National University of Computer & Emerging Sciences, Karachi

FALL 2024, LAB MANUAL – 10

NAÏVE BAYES

COURSE CODE :	**AL3002**
INSTRUCTOR :	**Usama Bin Umar**

OBJECTIVE

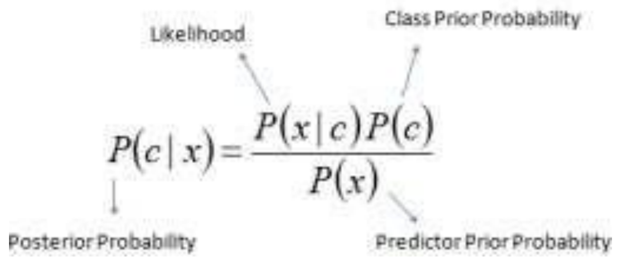
1. Implementation of Naïve Bayes
2. Types of Naïve Bayes
3. NLP and Recommendation System

NAÏVE BAYES

The Naïve Bayes classifier is a popular supervised machine learning algorithm used for classification tasks such as text classification. It belongs to the family of generative learning algorithms, which means that it models the distribution of inputs for a given class or category. This approach is based on the assumption that the features of the input data are conditionally independent given the class, allowing the algorithm to make predictions quickly and accurately.

Naive Bayes is a probabilistic machine learning algorithm that is based on Bayes' theorem. It is a classification technique that assumes independence among the features used to describe an observation. Despite its seemingly naive assumption of feature independence (hence the name "naive"), it has been found to work well in practice, especially for text classification problems like spam filtering and sentiment analysis.

The basic idea behind Naive Bayes can be explained using Bayes' theorem. Given a class variable C and a dependent feature vector X, Bayes' theorem states:


Likelihood
Class Prior Probability
$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of the predictor given class.
- P(x) is the prior probability of the predictor.

The "naive" part comes from the assumption that the features in X are conditionally independent given the class C, which simplifies the computation of P(X|C) as the product of the individual probabilities of each feature given the class.

Naive Bayes is widely used in various applications due to its simplicity, speed, and the fact that it often performs surprisingly well, especially in situations where the independence assumption holds reasonably well.

Naive Bayes Classifier Assumptions:

Feature Independence (Naive Assumption): The algorithm assumes that the features used for classification are conditionally independent given the class label. This is a simplifying assumption that allows for efficient computation of probabilities.

Probabilistic Predictions: Instead of providing a definite classification, Naive Bayes outputs probabilities. It calculates the probability of an observation belonging to each class and assigns the class with the highest probability as the predicted class.

Works well with Categorical Data: Naive Bayes is particularly suitable for categorical data, such as word counts in text classification problems.

Fast Training and Prediction: Naive Bayes is computationally efficient, making it fast for both training and prediction. This makes it suitable for large datasets.

Gaussian Naïve Bayes Classifier

```python
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()

modelNB = NB.fit(x_train,y_train)
prediction= modelNB.predict(x_test)

from sklearn.metrics import accuracy_score
print("====================Training Accuarcy============")
trac=NB.score(x_train,y_train)
print(trac)
print("====================Testing Accuarcy============")
teacNB=accuracy_score(y_test,prediction)
print(teacNB)
```

TYPES OF NAÏVE BAYES

Gaussian Naive Bayes: Gaussian is used in classification tasks and it assumes that feature values follow a Gaussian distribution.

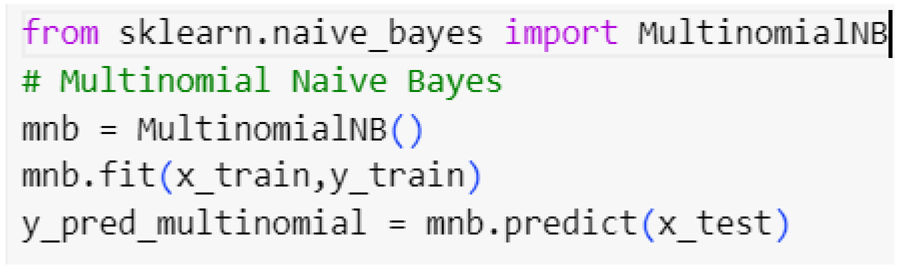
Multinomial Naive Bayes: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".

