Intent Classification and Sequence Labelling

Abhijit Nayak
Indiana University Bloomington
nayakab@iu.edu

Abhisek Panigrahi
Indiana University Bloomington
abpani@iu.edu

Abstract

This paper contains the implementations of both static and contextual word embeddings for intent classification. Besides, word embedding techniques we also perform some linguistics analysis on the corpus and a downstream task Sequence labelling. For sequence labelling, we have used the BIO tagging. The very first approach is to implement the static word embedding techniques like Skipgram (SG) and Count Vectorizer (CV) and then sentence bert (SBERT) a contextual embedding method to classify the intents of the user. Implementation of these 2 embedding techniques (static, contextual) helped us compare the performance of the classification models in each of the methods (SG, CV and SBERT) of each technique. Finally, a downstream task like Sequence labelling was performed by using Beginning Inside Outside (BIO) tagging approach to identify a new entity from the corpus itself.

1 Introduction

The computerized categorization of text data based on client goals is known as intent classification. Natural language processing (NLP) research has long recognized it as a difficult topic. This issue is one of the first steps in creating machines that can understand our language. This strategy can be used in a variety of ways, one of which is to help businesses flourish. For example, if you can identify a potential customer's buy intent early on, you'll have a better chance of turning them into full-fledged clients. Machine learning and natural language processing are used in intent classification to automatically link words or sentences with a specific intent. A machine learning model, for example, can learn that words like buy and acquire are frequently related with the intent to purchase. If you're looking at client data, you might choose the tags Interested, Need Information, Unsubscribe, Wrong Person, Email Bounce, Auto reply, etc. Sequence labeling is a pattern recognition problem that entails assigning a categorical label to each element of a sequence of observed values using an algorithm (BIO tagging). So, sequence labelling is a downstream task of intent classification which we performed here by extracting the intent first and then using that intent to extract all the relevant entities from the document.

2 Research Questions

1. Performing linguistic analysis on the corpus like word frequencies, n-grams and word cloud visualization per specific intents.

2. Comparing different word embeddings: static (Skipgram, Count Vectorizer) to contextual (Sentence Bert).

3. Carrying out a downstream task like sequence labelling to identify a new entity from a set of documents using BIO tagging (Beginning Inside Outside).

3 Data Collection

We have collected the data set from Kaggle. The train, test and val files were all in json formats. There were 150 "intents" in other words classes. Some data snippet shown below:

1. (Text): "what is the projected time frame for the flight to land".

[Intent]: "flight status"

2. (Text): "what do i need to make chicken noodle casserole".

[Intent]: "ingredients list"

3. (Text): "am i talking with real person or ai".

[Intent]: "are you a bot"

The data we collected are labelled. Each of the json files contain 150 intents in equal proportions. Since there are 150 classes and the data set has 15000 rows in the train set. We performed our classification models on the full train, test and val

set as well as randomly chose 3 intents (classes) as shown above for granular analysis, classification and evaluation of the models too.

4 DATA PRE-PROCESSING

The very first step we took was to identify if the input data had any emails, URLs, mentions or hashtags in them. We used count vectorizer to create a bag of words and used regular expressions to check for the above mentioned patterns in them. Once these were identified (only 1 hashtag was identified in the train set), we moved on to the actual processing of the data. Then, punctuations were removed from the text and the text was converted to lowercase. Nltk library was used then to perform tokenization on the cleaned text and stopwords removal followed the tokenization process.

5 LINGUISTIC ANALYSIS

We performed 3 different types of linguistic analysis on the whole corpus as well as for the 3 specific intents: ”flight status”, ”ingredients list”, ”are you a bot”. This practice is also known as corpus linguistic analysis, and it transforms textual data into word frequencies, pairs of words most occurring together and patterns largely impossible to note in conventional reading. We used the aforementioned techniques in our work to identify the tacit patterns from the input corpus.

