Precog Recruitment Task 2025 Report
Language Representations

Name: Anirudh Sankar
Roll No.: 2023111024

I. Introduction

The following report details my methodologies I used, the insights I've gained and the results I observed while completing the Programming Task part of Precog Recruitment Task 2025.
The following parts were attempted and completed:

1) Part 1: Dense Representations
2) Part 2: Cross-lingual Alignment
3) Bonus: Harmful Word Associations

II. Part 1: Dense Representations

For this section, I decided to use the English News 2020 1M Corpus from the Leipzig Corpora Collection. The corpus was made using text material from news website. This lead to some interesting observation to wards the end

A. Constructing a Co-occurrence Matrix

There are two main approaches followed for calculating co-occurrence matrices:

- Bag of Words and,
- Term Frequency Inverse Document Frequency (TF-IDF)

I chose to use the Bag of Words method to construct my matrix as the corpus is presented as one document with many unrelated sentences. This meant a simple bag of words approach would be better

However to mitigate the issue of stop-words (like 'a', 'an', 'the', 'is', etc.) which appear very frequently leading to high co-occurrence matrix scores, these word were removed when calculating co-occurrence values.

After filtering out stop-words from the corpus of 1 million sentences, 304,951 unique tokens. This implies that the co- occurrence matrix has 92,995,112,401 entries.

Making a numpy array of almost 93 billion entries of type float64 needs 693 GB of memory.

So my next approach was to assign an id for each unique token identified in the corpus and then use a Python dictionary to store only the non-zero values. The keys of the dictionary are tuples of the two token IDs and the value is the corresponding co-occurrence number.

The no. of key:value pairs in the dictionary i.e. its length will give us an idea of how many non-zero entries are there in the co-occurrence matrix.

The length of the dictionary was found to be 18,805,223. This means out of 93 billion entries only around 19 million entries have non-zero values which is just 0.02% of the entire matrix i.e. the matrix is very sparse.

Thus I decided to use Python packages to represent the sparse matrix to perform mathematical operations on the matrix more easily.

B. Dimensionality Reduction

Now that we have the co-occurrence matrix. Each row is a vector representing the word. However this may not be useful as each word contains 304,951 features (i.e. no. of unique tokens in the corpus). Thus we need to remove to features that may not contribute much to the word embedding

The reduction of features is called 'Feature Selection' and is commonly done via 'Dimensionality Reduction'. There are multiple methods of reducing dimensions for co-occurrence matrices:

- Singular Value Decomposition (SVD)
- Principle Component Analysis (PCA)
- t-distributed Stochastic Neighbor Embedding (t-SNE)

I decided to perform Singular Value Decomposition as it can directly be applied to spare matrices without centering the data which can be computationally expensive

SVD of co-occurrence matrix A is given by,

$$A = U\Sigma V^\top,\ \Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) \tag{1}$$

Where

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0 \tag{2}$$

The singular values tell us the importance of that feature. Since they are always in decreasing order, the first k columns represent the k most important features and the rows form a basis for a vector space of k dimensions.
Thus if the dimensions we need are d, then to extract word embeddings, we select the first d columns of U.
Thus we now have d-dimensional word embeddings of unique tokens in the corpus

SVD was performed and word embeddings of dimensions 30, 100, 300 and 500 were extracted.

C. Evaluate quality of the embeddings

Word embeddings are useless if we cannot extract meaningful information out of them. One way to verify if the embeddings have captured information is to check if embeddings of words with similar meanings appear closer together.
This can be done by calculating the distance between vector. Vectors of words with similar meanings must show lower distances. There are two kinds of distance calculations:

- Scale-Variant: Accounts for Magnitude and Direction.
  Eg: Manhattan and Euclidean Distance
- Scale-Invariant: Accounts for Direction only.
  Eg; Cosine Similarity

Since word-embeddings are sparse vectors in high-dimensional spaces, they may have big differences in magnitude based simply on frequency of the word. Thus I've decided to use Cosine Similarity as the primary metric to study similarity of word embeddings to evaluate them
The word embeddings are evaluated with 3 of the popular datasets of human annotated similarity scores

1) MEN dataset (3000 word pairs)
2) Simlex-999 (999 word pairs)
3) wordsim353 (353 word pairs)

Before we proceed, there two key points to be noted.

