Cuneiform Classification

Keegan Moseley

Dataset Reminder

- My dataset has two columns: Cuneiform and Lang
- Cuneiform is an ancient writing system, used to write several different languages
- 7 Categories
    - SUX - 53673
    - NEA - 32966
    - STB - 17817
    - LTB - 15947
    - NEB - 9707
    - MPB - 5508
    - OLB - 3803

Akkadian Dialects (NEA, STB, LTB, NEB, MPB, OLB)

OBJECTIVES

- Primary Effort: Build a model to predict the language/dialect of input cuneiform
- Secondary Efforts:
  - Create a model to predict the Language of input cuneiform
    - SUX (Summerian)
    - AKK (Akkadian)
  - Create a basic visualization of the embeddings of these symbols

Actions and Results

- Actions
  - Most of my time was spent tuning preprocessing parameters, applying dimension reduction techniques, and determining which classification algorithm to use.
    - This work was done for the classification model of all the labels.
- Results:
  - Unexpected outcomes from most dimension reduction techniques.
  - Successfully accomplished my three goals:
    - Primary:
      - Classification Model for All Labels
    - Secondary
      - Classification Model for Summerian and Akkadian
      - Cuneiform Embedding Model with CBOW

Machine Learning Pipeline

Most of my time was spent determining which combination of :

- Preprocessing techniques
- Preprocessing hyperparameters
- Dimension reduction techniques
- Dimension reduction hyperparameters
- Classification Algorithms
- Classification Algorithm Hyperparameters

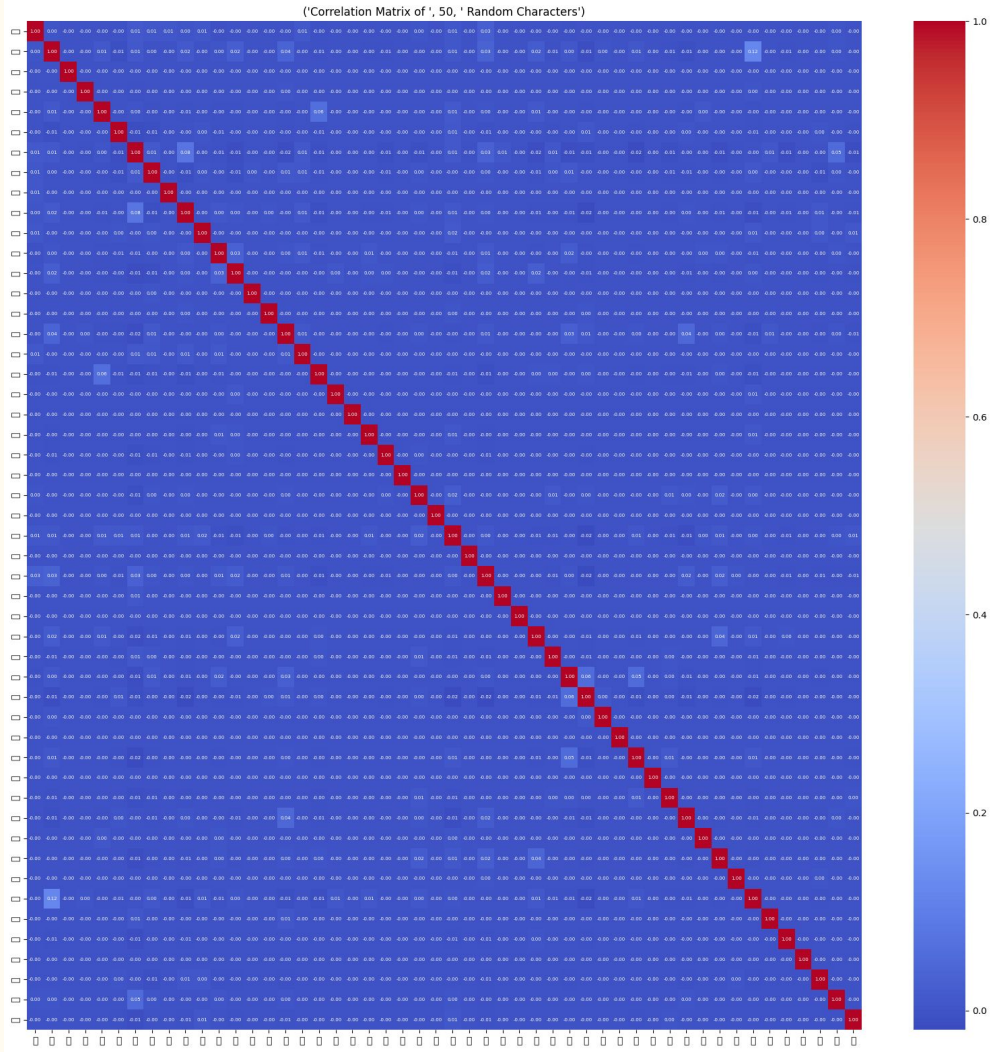
Would yield the best result for my final models.