5.1 Word Frequencies

After all the pre-processing techniques, we iterated through the cleaned tokenized text and applied Counter from collections module of Python to create a word frequencies dictionary for the whole corpus. Then we filtered the word frequencies for specific intents. Some of the plottings of the word frequencies per intents can be seen below:

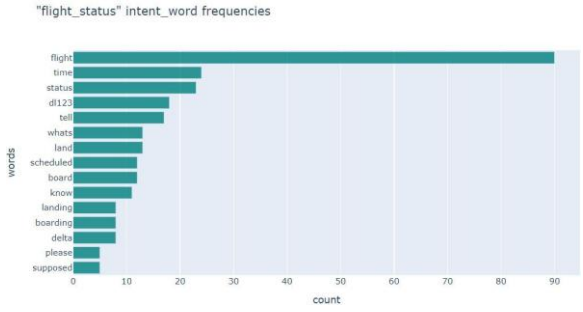

Figure 1: Flight Status - Word Frequencies Plot

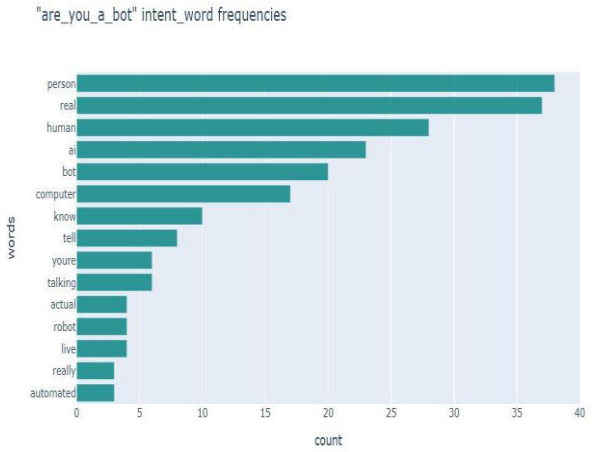

Figure 2: Are You a Bot - Word Frequencies Plot

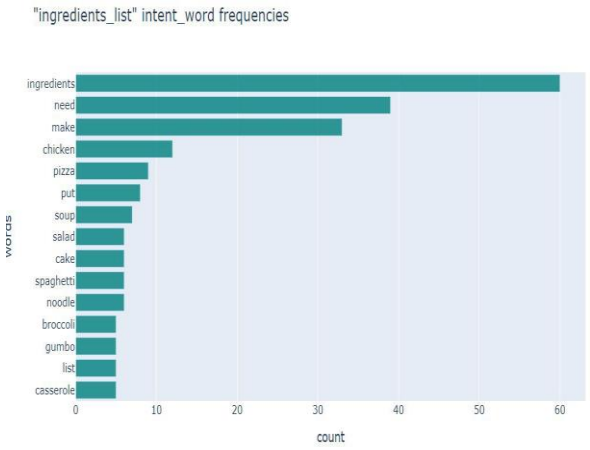

Figure 3: Ingredients List - Word Frequencies Plot

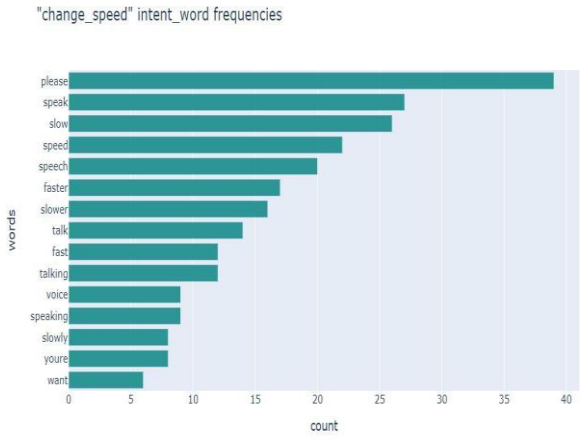

Figure 4: Change Speed - Word Frequencies Plot

5.2 N-Grams

In a document, N-grams are continuous sequences of words, symbols, or tokens. They can be defined as the adjacent sequences of items in a document in technical terms.Here, we implemented bigrams (two words), trigrams (three words) and four words for specific intents. The implementation was done with the help of Count Vectorizer where we dynamically passed the ngram range parameter. Some of the bar plots of the top 10 respective n-grams are as follows:

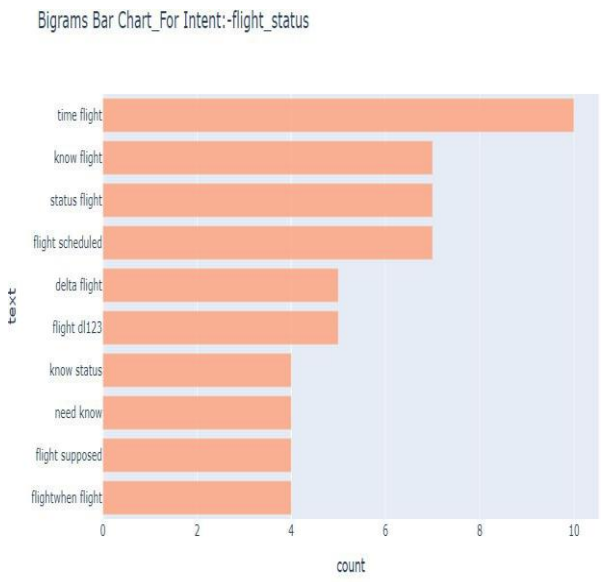

Figure 5: Flight Status - Bigrams Plot

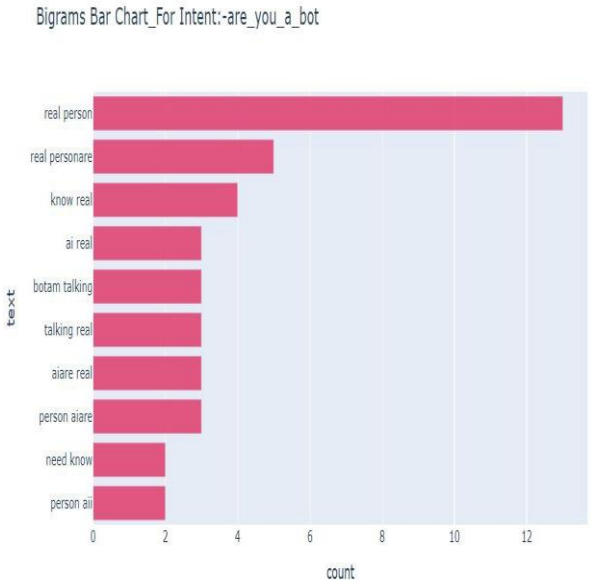

Figure 6: Are You a Bot - Bigrams Plot

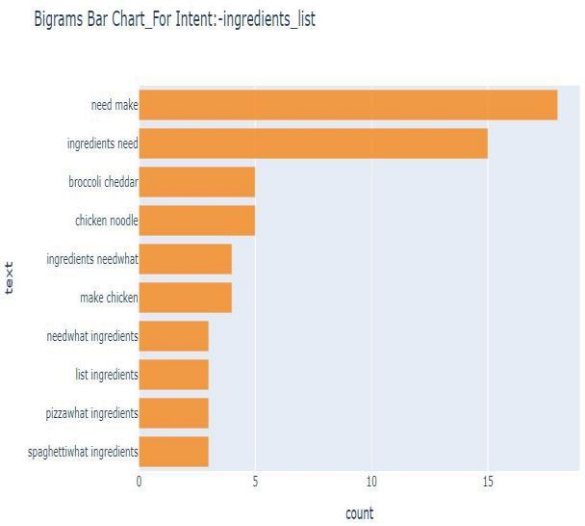

Figure 7: Ingredients List - Bigrams Plot

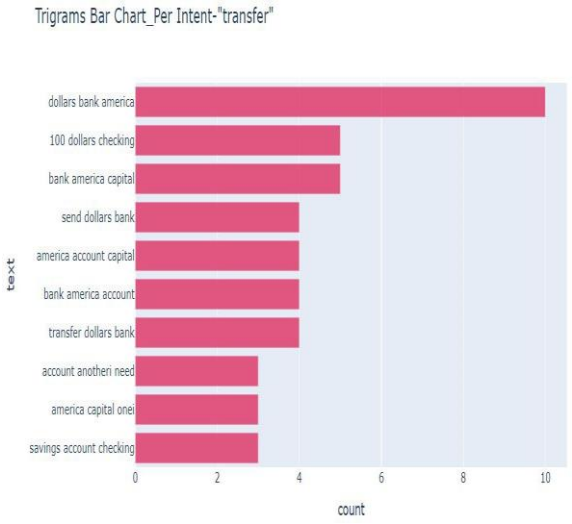

Figure 8: Change Speed - Trigrams Plot

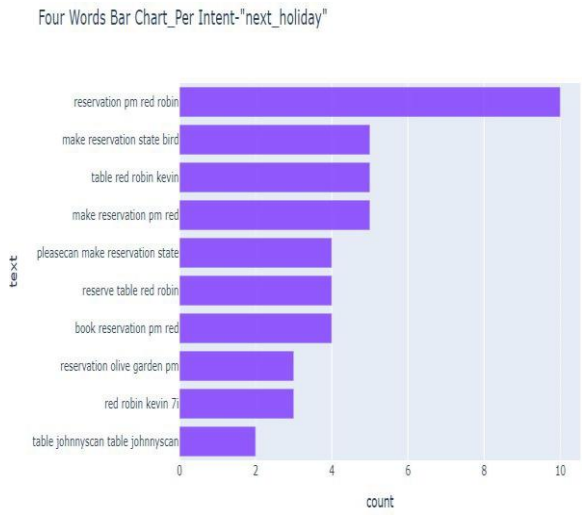

Figure 9: Change Speed - Four Words Plot

5.3 WordCloud

WordCloud is a visualization technique used in processing of textual data in Natural language processing which shows the most frequent words with bigger and bolder letters, and in different colors too. In other words, it is a simple tool to identify the focus of written material. To implement this, we used wordcloud package of python. Some of the plottings of the word clouds per intents can be seen below:

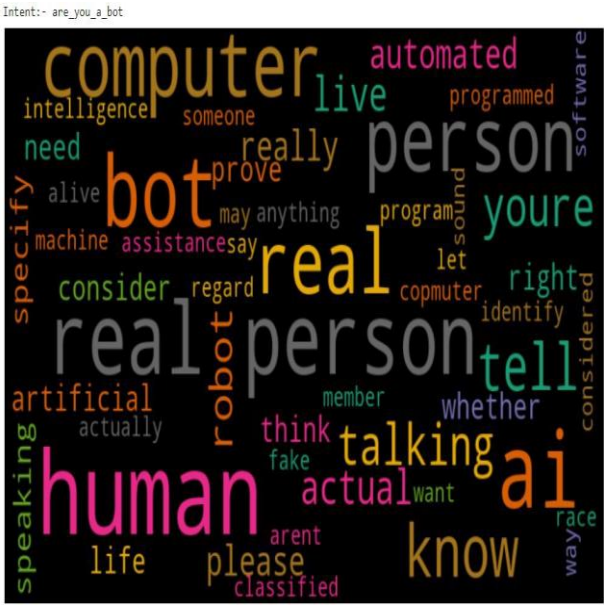

Figure 10: Are You a Bot - WordCloud

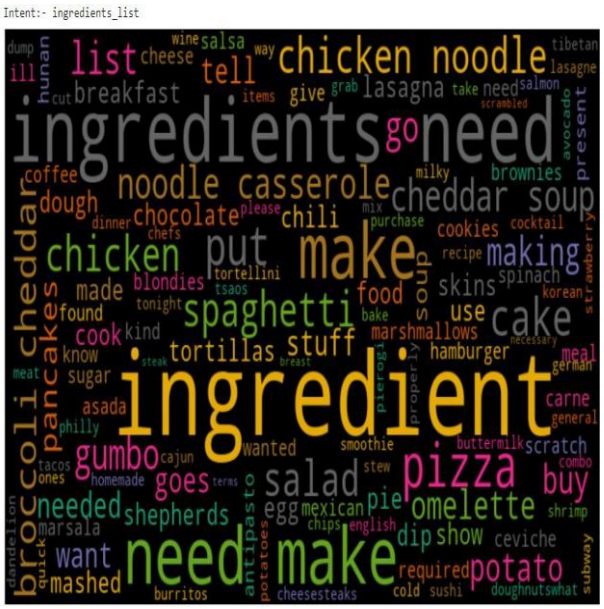

Figure 11: Ingredients List - WordCloud

Figure 12: Change Speed - WordCloud

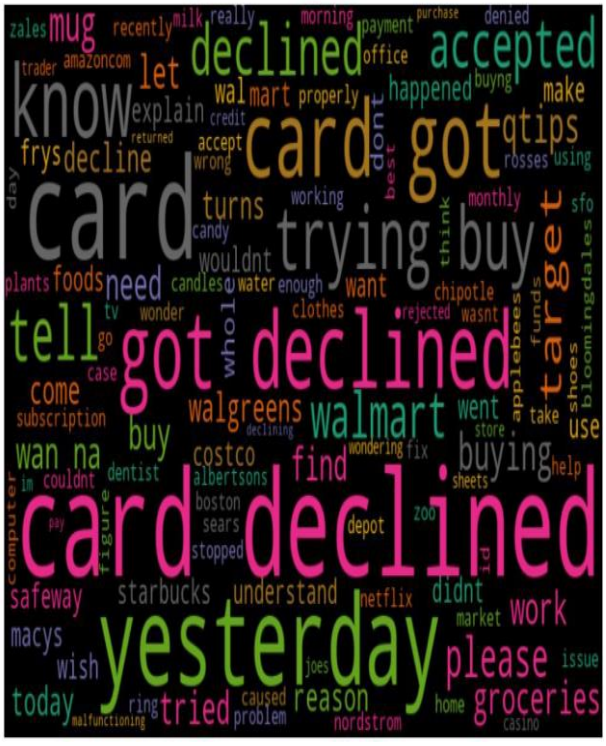

Figure 13: Change Speed - WordCloud

6 MODELLING WITH DIFFERENT WORD EMBEDDINGS

We implemented SkipGram, Count Vectorizer from static word embeddings and Sentence Bert from Contextual Embeddings for all the 150 intents (data set) and the subset of data which we discussed earlier for 3 specific intents : "flight status", "ingredients list", "are you a bot". The number of rows for these three intents data sets were 300, 60 and 90 respectively for train, val and test.

6.1 Static Word Embeddings

Each word type is mapped to a single vector by a (static) word embedding. These vectors generated by Glove, Word2Vec (word embedding models) are dense whereas vectors generated by Count Vectorizer, TFIDF (Term Frequency Inverse Document Frequency), Bag of Words are sparse matrices.

6.1.1 SkipGram

It employs a classifier to predict which words appear in the context of a target word and which words appear in the context of a target word. A dense vector representation of words is included in this classifier (embeddings). It can capture a word's context in a document, as well as its semantic and syntactic similarities to other words. Words that appear in comparable contexts will have vector representations that are highly similar. We used Skpigram from Word2vec package to create the Skipgram embeddings for the input corpus. The parameters we used were a vector size of 100, window of 5, sg = 1 and 4 workers. After the skipgram model is built on the input texts with these parameters we combined the vector representations of each word together and got a new one that represented the document as a whole. Here, we computed a weighted average (tfidf score) where each weight gave the importance