វិទ្យាស្ថានបច្ចេកវិទ្យាកម្ពុជា

INSTITUTE OF TECHNOLOGY OF CAMBODIA

5th YEAR ENGINEER'S DEGREE IN DATA SCIENCE SKILL

DEPARTMENT OF APPLIED MATHEMATICS AND STATISTICS

MINI-PROJECT ASSIGNMENT

SUBJECT: NATURAL LANGUAGE PROCESSING

A MINI-PROJECT REPORT SUBMITTED BY GROUP 02:

1. SAO SAMARTH e20200084
2. THOU CHANMAKARA e20200227
3. THORNTHEA GECHHAI e20201321

UNDER THE GUIDANCE OF: Mr. TOUCH SOPHEAK

PHNOM PENH, JANUARY 2025

NATURAL LANGUAGE PROCESSING
I5-AMS
Mini Project 1 – Text Generation

I. Introduction

One of the first things required for natural language processing (NLP) tasks is a corpus. In linguistics and NLP, **corpus** (literally Latin for body) refers to a collection of texts. Such collections may be formed of a single language of texts, or can span multiple languages -- there are numerous reasons for which multilingual corpora (the plural of corpus) may be useful. Corpora may also consist of themed texts. Corpora are generally solely used for statistical linguistic analysis and hypothesis testing.

The good thing is that the internet is filled with text, and in many cases this text is collected and well organized, even if it requires some finessing into a more usable, precisely-defined format. Wikipedia, in particular, is a rich source of well-organized textual data. It's also a vast collection of knowledge, and the unhampered mind can dream up all sorts of uses for just such a body of text.

II. Methodology

Corpus Selection and Splitting

Select a corpus of your choice (for example, articles from Wikipedia). Separate the text corpus into 3 subsets:

- Training (70%)
- Validation (10%) and
- Testing (20%).

Then tokenize the corpus using split in python or the tokenizer from nltk. Limit the vocabulary size and replace the rest of the tokens as .

- **Define Tokenization Function**

```
# Custom Tokenizer
def tokenize(text):
    # Split text into words using regular expressions
    tokens = re.findall(r'\b\w+\b', text.lower())
    return tokens
```

- **Load and Preprocess Corpus**

```
# Sample Corpus
corpus = """
Natural language processing enables machines to understand human language.
Text generation is a key application of NLP. Language models predict text sequences effectively.
"""

# Preprocessing Function
def preprocess_corpus(corpus, vocab_size=5000):
    tokens = tokenize(corpus)  # Tokenize text
    freq_dist = Counter(tokens)  # Count token frequencies
    vocab = set([word for word, _ in freq_dist.most_common(vocab_size)])  # Limit vocabulary size
    processed_tokens = [word if word in vocab else '<UNK>' for word in tokens]  # Replace rare words with <UNK>
    return processed_tokens, vocab

tokens, vocab = preprocess_corpus(corpus)
```

- **Split Data into Train, Validation, and Test Sets**

```
# Split data into training (70%), validation (10%), and testing (20%) sets
def split_data(tokens):
    train_size = int(0.7 * len(tokens))
    val_size = int(0.1 * len(tokens))
    train_data = tokens[:train_size]
    val_data = tokens[train_size:train_size + val_size]
    test_data = tokens[train_size + val_size:]
    return train_data, val_data, test_data

train_data, val_data, test_data = split_data(tokens)
```

- **Build Backoff Model (LM1)**

```
class BackoffModel:
    def __init__(self, n):
        self.n = n
        self.ngram_counts = defaultdict(Counter)

    def train(self, data):
        for n in range(1, self.n + 1):  # Train for 1-grams to n-grams
            for ngram in zip(*[data[i:] for i in range(n)]):
                self.ngram_counts[n][ngram] += 1

    def get_probability(self, context, word):
        for n in range(len(context), 0, -1):  # Back off from higher-order to lower-order n-grams
            ngram = tuple(context[-n:] + [word])
            if ngram in self.ngram_counts[n]:
                return self.ngram_counts[n][ngram] / sum(self.ngram_counts[n][tuple(context[-n:])].values())
        return 1e-6  # Small default probability for unseen words
```