- Our embeddings are generated from news sources which are in general very different from normal speech and do not touch upon many mundane topics
- Out of the 9.25 million unique word pairs, only a small fraction of pairs are tested (max. 3000 pairs in MEN dataset which is just 0.0333% of pairs)
  Note: The 9.25 million figure is obtained by subtracting same-word pairs and dividing the remainder by 2 since if (x, y) is a word pair, then (y, x) also has the same value in the matrix i.e. the co-occurrence matrix is symmetric.

Thus it is expected that scores are not as good as general purpose pre-trained word embeddings.

The output of Cosine Similarity lie in the range [-1, 1]. But the human annotated scores of the datasets may have different distributions, so how do we compare them ?
To tackle this problem, I have decided to use Spearman Rank Correlation.
Suppose we have two random variables X and Y which have two distinct probability distributions. For example:

$$X = [1.2, 3.4, 0.5, 2.1] \text{ and } Y = [15.2, 89.4, 2.3, 45.6] \quad (3)$$

After applying rank transformations, they become,

$$X_{ranks} = [2, 4, 1, 3] \text{ and } Y_{ranks} = [2, 4, 1, 3] \quad (4)$$

The Spearman Rank Correlation is calculated as

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

where ρ is Spearman's rank correlation coefficient,
d_i is the difference between the ranks of each pair, i.e. (rank(X_i) - rank(Y_i)),
n is the number of paired observations. (5)

For X and Y, this would give a perfect Spearman correlation of 1.0, despite very different original distributions of X and Y.

By using this method we can see the correlation between cosine similarity of our generated word embeddings and human annotated scores of the used datasets. The following table summarizes the results for word embeddings of various dimensions across the three datasets

The results of this analysis are presented in Table I

We see two interesting results from this exercise:

1) The word embeddings almost universally perform worst on SimLex-999 across all dimensions.
   This because co-occurrence matrices capture relatedness than strict similarity. To quote the website of SimLex-999 dataset :
   SimLex-999 provides a way of measuring how well models capture similarity, rather than relatedness or association.
   To put it simply with an example, word pairs like "doctor-hospital" have higher cosine similarity than "doctor-surgeon" because they occur more frequently together. But common sense dictates "doctor" is more similar to "surgeon" than "hospital"
   The embeddings perform better on wordsim353 and MEN datasets as they account for relatedness also.

2) We also notice that improvements start to diminish as no. of dimensions increases.
   Since the jump from 300-dimensions to 500-dimensions only provides minor improvement in two of the 3 datasets. I have chosen to use 300-dimensional embeddings for further analysis as it is a good trade-off between performance and memory usage.

Simply throwing around numbers is of no use, the Spearman Correlation ranges from [-1, 1] where 1 suggests the two distributions are the same, and -1 indicates the distributions are inverses of each other. But how do these results compare to popular pre-trained vectors ? This is what we investigate next

I
Correlation Results for Various Datasets and Word Embedding Dimensions

Dataset (Word Pairs Matched)	Embedding Dimension	Correlation	P-value
MEN (2998/3000)	30-dim	0.1798	<0.001
	100-dim	0.2725	<0.001
	300-dim	0.3680	<0.001
	500-dim	0.4306	<0.001
SimLex-999 (994/999)	30-dim	0.0817	0.01
	100-dim	0.1061	<0.001
	300-dim	0.1686	<0.001
	500-dim	0.1814	<0.001
WordSim353 (335/353)	30-dim	0.1900	0.0005
	100-dim	0.2775	<0.001
	300-dim	0.3046	<0.001
	500-dim	0.3103	<0.001

