translations_df['output_language_key'] == to_language) & (translations_df['input_sentence_id'] == sentence_id)]
```

	input_sentence_id	input_language_key	input_text	output_text	output_sentence_id	output_language_key
2160	717	EN	We are good friends	Nós somos bons amigos	718	PT

Fig. 10: Translating Matched Sentence to Portuguese

We tried many examples and the results were similar, here is another example where we we altered our known sentence ”'I would like dessert” to ”'I would love dessert”:



Cosine Similarity and Euclidean Distances

```
#Find and print Euclidean Distances
e_distance = []
e_distance.append(euclidean_distances(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Least Distance: " + str(np.min(e_distance)))
print("Bottom 2 Distances: " + str(np.sort(e_distance, axis=None)[0:2]))
percent_diff = round(((np.sort(e_distance, axis=None)[1]/np.sort(e_distance, axis
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_min = np.argmin(e_distance)
print("Index of Least Distance: " + str(index_min))
print("Best Euclidean Match: '" + corpus[index_min+1] +"'")
```

```
Least Distance: 0.772589293273
Bottom 2 Distances: [ 0.77258929  1.13168902]
Difference between top 2 guesses: 46.48%
Index of Least Distance: 4
Best Euclidean Match: 'i would like dessert'
```

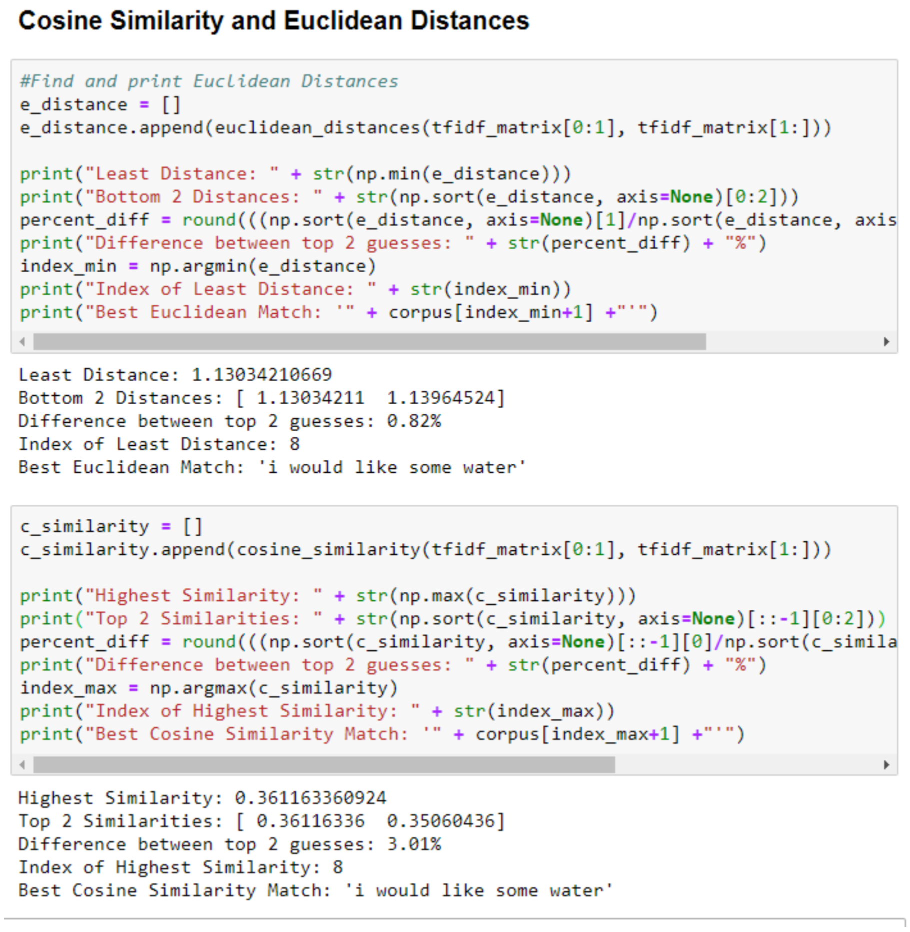
```
c_similarity = []
c_similarity.append(cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Highest Similarity: " + str(np.max(c_similarity)))
print("Top 2 Similarities: " + str(np.sort(c_similarity, axis=None)[::-1][0:2]))
percent_diff = round(((np.sort(c_similarity, axis=None)[::-1][0]/np.sort(c_simila
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_max = np.argmax(c_similarity)
print("Index of Highest Similarity: " + str(index_max))
print("Best Cosine Similarity Match: '" + corpus[index_max+1] +"'")
```