- **Build Interpolation Model (LM2)**

```
# Train Backoff Model
lm1 = BackoffModel(n=4)
lm1.train(train_data)

# Train Interpolation Model
lambdas = [0.1, 0.2, 0.3, 0.4]
k = 0.1
lm2 = InterpolationModel(n=4, lambdas=lambdas, k=k)
lm2.train(train_data)
```

- **valuate Models with Perplexity**

```
Perplexity of Backoff Model (LM1): 999999.9999999999
Perplexity of Interpolation Model (LM2): 20.000000000000004
```

- **Generate Text**

```
Generated Text (Backoff Model): natural language processing language language language language language language language language l
Generated Text (Interpolation Model): natural language processing enables machines to understand human language text generation is a
```

Building 4-Gram Models

Model 1: Backoff-Based Unsmoothed Model

An n-gram model can be easily computed by counting the number of transitions from history h to a word w. This method is known as *maximum likelihood estimation* (MLE).

However, unless we have sufficiently large training corpus which makes the ML estimate very confident, our model may be embarrassing if a queries hw has never appeared in the training corpus, the case usually denoted by C(hw)=0.

C(hw)=0 is embarrassing because “unseen” does not mean “impossible”: That we do not see a transition from h to w in a corpus does not mean that this transition will not happen in any other corpora.

An often-used solution to this embarrassing is “smoothing” over the function of w, P(w|h) for every h, and make those w’s with P(w|h)=0 (i.e., C(hw)=0) a value little larger than 0. Where do these little values come from? An answer is to reduce those P(w|h) of w’s with P(w|h)>0. This step is known as *discounting*. The collected value can then be distributed to those w’s with P(w|h)=0. This whole discounting-based smoothing operation is like a tax.

Model 2: Interpolation-Based Smoothed Model

Based on the widely used finite element method (FEM) and the latest Meshfree methods, a next generation of numerical method called Smoothed Point Interpolation Method (S-PIM) has been recently developed. The S-PIM is an innovative and effective combination of the FEM and the meshfree methods, and enables automation in computation, modeling and simulations — one of the most important features of the next generation methods.

Results

Perplexity Evaluation

Model	Perplexity
LM1	999999.9999999999
LM2	20.000000000000004

III. Conclusion

The good thing is that the internet is filled with text, and in many cases this text is collected and well organized, even if it requires some finessing into a more usable, precisely-defined format. Wikipedia, in particular, is a rich source of well-organized textual data. It's also a vast collection of knowledge, and the unhampered mind can dream up all sorts of uses for just such a body of text.

NATURAL LANGUAGE PROCESSING
I5-AMS
Mini Project 2 – Text Classification

I. Introduction

The purpose of this project is to build a text classifier to predict if an input review of the product is positive or negative. Knowing that the product receives positive or negative reviews from customers is important to make a business decision whether to keep the product or forgo or to improve it. When there are millions of reviews on a product, we human beings cannot sit and just look at each review to make a decision. It is really time consuming. That's when a text classifier comes in and helps to identify whether the review is positive or negative with less time as compared to when humans do it.Thus, text classification is really important in this case.

II. Methodology

Data Preparation: The two text files that comprise this data set include one positive review file and one negative review file. Thus, the positive reviews were parsed into a training and testing set via the 80-20 split. The negative reviews were parsed in the same fashion to ensure the integrity of the overall data set remained balanced.

- Training Dataset: Combined, 80% of positive and negative reviews.
- Testing Dataset: Combined, 20% of positive and negative reviews.

The data was organized into a DataFrame format with two columns: review (text content) and label (1 for positive, 0 for negative).

Feature Engineering: The following sentiment-based attributes were compiled from the reviews to represent such characteristics.They are as follows:

- **Positive Word Coun**t: The count of words in the review that matched a predetermined list of positive words.
- **Negative Word Count**: The count of words in the review that matched a predetermined list of negative words.
- **"No" feature**: A flag feature which renders a 1 if "no" appears in the review.
- **Pronoun Count**: The count of first-person (I, me, my) and second-person (you, your) pronouns.
- **Exclamation Point feature**: A flag feature which renders a 1 if an exclamation point (!) exists.