Pipeline Parameters

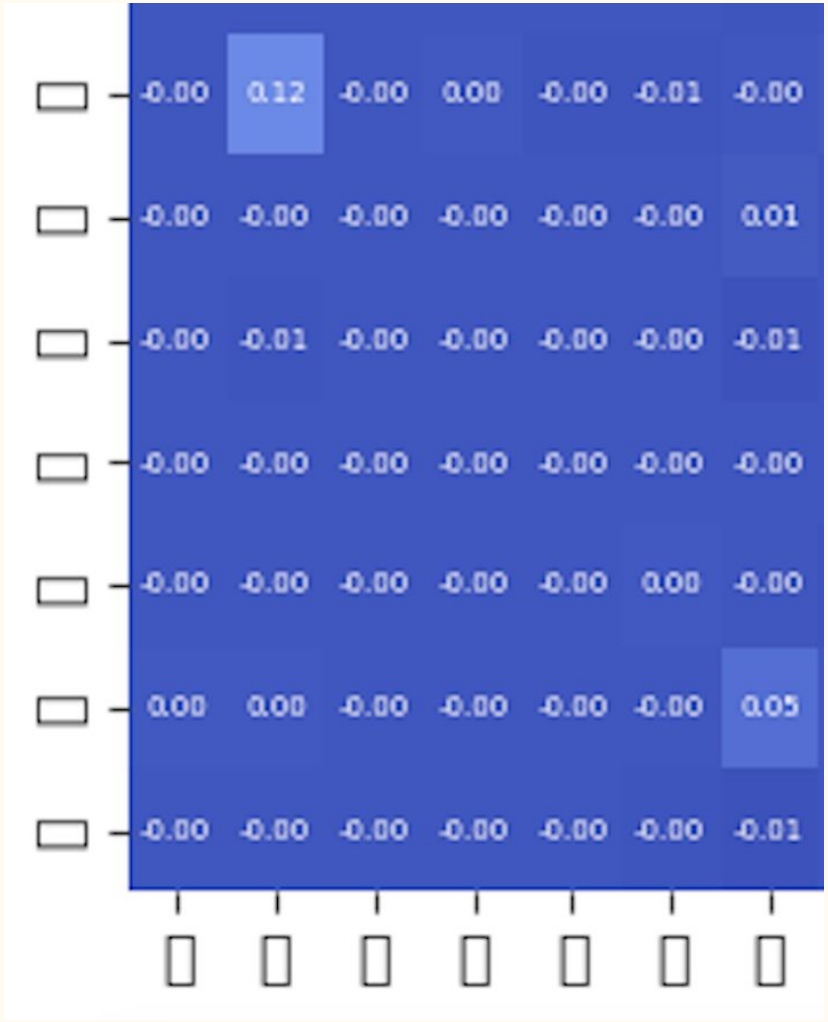
- Preprocessing techniques
  - TF-IDF Vectorization vs Count Vectorization
    - Minimum and Maximum Document Frequency
  - Outlier Detection
    - Isolation Forest and Local Outlier Factor
      - Contamination
    - Effects of not including outlier detection
  - Dimension Reduction
    - PCA, Truncated SVD, UMAP, (limited) TSNE, and LLE
      - How many dimensions to reduce to, and which hyperparameters to pick for each
- Model Selection
  - Logistic Regression, Random Forest, Histogram Gradient Boosted Tree, K-Nearest Neighbors
    - Apply a randomized grid search for each
- Fed multiple different combinations of these preprocessing techniques into the model grid searches
  - Used results to develop intuitions about which techniques and models to focus on for further tuning

