NLP Assignment-1

Name: Akshara B

Roll No: 211AI012

1. HMM POS tagging using NLTK toolkit

The dataset I used is “hindi.pos” from Indian corpus present is nltk library. It is available for open-source use on Kaggle.
https://www.kaggle.com/datasets/nltkdata/indian-corpus

Then I transformed the dataset into a list containing list of sentences with word,tag as a tuple.

```
taggedSet = "hindi.pos"
wordSet = indian.sents(taggedSet)
count = 0
for sen in wordSet:
    count = count + 1
    sen = "".join(
        [
            " " + i if not i.startswith("'") and i not in string.punctuation else i
            for i in sen
        ]
    ).strip()
    print(count, sen, "sentences")
print("Total sentences in the tagged file are", count)

data = indian.tagged_sents(taggedSet)
print("data is : ",end=" ")
print(data)
```

```
435 गुजरात प्रदश कांग्रस क अध्यक्ष साडा पटल का गुरुवार प्रात: ४ बजकर ५० मिनट पर निधन हा गया । sentence
436 वे ७० वर्ष के थे । पटेल के परिवार में उनकी पत्नी, पुत्र और २ पुत्रियां हैं । sentences
437 गुजरात कांग्रेस के प्रवक्ता जयंती लाल परमार के मुताबिक पटेल को दिल का दौरा पड़ने के बाद सोमवार को स्थानी
438 पटेल पूर्व में सूरत से लोकसभा के लिए चुने जा चुके थे तथा ३ बार विधानसभा के सदस्य भी रहे । sentences
439 वे १९९० में विधानसभा में प्रतिपक्ष के नेता भी थे । sentences
```

```
539 न्यू जीलैंड की तरफ से टफी और फ्रेंक्लीन ने तीन-तीन और हैरिस ने दो विकेट लिये । sentences
540 अनवर को विसेट ने रन आउट किया । sentences
Total sentences in the tagged file are 540
data is :  [[('पूर्ण', 'JJ'), ('प्रतिबंध', 'NN'), ('हटाओ', 'VFM'), (':', 'SYM'), ('इराक', 'NNP')], [('संयुक्त', 'NNC'), ('राष्ट्र', 'NN'), ('।', 'SYM')], ...]
```

+ Code + Text

Now this list is shuffled and separated into with 20% test and 80% train datasets

```
from sklearn.model_selection import train_test_split
tagged_sentences = data
print(tagged_sentences)
traindataset , testdataset = train_test_split(tagged_sentences, shuffle=True, test_size=0.2) #Splitting test and train dataset

HmmModel = nltk.HiddenMarkovModelTagger.train(traindataset)

correct_labels = [tag for sentences in testdataset for word, tag in sentences]

predicted_labels=[]
for sentences in testdataset:
    predicted_labels += [tag for _, tag in HmmModel.tag([word for word, _ in sentences])]

from sklearn.metrics import classification_report
print (classification_report(correct_labels, predicted_labels))
```

```
[[('पूर्ण', 'JJ'), ('प्रतिबंध', 'NN'), ('हटाओ', 'VFM'), (':', 'SYM'), ('इराक', 'NNP')], [('संयुक्त', 'NNC'), ('राष्ट्र', 'NN'), ('।', 'SYM')], ...]
              precision    recall  f1-score   support
```

Then we apply train the HmmModel = nltk.HiddenMarkovModelTagger.train(traindataset) to train HMM model on train set and we apply model on test dataset to get predicted labels. Then we print a classification report with label specific accuracies and overall accuracy of 82%

```
[5]
[[('पूर्ण', 'JJ'), ('प्रतिबंध', 'NN'), ('हटाओ', 'VFM'), (':', 'SYM'), ('इराक', 'NNP')], [('संयुक्त', 'NNC'), ('राष्ट्र', 'NN'), ('।', 'SYM')], ...]
              precision    recall  f1-score   support

                   0.00      0.00      0.00         0
          CC       0.98      0.97      0.98        66
        INTF       0.00      0.00      0.00         3
          JJ       0.85      0.51      0.64        67
         JVB       0.52      0.70      0.60        20
         NEG       0.59      1.00      0.74        10
        NLOC       1.00      0.50      0.67         6
          NN       0.87      0.78      0.82       355
         NNC       0.64      0.59      0.61        58
         NNP       0.85      0.78      0.82       171
        NNPC       0.71      0.53      0.61        60
         NVB       0.45      0.67      0.54        36
        PREP       0.96      0.97      0.96       408
         PRP       0.84      0.82      0.83        72
        PUNC       0.87      0.89      0.88       117
          QF       0.60      0.50      0.55         6
       QFNUM       0.84      0.84      0.84       108
          QW       0.00      0.00      0.00         2
          RB       0.73      0.69      0.71        16
          RP       0.95      0.95      0.95        20
         SYM       0.90      0.53      0.67        17
        VAUX       0.81      0.93      0.87        95
         VFM       0.83      0.87      0.85       148
         VJJ       0.67      0.29      0.40         7
         VNN       0.55      0.60      0.58        35
         VRB       0.73      0.85      0.79        13

    accuracy                           0.82      1916
   macro avg       0.68      0.64      0.65      1916
weighted avg       0.85      0.82      0.83      1916
```

Here we see that our dataset has very high accuracy for labels CC,VAUX,VFM,PUNC,PREP,NN,NNP.