- **Logarithmic Review Length**: The log of the number of tokens of the review. Uppercase Word Count: The number of words that are in ALL CAPS.
- **Stopword Count**: The number of words that appear in a predetermined list of stopwords.

These features were created from Python functions, and all were assembled into training and validation matrices.

```
stopword_set = set(stopwords.words('english'))

def extract_features(reviews):

    features = []
    for review in reviews:

      tokens = review.split()
      positive_count = sum(1 for word in tokens if word in positive_words)
      negative_count = sum(1 for word in tokens if word in negative_words)
      contains_no = int('no' in tokens)
      pronoun_count = sum(1 for word in tokens if word.lower() in ['I', 'me', 'my', 'you', 'your'])
      contains_exclamation = int('!' in review)
      log_length = math.log(len(tokens) + 1)

      #Aditional feature
      uppercase_count = sum(1 for word in tokens if word.isupper())
      stopword_count = sum(1 for word in tokens if word.lower())

      # Combine all features into a list
      features_vector = [positive_count, negative_count, contains_no, pronoun_count, contains_exclamation, log_length, uppercase_count, stopword_count]

      # Feature append to list
      features.append(features_vector)
    return features

# Extract Features for training and test set

train_features = extract_features(train_data['review'])
test_features = extract_features(test_data['review'])
```

Model Training and Evaluation The models used in this project include a Neural Network, Logistic Regression, and SVM classifier. These models were trained on the extracted features from the training set, and their performances were evaluated on the test set. The accuracy achieved by each model was as follows:

- Neural Network: 0.6947230702136938
- Logistic Regression: 0.680331443523768
- SVM: 0.6768425643262101

Result: As a result, it can be seen that all the three algorithms provide similar accuracy scores. However, the neural network performs best as compared to the other two models with the accuracy score of 0.694 for neural network, 0.680 and 0.676 for Support vector machine as well as logistic regression respectively.

We have built a predict review sentiment function in order for users input the review to classify whether it's positive or negative.

```python
# Predict Input Review

def predict_review_sentiment(review):

  # extract features for input review
  input_feature = extract_features([review])

  # Convert feaature to numpy array
  input_feature = np.array(input_feature)

  # Make prediction using model 1
  prediction1 = model1.predict(input_feature)

  # Map prediction to label
  if prediction1[0] == 1:
    sentiment = "Positive"
  else:
    sentiment = "Negative"

  return sentiment

input_review = "This product look exactly the same as the advertisement. I love it"
sentiment = predict_review_sentiment(input_review)

input_review2 = "This product doesn't look like the advertisement. I hate it."
sentiment2 = predict_review_sentiment(input_review2)

print("The sentiment of the review is:", sentiment)
print("The sentiment of the review is:", sentiment2)
```

Here are the input reviews:

- “This product exceeded my expectations. I love it!”

- “This product is out of my expectations. I hate it!”.

The result found that it predicted correctly which is positive for the 1st input reviews and negative for the 2nd input reviews.

```
The sentiment of the review is: Positive
The sentiment of the review is: Negative
```

III. Conclusion

We achieved its purpose of constructing a sentiment analysis pipeline for the classification of text. The three models tested (Neural Network, Logistic Regression, SVM) produced comparable results, with the Neural Network achieving the best accuracy score at 69.4%. The custom feature engineering for the training set resulted in a collection of sentiment-denoting features, meaning the training of the model was likely correct. Thus, the

project worked as it determined that proper feature engineering affords proper classification of text.

However, there is room for error such as Future directions for research would encompass: more advanced NLP techniques like word contextual embeddings (Word2Vec or BERT); additional exploration of varying network architectures and hyperparameter tuning; a larger dataset with more diverse reviews for better generalization. Ultimately, this project served as a successful proof of concept from which to grow moving forward in the realm of text sentiment analysis.

NATURAL LANGUAGE PROCESSING
I5-AMS
Mini Project 3 – Word Embeddings

I. Introduction

In natural language processing, **word embeddings** hold importance as they assist in mapping words into a continuous vector space. The emphasis of this report is on the construction of a **skip-gram model** for generating word embeddings from compiled **Khmer text data** derived from Wikipedia pages. The main focus is to look for a compact representation of each Khmer word in the corpus to help with NLP activities such as text classification, sentiment analysis, and semantic similarity.