Bernoulli Naive Bayes: The binomial model is useful if your feature vectors are boolean (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

Complement Naive Bayes: It is an adaptation of Multinomial NB where the complement of each class is used to calculate the model weights. So, this is suitable for imbalanced data sets and often outperforms the MNB on text classification tasks.

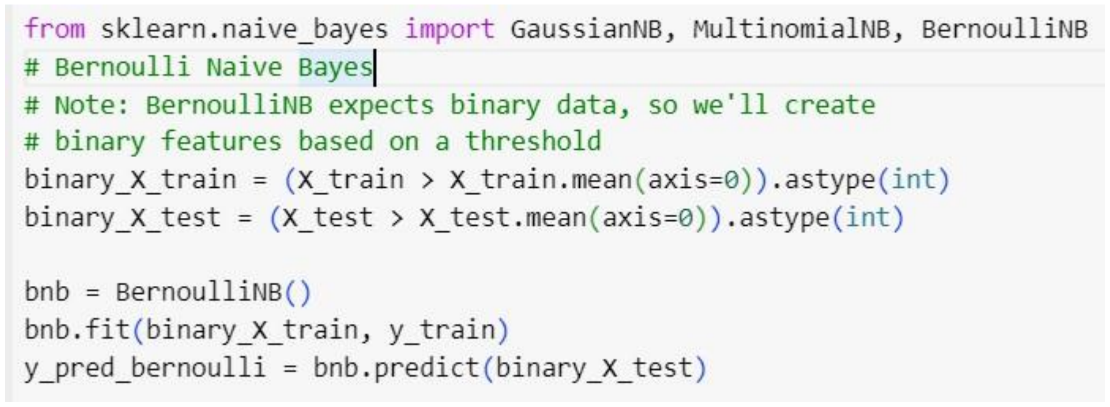
Categorical Naive Bayes: Categorical Naive Bayes is useful if the features are categorically distributed. We have to encode the categorical variable in the numeric format using the ordinal encoder for using this algorithm.

Multinomial Naïve Bayes Classifier



```python
from sklearn.naive_bayes import MultinomialNB
# Multinomial Naive Bayes
mnb = MultinomialNB()
mnb.fit(x_train,y_train)
y_pred_multinomial = mnb.predict(x_test)
```

Bernoulli Naïve Bayes Classifier



```python
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
# Bernoulli Naive Bayes
# Note: BernoulliNB expects binary data, so we'll create
# binary features based on a threshold
binary_X_train = (X_train > X_train.mean(axis=0)).astype(int)
binary_X_test = (X_test > X_test.mean(axis=0)).astype(int)

bnb = BernoulliNB()
bnb.fit(binary_X_train, y_train)
y_pred_bernoulli = bnb.predict(binary_X_test)
```

Multinomial Naïve Bayes Classifier

```
from sklearn.naive_bayes import ComplementNB
# Multinomial Naive Bayes
cnb = ComplementNB()
cnb.fit(x_train,y_train)
y_pred_cnb = cnb.predict(x_test)
```

NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is a field of artificial intelligence (AI) that focuses on the interaction between computers and human languages. The goal of NLP is to enable computers to understand, interpret, and generate human-like text or speech. It involves the development of algorithms and models that can process, analyze, and derive meaning from natural language data.

APPLICATIONS OF NLP

Text Processing: Breaking down text into individual words (tokenization), identifying sentence boundaries, and other basic text manipulations.

Part-of-Speech Tagging (POS): Assigning grammatical categories (such as noun, verb, adjective) to each word in a sentence.

Named Entity Recognition (NER): Identifying and classifying entities (e.g., names of people, organizations, locations) in text.

Sentiment Analysis: Determining the sentiment expressed in a piece of text, whether it is positive, negative, or neutral.

Machine Translation: Automatically translating text from one language to another.

Speech Recognition: Converting spoken language into written text.

COMPONENTS OF NLP

There are the following two components of NLP

1. Natural Language Understanding (NLU)

Natural Language Understanding (NLU) helps the machine to understand and analyse human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.

NLU mainly used in Business applications to understand the customer's problem in both spoken and written language.

NLU involves the following tasks -

- It is used to map the given input into useful representation.
- It is used to analyze different aspects of the language.