```
Highest Similarity: 0.70155289196
Top 2 Similarities: [ 0.70155289  0.35963998]
Difference between top 2 guesses: 95.07%
Index of Highest Similarity: 4
Best Cosine Similarity Match: 'i would like dessert'
```

Fig. 11: Sentence Comparison 2

Again, both methods were effective at finding the result we anticipated: ”I would like dessert.” However, the next-closest Euclidean Distance was only 46.48% more than the best guess, but for Cosine Similarity the best guess was 95.07% better than the next-best guess.

We also thought that it would be interesting to try a sentence that wasn't even close to one to which we knew the translation. For this we altered ”I would like some water” to ”I would like some blueberry muffins and orange juice, please.” We knew this wouldn't be pretty, here is the result:



Cosine Similarity and Euclidean Distances

```
#Find and print Euclidean Distances
e_distance = []
e_distance.append(euclidean_distances(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Least Distance: " + str(np.min(e_distance)))
print("Bottom 2 Distances: " + str(np.sort(e_distance, axis=None)[0:2]))
percent_diff = round(((np.sort(e_distance, axis=None)[1]/np.sort(e_distance, axis
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_min = np.argmin(e_distance)
print("Index of Least Distance: " + str(index_min))
print("Best Euclidean Match: '" + corpus[index_min+1] +"'")
```

```
Least Distance: 1.13034210669
Bottom 2 Distances: [ 1.13034211  1.13964524]
Difference between top 2 guesses: 0.82%
Index of Least Distance: 8
Best Euclidean Match: 'i would like some water'
```

```
c_similarity = []
c_similarity.append(cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Highest Similarity: " + str(np.max(c_similarity)))
print("Top 2 Similarities: " + str(np.sort(c_similarity, axis=None)[::-1][0:2]))
percent_diff = round(((np.sort(c_similarity, axis=None)[::-1][0]/np.sort(c_simila
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_max = np.argmax(c_similarity)
print("Index of Highest Similarity: " + str(index_max))
print("Best Cosine Similarity Match: '" + corpus[index_max+1] +"'")
```

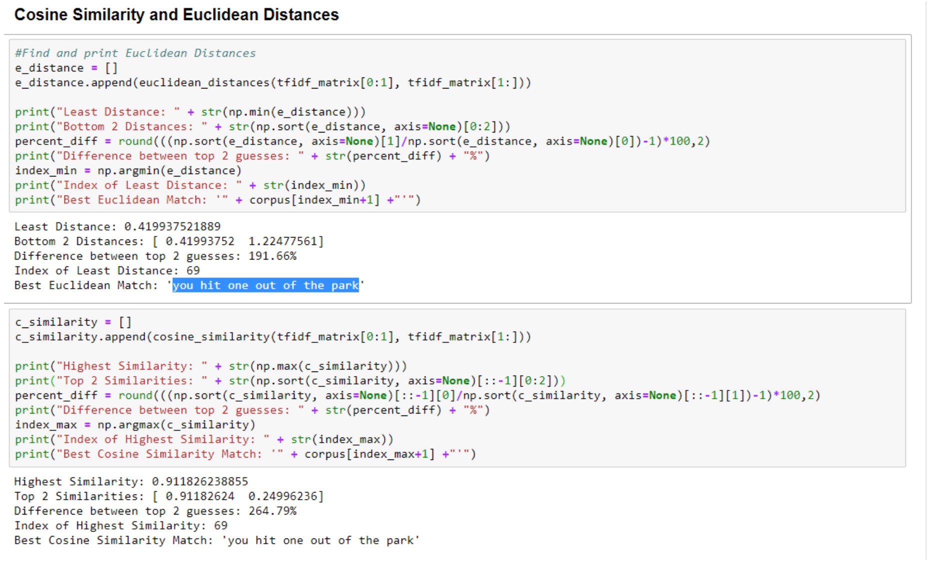
```
Highest Similarity: 0.361163360924
Top 2 Similarities: [ 0.36116336  0.35060436]
Difference between top 2 guesses: 3.01%
Index of Highest Similarity: 8
Best Cosine Similarity Match: 'i would like some water'
```

Fig. 12: Sentence Comparison 3

The good news is that both methods found what we hoped: ”I would like some water.” It’s also notable that the Cosine Similarity did perform very slightly better than Euclidean Distance in terms of the difference to the next-highest match: 3.01% for Cosine Similarity vs. 0.83% for Euclidean Distance. However, looking at the quality of match for Cosine Similarity says that these sentences are only 36.12% similar, which is a very low similarity compared to the over 70% of the other two matches. We would need to strongly discourage anyone taking this translation as a close match to what is entered.

Finally, we wanted to list the point of PST, translating a sentence that is hard to translation from one language to another. In this case we chose ”You really hit one out of the park”. Google Translate shows this in Portuguese as ”Você realmente bateu um fora do Parque”, which still would not give a native

Portuguese speaker any idea what that sentence means to them, unless they knew American Baseball. Here's how PST Performs:



Cosine Similarity and Euclidean Distances

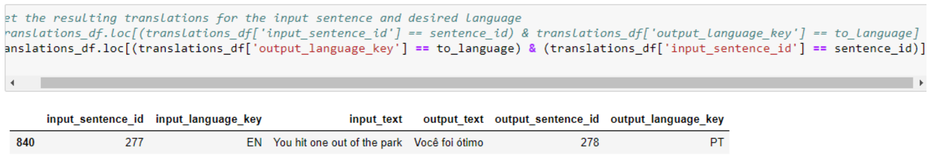
```
#Find and print Euclidean Distances
e_distance = []
e_distance.append(euclidean_distances(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Least Distance: " + str(np.min(e_distance)))
print("Bottom 2 Distances: " + str(np.sort(e_distance, axis=None)[0:2]))
percent_diff = round(((np.sort(e_distance, axis=None)[1]/np.sort(e_distance, axis=None)[0])-1)*100,2)
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_min = np.argmin(e_distance)
print("Index of Least Distance: " + str(index_min))
print("Best Euclidean Match: '" + corpus[index_min+1] +"'")
```