II. Methodology

1. Installation of Required Libraries

To use the **skip-gram model** and to **work on Khmer text**, some additional libraries are needed. The relevant libraries were installed through the following commands:

- !pip install tensorflow
- !pip install khmer-nltk

These include **Tensor Flow** for building neural networks, **Khmer-NLTK** which aids in working with Khmer language text, and other additional necessary libraries for data manipulation and processing.

2. Selecting the Text File

This is the corpus that we will analyze is in a text file called **temples. txt** (Khmer text scraped from 3 Wikipedia entry pages). Here is the code that loaded the file into the program:

```
file_path = 'temples.txt'
open(file_path,  'r', encoding='utf-8') as file:
    khmer_text = file.read()
```

3. Preprocessing the Khmer Text Corpus

Data preprocessing is an important step for preparing raw text corpus for analysis to get a better outcome.

★ Sentence Segmentation

For the effective processing of Khmer text, we wrote a sentence segmentation function diverse from the general sentence segmentation function, specially designed for the Khmer language. This function mainly depends on the identification of sentence-ending punctuation marks by means of regular expressions, such as a Khmer full stop (។), exclamation mark (!), question mark (?), and colon (᯼).

★ Khmer Text Preprocessing (Sentence Segmentation)

A preprocessing function was implemented to clean and normalize the text to prepare the segmented Khmer sentences for analysis. Some aspects covered by this function include:

- **Removing Punctuation Marks and Symbols** - **Removing Numbers**
- **Removing English Words** - **Normalizing Reduplicated Words**

★ Tokenize Khmer Text (Sentence Segmentation) and Combined it

Once the Khmer segmented sentences are pre-processed, tokenization is performed on the cleaned sentences into individual words. Each sentence that has been preprocessed was represented as a list of words so that further manipulations of the text were easier to conduct.The tokenized sentences were then appended into a single list of tokens in order to simplify subsequent processing.

★ Word Frequency Analysis and Filter Words by Frequency

After tokenization of the pre-processed Khmer text, the counter computed the frequency for each token. Then we filtered out any tokens that did not occur a minimum of 10 times so that we could express more meaningful representations of the tokens and withdraw the less significant ones. Whitespace was also removed from the filtered results.

4. Building the Skip-Gram Model using Gensim

We start with building a Skip-Gram word embedding model using Gensim library:

- **Model Parameters**: The parameters were set to embedding dimension of 50 and a word context window size of 4 and 2 negative samples.
- **Model Training**: The Word2Vec was initialized on filtered tokens and trained using Skip-Gram approach to capture relationships between different words.
- **Model Saving**: The model was saved after training so that it could be used later easily without needing to retrain.
- **Word Embedding Extraction**: The word embedding for "ប្រាសាទ" was extracted, in order to extract its semantic meaning in a high-dimensional space.