of the word with respect to the corpus. Then for all the train, test and val sets we created a list of vectors and scaled them to have zero mean and unit standard deviation. Finally, these word embeddings were fed into classification models (Random Forest, Logistic Regression) for training, validation and test predictions.

6.1.2 Count Vectorizer

The scikit-learn toolkit in Python has a fantastic utility called CountVectorizer. It is used to convert a text into a vector based on the frequency (count) of each word that appears throughout the text. This is useful when dealing with a large number of such texts and converting each word into a vector (for using in further text analysis). It builds a matrix with each unique word represented by a column and each text sample from the document represented by a row. The count of the word in that particular text sample is the value of each cell. We used the parameters: 'analyzer' as word and ngram range as (2,2) in Count vectorizer. It was also done for the full dataset and the three intents data set.

6.2 Contextual Word Embeddings

Siamese BERT-Networks are a variant of the BERT network that uses Siamese and triplet networks to generate semantically meaningful sentence embeddings. Before going in details to Sentence BERT we can talk about some details mentioned in BERT's paper. The BERT framework includes two steps: pre-training and fine-tuning. The model is trained on unlabeled data across different pre-training tasks during the pre-training process, and for fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.

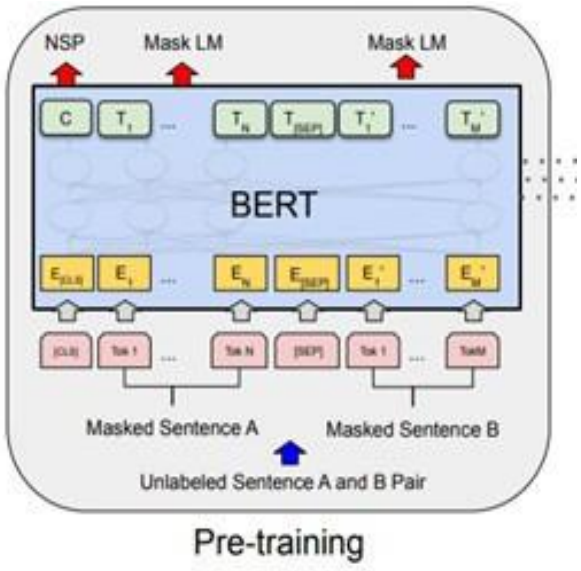

Figure 14: BERT Architecture

BERT is pre-trained on two unsupervised tasks: Masked LM and Next Sentence Prediction (NSP). To train a deep bidirectional representation, we simply mask a random percentage of the input tokens and then predict those masked tokens. As in a standard language model, the final hidden vectors