Data Correlations

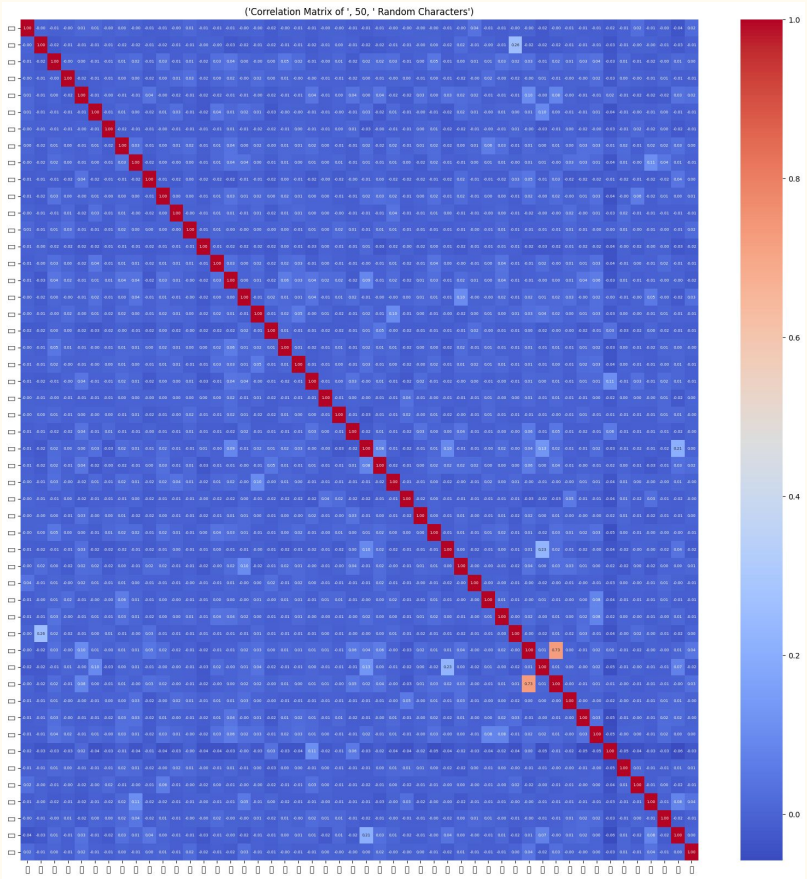
- Reminder: Reduces dimensions creating uncorrelated principal components. Relies on linearly correlated columns.
- Explored Correlations in the Columns:
  - Took several random samples of 50 columns of a TF-IDF Document Term Matrix, and created a correlation matrix

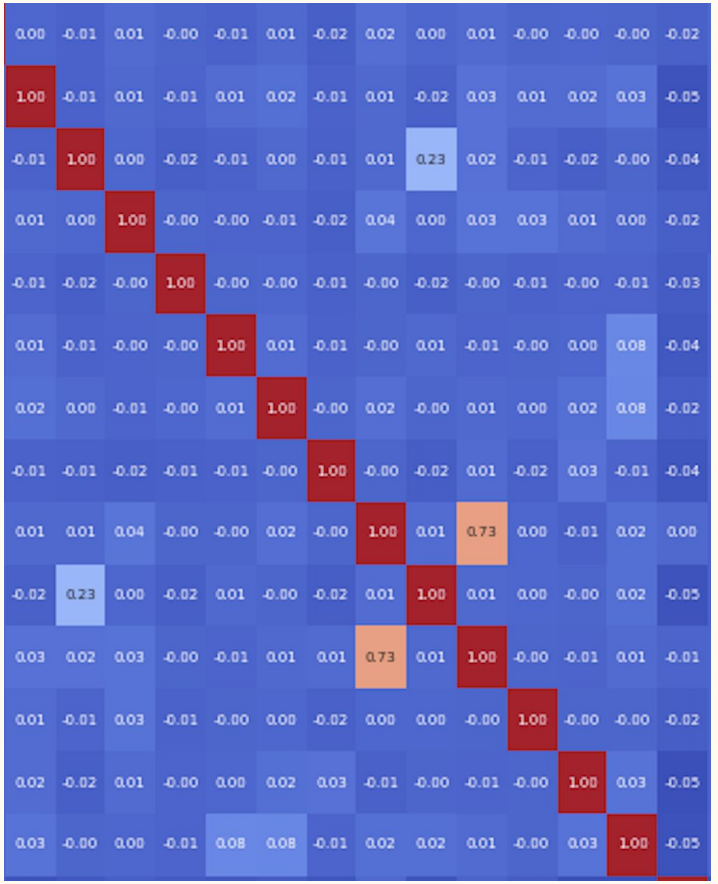

('Correlation Matrix of ', 50, ' Random Characters')