To carry out this evaluation, I picked out for popular word-embeddings

1) word2vec vectors (300-dimensions trained on Google News)
2) GloVe (840B tokens, 300-dimensions trained on Common Crawl)
3) fasttext (300-dimensions trained on Common Crawl)
4) BERT ('bert-base-uncased' pre-trained model hosted on HuggingFace with the context: " 'X' is similar to 'Y' ")

The first 3 being static embeddings, we can use the cosine-similarity method from before. BERT poses some unique challenges in this analysis as it a contextual word embedding. Thus we give the context mentioned above with one of the words of the dataset as X and Y is set as special token [MASK] we then use the fill-mask functionality with target supplied as the other word from the dataset and record the model's confidence score.
This confidence score is also another distribution but as discussed before, the rank transformation takes care of that

I now present the results of this analysis in Table II

We gather some insights fromthese results too:

- The pre-trained word embeddings also perform worse across the board on SimLex-999 dataset. This demonstrates the difficulty of representing strict similarities via corpus based training of word vectors.
- The BERT embedding performs worse in comparison to static embeddings due to its contextual nature. The supplied context may not have been optimal. Another key difference that might contribute to worse performance in similarity test is that BERT uses sub-word tokenization (i.e. running is split into "run" and "-ing")

Now I would like to move to provide some visualizations

- 2D Visualization using Matplotlib using t-SNE
- 3D Visualization using Plotly and TensorBoard

The 2D-visualization was generated after running the t-SNE algorithm for maximum 1000 iterations is shown in Fig 1

The 3D-visualization was carried out with the help of TensorBoard as it more interactive enabling rotation of the vector space along all three axes and searching through the words in the vector space to see its nearest neighbors. It can reduce dimensionality of embeddings using PCA and t-SNE. Here for demonstration I have inserted screenshots of 3D space obtained after PCA is performed.
See Figs 2, 3 and 4

III. Part 2: Cross-lingual Alignment

Cross-lingual alignment involves finding a Transformation, such vector space of one language becomes more similar to that of other language.
For example, if we have "cat" in English space and बिल्ली in Hindi space. The goal is to find a transformation which when applied to English space makes the embedding for "cat" and "बिल्ली" have greater cosine similarity

There are both Supervised and Unsupervised methods of doing this:

- Supervised Method: Procrustes Alignment
- Unsupervised Method: Cross-domain Similarity Local Scaling (CSLS)

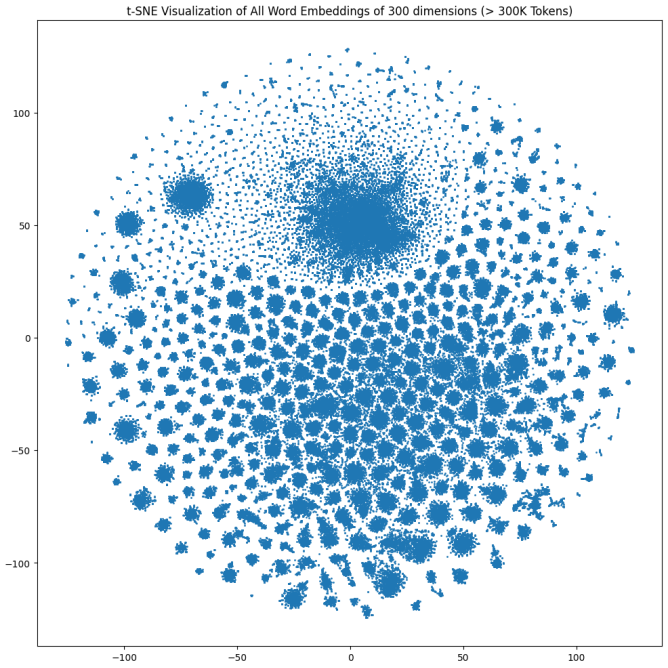
I have decided to use Facebook Research's MUSE project (https://github.com/facebookresearch/MUSE) to obtain a bilingual dictionary for Hindi and English and then performed iterative Procrustes Alignment to generate a transformation W that transforms the English embeddings into the Hindi embedding vector space.
For the embeddings I decided to use fastText Wiki word vectors as this would ensure accurate translations because Wikipedia article on which the vectors were trained have translations in multiple languages.