2. Natural Language Generation (NLG)
Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization.

NLP PREPROCESSING

- **Tokenization**
- Sentence Tokenization
- Word Tokenization
- **Text Cleaning and Normalization**
- Text Lowercasing
- Removing Punctuation
- Removing Special Characters and Numbers
- Removing Stopwords
- Stemming or Lemmatization
- Handling Contractions
- Spell Checking and Correction
- Removing HTML Tags (for web-based text)
- Removing Redundant Whitespace
- POS tagging

NLTK

NLTK stands for Natural Language Toolkit, and it is a powerful Python library for working with human language data (text). NLTK provides easy-to-use interfaces to over 50 corpora and lexical resources, such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and more.

TOKENIZATION

Tokenization in Natural Language Processing (NLP) is the process of breaking down a sequence of text into individual units, known as tokens. A token can be a word, a subword, or even a character, depending on the level of granularity chosen for the analysis. Tokenization is a fundamental step in NLP that allows a computer to understand and process human language by treating individual elements as separate entities.

TYPES OF TOKENIZATION

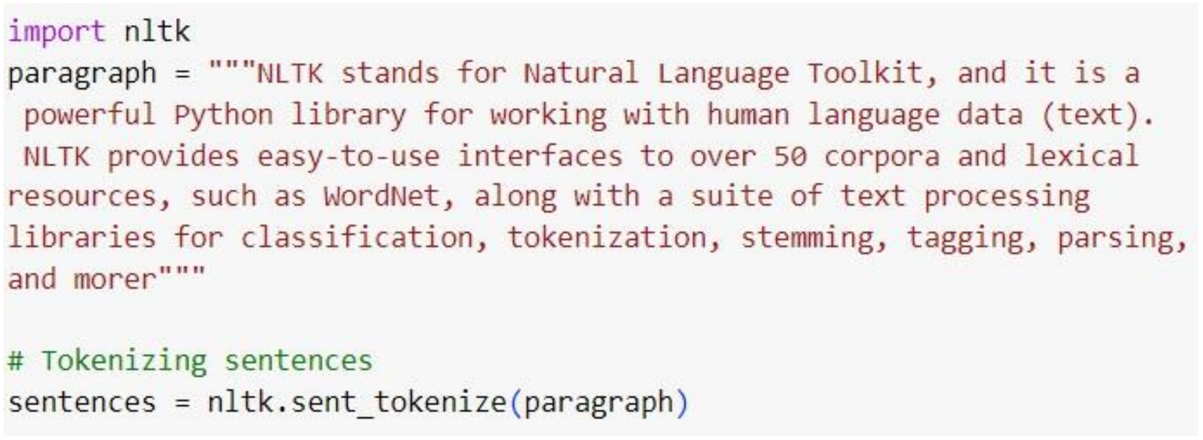
Sentence Tokenization: Divides text into sentences.

Example:

Input: "NLP is fascinating. It involves processing text."

Output: ["NLP is fascinating.", "It involves processing text."]

Sentence Tokenization / Segmentation



```
import nltk
paragraph = """NLTK stands for Natural Language Toolkit, and it is a
 powerful Python library for working with human language data (text).
 NLTK provides easy-to-use interfaces to over 50 corpora and lexical
resources, such as WordNet, along with a suite of text processing
libraries for classification, tokenization, stemming, tagging, parsing,
and morer"""

# Tokenizing sentences
sentences = nltk.sent_tokenize(paragraph)
```

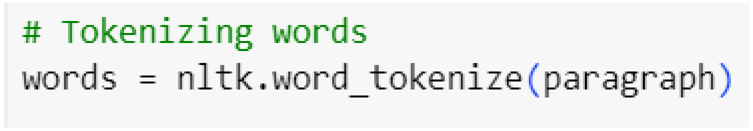
Word Tokenization: Breaks text into individual words.

Example:

Input: "she is a girl."

Output: ["she", "is", ,"a", "girl."]

Word Tokenization / Segmentation



```
# Tokenizing words
words = nltk.word_tokenize(paragraph)
```

Character Tokenization: Breaks text into individual characters.

Example:

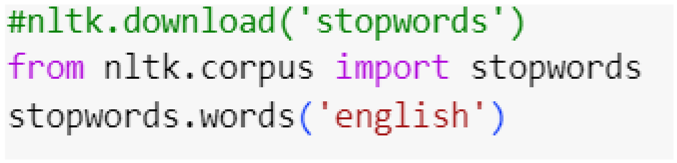
Input: "Tokenization"

Output: ['T', 'o', 'k', 'e', 'n', 'i', 'z', 'a', 't', 'i', 'o', 'n']

STOP WORD ANALYSIS

In English, some words appear more frequently than others such as "is", "a", "the", "and". As they appear often, the NLP pipeline flags them as stop words. They are filtered out so as to focus on more important words.