1.0
0.8
0.6
0.4
0.2
0.0


-0.00 0.12 -0.00 0.00 -0.00 -0.01 -0.00
-0.00 -0.00 -0.00 -0.00 -0.00 -0.00 0.01
-0.00 -0.01 -0.00 -0.00 -0.00 -0.00 -0.01
-0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00
-0.00 -0.00 -0.00 -0.00 -0.00 0.00 -0.00
0.00 0.00 -0.00 -0.00 -0.00 -0.00 0.05
-0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.01

Reduced the DTM to the 81 most common terms via min_df



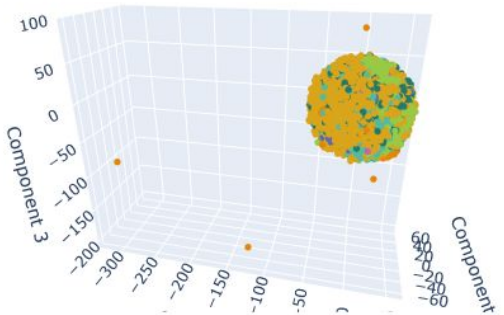


Correlation Conclusions

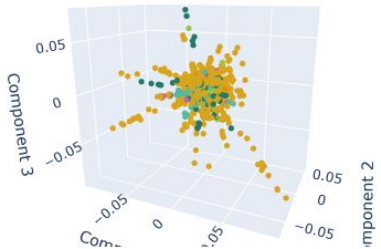
- There are a few instances of significant linear correlations, so PCA could be applied to at least slightly reduce the dimensionality.
- Decided to experiment with Truncated SVD as well for dimension reduction. This technique uses SVD (Singular Value Decomposition) to reduce dimensions
  - Chose this technique since it can provide better results with sparse matrices, and it is commonly used with Natural Language Processing
  - Also allows the user to view how much variance is captured by each dimension, just like with PCA
- Ended up inputting PCA and Truncated SVD dimension reductions to the models. I experimented with a large range of dimensions, from 50 - 400 due to the relatively low correlations for most columns.

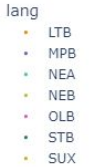
Dimension Reductions - Poor Visualization Results

Visualized TSNE of Language Training Data (Perplexity=10)



Visualized LLE of Language Training Data





Response to poor visualization results

- Despite having investing a lot of time tuning hyperparameters of dimension reductions in 3 dimensions, I failed to get a visualization where the different labels separate from one another.
- I decided to reduce to a larger range of dimensions. I experimented using a wide range of dimensions, from 15 - 50.

Model Selection Results

- After all that time invested in preprocessing tuning and dimension reduction techniques, I saw the best performance with:
  - Not reducing dimensions with anything other than min_df when vectorizing.
    - Using min_df = 25 eliminated nearly 200 columns, and the resulting grid search results were only slightly worse than not having a min_df.
      - I decided that this tradeoff was worth it. With a lessened curse of dimensionality, I hoped to further tune model hyperparameters to produce an even stronger model.
  - 2% Outlier Removal
  - HistGradientBoost Classification Model
    - Histogram Gradient Boost is a modification of Gradient Boost, which is faster & thus easier to use on large datasets
- Similar results when using TF-IDF Vectorization and Count Vectorization, even with instances of Count Vectorized DTMs producing better results
  - Decided to create a final model with each of these vectorization techniques.

