

**CSE523 Machine Learning**

**Weekly Report 4**

**Group Name: Precision Précis**

| **NAME** | **ENROLLMENT NUMBER** |
| --- | --- |
| Twinkle Popat | AU2040216 |
| Kathan Bhavsar | AU2040170 |
| Rushali Moteria | AU2040210 |
| Neel Buddhdev | AU2040176 |

1. **Tasks performed in the week.**

The report will discuss the implementation of Sentence Level Annotation using Named Entity Recognition, followed by the implementation of the SVM model.

1. **Sentence Level Annotation**

We are performing sentence-level annotation using Named Entity Recognition.

Firstly, We are obtaining a reference summary for the text, tokenizing the summary into words, and using part-of-speech (POS) tagging to tag each word with its grammatical role in the sentence. The code then performs chunking using the Named Entity Chunker in NLTK, which identifies chunks of words that represent named entities such as organizations, persons, locations, dates, and times. The code then adds these entities to a list called entity[i] for each reference summary.

| entity\_types = ['ORGANIZATION', 'PERSON', 'LOCATION', 'DATE', 'TIME']  summary = refehighlights[i]   words = nltk.word\_tokenize(summary)  tagged = nltk.pos\_tag(words)  entities = nltk.chunk.ne\_chunk(tagged)  for subtree in entities.subtrees():  if subtree.label() in entity\_types:  entity[i].append(subtree.label()) |
| --- |

Then we create a binary dictionary for each sentence in the text and initialize it to 0. After that, we loop over each sentence, tokenize it into words, perform POS tagging on the words, and then apply NER using the same Named Entity Chunker as in the previous step.

| *# NER for text*  text = article[i]  binDict = {}  sentences = nltk.sent\_tokenize(text)  for sent in sentences:  binDict[sent] = 0   words = [nltk.word\_tokenize(sent) for sent in sentences]   *# Perform part-of-speech (POS) tagging on the words*  pos\_tags = [nltk.pos\_tag(sent) for sent in words]   *# Perform named entity recognition (NER) on the POS tagged words*  ne\_tags = [nltk.ne\_chunk(tagged) for tagged in pos\_tags]  entity\_types = entity[i] |
| --- |

Finally, we check whether the identified entity type is in the entity\_types list for the given reference summary. If it is present, we set the value of the corresponding sentence in the binary Dict defined above to 1 else, it will remain 0.

| for k,ne in enumerate(ne\_tags):  for subtree in ne.subtrees():  if subtree.label() in entity\_types:  binDict[sentences[k]] = 1 |
| --- |

1. **Count vectorizer and SVM**

We use the count vectorization method for feature extraction. It involves the initial preprocessing, like the removal of stop words and sentence tokenization. In this method, a dictionary of unique words from the text is formed and the frequency of each word in the sentence is calculated. It forms a vector where the columns represent the number of unique words and the rows represent different sentences. We have summed up the frequencies of all words. The following is the code for the count vectorizer.

| sentences = nltk.sent\_tokenize(article[i])   for sent in sentences:  count\_dict = {}  sum1 = 0  for word in sent:  if((word not in stopwords) and (word not in punctuation)):  if(word in count\_dict):  count\_dict[word] += 1  else:  count\_dict[word] = 1   sum1 = sum(count\_dict.values())  cv\_arr[i].append(sum1) |
| --- |

The count vectorizer array and the binary array formed earlier are fed to the SVM model to train it. Initially, for training, we have taken 500 articles. For the implementation of the SVM model, we have used the linear kernel. The Y-train input of the SVM is the binary array while the X-train input is the count vectorizer. Then another article was used for testing the model.

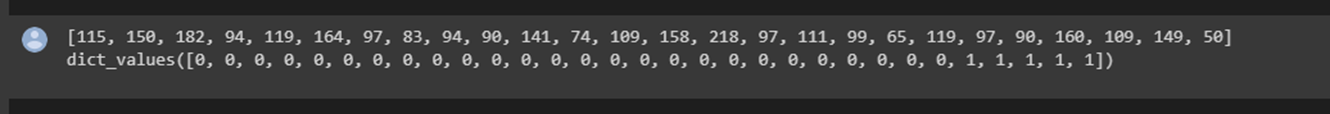
Before implementing the count vectorizer we converted the two arrays to fixed-sized matrices which can be fed into the SVM model. The following code snippet shows the code implementation.

The following is the code snippet for SVM.

| import numpy as np from sklearn import svm  svm\_model = svm.SVC(kernel='linear') svm\_model.fit(x\_train, y\_train) |
| --- |

1. **Outcomes of the tasks performed.**

The following are the outcomes for the binary array formed and the count vectorizer array formed. As stated above, in the count vectorizer, we have summed up the frequencies of the individual words.



The first array is the count vectorizer array and the second array is the binary array.

The following are the results of the SVM model. In this array 1 represents that the sentence is important and 0 represents that the sentence is not important.



1. **Tasks to be performed in the upcoming week.**

In the upcoming week, we will solve perform sentence-level annotation using topic modeling and sentence similarity. In addition, the size of the array generated by the SVM model is not the same as the size of the binary array, so we will also resolve the issue in the upcoming week.