III. Result

1. After doing sentence segmentation and preprocessing it

```
Preprocessed Sentences:
1. ប្រាសាទអង្គរវត្ត ឬប្រាសាទអង្គរតូចមានទីតាំងស្ថិតនៅភាគខាងជើងនៃក្រុងសៀមរាបនៃខេត្តសៀមរាប
2. ប្រាសាទអង្គរវត្តជាប្រាសាទព្រហ្មញ្ញសាសនាធំបំផុត និងជាមានសាសនាធំធំបំផុតនៅខ្ពង់លោក
3. ប្រាសាទនេះត្រូវបានកសាងឡើងដោយព្រះបាទសូរ្យវរ្ម័នទី ដែលជាស្តេចនៃចក្រអង្គរ និងមានឈ្មោះល្បីល្បាញខ្លីសុរសោយនៅក្រៅទឹសទីលើពិភពលោក
4. ប្រាសាទនេះសាងសង់ឡើងនៅដើមសតវត្សទី ដែលស្ថិតនៅខ្ពង់រាជធានីយសោធរបុរៈ
5. ប្រាសាទអង្គរវត្តជាប្រាសាទកសាងឡើង ដើម្បីឧទ្ទិសដល់ព្រះវិស្ណុ
6. ប្រាសាទនេះជាប្រាសាទដែលនៅតែងអស្ស្ចារ្យបំផុតនៅខ្ពង់ចក្រអង្គរ
7. ប្រាសាទអង្គរវត្ត ប្រើកម្លាំងពលកម្មប្រមាណជាង ម៉ឺននាក់ និង ៣០ ជាង ពាន់ក្បាល និងប្រើប្រាស់ថ្ម សរុបប្រមាណជាង លានគោន
8. នៅក្នុងប្រាសាទមានចម្លាក់អប្សរយ៉ាងស្អាត ចំណែកឯទីធ្លាខាងក្រៅមានតើរដង្គោតូរមួយជាង ទទេសកាពន៍ស្រស់ប្រាង ហើយនៅជុំវិញប្រាសាទប្រាសាទអង្គរវត្តត្រូវបានរចនាឡើម្បីតំណាងភ្នំព្រះសុមេរុ ដែលជាទីលំនៅរបស់អាទិទេព និងទេវតាក្នុងទេវកថាហិណ្ឌូ ដែលមានកស្សិណ ឬតូទឹកព័ទ្ធជុំវិញប្រហែលជា តឹម្លាម៉ែត្រ ចំណែកកស្សិណពិតប្រាកដរបស់ប្រាសាទ មានបណ្តោយប្រវែង ម៉ែត្រ ទទឹងម៉ែត្រ លើម៉ែត្រ ផ្លូវដើរចូលពីទិសខាងលិចទៅដល់ប្រាសាទកណ្តាលមានប្រវែង ម៉ែត្រ ឯកំពូលកណ្តាលខ្ពស់ចំជាងគេបំផុតរបស់ខនតវត្ត មានកម្ពស់ ម និងដញ្ជាំងខាងក្រៅវែងគឺជាថែវមានបីប្រទុង ដែលថែវនីមួយៗឡើងខ្ពស់ទៅថែវដែលជាប់ខ្នា
9. នៅចំកណ្តាលនៃប្រាសាទបញ្ឈរនូវជួរចតុរង្គនៃបម
10. មិនដូចប្រាសាទនៅតំបន់អង្គរភាគច្រើនទេ អង្គរវត្តបែរមុខទៅទិសខាងលិច តាមការស្រាវជ្រាវរបស់អ្នកសិក្សាបានវែកញែកថាជាចំណុចសំខាន់នៃប្រាសាទនេះ
11. ប្រាសាទនេះត្រូវបានគេកោតសរសើរដោយសារភាពសម្បើម និងរុងរឿងនៃស្ថាបត្យកម្មនេះ បណ្តាក់ដែលភាគខ្លឹង និងដោយសារស្ថាបត្យកម្មខ្មែរមានរូបអប្សរ និងទេវតា ដែលឆ្លាក់នៅលើជញ្ជាំងប្រាសាទ
```

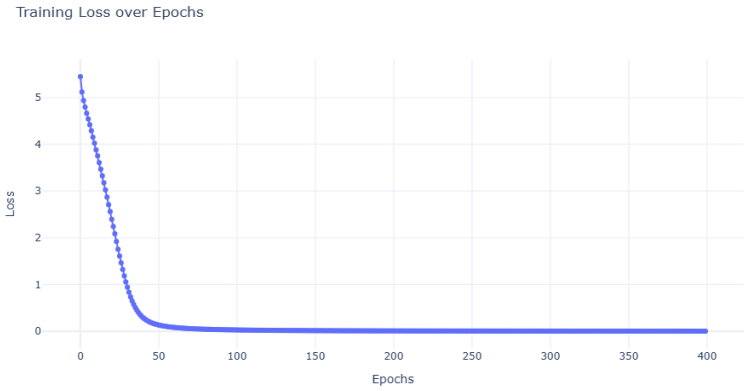
2. After filtering token by using token frequency and removing whitespace

