

## Practical No. 05: Implement HMM for POS tagging. Build a Chunker

Date: /1 /2024

### Program:

```
import nltk

from nltk.corpus import treebank

from nltk.tag import hmm

nltk.download('treebank')

train_data = treebank.tagged_sents()

trainer = hmm.HiddenMarkovModelTrainer()

hmm_tagger = trainer.train(train_data)

sentence = "The quick brown fox jumps over the lazy dog".split()

pos_tags = hmm_tagger.tag(sentence)

print("POS Tags:", pos_tags)
```

### Output:

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data] Unzipping corpora/treebank.zip.
/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:333: RuntimeWarning: overflow encountered in cast
  X[i, j] = self._transitions[si].logprob(self._states[j])
/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:335: RuntimeWarning: overflow encountered in cast
  O[i, k] = self._output_logprob(si, self._symbols[k])
/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:331: RuntimeWarning: overflow encountered in cast
  P[i] = self._priors.logprob(si)
POS Tags: [('The', 'DT'), ('quick', 'JJ'), ('brown', 'NNP'), ('fox', 'NNP'), ('jumps', 'NNP'), ('over', 'NNP'), ('the', 'NNP'), ('lazy', 'NNP'), ('dog', 'NNP')]
/usr/local/lib/python3.10/dist-packages/nltk/tag/hmm.py:363: RuntimeWarning: overflow encountered in cast
  O[i, k] = self._output_logprob(si, self._symbols[k])
```

## Practical No. 10: Implement RNN for sequence labelling and show some output

Date: /1 /2024

### Program:

```
import numpy as np

# Sample data (replace with your actual sequence labeling data)
# Sequences: [['word1', 'word2', ...], ...]
# Labels: [['label1', 'label2', ...], ...]

sequences = [['The', 'quick', 'brown', 'fox'], ['jumps', 'over', 'the', 'lazy', 'dog']]
labels = [['DET', 'ADJ', 'ADJ', 'NOUN'], ['VERB', 'ADP', 'DET', 'ADJ', 'NOUN']]

# Create vocabulary and label dictionaries
word_to_index = {}
label_to_index = {}
index_to_label = {}

for seq, lab in zip(sequences, labels):
    for word in seq:
        if word not in word_to_index:
            word_to_index[word] = len(word_to_index)

    for label in lab:
        if label not in label_to_index:
            label_to_index[label] = len(label_to_index)
            index_to_label[len(label_to_index)-1] = label

# Convert data to numerical representations
X = [[word_to_index[word] for word in seq] for seq in sequences]
y = [[label_to_index[label] for label in lab] for lab in labels]

# Pad sequences to ensure uniform length
max_len = max(len(seq) for seq in X)
X = [seq + [0] * (max_len - len(seq)) for seq in X]
y = [lab + [0] * (max_len - len(lab)) for lab in y]

import numpy as np
```

```

# Simulate RNN calculations (replace with actual RNN implementation)

# Example: a basic RNN using NumPy

def simple_rnn(input_seq, weights, bias):

    hidden_state_size = weights[1].shape[0] # Get the size of the hidden state from the recurrent weight matrix

    hidden_state = np.zeros(hidden_state_size) # Initialize hidden state with the correct size


    outputs = []

    for word_index in input_seq:

        input_vector = np.zeros(len(word_to_index))

        input_vector[word_index] = 1 #one-hot encoding

        hidden_state = np.tanh(np.dot(input_vector, weights[0]) + np.dot(hidden_state, weights[1]) + bias[0])


        # Predict label using the hidden state (replace with more appropriate prediction method)

        output_probs = np.dot(hidden_state, weights[2]) + bias[1]

        predicted_label_index = np.argmax(output_probs)

        outputs.append(predicted_label_index)

    return outputs


# Randomly initialize weights and biases (replace with training)

hidden_state_size = 10 # Define the desired size of the hidden state

weights = [np.random.rand(len(word_to_index), hidden_state_size), np.random.rand(hidden_state_size,
hidden_state_size), np.random.rand(hidden_state_size, len(label_to_index))] # Ensure weights are compatible with
the hidden state size

bias = [np.random.rand(hidden_state_size), np.random.rand(len(label_to_index))]


