Aligning Word Vectors on Low-Resource Language, Bodo

Group F:
Vishesh Thakur
Vijay Rakshit
Shaleen Malik

Problem Statement

- Use of Wiktionary to construct high-quality bilingual lexicons suitable for training and evaluating BLI models from 298 languages to English (253 for the first time).

- Train the largest collection of BLI models. Grave et al provided pre-trained word vectors in 157 languages, BLI models trained for these languages to English. (112 languages had no BLI models trained due to absence of data for training/evaluation).

Background

MUSE Corpus:

This collection provides bilingual lexicons between 45 languages and English. This lexicon is generated from a machine translation system, and therefore suffers from a number of problems.

1. Many of the mappings in the lexicon do not contain real words in either the source or target language.
2. The distribution of words is inconsistent between languages, with many languages containing only proper nouns in their training and test sets.

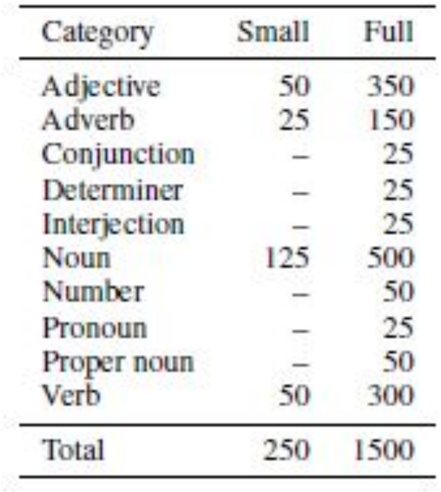
Wiktionary dataset:

Wiktionary contains user annotated datasets in 8166 languages along with the POS tags, and we aim to build our lexicons on this dataset.

Dataset Used

Data Extraction: user entered data

Train/Test splits: To populate the small test set, we select the most frequent words from each category. The remaining words in the large test set are sampled uniformly from the 10,000 most common words for each category.



Category	Small	Full
Adjective	50	350
Adverb	25	150
Conjunction	–	25
Determiner	–	25
Interjection	–	25
Noun	125	500
Number	–	50
Pronoun	–	25
Proper noun	–	50
Verb	50	300
Total	250	1500

Experiments

Setup: Common crawl vectors aligned to English language vectors trained on the common crawl data. Iterative normalization processing procedure used to transform the source and target language vectors before learning.

Experiments:

1. Impact of size of BLI training dataset on model performance
2. Comparing quality of MUSE and Wiktionary lexicons
3. Training BLI models on 112 new, previously unstudied languages

Experimental Results and Inferences

1. Training Dictionary Size: BLI accuracy rapidly increases as the number of training samples reaches 5k, and then tapers off. After 20k training points, there is minimal improvement and the performance occasionally decreases due to statistical randomness.

2. MUSE vs Wiktionary: MUSE training set outperforms Wiktionary training set 22/45 times.
   a. First, the effect only happens when the MUSE training set is much larger than the Wiktionary training set.
   b. Wiktionary is naturally biased towards containing the "dictionary" (i.e. unconjugated) forms of words. Slova is a fusional language with many inflected forms for each word, and this helps explain the smaller size of the wiktionary dataset.

Experimental Results and Inferences

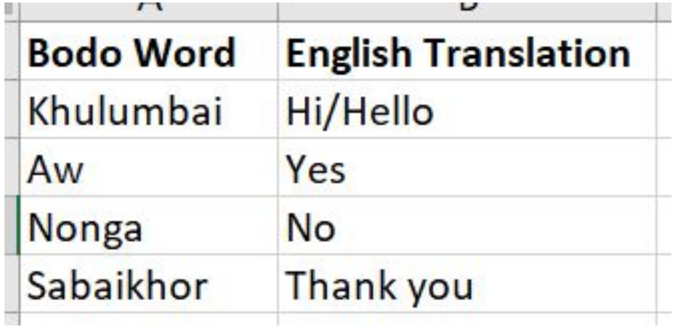
Three alignment models implemented:

- Procrustes (Xing et al., 2015)
- Bootstrap Procrustes (Glavaš et al., 2019; Vuli´c et al., 2019)
- VecMap (Artetxe et al., 2018a)

Inferences:

- Thirteen languages achieve an accuracy on the full test set greater than 30. An additional 2 languages achieve an accuracy on the small test set greater than 30.
- We observe that the higher-resource languages (top of table) tend to have better BLI performance than the lower resource languages (bottom of table).
- performance drop off long before this 5k training dictionary size mark.