To find the Procrustes Alignment, we must solve the Orthogonal Procrustes Problem, i.e. we need to fine optimal Orthogonal rotation matrix R such that

$$\underset{\mathbf{R}}{\text{minimize}} \|\mathbf{X} - (\mathbf{YR})\|_F \tag{6}$$

II
Correlation Results for Various Datasets and Pre-Trained Embedding

Dataset	Embedding Method	Pairs Matched	Correlation	P-value
MEN	Word2Vec	2946/3000	0.7820	< 0.001
	GloVe	3000/3000	0.8049	< 0.001
	fastText	3000/3000	0.8427	< 0.001
	BERT	2812/3000	0.5712	< 0.001
SimLex-999	Word2Vec	999/999	0.4420	< 0.001
	GloVe	999/999	0.4083	< 0.001
	fastText	999/999	0.4644	< 0.001
	BERT	980/999	0.1116	0.0004
WordSim353	Word2Vec	353/353	0.7000	< 0.001
	GloVe	353/353	0.7379	< 0.001
	fastText	353/353	0.7805	< 0.001
	BERT	332/353	0.4443	< 0.001



1. t-SNE Visualization of 300-dimensional word embeddings generated using co-occurrence matrix

where,
X and **Y** are the two vector spaces and,
$\|\cdot\|_F$ denotes the Frobenius norm.

This is equivalent to finding the closest orthogonal matrix to $M = BA^T$ i.e. what we need to find is:

$$\min_{\mathbf{W}} \|\mathbf{W} - \mathbf{M}\|_F \tag{7}$$

This can be done easily by performing SVD on M,

$$M = U\Sigma V^T \tag{8}$$

And W is given by,

$$W = UV^T \tag{9}$$

In iterative Procrustes, we perform this alignment multiple times, that is we find W to align English embeddings X to Hindi vector space Y and then apply Procrustes again on the transformed vectors XW to find better alignment.

After 30 iterations,
Average cosine similarity before alignment: 0.0078
Average cosine similarity after alignment: 0.4728

We see that now the similarity between words and its translations have improved. There we can say some alignment has taken place. However, using a more robust Procrustes implementation may yeild better results.

IV. Bonus Task: Harmful Associations

Harmful associations creep up in word-embeddings through training due to inherent biases that exists in the

word. Identifying these biases can help towards developing safer AI models.

For identifying biases in word embeddings, I have decided to make use of The Word Embedding Fairness Evaluation Framework (WEFE): https://github.com/dccuchile/wefe
WEFE contains multiple datasets for evaluating biases but for the purpose of this task I have decided to use the Word Embedding Association Tests (WEAT) for evaluating biases
The orignial paper which introduced WEAT (https://arxiv.org/pdf/1608.07187) provides the words they used the for evaluating biases.
For the task I have decided to replicate one of test used in the paper, *Replicating Implicit Associations for Career and Family* in two types of word embeddings:

1) Static: word2vec vectors (300-dimensions trained on Google News)
2) Contextual: BERT ('bert-base-uncased' pre-trained model hosted on HuggingFace)

The words used in the experiment are:

- Male names: John, Paul, Mike, Kevin, Steve, Greg, Jeff, Bill.
- Female names: Amy, Joan, Lisa, Sarah, Diana, Kate, Ann, Donna.
- Career words : executive, management, professional, corporation, salary, office, business, career.
- Family words : home, parents, children, family, cousins, marriage, wedding, relatives

For word2vec embeddings, we use cosine similarity as the metric of evaluation as before

