Based on the PDF provided, here are the experiments formatted as requested. I have included the 13 experiments listed in the main section, as they are explicitly labeled "EXPERIMENT".

### EXPERIMENT-1: Implementation of Tokenization using UNIX commands

**Code:**

Bash

# 1. Create a text file text1.txt  
echo "This is an example text file." > text1.txt  
  
# 2. Use tr utility to replace characters  
echo "Hello World!" > example.txt  
tr 'o' 'x' < example.txt  
  
# 3. Squeeze repeats and complement options  
echo "aaabbbccc" | tr -s 'a'  
echo "abc123" | tr -c 'a-z' 'X'  
  
# 4. Transform lowercase to uppercase  
tr a-z A-Z < text1.txt  
  
# 5. Create text2.txt and sort  
echo -e "banana\napple\norange\nkiwi\ngrape" > text2.txt  
sort text2.txt  
  
# 6. Sort and display unique lines  
sort -u text2.txt  
  
# 7. Sort, unique lines with frequency count  
sort text2.txt | uniq -c  
  
# 8. Obtain and display tokens in text1.txt (one per line)  
tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$'  
  
# 9. Display tokens in sorted order  
tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' | sort  
  
# 10. Display unique tokens in sorted order  
tr -sc 'A-Za-z' '\n' < text1.txt | grep -v '^$' | sort | uniq

**Output:**

Plaintext

HellxWxrld!  
abbbccc  
XXX123  
THIS IS AN EXAMPLE TEXT FILE.  
  
apple  
banana  
grape  
kiwi  
orange  
  
apple  
banana  
grape  
kiwi  
orange  
  
1 apple  
2 banana  
1 grape  
3 kiwi  
1 orange  
  
This  
is  
an  
example  
text  
file  
  
This  
an  
example  
file  
is  
text  
  
This  
an  
example  
file  
is  
text

### EXPERIMENT-2: Implement a word tokenization using regular expressions

**Code:**

Python

import nltk  
from nltk.tokenize import RegexpTokenizer  
from nltk.tokenize import word\_tokenize  
import re  
  
# Ensure necessary NLTK data is downloaded  
nltk.download('punkt')  
nltk.download('stopwords')  
from nltk.corpus import stopwords  
  
# 1. RegexpTokenizer Example  
s = "Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\n\nThanks."  
tokenizer = RegexpTokenizer(r'\w+|\$[\d\.]+|\S+')  
print(tokenizer.tokenize(s))  
  
# 2. word\_tokenize Example  
sentence = """At eight o'clock on Thursday morning... Arthur didn't feel very good."""  
tokens = nltk.word\_tokenize(sentence)  
print(tokens)  
  
# 3. Custom Regex Examples  
text = "This is V.C.E C.S.E I am a Student. I paid a fees of 13.0"  
print(re.findall(r"(?:[A-Z]\.)+[A-Z]", text))   
  
text2 = "That U.S.A poster-print costs $12.40... which is 3.45."  
print(re.split(r"\s", text2)) # split by whitespace  
print(re.findall(r"\w+-\w+", text2)) # hyphenated words  
print(re.findall(r"(?:\w+-\w+)", text2))  
  
# 4. Filtering Stopwords  
li = []  
for w in word\_tokenize(text2):  
 if w not in stopwords.words('english'):  
 li.append(w)  
print(li)

**Output:**

Plaintext

['Good', 'muffins', 'cost', '$3.88', 'in', 'New', 'York', '.', 'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']  
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', '...', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']  
['V.C.E', 'C.S.E']  
['That', 'U.S.A', 'poster-print', 'costs', '$12.40...', 'which', 'is', '3.45.']  
['poster-print']  
['poster-print']  
['That', 'U.S.A', 'poster-print', 'costs', '$', '12.40', '...', '3.45', '.']