```
Least Distance: 0.419937521889
Bottom 2 Distances: [ 0.41993752  1.22477561]
Difference between top 2 guesses: 191.66%
Index of Least Distance: 69
Best Euclidean Match: 'you hit one out of the park'
```

```
c_similarity = []
c_similarity.append(cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:]))

print("Highest Similarity: " + str(np.max(c_similarity)))
print("Top 2 Similarities: " + str(np.sort(c_similarity, axis=None)[::-1][0:2]))
percent_diff = round(((np.sort(c_similarity, axis=None)[::-1][0]/np.sort(c_similarity, axis=None)[::-1][1])-1)*100,2)
print("Difference between top 2 guesses: " + str(percent_diff) + "%")
index_max = np.argmax(c_similarity)
print("Index of Highest Similarity: " + str(index_max))
print("Best Cosine Similarity Match: '" + corpus[index_max+1] +"'")
```

```
Highest Similarity: 0.911826238855
Top 2 Similarities: [ 0.91182624  0.24996236]
Difference between top 2 guesses: 264.79%
Index of Highest Similarity: 69
Best Cosine Similarity Match: 'you hit one out of the park'
```

Fig. 13: Sentence Comparison 4



```
et the resulting translations for the input sentence and desired language
ranslations_df.loc[(translations_df['input_sentence_id'] == sentence_id) & translations_df['output_language_key'] == to_language]
anslations_df.loc[(translations_df['output_language_key'] == to_language) & (translations_df['input_sentence_id'] == sentence_id)]
```

	input_sentence_id	input_language_key	input_text	output_text	output_sentence_id	output_language_key
840	277	EN	You hit one out of the park	Você foi ótimo	278	PT

Fig. 14: Translation for Colloquialism

The sentence we input was very similar to the PST entry "You hit one out of the park", which is reflected in the displayed scores. More importantly, PST returned "Você foi ótimo", which means "You were great" in English, which is the intent of the colloquialism.

Now that we have performed our analysis we are ready to decide on a final method.

All code for our comparisons can be found at this URL in the References section: [20]

4.2 Conclusions

Cosine Similarity seems to outperform Euclidean Distance as a determination of sentence similarity, so that is the metric we will focus on for determining closeness of input sentences to our known sentences. It makes sense that Cosine Similarity would be a more ideal fit than Euclidean Sentences because every unexpected word in Euclidean Distance comparison is penalized, but in Cosine Similarity these are ignored and we focus only on the matches. This helps cut through the noise and return a more confident match in what we want.

In addition, by testing a very bad sentence match we touched on the possibility of using a similarity threshold of some kind, where perhaps in future iterations of the program we could warn users not to pay too much attention to a sentence match less than some percentage. Over 70% match seems like a good match, below 40% seems like a bad match, but we need to compare a lot more data to determine what that threshold might make sense to be.

We have also shown how PST could be used to overcome some of the hazards around langauge translation today, ignoring colloquialisms and focusing on the intent of the sentence.

Given our work so far, we now have a method to find a sentence ID that is our closest match to an input sentence, then take that sentence ID and find all the applicable known translations for that sentence. We can do this for all 200 sentences in our database, and can display how close of a match we have made using Cosine Similarity, which we have identified as our chosen similarity determination method. This is enough to demonstrate the basic concept of PST, and can be expanded on for future work.

5 Ethical Considerations

The use of Natural Language Processing (NLP) in regards to language transations raises several ethical issues. Like many other fields in data science, practicioners of NLP must worry about the core questions around gathering data, as well as a second, more-specific ethical concern around their topic (in the case of this paper, translation of language). First we will address the core questions around ethical data science behaviors, then the more specific issue around our translation project.

In order to find a framework in which to address ethical considerations, it would be helpful to have a template, or some core questions to answer. Margot Mieske proposes in her paper "A Quantitative Study of Data in the NLP community" five key questions every NLP programmer must answer: [8]