# Example usage: predict labels for a sequence

predicted_labels = simple_rnn(X[0], weights, bias)

print(predicted_labels) # output as indexes


#convert prediction to label

print([index_to_label[pred] for pred in predicted_labels])

```

### Output:

```
[1, 1, 1, 1, 1]
```

```
['ADJ', 'ADJ', 'ADJ', 'ADJ', 'ADJ']
```

## Practical No. 09: Implement a movie reviews sentiment classifier

Date: /1 /2024

### Program:

# prompt: Implement a movie reviews sentiment classifier

```
import nltk

import random

import numpy as np

import pandas as pd

from nltk.corpus import movie_reviews

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Download necessary NLTK data (if not already downloaded)

nltk.download('movie_reviews')

nltk.download('punkt')


# Prepare the data

documents = [(list(movie_reviews.words(fileid)), category)

              for category in movie_reviews.categories()

              for fileid in movie_reviews.fileids(category)]

random.shuffle(documents)


all_words = []

for w in movie_reviews.words():

    all_words.append(w.lower())

all_words = nltk.FreqDist(all_words)

word_features = list(all_words.keys())[:3000]
```

```

def find_features(document):
    words = set(document)

    features = {}

    for w in word_features:
        features[w] = (w in words)

    return features

featuresets = [(find_features(rev), category) for (rev, category) in documents]

training_set = featuresets[:1900]

testing_set = featuresets[1900:]

# Train the classifier

classifier = nltk.NaiveBayesClassifier.train(training_set)

print("Classifier accuracy percent:", (nltk.classify.accuracy(classifier, testing_set))*100)

classifier.show_most_informative_features(15)

# Example usage

example_text = "This movie was absolutely terrible. The acting was awful and the plot was confusing."

example_features = find_features(word_tokenize(example_text.lower()))

prediction = classifier.classify(example_features)

print(f"Prediction for '{example_text}': {prediction}")

```

## Output:

```

[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data] Unzipping corpora/movie_reviews.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
Classifier accuracy percent: 82.0
Most Informative Features
      sucks = True          neg : pos   =    10.4 : 1.0
      frances = True        pos : neg   =     8.9 : 1.0
  unimaginative = True     neg : pos   =     8.5 : 1.0
      annual = True        pos : neg   =     8.2 : 1.0
      stellan = True       pos : neg   =     8.2 : 1.0
      groan = True         neg : pos   =     7.8 : 1.0
      sexist = True        neg : pos   =     7.8 : 1.0
  silverstone = True       neg : pos   =     7.8 : 1.0
  schumacher = True        neg : pos   =     7.5 : 1.0
      idiotic = True       neg : pos   =     7.1 : 1.0
      shoddy = True        neg : pos   =     7.1 : 1.0
  atrocious = True         neg : pos   =     6.7 : 1.0
      regard = True        pos : neg   =     6.5 : 1.0
      turkey = True        neg : pos   =     6.4 : 1.0
      kombat = True        neg : pos   =     6.4 : 1.0
Prediction for 'This movie was absolutely terrible. The acting was awful and the plot was confusing.': neg

```

## Practical No. 02: Find the synonym /antonym of a word using WordNet.

Date: 03/09/2024

### Program:

```
import nltk

from nltk.corpus import wordnet

def find_synonyms_antonyms(word):

    synonyms = []

    antonyms = []

    # Retrieve all synsets for the word

    for syn in wordnet.synsets(word):

        for lemma in syn.lemmas():

            synonyms.append(lemma.name()) # Add the synonym

            if lemma.antonyms(): # Check if antonyms exist

                antonyms.append(lemma.antonyms()[0].name()) # Add the antonym

    print(f"Synonyms of {word}: {set(synonyms)}")

    print(f"Antonyms of {word}: {set(antonyms)}")