### EXPERIMENT-3: Implement Minimum Edit Distance (MED) algorithm for spelling correction

**Code:**

Python

import nltk  
  
def find\_minimum\_edit\_distance(word1, word2):  
 distance = nltk.edit\_distance(word1, word2)  
 return distance  
  
# Example usage  
word1 = "kitten"  
word2 = "sitting"  
min\_edit\_distance = find\_minimum\_edit\_distance(word1, word2)  
  
print(f"The minimum edit distance between '{word1}' and '{word2}' is: {min\_edit\_distance}")  
  
# Manual DP Implementation logic (as shown in record)  
source = 'kitten'  
target = 'sitting'  
m = len(source)  
n = len(target)  
dp = [[0 for i in range(m+1)] for j in range(n+1)]  
  
for i in range(m+1):  
 dp[0][i] = i  
for j in range(n+1):  
 dp[j][0] = j  
  
for i in range(1, n+1):  
 for j in range(1, m+1):  
 if source[j-1] == target[i-1]:  
 cost = 0  
 else:  
 cost = 1  
 dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+cost)  
  
print('Edit Distance (DP): ', dp[n][m])

**Output:**

Plaintext

The minimum edit distance between 'kitten' and 'sitting' is: 3  
Edit Distance (DP): 3

### EXPERIMENT-4: Implement n-gram language model

**Code:**

Python

import nltk  
from nltk.corpus import brown  
from nltk.util import bigrams  
from nltk.lm.preprocessing import pad\_both\_ends, padded\_everygram\_pipeline  
from nltk.lm import MLE  
  
nltk.download('brown')  
nltk.download('punkt')  
  
corpus = brown.sents(categories="news")  
test\_sentence = ['There', "wasn't", 'a', 'bit', 'of', 'trouble', 'in', 'Texas']  
  
# Generate bigrams for test sentence  
test\_sentence\_bigrams = list(bigrams(pad\_both\_ends(test\_sentence, n=2)))  
print(test\_sentence\_bigrams)  
  
# Train the model  
train\_data, vocab = padded\_everygram\_pipeline(2, corpus)  
lm = MLE(2)  
lm.fit(train\_data, vocab)  
  
print("Number of words in vocabulary is:", len(lm.vocab))  
  
prob = 1  
for t in test\_sentence\_bigrams:  
 score = lm.score(t[1], [t[0]])  
 print(score)  
 prob \*= score  
  
print(prob)

**Output:**

Plaintext

[('<s>', 'There'), ('There', "wasn't"), ("wasn't", 'a'), ('a', 'bit'), ('bit', 'of'), ('of', 'trouble'), ('trouble', 'in'), ('in', 'Texas'), ('Texas', '</s>')]  
Number of words in vocabulary is: 14397  
0.011464417045208739  
0.017241379310344827  
0.3333333333333333  
0.002508780732563974  
0.2857142857142857  
0.001053001053001053  
0.375  
0.001584786053882726  
0.125  
3.6943506664397765e-15

### EXPERIMENT-5: Implement Naïve Bayes classification for sentiment analysis

**Code:**

Python

import nltk  
from nltk.corpus import movie\_reviews  
import random  
  
nltk.download('movie\_reviews')  
  
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 = nltk.FreqDist(w.lower() for w in movie\_reviews.words())  
word\_features = list(all\_words)[:2000]  
  
def document\_features(document):  
 document\_words = set(document)  
 features = {}  
 for word in word\_features:  
 features[f'contains({word})'] = (word in document\_words)  
 return features  
  
featuresets = [(document\_features(d), c) for (d, c) in documents]  
train\_set, test\_set = featuresets[100:], featuresets[:100]  
  
classifier = nltk.NaiveBayesClassifier.train(train\_set)  
  
confusion\_matrix = {"tp": 0, "tn": 0, "fp": 0, "fn": 0}  
  
for i in range(len(test\_set)):  
 predicted = classifier.classify(test\_set[i][0])  
 actual = test\_set[i][1]  
 # print(predicted, actual) # Optional: print each prediction  
   
 if predicted == 'pos' and actual == 'pos':  
 confusion\_matrix['tp'] += 1  
 elif predicted == 'neg' and actual == 'neg':  
 confusion\_matrix['tn'] += 1  
 elif predicted == 'pos' and actual == 'neg':  
 confusion\_matrix['fp'] += 1  
 else:  
 confusion\_matrix['fn'] += 1  
  
print("\nConfusion Matrix:")  
print("True Positives (TP): ", confusion\_matrix["tp"])  
print("True Negatives (TN): ", confusion\_matrix["tn"])  
print("False Positives (FP): ", confusion\_matrix["fp"])  
print("False Negatives (FN): ", confusion\_matrix["fn"])