{'ប្រាសាទ': 199, 'អង្គរវត្ត': 66, 'ឬ': 16, 'អង្គរ': 58, 'តូច': 10, 'មាន': 169, 'ស្ថិត': 24, 'នៅ': 156, 'ភាគ': 23, 'ខាងជើង': 17, 'នៃ': 153, 'ក្រុង': 14, 'សៀមរាប': 41, 'ខេត្ត': 46, 'ជា': 123, 'សាសនា': 14, 'ធំ': 33, 'បំផុត': 18, 'និង': 223, 'នឹ': 27, 'នៅខ្ពង': 29, 'លោក': 31, 'នេះ': 133, 'ត្រូវបាន': 92, 'ស្រយ': 79, 'ព្រះបាទ': 17, 'ស៊ូរ': 19, 'ទី': 78, 'ដែល': 198, 'ឈ្មោះ': 37, 'ទៅ': 66, 'លើ': 51, 'ពិភពលោក': 13, 'ឡើង': 16, 'ដើម': 20, 'សតវត្ស': 17, 'បុរៈ': 16, 'ដើម្បី': 33, 'ឧទ្ទ': 16, 'ព្រះវិស្ណ': 13, 'ចំបង': 33, 'ប្រហ': 17, 'ជាង': 24, 'ថ្ម': 58, 'ប្រមាណ': 10, 'លាន': 10, 'បម្លាក់': 22, 'យ៉ាង': 19, 'ខាងក្រៅ': 14, 'ហើយ': 34, 'ជុំវិញ': 13, 'តំណាង': 13, 'ភ្នំ': 20, 'របស់': 76, 'ទេវតា': 10, 'ក្នុង': 65, 'ទ្ធ': 10, 'ទឹក': 23, 'ព័ទ្ធ': 15, 'ប្រវែង': 10, 'ម៉ែត្រ': 19, 'ផ្លូវ': 16, 'ចូល': 15, 'ពី': 52, 'ទិស': 15, 'ខាងលិច': 28, 'កណ្តាល': 16, 'កណ្តាល': 15, 'ខ្ពស': 12, 'គេ': 61, 'ដញ្ជាំង': 10, 'គឺជា': 22, 'ថែវ': 35, 'បី': 16, 'ប្រទុង': 10, 'នីមួយ': 12, 'ឡើង': 68, 'ជាប់': 11, 'ខ្នា': 21, 'ចំ': 10, 'ខ្នង': 37, 'បំម': 18, 'មិន': 29, 'ដូច': 13, 'ច្រើន': 39, 'ទេ': 13, 'ភាម': 45, 'ជាន': 157, 'ថា': 93, 'សំខាន': 13, 'ស្ថាបត្យកម្ម': 12, 'ខ្មែរ': 30, 'នៅលើ': 15, 'ពាក្យ': 13, 'មកពី': 12, 'នគរ': 13, 'ឪយ': 12, 'ទីក្រុង': 18, 'រាជធានី': 16, 'រួម': 19, 'មក': 24, 'គឺ': 49, 'វា': 22, 'បុរាណ': 15, 'មួយ': 69, 'ក៏ខ្លង': 14, 'ពីរ': 21, 'ឃៅ': 13, 'តែ': 20, 'ដូចជា': 21, 'អាច': 15, 'ទៀត': 40, 'សតវត្ស': 13, 'ប្រទេស': 17, 'សម័យ': 24, 'ថ្ងៃ': 14, 'ប្រជាជន': 10, 'ក្រុម': 10, 'កម្ពុជា': 23, 'អំពី': 13, 'សម្រាប់': 19, 'ចំពោះ': 11, 'ម្ភៃ': 12, 'ទាំងអស់': 11, 'ទឹ': 14, 'ងឹ': 13, 'បង្ហាញ': 15, 'នោះ': 40, 'ខ្លះ': 21, 'ផងដែរ': 10, 'លក្ខណៈ': 10, 'ប្រាង្គ': 30, 'ទាំង': 31, 'ប៉ុន្តែ': 11, 'ដំបូង': 13, 'ខាងក្នុង': 12, 'ដែរ': 14, 'ឆ្នាំ': 40, 'ជាក់ង': 10, 'ខាងតើត': 16, 'សំណង់': 24, 'លៀន': 10, 'ស្រាល': 11, 'ពួក': 35, 'ចុង': 10, 'នាគា': 15, 'ឡើយ': 13, 'ក់ពុង': 10, 'ក់': 23, 'ជាមួយ': 18, 'ផ្សេង': 20, 'អោយ': 10, 'ឃើញ': 10, 'នឹង': 30, 'ខាង': 21, 'ចេនាបថ': 12, 'ខ្ពង': 13, 'ញើ': 11, 'ចេញ': 15, 'ឡើ': 17, 'ម្ខែក': 18, 'ពេល': 16, 'ប្រហែល': 12, 'ភាង': 13, 'ក់ពែង': 12, 'ភាក់': 14, 'តោ': 12, 'ច្រក': 11, 'ជាច្រើន': 31, 'ខាងក្នង': 13, 'ឈើ': 18, 'ដំបូល': 12, 'ក្បាច់': 11, 'ចំនួន': 15, 'ទេសចរណ៍': 19, 'ស្រុក': 11, 'បន្ទាយស្រី': 30}