corresponding to the masked tokens are fed into an output SoftMax over the vocabulary. Language model does not capture the relationship between 2 sentences. Let's have a look at BERT input representation: The input embeddings are the sum of the token embeddings, the segmentation embeddings, and the position embeddings.

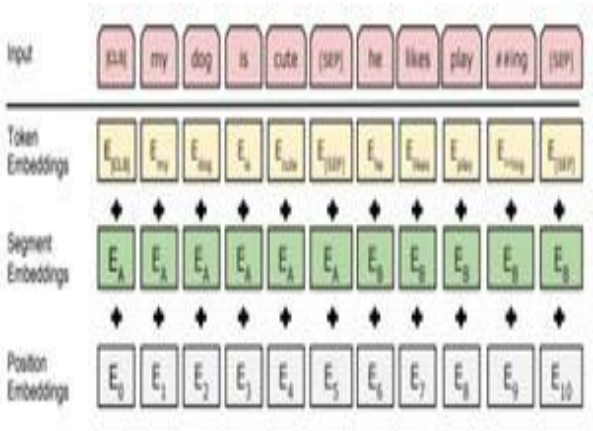

Figure 15: BERT Input Format

So, from the paper of Siamese BERT it is evident that SBERT adds a pooling operation to the output of BERT to derive a fixed sized sentence embedding. There are 3 types of pooling strategies: Using the output of the CLS-token, computing the mean of all output vectors(MEAN-strategy), and computing max-over-time of the output vectors(MAX-strategy). Then for classification task we concatenate the sentence embeddings u and v with the element-wise difference —u-v— and multiply it with the trainable weight Wt .

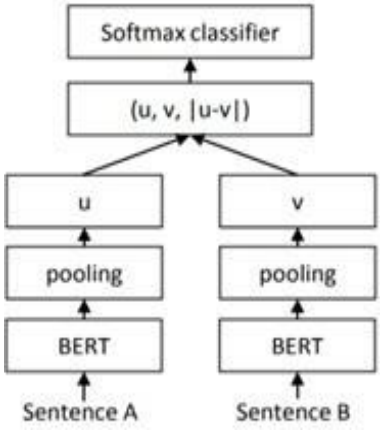

Figure 16: Sentence BERT Architecture

We gave out cleaned and tokenized sentences to SBERT model and then we used Random Forest classifier and Logistic Regression on our embeddings to find the intents for the sentence inputs.

7 SEQUENCE LABELLING

This task uses the idea of transfer learning, i.e., first pre-training a large neural network in an unsupervised way, then fine tuning that neural network on a task of interest. BERT in this case is a neural network that has been pre-trained on two tasks: masked language modeling and next sentence prediction. We will now fine-tune this network using a NER dataset generated from our intent corpus. We created a labeled dataset because fine tuning is a supervised task. At the word level, named entity recognition employs a specific annotation scheme. BIO-tagging is the annotation scheme that we used. Each tag indicates whether the corresponding word is located inside, outside, or at the start of a specific named entity. This is due to the fact that there are instances where named entities comprise of more than one word.

We are trying to find entity money from the sentences. Let's have a look at an example. If there is sentence like "1000 dollars is a lot of money", then the corresponding tags would be [B-mon, I-mon, O, O, O, O, O]. "B-mon" means the word 1000 is at the beginning of our money entity and "I-mon" means the word "dollars" is inside the entity and "O" means rest are outside. So we extracted over 140 sentences which had our custom entity from the corpus and manually tagged the documents in BIO-tag format. We created a custom data format where in addition to the encoded output from BERT with input ids and attention mask we get labels padded to MAX LENGTH. We selected 64 as max length for our embeddings and for padding we used -100. We processed the inputs in batch size of 4 to our token classification model. Then we trained our model on 128 sentences with 0.93 training accuracy based on the predicted custom entity tags. The model gives output in probabilistic distribution of labels for all the tokens and then we take argmax to find the predicted label for each token. Then we got around 0.4 validation accuracy on our validation data consisting of 13 sentences and processed in batch size of 2. We used 2 pretrained BERT models such BertForTokenClassification and DistilBertForTokenClassification and chose the one which gave us better predictions. The example of the output on unseen text can be

seen in figure 20.