# Example usage

# find_synonyms_antonyms("Bad")

text = str(input("Enter a word to find its synonyms and antonyms: "))

find_synonyms_antonyms(text)
```

### Output:

Enter a word to find its synonyms and antonyms: good

Synonyms of good: {'unspoiled', 'adept', 'goodness', 'commodity', 'respectable', 'upright', 'effective', 'ripe', 'honest', 'near', 'practiced', 'undecomposed', 'unspoil', 'good', 'soundly', 'thoroughly', 'proficient', 'just', 'safe', 'beneficial', 'honorable', 'dear', 'secure', 'dependable', 'salutary', 'trade\_good', 'right', 'well', 'in\_effect', 'expert', 'full', 'serious', 'skilful', 'estimable', 'in\_force', 'sound', 'skillful'}

Antonyms of good: {'evilness', 'evil', 'bad', 'ill', 'badness'}

## Practical No. 07: Implement Semantic Role Labeling (SRL) to Identify Named Entities

Date: /1 /2024

### Program:

```
import nltk

# Download necessary NLTK data (if not already downloaded)
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
nltk.download('conll2000')

from nltk.chunk import conlltags2tree, tree2conlltags
from pprint import pprint

def named_entity_recognizer(text):
    # Tokenize the text
    tokens = nltk.word_tokenize(text)

    # Part-of-speech tagging
    pos_tags = nltk.pos_tag(tokens)

    # Named Entity Recognition using ne_chunk
    named_entities = nltk.ne_chunk(pos_tags, binary=True) # Use binary=True for simpler output
    iob_tagged = tree2conlltags(named_entities)
    pprint(iob_tagged)

    # Print the results (you can modify this to return the results in a different format)
    #print(named_entities)

# Example usage
text = "Barack Obama was born in Honolulu, Hawaii. He studied at Columbia University."
named_entity_recognizer(text)
```

## Output:

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]   /root/nltk_data...
[nltk_data] Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data] Package words is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]   /root/nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data]   date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package conll2000 to /root/nltk_data...
[nltk_data] Package conll2000 is already up-to-date!
[('Barack', 'NNP', 'B-NE'),
 ('Obama', 'NNP', 'I-NE'),
 ('was', 'VBD', 'O'),
 ('born', 'VBN', 'O'),
 ('in', 'IN', 'O'),
 ('Honolulu', 'NNP', 'B-NE'),
 (',', 'O'),
 ('Hawaii', 'NNP', 'B-NE'),
 (',', 'O'),
 ('He', 'PRP', 'O'),
 ('studied', 'VBD', 'O'),
 ('at', 'IN', 'O'),
 ('Columbia', 'NNP', 'B-NE'),
 ('University', 'NNP', 'I-NE'),
 (',', 'O')]
```



**Practical No. 03: Demonstrate a bigram / trigram language model. Generate regular expression for a given text.**

**Date: 10/09/2024**

**Program:**

```
import nltk

nltk.download('punkt') # nltk.download('punkt_tab')

from nltk import ngrams

from nltk.tokenize import word_tokenize

sentence = "N-grams enhance language processing tasks."

tokens = word_tokenize(sentence)

bigrams = list(ngrams(tokens, 2))

trigrams = list(ngrams(tokens, 3))

print("Bigrams: ",bigrams)

print("Trigrams: ",trigrams)
```

**Output:**

```
Bigrams: [('N-grams', 'enhance'), ('enhance', 'language'), ('language', 'processing'), ('processing', 'tasks'), ('tasks', '.')]

Trigrams: [('N-grams', 'enhance', 'language'), ('enhance', 'language', 'processing'), ('language', 'processing', 'tasks'), ('processing', 'tasks', '.')]

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data] Package punkt is already up-to-date!
```