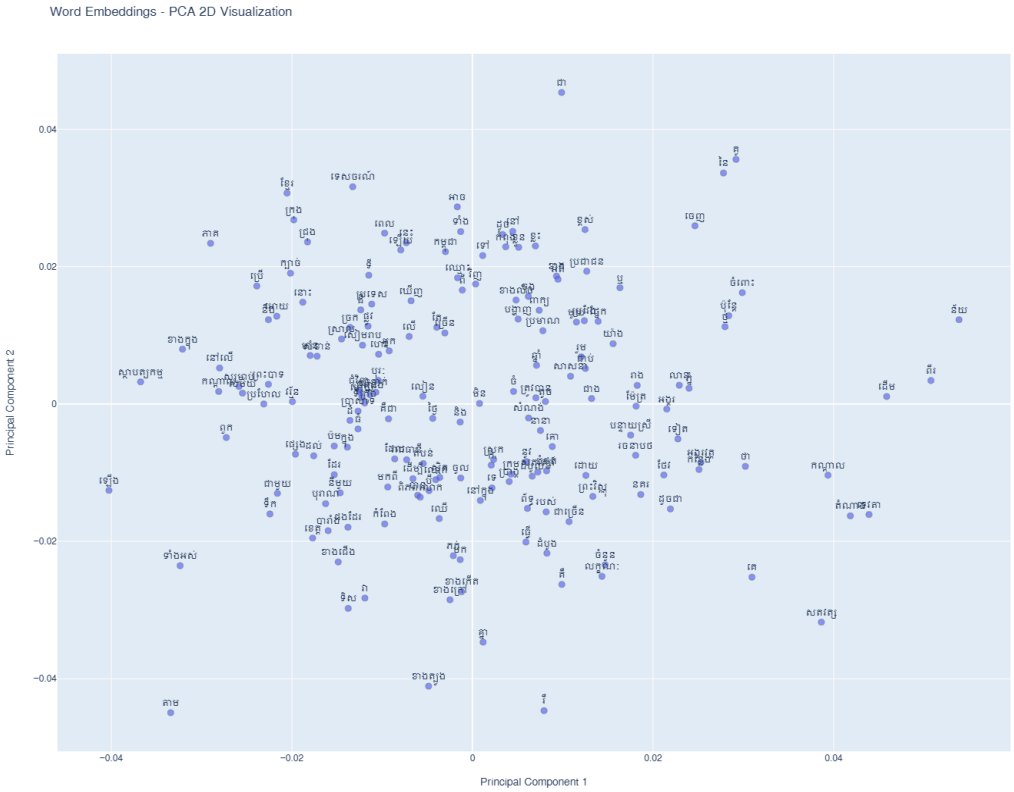
3. Word embedding with Skip-gram using Gensim