**Output:**

Plaintext

Confusion Matrix:  
True Positives (TP): 33  
True Negatives (TN): 44  
False Positives (FP): 13  
False Negatives (FN): 10

### EXPERIMENT-6: Implement POS tagging using HMM

**Code:**

Python

# 6.a) Basic usage of HMM trainer  
import nltk  
from nltk.corpus import brown  
from nltk.tag import hmm  
  
nltk.download('brown')  
nltk.download('punkt')  
  
sentences = brown.tagged\_sents()  
trainer = hmm.HiddenMarkovModelTrainer()  
tagger = trainer.train(sentences)  
  
text = "this is a sample sentence for POS tagging in python"  
words = nltk.word\_tokenize(text)  
tags = tagger.tag(words)  
  
for word, tag in tags:  
 print(f"{word}: {tag}")  
  
# 6.b) Accuracy testing  
brown\_tagged\_sentences = brown.tagged\_sents(categories='news')  
size = int(len(brown\_tagged\_sentences) \* 0.9)  
train\_sentences = brown\_tagged\_sentences[:size]  
test\_sentences = brown\_tagged\_sentences[size:] # Fixed slicing from record to make sense  
  
trainer = hmm.HiddenMarkovModelTrainer()  
tagger = trainer.train(train\_sentences)  
print("Accuracy:", tagger.accuracy(test\_sentences))

**Output:**

Plaintext

this: DT  
is: BEZ  
a: AT  
sample: NN  
sentence: NN  
for: IN  
POS: AT  
tagging: AT  
in: AT  
python: AT  
  
Accuracy: 0.98089945979386

### EXPERIMENT-7: Implement CKY parsing algorithm

**Code:**

Python

def print\_chart(chart, n):  
 for i in range(n):  
 for j in range(n + 1):  
 print(chart[i][j], end="\t")  
 print()  
  
def CKY\_PARSE(words, grammar):  
 n = len(words)  
 table = [[set() for \_ in range(n + 1)] for \_ in range(n + 1)]  
  
 # Fill diagonal with preterminal rules  
 for i in range(n):  
 word = words[i]  
 for lhs, rhs in grammar:  
 if rhs == (word,):  
 table[i][i + 1].add(lhs)  
  
 # Fill upper cells using binary rules  
 for length in range(2, n + 1):  
 for i in range(n - length + 1):  
 j = i + length  
 for k in range(i + 1, j):  
 for lhs, rhs in grammar:  
 if len(rhs) == 2:  
 if rhs[0] in table[i][k] and rhs[1] in table[k][j]:  
 table[i][j].add(lhs)  
 return table  
  
sentence = "the dog chased the cat"  
words = sentence.split()  
n = len(words)  
  
grammar = [('S', ('NP', 'VP')),   
 ('NP', ('DET', 'NOMINAL')),   
 ('VP', ('VERB', 'NP')),   
 ('NOMINAL', ('cat',)),   
 ('NOMINAL', ('dog',)),   
 ('VERB', ('chased',)),   
 ('DET', ('the',))]  
  
chart = CKY\_PARSE(words, grammar)  
print(words, "\n")  
print\_chart(chart, n)  
  
start\_symbol = 'S'  
if start\_symbol in chart[0][n]:  
 print("The sentence is grammatically correct.")  
else:  
 print("The sentence is not grammatically correct.")

**Output:**

Plaintext

['the', 'dog', 'chased', 'the', 'cat']   
  
set() {'DET'} {'NP'} set() set() {'S'}   
set() set() {'NOMINAL'} set() set() set()   
set() set() set() {'VERB'} set() set()   
set() set() set() set() {'DET'} {'NP'}   
set() set() set() set() set() {'NOMINAL'}   
  
The sentence is grammatically correct.