Stop Words

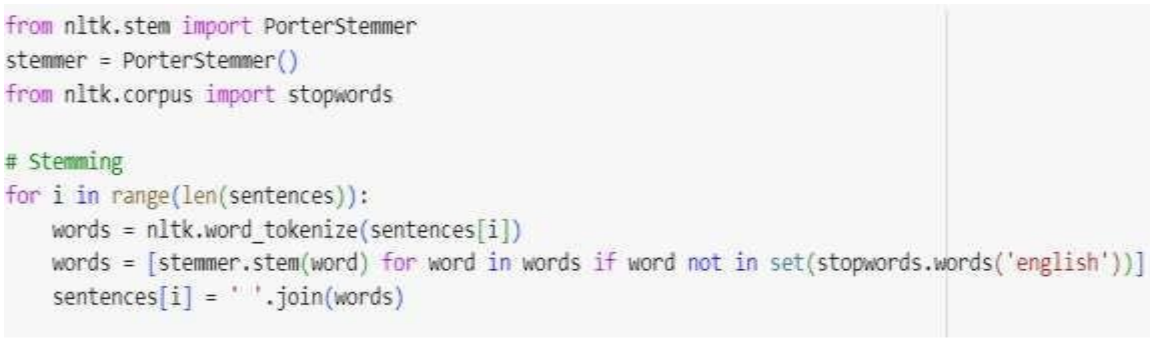


```python
#nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords.words('english')
```

Stemming helps in preprocessing text. The model analyzes the parts of speech to figure out what exactly the sentence is talking about.

Stemming normalizes words into their base or root form. In other words, it helps to predict the parts of speech for each token. For example, intelligently, intelligence, and intelligent. These words originate from a single root word 'intelligen'. However, in English there's no such word as 'intelligen'.

Stemming



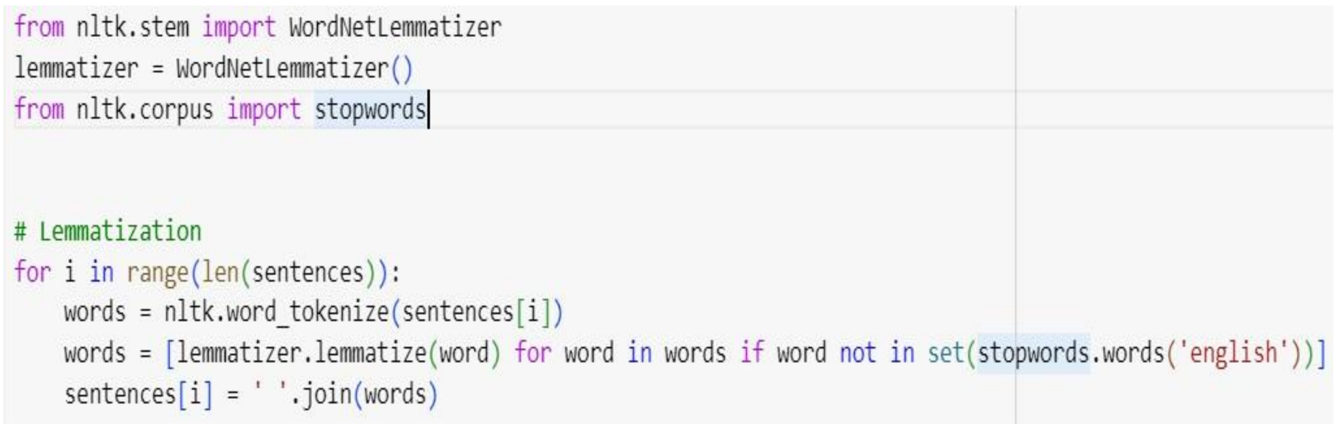
```python
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
from nltk.corpus import stopwords

# Stemming
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [stemmer.stem(word) for word in words if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)
```

LEMMATIZATION

Lemmatization removes inflectional endings and returns the canonical form of a word or lemma. It is similar to stemming except that the lemma is an actual word. For example, 'playing' and 'plays' are forms of the word 'play'. Hence, play is the lemma of these words. Unlike a stem (recall 'intelligen'), 'play' is a proper word.