- Has data been collected?
- How was this data collected and processed?
- Was previously available data used/extended – which one?
- Is a link or a contact given?
- Where does it point (private page, research institute, official repository)?

For this project the initial set of 200 English Sentences data were gathered from three travel websites. [11,12,13]. Travel sites were chosen in order to get a good spread of what sentences people might find useful. Translations were then done using a combination of Google Translate [14] and a human expertise in Portuguese and Hindi in order to determine if the Google Translation made sense. If the sentence did not make sense as Google Translated it, we entered a more sensible translation. The original list of sentences and translations can be found in our public Github Repository here:

`https://github.com/coarib/SMU_Masters_PST/blob/master/data/processed/PST_workbook.xlsx`

For the specifics around ethical considerations for NLP we turn to Jochen L. Leidner and Vassilis Plachouras and their paper "Ethical by Design: Ethics Best Practices for Natural Language Processing" [15]. Leidner and Plachouras propose that since NLP pertains to human language and touches every part of human life it has a specific ethics dimension, therefore automation and errors also become ethical topics. We need to make sure our translations and base sentences are unbiased and fair, without discrimination based on age, race, or gender. This is why we chose to target trave sites that hopefuly won't favor certain demographics. At the very least we are transparent about from where we pulled the sentences.

This is also why we carefully scanned each translation for fairness to make sure our data was reviewed before putting it out to the public, and why we keep a tight control over the sentences that might be suggested for a user. As it is, this proof-of-concept system is not something I'd like to put out into the world for general use. With only 200 sentences the risks around mistranslating something are too high, there is the potential for widepread confusion if our suggestions aren't close enough to the input request. Before going public we would need to follow Leidner and Plachouras's advice and establish an ethics review board that establish a process for reviewing and implementing new translation requests and helping monitor issues around translations on the site.

6 Conclusions and Other Work

Through this proof-of-concept we have shown the potential around Pristine Sentence Translations, and have overcome some of the technical hurdles of implementing the translation system. There is plenty of room for improvement, but it explores a fresh take on translations that is both simple in nature and effective for solving issues around hard-to-translate words and colloquialisms.

This Pristine Sentence Translations proof-of-concept is only built for about 200 sentences which could be translated from/to English, Portuguese and Hindi languages. This model could be expanded by adding more data and by incorporating more languages for translations. The greater the number of sentences in the database, the greater the chance of finding a significant match to an imput sentence. Tens of thousands of sentences would be needed in order to function as a serivceable translation algorithm, and with the evolution of language this would need to be added onto constantly. Methods of discovering translations might include crowdsourcing to large groups knowledgeable in both languages (translators, teachers, bilingual students), or crawling for translations on travel sites, online movie scripts, instruction booklets, etc.

One could try dropout and other forms of regularization techniques to mitigate over-fitting, or perform with hyperparamter tuning. One could also play with learning rate, batch-size, number of epochs etc.

It would be interesting to see how the model would perform when built using Attention.

References

1. Google's new translation software is powered by brainlike artificial intelligence (2016, September 27), Available at: `https://www.sciencemag.org/news/2016/09/google-s-new-translation-software-powered-brainlike-artificial-intelligence?r3f_986=https://www.google.com/`. Last accessed 4 Feb 2019