### EXPERIMENT-8: Implement PCKY parsing algorithm

**Code:**

Python

def print\_chart(chart, n, non\_terminals):  
 for p in range(n + 1):  
 for q in range(n + 1):  
 print(f'[{p}, {q}]:', end=" ")  
 for nt in non\_terminals:  
 if chart[p][q][nt] > 0:  
 print(f'{{ {nt}: {chart[p][q][nt]} }}', end="")  
 print()  
 print()  
  
def PCKY\_PARSE(words, grammar, non\_terminals):  
 n = len(words)  
 print(words, "\n")  
 table = [[{nt: 0.0 for nt in non\_terminals} for \_ in range(n + 1)] for \_ in range(n + 1)]  
  
 # Fill diagonal  
 for j in range(1, n + 1):  
 for lhs, rhs, pr in grammar:  
 if rhs == (words[j - 1],):  
 table[j - 1][j][lhs] = pr  
  
 # Fill upper cells  
 for length in range(2, n+1):   
 for i in range(n - length + 1):  
 j = i + length  
 for k in range(i + 1, j):  
 for lhs, rhs, pr in grammar:  
 if len(rhs) == 2 and table[i][k][rhs[0]] > 0 and table[k][j][rhs[1]] > 0:  
 prob = pr \* table[i][k][rhs[0]] \* table[k][j][rhs[1]]  
 if table[i][j][lhs] < prob:  
 table[i][j][lhs] = prob  
 return table  
  
sentence = "the flight includes a meal"  
words = sentence.split()  
n = len(words)  
  
grammar = [  
 ('S', ('NP', 'VP'), 0.80),   
 ('NP', ('DET', 'NOMINAL'), 0.30),   
 ('VP', ('VERB', 'NP'), 0.20),   
 ('NOMINAL', ('meal',), 0.01),   
 ('NOMINAL', ('flight',), 0.02),   
 ('VERB', ('includes',), 0.05),   
 ('DET', ('the',), 0.40),   
 ('DET', ('a',), 0.40)  
]  
  
non\_terminals = ['S', 'NP', 'VP', 'DET', 'NOMINAL', 'VERB']  
chart = PCKY\_PARSE(words, grammar, non\_terminals)  
print\_chart(chart, n, non\_terminals)  
  
if chart[0][n]['S'] > 0:  
 print("The sentence is grammatically correct.")

**Output:**

Plaintext

['the', 'flight', 'includes', 'a', 'meal']   
  
...  
[0, 1]: { DET: 0.4 }  
[0, 2]: { NP: 0.0024 }  
[0, 5]: { S: 2.3040000000000003e-08 }  
[1, 2]: { NOMINAL: 0.02 }  
[2, 3]: { VERB: 0.05 }  
[2, 5]: { VP: 1.2000000000000002e-05 }  
[3, 4]: { DET: 0.4 }  
[3, 5]: { NP: 0.0012 }  
[4, 5]: { NOMINAL: 0.01 }  
...  
The sentence is grammatically correct.

### EXPERIMENT-9: Implementation of Computing cosine similarity between the words using term document matrix

**Code:**

Python

import nltk  
import random  
import math  
from nltk.corpus import brown, stopwords  
  
nltk.download('brown')  
nltk.download('stopwords')  
  
doc\_names = ['ca01', 'ca02', 'ca03', 'ca04']  
  
def extract\_words(document):  
 all\_terms\_list = brown.words(fileids=document)  
 only\_words\_list = [w.lower() for w in all\_terms\_list if w.isalpha()]  
 stopwords\_list = stopwords.words('english')  
 final\_terms\_list = [w for w in only\_words\_list if w not in stopwords\_list]  
 return final\_terms\_list  
  
def freq(word, document):  
 d\_terms = extract\_words(document)  
 fdist = nltk.FreqDist(d\_terms)  
 return fdist[word]  
  
vocab = set()  
for doc in doc\_names:  
 vocab.update(extract\_words(doc))  
  
print("Length of vocabulary =", len(vocab))  
  
word1 = list(vocab)[random.randint(0, len(vocab)-1)]  
word2 = list(vocab)[random.randint(0, len(vocab)-1)]  
  
print("word-1:", word1)  
print("word-2:", word2)  
  
word1\_vector = [freq(word1, doc) for doc in doc\_names]  
word2\_vector = [freq(word2, doc) for doc in doc\_names]  
  
print("word-1-vector:", word1\_vector)  
print("word-2-vector:", word2\_vector)  
  
dot\_product = sum(word1\_vector[i] \* word2\_vector[i] for i in range(len(doc\_names)))  
vector1\_len = math.sqrt(sum(w\*\*2 for w in word1\_vector))  
vector2\_len = math.sqrt(sum(w\*\*2 for w in word2\_vector))  
  
if vector1\_len \* vector2\_len != 0:  
 cos\_theta = dot\_product / (vector1\_len \* vector2\_len)  
else:  
 cos\_theta = 0  
  
print(f"cos\_theta({word1}, {word2}) =", cos\_theta)