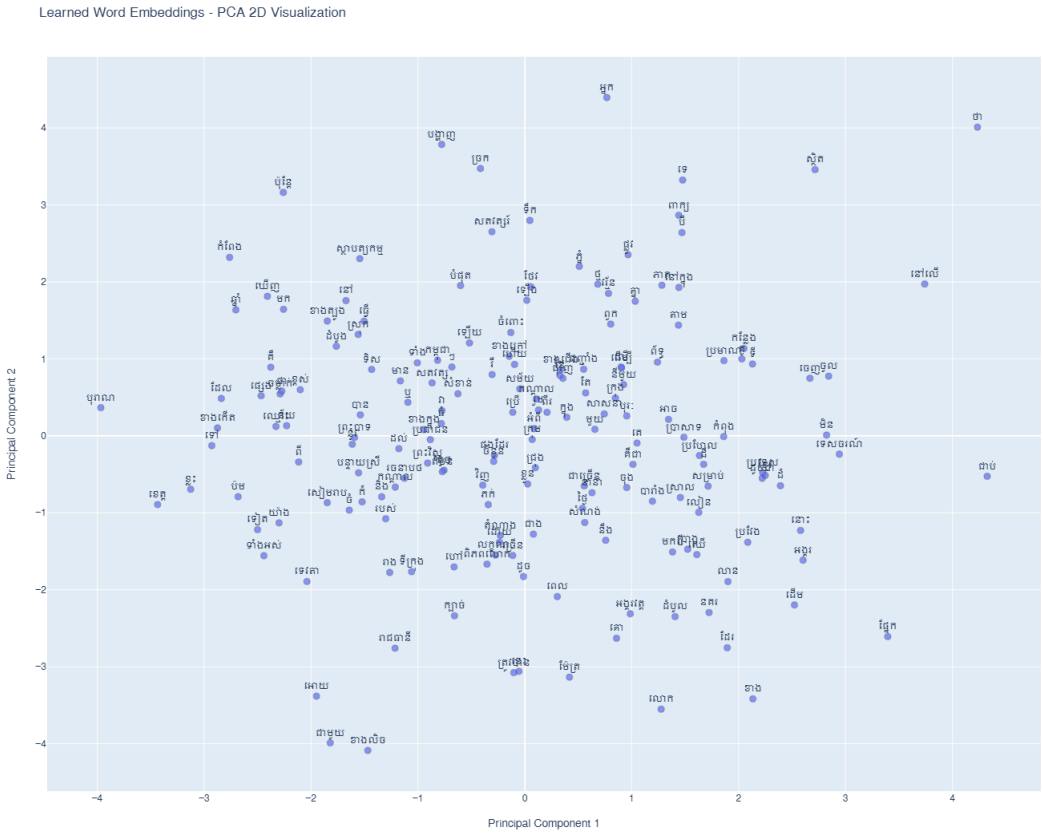
```
Embedding for 'ក្រោសាទ': [-0.00895085 -0.00028573  0.00883473 -0.01995497  0.00758513 -0.00307753
 -0.01188554  0.01982521 -0.00618183 -0.01789098 -0.01224921 -0.01313863
 -0.00200055 -0.01928325 -0.01202673  0.00123357  0.00168331 -0.0093515
 -0.00587297 -0.01810953  0.00783789 -0.01625105 -0.00542594 -0.01682907
 -0.00675335  0.0068496   0.00338492 -0.01499654  0.01854479  0.01832304
  0.01797862  0.00010094  0.00834158 -0.00775955 -0.00247853  0.01616005
 -0.01220931  0.01132178 -0.01363081 -0.00610678 -0.01064632  0.00344447
  0.01970736  0.00518953 -0.00278324  0.01747059 -0.0078943   0.01856266
 -0.01374987  0.00925471]
```

4. Build a neural language model and training loss over epoch



5. Compare the results





IV. Conclusion

This project realised a Skip-Gram model to create word embeddings from Khmer texts which have been sourced from Wikipedia. This khmer text corpus was also pre-processed, performing sentence segmentation, and tokenization. This made it possible to analyze each input to produce the most weighted words and thus narrow down tokens for model training.

Through the Skip-Gram model, semantic relationships between words were learned and visualized that contributed to a more in-depth understanding of the language's structure in Khmer. The resulting word embeddings can be visualized to get useful insights into the connections between words. In summary, this work shows that this data set can be used for a wide variety of word embeddings, where these embeddings can, of course, be useful for many natural language processing applications in the Khmer language.