Since the NLTK HMM model is generalized for any language and its pretrained I think its easier for the model to predict the labels such as CC, PUNC and NN which have same semantic relationship with the rest of the sentence in any language, Hence the high accuracy

The other tags such as JVB,VJJ, QF, QW, have very few instances in the dataset hence model couldn't learn the relationship with these tags hence slightly lower accuracies.

2. POS tagging using SVM classifier

I used the same dataset as the previous one and the preprocessing steps are same until I split it into test and train datasets (20-80).

```
import nltk
import random
from nltk.corpus import indian
from sklearn.svm import SVC
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split


tagged_sents = indian.tagged_sents('hindi.pos')


def extract_features(sentence, index):
    word = sentence[index][0]
    features = {
        'word': word,
        'prev_word': sentence[index - 1][0] if index > 0 else '',
        'next_word': sentence[index + 1][0] if index < len(sentence) - 1 else ''
    }
    return features


data = []
for sent in tagged_sents:
    for i in range(len(sent)):
        features = extract_features(sent, i)
        data.append((features, sent[i][1]))


random.shuffle(data)
train_size = int(0.8 * len(data))
train_data, test_data = data[:train_size], data[train_size:]
```

Then here I applied one hot encoding for converting categorical features to vectors. Initially I used vectorizer as embedder but it resulted in lower

accuracy(53%). so upon research I understood that dense embedder like one hot encoding is more suitable. This embedder is applied on X_test and X_train. (Note: here tag value is used as label and tokens act as features).

```
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
X_train = encoder.fit_transform([list(features.values()) for features, _ in train_data])
X_test = encoder.transform([list(features.values()) for features, _ in test_data])


y_train = [tag for _, tag in train_data]
y_test = [tag for _, tag in test_data]
```

Now SVM classifier model with linear kernel is trained with X_train and Y_train. We calculate the predicted labels for X_test and note the accuracy (85%)

```
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2%}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` wa
  warnings.warn(
Accuracy: 84.70%
```

Here also we can see that tags with less data (no. of available tags of specific tag type given as support in classification report) have lesser accuracy compared to tags with more data.

	precision	recall	f1-score	support
	0.00	0.00	0.00	7
CC	0.98	0.95	0.97	66
INTF	1.00	0.44	0.62	9
JJ	0.64	0.65	0.64	72
JVB	0.75	0.55	0.63	33
NEG	0.90	0.82	0.86	11
NLOC	1.00	0.62	0.77	8
NN	0.72	0.91	0.80	360
NNC	0.61	0.49	0.54	51
NNP	0.89	0.82	0.85	152
NNPC	0.73	0.52	0.60	62
NVB	0.61	0.51	0.56	39
PREP	0.98	0.99	0.99	380
PRP	0.99	0.86	0.92	80
PUNC	0.94	0.99	0.96	110
QF	1.00	0.50	0.67	4
QFNUM	0.92	0.85	0.88	80
QW	0.00	0.00	0.00	2
RB	0.82	0.82	0.82	11
RP	1.00	1.00	1.00	22
SYM	1.00	0.68	0.81	19
VAUX	0.84	0.91	0.88	101
VFM	0.84	0.85	0.84	150
VJJ	1.00	0.45	0.62	11
VNN	1.00	0.52	0.68	29
VRB	1.00	0.85	0.92	13
accuracy			0.85	1882
macro avg	0.81	0.68	0.72	1882
weighted avg	0.85	0.85	0.84	1882

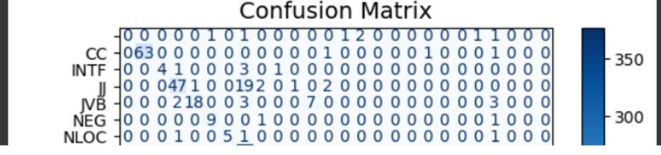
In the end I printed a confusion matrix and it gives some interesting insights.

```
import itertools
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)

# Create a ConfusionMatrixDisplay with adjusted font size
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot(cmap=plt.cm.Blues, values_format="d")  # You can specify cmap and values_format
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.xlabel('Predicted Labels', fontsize=12)  # Set the font size for x-axis label
plt.ylabel('True Labels', fontsize=12)  # Set the font size for y-axis label
plt.title('Confusion Matrix', fontsize=14)  # Set the font size for the title

plt.show()
```



We can see Almost(19/71) JJ (adjectives) are wrongly classified as nouns. This is expected because POS tags of few adjectives are also nouns depending on the semantics of sentence ex: word “sundar”. Especially since its in Hindi its tough to detect the proper semantics without certain set of rules specific to the language. But the model tries its best. Similar case can be seen with VNN, where few VNN labels are classified as NN. Overall, The model has a scope of improvement.