HistGradientBoost Grid Results

Grid Results of Gradient Boost Model, with TFIDF and No Dimension Reduction

	params	mean_test_score	rank_test_score
0	{'random_state': 251, 'max_iter': 200, 'max_depth': 100, 'loss': 'log_loss', 'learning_rate': 0.1}	0.794158	1
8	{'random_state': 251, 'max_iter': 200, 'max_depth': 200, 'loss': 'log_loss', 'learning_rate': 0.1}	0.794158	1
1	{'random_state': 251, 'max_iter': 200, 'max_depth': 100, 'loss': 'log_loss', 'learning_rate': 0.2}	0.790683	3

No Min-Df

mean_test_score
0.794158
0.794158
0.794158

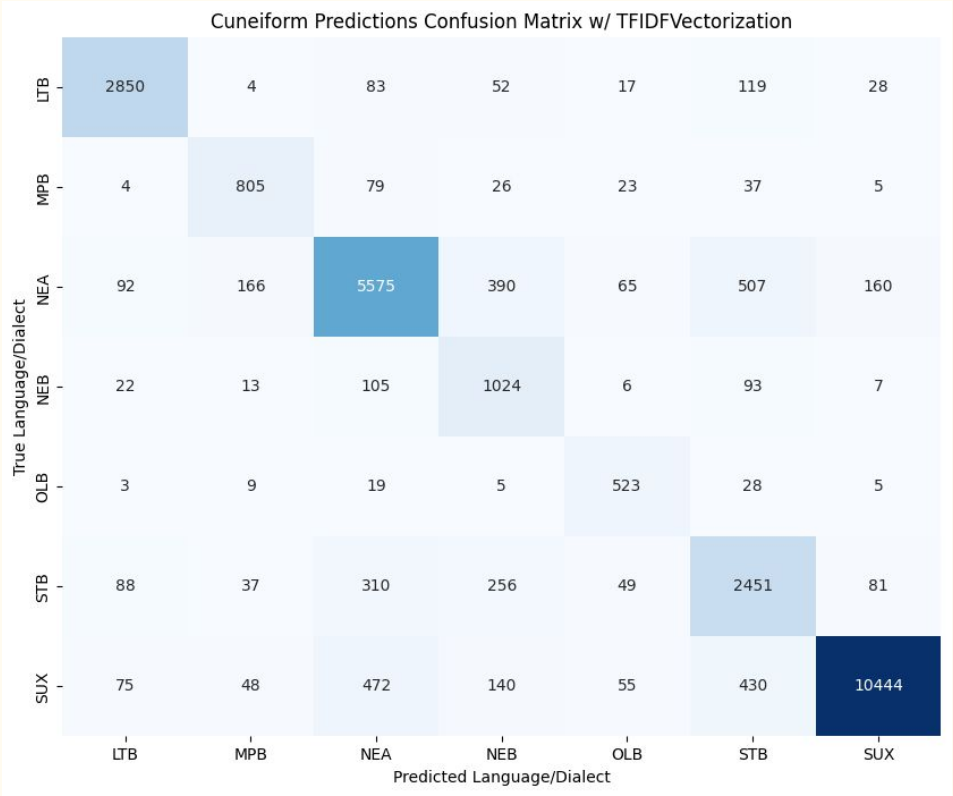
Min-Df = 25

mean_test_score
0.793220
0.793220
0.793220

Final Model Results After Further Tuning, using TF-IDF

	precision	recall	f1-score	support
LTB	0.90	0.91	0.91	3134
MPB	0.82	0.74	0.78	1082
NEA	0.80	0.84	0.82	6643
NEB	0.81	0.54	0.65	1893
OLB	0.88	0.71	0.79	738
STB	0.75	0.67	0.71	3665
SUX	0.90	0.97	0.93	10730
accuracy			0.85	27885
macro avg	0.84	0.77	0.80	27885
weighted avg	0.85	0.85	0.84	27885