**Practical No. 04: Perform Lemmatization and Stemming. Identify parts-of Speech using Penn Treebank tag set.**

**Date: 21/10/2024**

**Program:**

```
import nltk

nltk.download('punkt')

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk import pos_tag, word_tokenize

# Initialize stemmer and lemmatizer

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

# Sample sentence

sentence = "The cats are running quickly."

tokens = word_tokenize(sentence)

# Perform stemming

stemmed_words = [stemmer.stem(word) for word in tokens]

print("Stemmed words:", stemmed_words)

# Perform lemmatization

lemmatized_words = [lemmatizer.lemmatize(word, pos='v') for word in tokens]

print("Lemmatized words:", lemmatized_words)
```

**Output:**

```
Stemmed words: ['the', 'cat', 'are', 'run', 'quickli', '.']

Lemmatized words: ['The', 'cat', 'be', 'run', 'quickly', '.']

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data] Package punkt is already up-to-date!
```

**Program:**

```
import nltk

# Download the specific resource 'averaged_perceptron_tagger_eng'
```

```
nltk.download('averaged_perceptron_tagger_eng')
```

```
pos_tags = pos_tag(tokens)
```

```
print("POS tags:", pos_tags)
```

### Output:

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
```

```
[nltk_data] /root/nltk_data...
```

```
[nltk_data] Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

```
POS tags: [('The', 'DT'), ('cats', 'NNS'), ('are', 'VBP'), ('running', 'VBG'), ('quickly', 'RB'), ('.', '.')]
```

### Program:

```
import nltk
```

```
from nltk.stem import WordNetLemmatizer, PorterStemmer
```

```
from nltk import pos_tag, word_tokenize
```

```
from nltk.corpus import wordnet
```

```
nlTK.download('wordnet')
```

```
nlTK.download('averaged_perceptron_tagger')
```

```
nlTK.download('punkt')
```

### Output:

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
[nltk_data] Package wordnet is already up-to-date!
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
```

```
[nltk_data] /root/nltk_data...
```

```
[nltk_data] Package averaged_perceptron_tagger is already up-to-
```

```
[nltk_data] date!
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
```

```
[nltk_data] Package punkt is already up-to-date!
```

```
True
```

### Program:

```
text = "Perform Lemmatization and Stemming. Identify parts-of Speech using Penn Treebank tag set."
```

```
tokens = word_tokenize(text.lower())
```

```
lemmatizer = WordNetLemmatizer()
```

```
stemmer = PorterStemmer()
```

```
lemmatized_words = [lemmatizer.lemmatize(token) for token in tokens]
```

```
print("Lemmatized words:", lemmatized_words)
```

### **Output:**

Lemmatized words: ['perform', 'lemmatization', 'and', 'stemming', '.', 'identify', 'parts-of', 'speech', 'using', 'penn', 'treebank', 'tag', 'set', '.']

### **Program:**

```
stemmed_words = [stemmer.stem(token) for token in tokens]
```

```
print("Stemmed words:", stemmed_words)
```

### **Output:**

Stemmed words: ['perform', 'lemmat', 'and', 'stem', '.', 'identifi', 'parts-of', 'speech', 'use', 'penn', 'treebank', 'tag', 'set', '.']

### **Program:**

```
pos_tags = pos_tag(tokens)
```

```
print("POS Tags:", pos_tags)
```

### **Output:**

POS Tags: [('perform', 'NN'), ('lemmatization', 'NN'), ('and', 'CC'), ('stemming', 'NN'), (',', ','), ('identify', 'VB'), ('parts-of', 'JJ'), ('speech', 'NN'), ('using', 'VBG'), ('penn', 'JJ'), ('treebank', 'NN'), ('tag', 'NN'), ('set', 'NN'), (',', ',')]

## Practical No. 08: Implement text classifier using logistic regression model.

Date: /1 /2024

### Program:

```
# prompt: Implement text classifier using logistic regression model

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score


# Sample data (replace with your actual data)

data = {'text': ['This is a positive sentence.', 'This is a negative sentence.', 'Another positive example.', 'A negative one.'],
        'label': [1, 0, 1, 0]} # 1 for positive, 0 for negative

df = pd.DataFrame(data)


# Create TF-IDF features

vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(df['text'])

y = df['label']


# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train a logistic regression model

model = LogisticRegression()

model.fit(X_train, y_train)


# Make predictions on the test set

y_pred = model.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")
```

```
# Example prediction

new_text = ['This is a new positive sentence.']

new_text_vectorized = vectorizer.transform(new_text)

prediction = model.predict(new_text_vectorized)

print(f"Prediction for '{new_text[0]}': {prediction[0]}")
```

**Output:**

Accuracy: 0.0

Prediction for 'This is a new positive sentence.': 1

## Practical No. 06: Implement Named Entity Recognizer.

Date: /1 /2024

### Program:

```
import nltk

# Download necessary NLTK data (if not already downloaded)
nltk.download('maxent_ne_chunker')
nltk.download('words')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
nltk.download('maxent_ne_chunker_tab')

def named_entity_recognizer(text):
    # Tokenize the text
    tokens = nltk.word_tokenize(text)

    # Part-of-speech tagging
    pos_tags = nltk.pos_tag(tokens)

    # Named Entity Recognition using ne_chunk
    named_entities = nltk.ne_chunk(pos_tags)

    # Print the results (you can modify this to return the results in a different format)
    print(named_entities)

# Example usage
text = "Barack Obama was born in Honolulu, Hawaii."
named_entity_recognizer(text)
```

### Output:

```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] /root/nltk_data...
[nltk_data] Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
```

[nltk\_data] Package words is already up-to-date!

[nltk\_data] Downloading package averaged\_perceptron\_tagger to

[nltk\_data] /root/nltk\_data...

[nltk\_data] Package averaged\_perceptron\_tagger is already up-to-

[nltk\_data] date!

[nltk\_data] Downloading package punkt to /root/nltk\_data...

[nltk\_data] Package punkt is already up-to-date!

[nltk\_data] Downloading package maxent\_ne\_chunker\_tab to

[nltk\_data] /root/nltk\_data...

[nltk\_data] Package maxent\_ne\_chunker\_tab is already up-to-date!

(S

(PERSON Barack/NNP)

(PERSON Obama/NNP)

was/VBD

born/VBN

in/IN

(GPE Honolulu/NNP)

,/,

(GPE Hawaii/NNP)

./.)



## Practical No. 01: Convert the text into tokens. Find the word frequency.

Date: 13/8/2024

### Program:

```
import nltk

from collections import Counter

text = """This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for
abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in
the target language before it can be used."""

print("\nThe generated tokenized words are: ")

words = nltk.word_tokenize(text)

print(words, "\n")

word_freq = Counter(words)

print(word_freq)
```

### Output:

The generated tokenized words are:

```
['This', 'tokenizer', 'divides', 'a', 'text', 'into', 'a', 'list', 'of', 'sentences', 'by', 'using', 'an', 'unsupervised', 'algorithm', 'to', 'build',
'a', 'model', 'for', 'abbreviation', 'words', ',', 'collocations', ',', 'and', 'words', 'that', 'start', 'sentences', '.', 'It', 'must', 'be', 'trained',
'on', 'a', 'large', 'collection', 'of', 'plaintext', 'in', 'the', 'target', 'language', 'before', 'it', 'can', 'be', 'used', '.']
```

```
Counter({'a': 4, 'of': 2, 'sentences': 2, 'words': 2, ',': 2, '.': 2, 'be': 2, 'This': 1, 'tokenizer': 1, 'divides': 1, 'text': 1, 'into': 1, 'list':
1, 'by': 1, 'using': 1, 'an': 1, 'unsupervised': 1, 'algorithm': 1, 'to': 1, 'build': 1, 'model': 1, 'for': 1, 'abbreviation': 1,
'collocations': 1, 'and': 1, 'that': 1, 'start': 1, 'It': 1, 'must': 1, 'trained': 1, 'on': 1, 'large': 1, 'collection': 1, 'plaintext': 1, 'in':
1, 'the': 1, 'target': 1, 'language': 1, 'before': 1, 'it': 1, 'can': 1, 'used': 1})
```