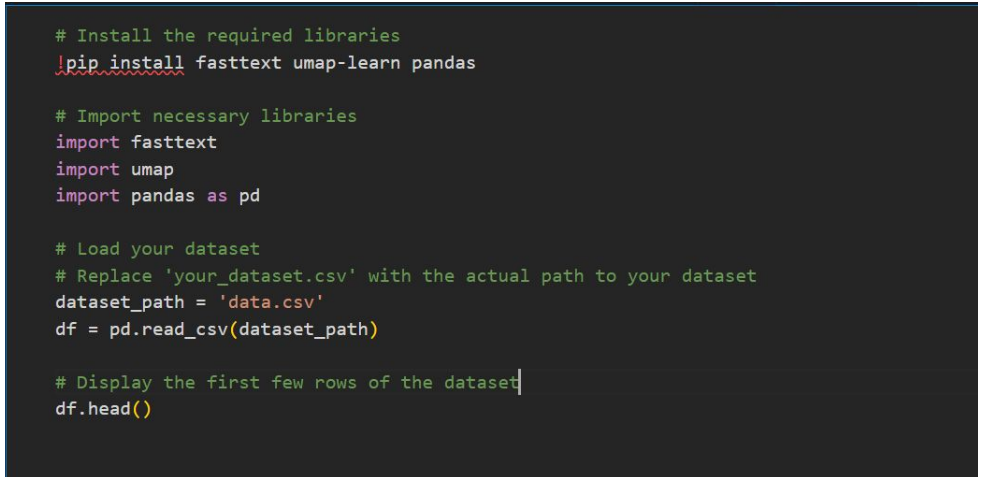
Dataset



Bodo Word	English Translation
Khulumbai	Hi/Hello
Aw	Yes
Nonga	No
Sabaikhor	Thank you

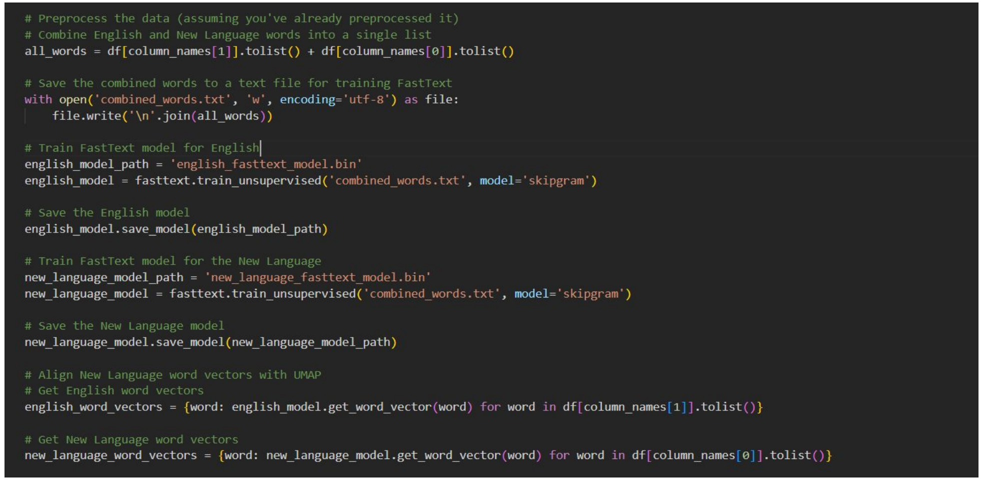
The dataset contains about 100 bodo words along with corresponding translations in English.