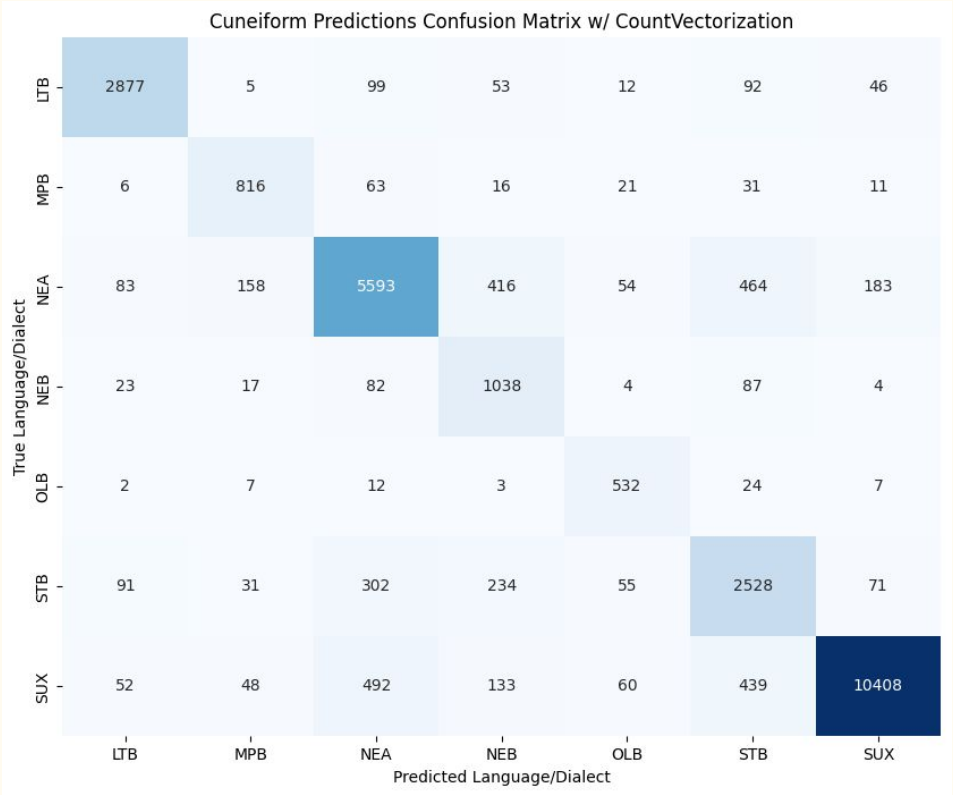
Average f1 score : 0.797304077647461



Final Model Results After Further Tuning, using Count Vectorization.

	precision	recall	f1-score	support
LTB	0.90	0.92	0.91	3134
MPB	0.85	0.75	0.80	1082
NEA	0.80	0.84	0.82	6643
NEB	0.83	0.55	0.66	1893
OLB	0.91	0.72	0.80	738
STB	0.76	0.69	0.72	3665
SUX	0.89	0.97	0.93	10730
accuracy			0.85	27885
macro avg	0.85	0.78	0.81	27885
weighted avg	0.85	0.85	0.85	27885