**Output:**

Plaintext

Length of vocabulary = 2023  
word-1: ignored  
word-2: interest  
word-1-vector: [0, 0, 1, 0]  
word-2-vector: [2, 0, 0, 0]  
cos\_theta(ignored, interest) = 0.0

### EXPERIMENT-10: Implementation of TF-IDF matrix for the given document set

**Code:**

Python

import nltk  
import random  
import math  
from nltk.corpus import brown, stopwords  
  
nltk.download('brown')  
nltk.download('stopwords')  
  
doc\_names = ['ca01', 'ca02', 'ca03', 'ca04']  
  
def extract\_words(document):  
 all\_terms\_list = brown.words(fileids=document)  
 only\_words\_list = [w.lower() for w in all\_terms\_list if w.isalpha()]  
 stopwords\_list = stopwords.words('english')  
 return [w for w in only\_words\_list if w not in stopwords\_list]  
  
def term\_freq(word, document):  
 d\_terms = extract\_words(document)  
 fdist = nltk.FreqDist(d\_terms)  
 return math.log10(fdist[word] + 1)  
  
def idf(word):  
 df = 0  
 for doc in doc\_names:  
 if word in extract\_words(doc):  
 df += 1  
 return math.log10(len(doc\_names) / df) if df != 0 else 0  
  
vocab = set()  
for doc in doc\_names:  
 vocab.update(extract\_words(doc))  
print("Length of vocabulary =", len(vocab))  
  
word1 = list(vocab)[random.randint(0, len(vocab)-1)]  
word2 = list(vocab)[random.randint(0, len(vocab)-1)]  
print("word-1:", word1)  
print("word-2:", word2)  
  
word1\_vector = [term\_freq(word1, doc) \* idf(word1) for doc in doc\_names]  
word2\_vector = [term\_freq(word2, doc) \* idf(word2) for doc in doc\_names]  
  
print("word-1-vector:", word1\_vector)  
print("word-2-vector:", word2\_vector)  
  
dot\_product = sum(word1\_vector[i] \* word2\_vector[i] for i in range(len(doc\_names)))  
vector1\_len = math.sqrt(sum(w\*\*2 for w in word1\_vector))  
vector2\_len = math.sqrt(sum(w\*\*2 for w in word2\_vector))  
  
cos\_theta = dot\_product / (vector1\_len \* vector2\_len) if (vector1\_len \* vector2\_len) != 0 else 0  
print(f"cos\_theta({word1}, {word2}) =", cos\_theta)

**Output:**

Plaintext

Length of vocabulary = 2023  
word-1: opelika  
word-2: compulsory  
word-1-vector: [0.1812381165789131, 0.0, 0.0, 0.0]  
word-2-vector: [0.0, 0.0, 0.1812381165789131, 0.0]  
cos\_theta(opelika, compulsory) = 0.0

### EXPERIMENT-11: Language Model Using Feed Forward Neural Network

**Code:**

Python

import tensorflow as tf  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
import numpy as np  
  
corpus = ['This is a simple example', 'Language modeling is interesting',   
 'Neural networks are powerful',   
 'Feed-forward networks are common in natural language processing']  
  
tokenizer = Tokenizer()  
tokenizer.fit\_on\_texts(corpus)  
total\_words = len(tokenizer.word\_index) + 1  
  
input\_sequences = []  
for line in corpus:  
 token\_list = tokenizer.texts\_to\_sequences([line])[0]  
 for i in range(1, len(token\_list)):  
 n\_gram\_sequence = token\_list[:i+1]  
 input\_sequences.append(n\_gram\_sequence)  
  
max\_sequence\_length = max([len(x) for x in input\_sequences])  
input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_length, padding='pre')  
  