Code Snippet



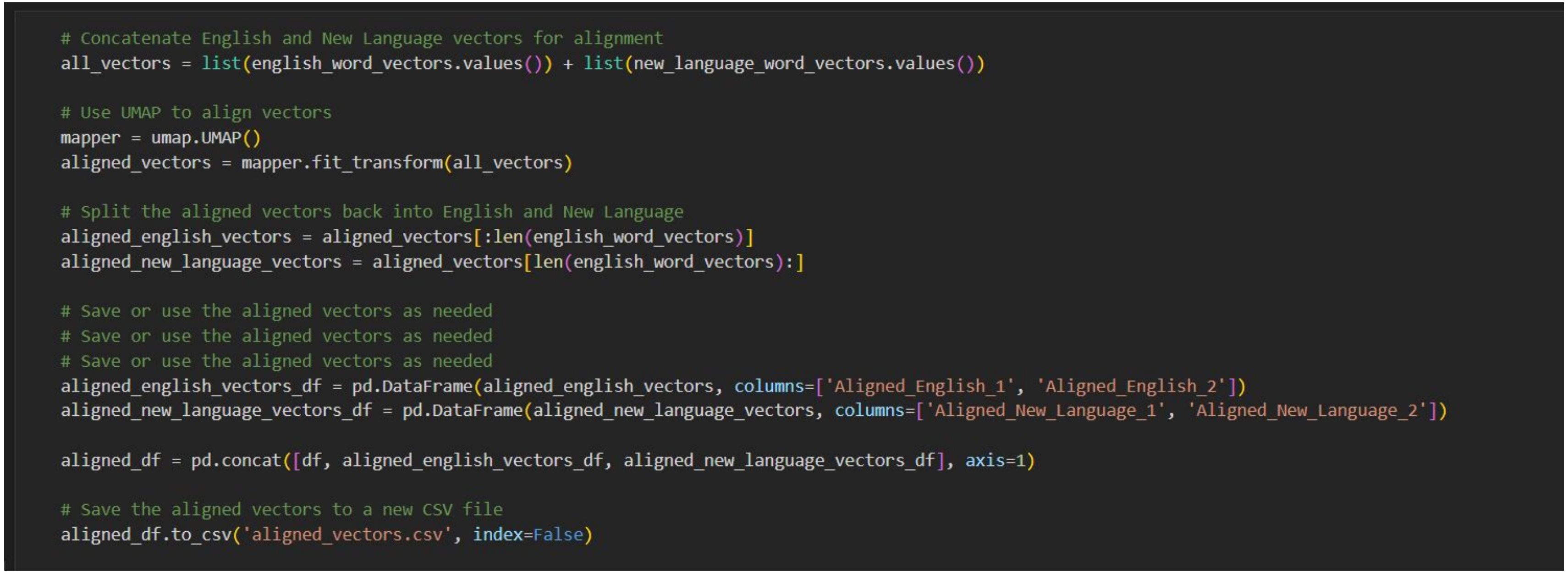
```
# Install the required libraries
!pip install fasttext umap-learn pandas

# Import necessary libraries
import fasttext
import umap
import pandas as pd

# Load your dataset
# Replace 'your_dataset.csv' with the actual path to your dataset
dataset_path = 'data.csv'
df = pd.read_csv(dataset_path)

# Display the first few rows of the dataset
df.head()
```

Code Snippet



```python
# Preprocess the data (assuming you've already preprocessed it)
# Combine English and New Language words into a single list
all_words = df[column_names[1]].tolist() + df[column_names[0]].tolist()

# Save the combined words to a text file for training FastText
with open('combined_words.txt', 'w', encoding='utf-8') as file:
    file.write('\n'.join(all_words))

# Train FastText model for English
english_model_path = 'english_fasttext_model.bin'
english_model = fasttext.train_unsupervised('combined_words.txt', model='skipgram')

# Save the English model
english_model.save_model(english_model_path)

# Train FastText model for the New Language
new_language_model_path = 'new_language_fasttext_model.bin'
new_language_model = fasttext.train_unsupervised('combined_words.txt', model='skipgram')

# Save the New Language model
new_language_model.save_model(new_language_model_path)

# Align New Language word vectors with UMAP
# Get English word vectors
english_word_vectors = {word: english_model.get_word_vector(word) for word in df[column_names[1]].tolist()}

# Get New Language word vectors
new_language_word_vectors = {word: new_language_model.get_word_vector(word) for word in df[column_names[0]].tolist()}
```