2. Found in translation: More accurate, fluent sentences in Google Translate (2016, November 15), Available at: `https://www.blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate`. Last accessed 4 Feb 2019
3. A ten-minute introduction to sequence-to-sequence learning in Keras, Available at: `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html`. Last accessed 4 Feb 2019
4. Figure 1. Available at: `https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/01/enc_dec_2.png`. Last accessed 22 Mar 2019
5. Figure 2. Available at: `https://cdn-images-1.medium.com/max/1600/1*37tROolA8uW7Nz2YpFsWqA.jpeg`. Last accessed 22 Mar 2019
6. Figure 3. Available at: `https://cdn-images-1.medium.com/max/1600/1*SQAxk6gSUM_AK2d53neYmA.jpeg`. Last accessed 22 Mar 2019
7. Figure 4, Available at: `https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/01/architecture.png`. Last accessd 22 Mar 2019
8. A Quantitative Study of Data in the NLP community, Margot Mieskes, Available at: `http://www.ethicsinnlp.org/workshop/EthNLP-2017.pdf`. Last accessed 25 Mar 2019
9. Google's new translation software is powered by brainlike artificial intelligence (2016, September 27), Available at: `https://www.sciencemag.org/news/2016/09/google-s-new-translation-software-powered-brainlike-artificial-intelligence?r3f_986=https://www.google.com/`. Last accessed 25 Mar 2019
10. Found in translation: More accurate, fluent sentences in Google Translate (2016, November 15), Available at: `https://www.blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate`. Last accessed 25 Mar 2019
11. 76 English Phrases for Traveling with Ease, Available at: `https://www.fluentu.com/blog/english/english-travel-phrases/`. Last accessed 25 Mar 2019
12. Travel English– 100 sentences you must know, Available at: `https://www.rong-chang.com/travel/`. Last accessed 25 Mar 2019
13. 100 Common English Phrases and Sentence Patterns (With Dialogue), Available at: `https://basicenglishspeaking.com/100-common-phrases-and-sentence-patterns/`. Last accessed 25 Mar 2019
14. Google Translate, Available at: `https://translate.google.com/`. Last accessed 25 Mar 2019

15. Ethical by Design: Ethics Best Practices for Natural Language Processing, Jochen L. Leidner and Vassilis Plachouras, Available at: `http://www.ethicsinnlp.org/workshop/pdf/EthNLP02.pdf/`. Last accessed 25 Mar 2019
16. How to Develop a Neural Machine Translation System from Scratch, by Jason Brownlee on January 10, 2018, Available at: `https://machinelearningmastery.com/develop-neural-machine-translation-system-keras/`. Last accessed 25 Mar 2019
17. SQL Alchemy, Available at: `https://www.sqlalchemy.org/`. Last accessed 25 Mar 2019
18. Found in translation: More accurate, fluent sentences in Google Translat, by Catherine Matacic on Sep. 27, 2016, Available at: `https://www.sciencemag.org/news/2016/09/google-s-new-translation-software-powered-brainlike-artificial-intelligence?r3f986=https://www.google.com/`. Last accessed 25 Mar 2019
19. What is Euclidean Distance?, On April 1, 2017, Available at: `WhatisEuclideanDistance?https://machinelearningtutorials.com/euclidean-distance/`. Last accessed 25 Mar 2019
20. Text Similarities, Meenu Ahluwalia and Brian Coari, Available at: `https://github.com/coarib/SMU_Masters_PST/blob/master/src/TextSimilarity_edited.ipynb`. Last accessed 25 Mar 2019
21. Jiawei Han, Micheline Kamber and Jian Pei: Data Mining: Concepts and Techniques. 3rd ed. Elsevier Inc., Chapter 2.4.7 - Cosine Similarity, Page 77 (2012)

A SQL Queries

Select the text and sentence IDs for a language key. Language keys can be EN, PT, or HI for English, Portuguese, or Hindi, respectively.

```
select text, sentence_id from Sentences where language_key='[Language_Key]'
```

Select every combination of matching translations between all sentences. Storing this query in a dataframe allows us to get all the translations for a sentence just by matching a sentence ID to the input_sentence_id column.

```
SELECT a.sentence_id as input_sentence_id,
a.language_key as input_language_key, a.text as input_text,
b.text as output_text, b.sentence_id as output_sentence_id,
b.language_key as output_language_key FROM masters.Translations
left join masters.Sentences a on a.sentence_id=Translations.sentence_id_1
left join masters.Sentences b on b.sentence_id=Translations.sentence_id_2
```