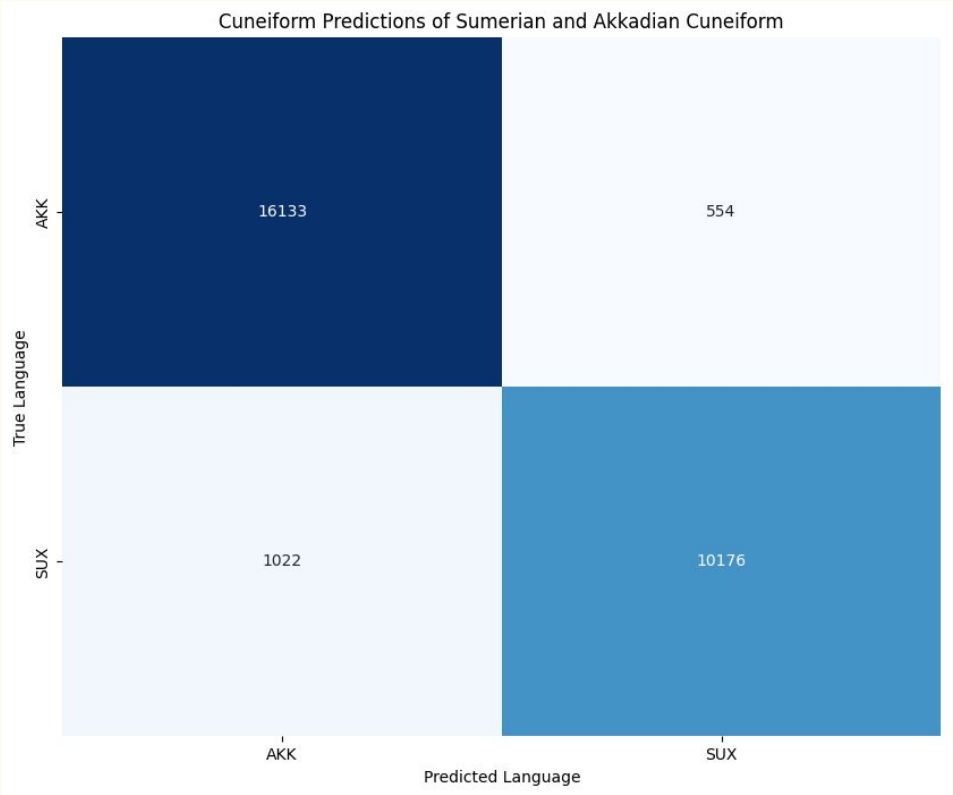
Average f1 score : 0.8070378641165845



Secondary Goal: Sumerian VS Akkadian Classification

	precision	recall	f1-score	support
AKK	0.97	0.94	0.95	17155
SUX	0.91	0.95	0.93	10730
accuracy			0.94	27885
macro avg	0.94	0.94	0.94	27885
weighted avg	0.94	0.94	0.94	27885

Average f1 score : 0.9407795342957026

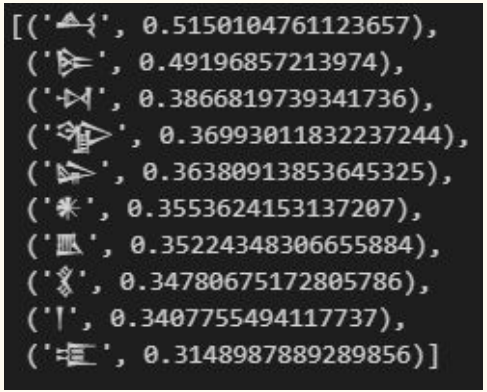


Secondary goal: Continuous Bag of Words Embedding

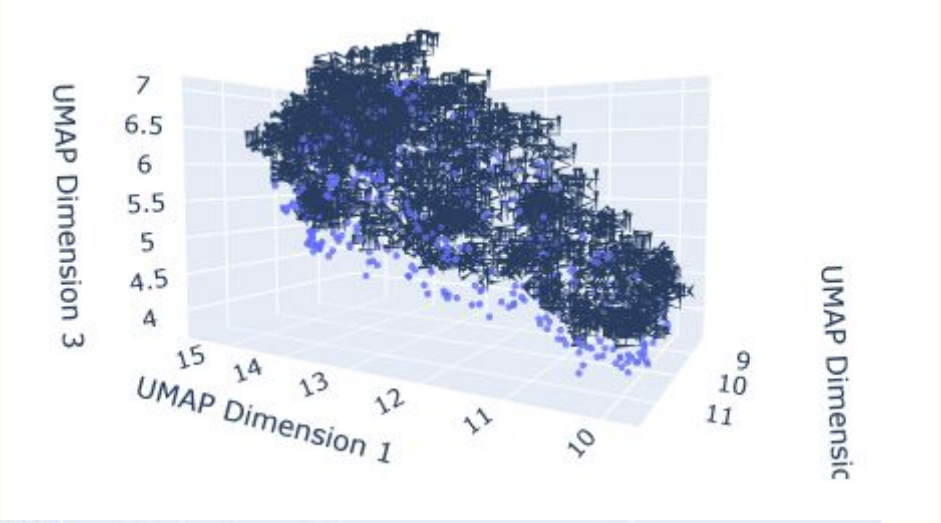
CBOW Embedding reminder:

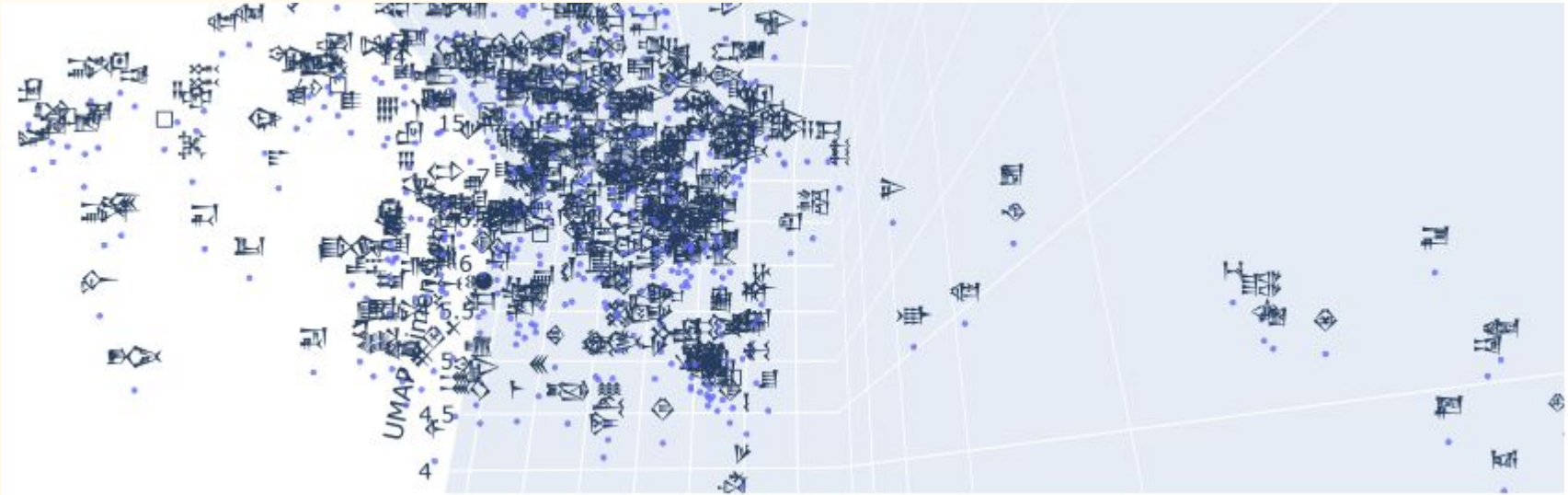
- Creates vectors for the meaning of a word, based on the context around it
- Allows for a measure of the similarity of two words.

I applied this embedding to the cuneiform data. It creates interesting opportunities for comparison. For example, I can get the 10 symbols with the most similar vectors to “□”, and a measure of their similarity:



UMAP Visualization of Cuneiform Embeddings





CONCLUSION AND ANALYSIS

- With my main cuneiform identification task, I ran into some unexpected results.
  - Worse results when reducing dimensions with techniques such as UMAP
    - Possible Explanations:
      - Few strong linear correlations
      - Random Forest and HistGradBoost are less affected by the curse of dimensionality, due to sampling features for splits and determining the most important feature to split by.
      - Hyperparameter tuning
  - Count Vectorization $>$ TF-IDF Vectorization
    - Lack of common terms. With vectorizing with max_df $= 0.15$, only three columns are eliminated.
      - TF-IDF penalizes common terms, with the assumption that common terms are less important for deriving meaning. Since so few terms are common, the additional context provided by TF-IDF is limited.
    - Many symbols are syllabic. For many words in these languages I've done the equivalent of tokenizing "apple" into "ap" & "ple".

Potential Space for Improvement

- Experiment with nonlinear dimension reduction techniques
  - Kernel PCA
- Train a model based on average embeddings
  - For each row of cuneiform, find the vectors of each symbol, and average their vectors.
  - May lead to better results due to:
    - An average “meaning” being found for each row
    - Lowers the dimensionality to however many vectors are created.
- Mistake: Grid training scores measured in “f1_weighted” instead of “f1_micro”

Does Cuneiform Identification Matter?

- I originally picked this topic purely out of personal interest.
  - Bachelors in History
  - Interest in Archeology & Material Culture
- If someone developed a better model to predict the language or dialect of cuneiform, it could have practical uses (especially if it is trained via images or integrated with an image transcription tool)
  - There are many more ancient artifacts than experts that can read cuneiform.
  - Looting remains a massive problem. Looting removes artifacts from their archeological context, permanently limiting how much information experts can learn from them.

Iraq Reclaims 17,000 Looted Artifacts, Its Biggest-Ever Repatriation

The cuneiform tablets and other objects had been held by the Museum of the Bible, founded by the family that owns the Hobby Lobby craft store chain, and by Cornell University.

Link

Additional Intersections of Machine Learning and History

- Identifying text from x-ray scans of delicate artifacts

- Reconstructing Damaged or Missing Ancient Inscriptions

- Deciphering lost languages