Code Snippet



```
# Concatenate English and New Language vectors for alignment
all_vectors = list(english_word_vectors.values()) + list(new_language_word_vectors.values())

# Use UMAP to align vectors
mapper = umap.UMAP()
aligned_vectors = mapper.fit_transform(all_vectors)

# Split the aligned vectors back into English and New Language
aligned_english_vectors = aligned_vectors[:len(english_word_vectors)]
aligned_new_language_vectors = aligned_vectors[len(english_word_vectors):]

# Save or use the aligned vectors as needed
# Save or use the aligned vectors as needed
# Save or use the aligned vectors as needed
aligned_english_vectors_df = pd.DataFrame(aligned_english_vectors, columns=['Aligned_English_1', 'Aligned_English_2'])
aligned_new_language_vectors_df = pd.DataFrame(aligned_new_language_vectors, columns=['Aligned_New_Language_1', 'Aligned_New_Language_2'])

aligned_df = pd.concat([df, aligned_english_vectors_df, aligned_new_language_vectors_df], axis=1)

# Save the aligned vectors to a new CSV file
aligned_df.to_csv('aligned_vectors.csv', index=False)
```

Novelty

The Existing dataset consisted 18 Indian languages out of the total of 22 scheduled Indian languages. As a novelty, we have included Bodo as a new language, which is extremely low resource. We have prepared a small dataset on Bodo, and evaluated it using the scored mentioned in the paper.

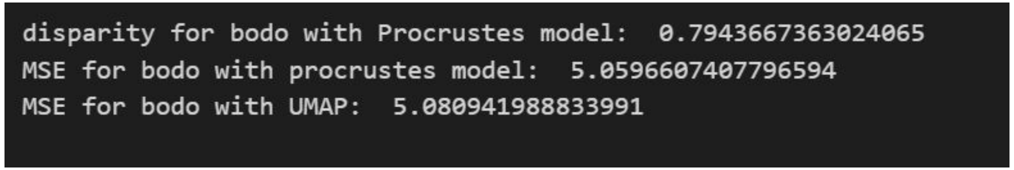
The Alignment MSE that we obtained 3.037.

We have used a new UMAP algorithm to learn a transformation from the embeddings of Bodo to English.

We used Procrustes and UMAP for alignment part, and our model seems to be working almost equally well as compared to Procrustes.

Novelty

The final metrics that we obtained after the alignment of Bodo are:



```
disparity for bodo with Procrustes model:  0.7943667363024065
MSE for bodo with procrustes model:  5.0596607407796594
MSE for bodo with UMAP:  5.080941988833991
```

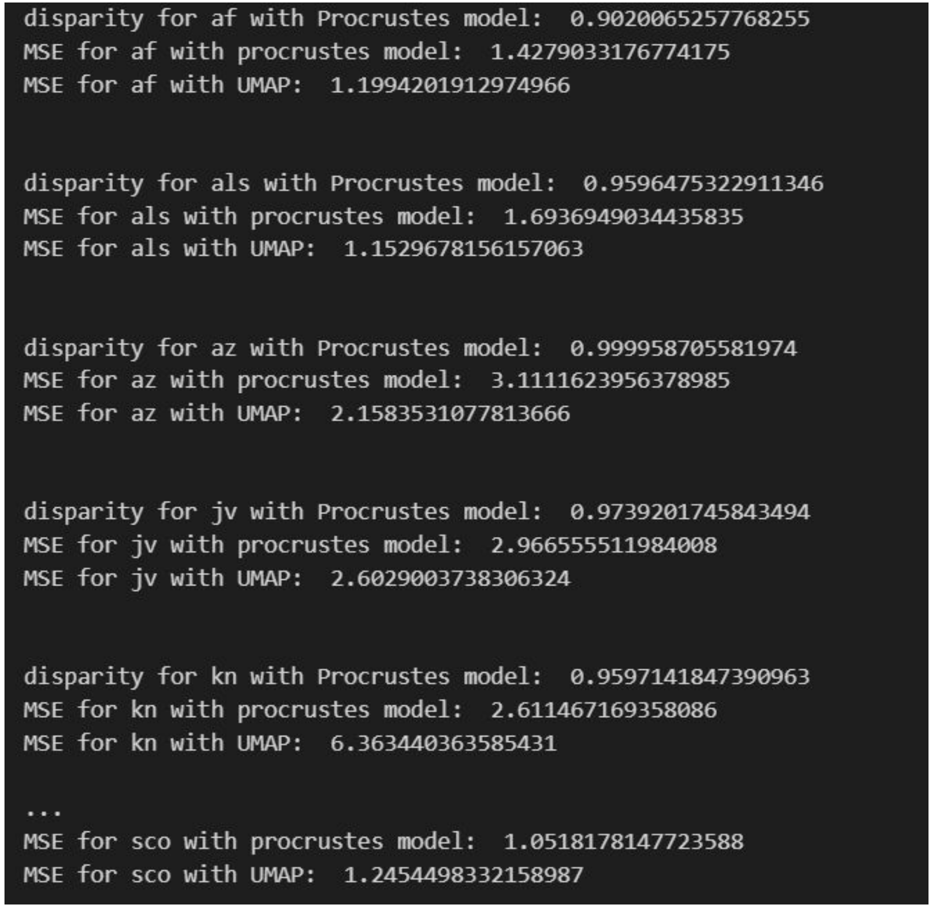
The contributions of the paper are:

- Provides high-quality human annotated translations for words in 298 languages to English.
- Use these lexicons to train and evaluate the largest published collection of aligned word embeddings on 157 different languages.

Our contributions:

- Provide a fully annotated dataset for Bodo language, a low resource language from India.
- Used this dataset to train and get the aligned embedding for Bodo with English language using FastText.
- Use a new algorithm for alignment (UMAP) and compare it to the existing methods.

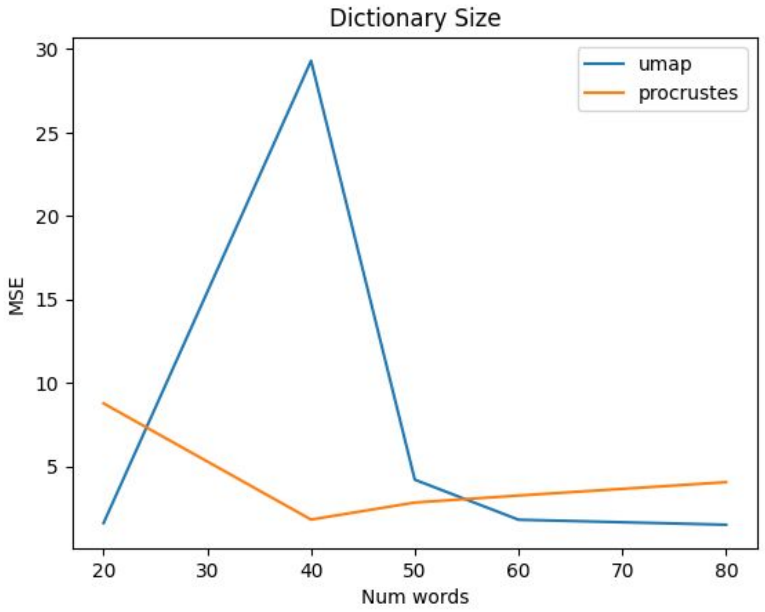
Novelty



```
disparity for af with Procrustes model:  0.9020065257768255
MSE for af with procrustes model:  1.4279033176774175
MSE for af with UMAP:  1.1994201912974966


disparity for als with Procrustes model:  0.9596475322911346
MSE for als with procrustes model:  1.6936949034435835
MSE for als with UMAP:  1.1529678156157063


disparity for az with Procrustes model:  0.999958705581974
MSE for az with procrustes model:  3.1111623956378985
MSE for az with UMAP:  2.1583531077813666


disparity for jv with Procrustes model:  0.9739201745843494
MSE for jv with procrustes model:  2.966555511984008
MSE for jv with UMAP:  2.6029003738306324


disparity for kn with Procrustes model:  0.9597141847390963
MSE for kn with procrustes model:  2.611467169358086
MSE for kn with UMAP:  6.363440363585431

...
MSE for sco with procrustes model:  1.0518178147723588
MSE for sco with UMAP:  1.2454498332158987
```

Novelty

We also tried to experiment with the behaviour that what is the comparison of MSE loss when we gradually increase the size of the dictionary. We experimented with dictionary sizes of 20, 40, 50, 60, 70 and 80. The results of UMAP vs Procrustes is as below:



Thankyou

Sabaikor