Lemmatization



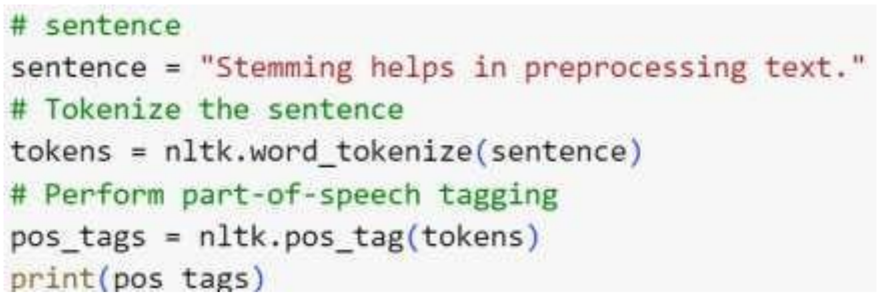
```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
from nltk.corpus import stopwords


# Lemmatization
for i in range(len(sentences)):
    words = nltk.word_tokenize(sentences[i])
    words = [lemmatizer.lemmatize(word) for word in words if word not in set(stopwords.words('english'))]
    sentences[i] = ' '.join(words)
```

POS TAGGING

POS tags contain verbs, adverbs, nouns, and adjectives that help indicate the meaning of words in a grammatically correct way in a sentence.

Pos Tagging



```
# sentence
sentence = "Stemming helps in preprocessing text."
# Tokenize the sentence
tokens = nltk.word_tokenize(sentence)
# Perform part-of-speech tagging
pos_tags = nltk.pos_tag(tokens)
print(pos_tags)
```

TASKS

TASK 1:

As part of your academic research, you are tasked with implementing the User Profile Correlation-Based Similarity (UPCSim) algorithm for movie recommendations, as outlined in a specific research paper.

Paper Understanding:

Carefully read the research paper to understand the theoretical foundations and methodology of the UPCSim algorithm. Take note of the key concepts, equations, and algorithmic steps.

Data Preparation:
Identify the dataset used in the research paper for experimentation. If a specific dataset is not mentioned, choose a relevant movie dataset (e.g., MovieLens) and preprocess it in a manner consistent with the paper's methodology.
Algorithm Implementation:
Implement the UPCSim algorithm based on the details provided in the research paper. Pay close attention to any pseudocode, equations, or specific instructions outlined in the paper.
Feature Extraction:
Extract the required features from the dataset to construct user profiles, following the methodology presented in the research paper. Consider aspects such as user ratings, genres preferences, and any additional features deemed crucial.
Correlation Calculation:
Calculate the correlation between user profiles using the UPCSim approach. Use appropriate correlation metrics mentioned in the paper, such as Pearson correlation coefficient or any other recommended similarity measures.
Recommendation Generation:
Implement the logic for generating movie recommendations based on the calculated user profile correlations. Ensure that your implementation aligns with the recommendation generation process outlined in the research paper.
Evaluation Metrics:
Apply the evaluation metrics specified in the research paper to assess the performance of your implementation. Compare the results with those reported in the paper and provide an analysis of any variations.
Challenges and Solutions:
Identify any challenges encountered during the implementation and propose solutions or adjustments based on your understanding of the algorithm and its application to movie recommendations.

TASK 2:

You are part of a development team tasked with creating a cutting-edge content recommendation system for a movie streaming platform. The platform caters to a diverse audience, from casual viewers to avid movie enthusiasts. The primary objective is to enhance the user experience by offering personalized movie recommendations that align with individual preferences, viewing habits, and thematic interests

Create a Content-Based Recommendation System for MovieLens Dataset

TASK 3:

You are a data scientist working on an email filtering system for a large email service provider. The goal is to automatically categorize incoming emails into "Spam" or "Ham" (non-spam) categories. The company has decided to leverage Natural Language Processing (NLP) techniques and a Multinomial Naive Bayes classifier for this task.
How would you design and implement a spam and ham classification system using NLP and

Multinomial Naive Bayes? Consider the challenges associated with distinguishing between legitimate and spam emails

TASK 4:

Imagine you are part of a team developing a sentiment analysis system for an e-commerce platform. The goal is to automatically classify product reviews into positive, negative, or neutral sentiments using a Naive Bayes classifier. This system will help the platform understand customer feedback more efficiently and improve the overall user experience.

Exercise:

Design and implement a Naive Bayes machine learning model for sentiment analysis based on product reviews.