X, y = input\_sequences[:, :-1], input\_sequences[:, -1]  
y = tf.keras.utils.to\_categorical(y, num\_classes=total\_words)  
  
model = tf.keras.Sequential([  
 tf.keras.layers.Embedding(total\_words, 50, input\_length=max\_sequence\_length-1),  
 tf.keras.layers.LSTM(100),  
 tf.keras.layers.Dense(total\_words, activation='softmax')  
])  
  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
model.fit(X, y, epochs=100, verbose=1)  
  
seed\_text = "Neural networks"  
next\_words = 7  
for \_ in range(next\_words):  
 token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]  
 token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_length-1, padding='pre')  
 predicted\_index = np.argmax(model.predict(token\_list, verbose=0), axis=-1)[0]  
   
 output\_word = ""  
 for word, index in tokenizer.word\_index.items():  
 if index == predicted\_index:  
 output\_word = word  
 break  
 seed\_text += " " + output\_word  
  
print(seed\_text)

**Output:**

Plaintext

Epoch 1/100 ... loss: 2.8825 - accuracy: 0.1250  
...  
Epoch 100/100 ... loss: 0.0587 - accuracy: 1.0000  
Neural networks are powerful in Natural Language Processing.

### EXPERIMENT-12: Implement Language Model Using RNN

**Code:**

Python

import tensorflow as tf  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
import numpy as np  
  
corpus = ['This is a simple example',  
 'Language modeling is interesting',  
 'Neural networks are powerful',  
 'Recurrent neural networks capture sequences well']  
  
tokenizer = Tokenizer()  
tokenizer.fit\_on\_texts(corpus)  
total\_words = len(tokenizer.word\_index) + 1  
  
input\_sequences = []  
for line in corpus:  
 token\_list = tokenizer.texts\_to\_sequences([line])[0]  
 for i in range(1, len(token\_list)):  
 n\_gram\_sequence = token\_list[:i+1]  
 input\_sequences.append(n\_gram\_sequence)  
  
max\_sequence\_length = max([len(x) for x in input\_sequences])  
input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_length, padding='pre')  
X, y = input\_sequences[:, :-1], input\_sequences[:, -1]  
y = tf.keras.utils.to\_categorical(y, num\_classes=total\_words)  
  
model = tf.keras.Sequential([  
 tf.keras.layers.Embedding(total\_words, 50, input\_length=max\_sequence\_length-1),  
 tf.keras.layers.LSTM(100),  
 tf.keras.layers.Dense(total\_words, activation='softmax')  
])  
  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
model.fit(X, y, epochs=100, verbose=1)  
  
seed\_text = "Recurrent neural networks"  
next\_words = 5  
  
for \_ in range(next\_words):  
 token\_list = tokenizer.texts\_to\_sequences([seed\_text])[0]  
 token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_length-1, padding='pre')  
 predicted = np.argmax(model.predict(token\_list, verbose=0), axis=-1)[0]  
   
 output\_word = ""  
 for word, index in tokenizer.word\_index.items():  
 if index == predicted:  
 output\_word = word  
 break  
 seed\_text += " " + output\_word  
  
print(seed\_text)

**Output:**

Plaintext

Epoch 1/100 ... loss: 2.8342 - accuracy: 0.0667  
...  
Epoch 100/100 ... loss: 0.3300 - accuracy: 1.0000  
Recurrent neural networks capture sequences well well well

### EXPERIMENT-13: Implement Perform Text Analytics

**Code:**

Python

from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
  
# Sample data  
documents = ["I love programming.", "Python is great for NLP.", "Text analytics is fun!"]  
labels = [1, 1, 0] # 1 for positive, 0 for neutral/negative  
  
# Vectorization  
vectorizer = CountVectorizer()  
X = vectorizer.fit\_transform(documents)  
  
# Model training  
clf = MultinomialNB()  
clf.fit(X, labels)  
  
# Prediction  
test\_docs = ["I enjoy coding.", "NLP is interesting."]  
X\_test = vectorizer.transform(test\_docs)  
predictions = clf.predict(X\_test)  
  
print(predictions)

**Output:**

Plaintext

[1 1]

**Sources**

1. <https://atmokpo.com/w/25283/>

2. <https://atmokpo.com/w/25283/>