For BERT, the context for Career words was provided as:
"[MASK] is a {career_word}"
and for Family words the context was:
"[MASK] has a {family_word}"
As before the evaluation metric is the confidence score of the transformer using the fill-mask pipeline having target word set as male and female name words

I now present the results in Table III

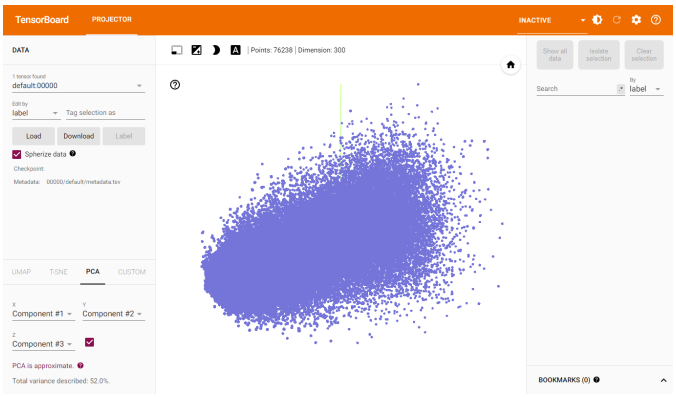
III
Gender Bias in Pre-trained Embeddings

Embedding Type	Category	Male Name Average	Female Name Average
Word2Vec Static	Career	0.0304	0.0144
	Family	0.0454	0.1077
BERT Contextual	Career	0.000105	0.000013
	Family	0.000105	0.000723

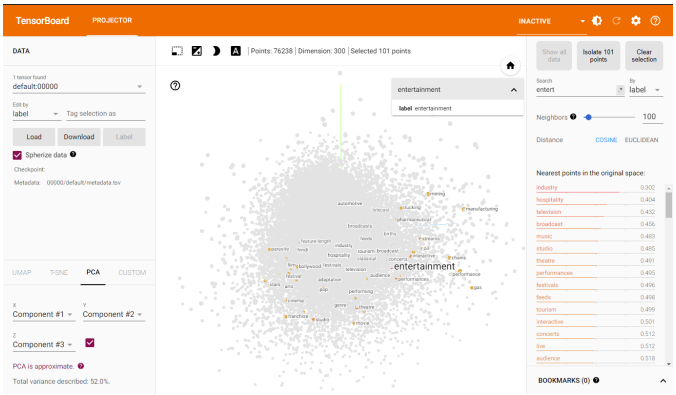
From these results we can clearly infer biases in both word2vec and BERT. The male name words score higher when combined with career words than female name words. And, female name words score higher when combined with family words than male name words. This shows glaring gender biases in both embeddings and may produce harmful associations if used in text generation tasks.

V. References

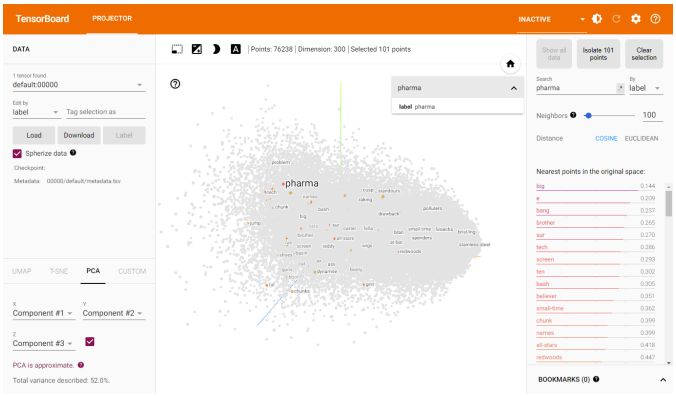
[1] Dimensionality Reduction Techniques — PCA, LCA and SVD
[2] SimLex-999 Website
[3] Racial Bias in BERT



2. Screenshot of TensorBoard screen showing all vectors



3. Screenshot of TensorBoard showing closest neighbors of 'entertainment'



4. Screenshot of TensorBoard showing closest neighbors of 'pharma'