8 RESULTS

For 150 Intents	ROCAUC Score	F1 Score
Random Forest (Skipgram)	0.964	0.662
Logistic Regression (Skipgram)	0.985	0.736
Random Forest (Count Vectorizer)	0.968	0.746
Logistic Regression (Count Vectorizer)	**0.99**	**0.78**

Figure 17: Random Forest - Logistic Regression - 150 Intents - Scores

The ROCAUC scores for each model with SG and CV are almost the same. But the F1 score varies and Logistic Regression with CV has the best scores for ROCAUC and F1 for 150 intents.

For 3 Intents	ROCAUC Score	F1 Score
Random Forest (Skipgram)	**1.0**	**1.0**
Logistic Regression (Skipgram)	1.0	0.949
Random Forest (Count Vectorizer)	0.999	0.950
Logistic Regression (Count Vectorizer)	**1.0**	**1.0**

Figure 18: Random Forest - Logistic Regression - 3 Intents - Scores

The ROCAUC scores for each model with SG and CV are almost the same. There is also very little variation in F1 score; Logistic Regression with CV and Random Forest with SG has the best scores for ROCAUC and F1 for 3 intents.

For 150 Intents	ROCAUC Score	F1 Score
Random Forest (Sentence Bert)	0.994	0.88
Logistic Regression (Sentence Bert)	**0.998**	**0.947**
For 3 Intents		
Random Forest (Sentence Bert)	1.0	1.0
Logistic Regression (Sentence Bert)	1.0	1.0

Figure 19: RF-LR SENTENCE BERT 150 and 3 Intents - Scores

Logistic Regression also performed better with Sentence Bert embeddings with 150 intents but both Random Forest and Logistic Regression with SBERT have very good scores for 3 intents.

```
1000 dollars is lot of money
('1000', 'I-mon')
('dollars', 'I-mon')
('is', 'O')
('lot', 'O')
('of', 'O')
('money', 'O')
```

Figure 20: Sequence Labelling Output

The Sequence Labelling model gives the first word as I-mon instead of B-mon because it has been trained on less data.But it predicted other tokens correctly as we required.

9 CONCLUSION

After all the implementations and careful analysis, it was observed that classification models built using the contextual word embeddings have better results than the ones built using static word embeddings. It was because Sentence Bert captures the contextual meaning of the sentences . It was also observed that Count Vectorizer with Logistic Regression (a fairly simple model) performed

better than SkipGram for both 150 intents and 3 intents. There are some challenges with this Intent Classification, one of which is it would struggle with out of scope intent classification. It could be solved by using embeddings of the word graph space of the classes via which the classification algorithm would be able to capture the inter class similarities and improve the detection of out of scope intents. The BIO tagging prediction accuracy can be improved by adding a CRF layer on top of the BERT output to get more accurate predictions.

10 REFERENCES

Haihong E and Peiqing Niu and Zhongfu Chen and Meina Song.*A Novel Bi-directional Interrelated Model for Joint Intent Detection and Slot Filling.* 2019. 1907.00390(1).

Zhiheng Huang and Wei Xu and Kai Yu. *Bidirectional LSTM-CRF Models for Sequence Tagging.* 1508.01991(1).

Zhiyong He and Zanbo Wang and Wei Wei and Shanshan Feng and Zianling Mao and Sheng Jiang. *A Survey on Recent Advances in Sequence Labelling from Deep Learning Models.* 2020.

Jacob Devlin and Ming-Wei Chang and Kenton Lee and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. Google AI Language.

Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* 2019. Department of Computer Science, Technische Universitat Darmstadt.

https://huggingface.co/transformers/v4.2.2/custom_datasets.html

https://huggingface.co/transformers/v4.2.2/custom_datasets.html#ft-native

https://www.kaggle.com/datasets/stefanlarson/outofscope-intent-classification-dataset

https://www.ahmedbesbes.com/blog/sentiment-analysis-with-keras-and-word-2-vec