MACHINE LEARNING INTERNSHIP

TASK1 TWITTER SENTIMENT

Create a sentiment analysis model for twitter data.Analyze tweets to understand public sentiment on specific topics.

python

import tweepy

Twitter API credentials (replace with your own credentials)

consumer_key = 'YOUR_CONSUMER_KEY'

consumer_secret = 'YOUR_CONSUMER_SECRET'

access_token = 'YOUR_ACCESS_TOKEN'

access_token_secret = 'YOUR_ACCESS_TOKEN_SECRET'

Authenticate to Twitter

auth = tweepy.OAuth1UserHandler(consumer_key, consumer_secret, access_token, access_token_secret)

api = tweepy.API(auth)

Fetch tweets for a specific keyword

def fetch_tweets(query, count=100):

tweets = tweepy.Cursor(api.search_tweets, q=query, lang='en', tweet_mode='extended').items(count)

tweet_data = []

```
    for tweet in tweets:
        tweet_data.append(tweet.full_text)
    return tweet_data
```

Example: Fetch tweets related to "Bitcoin"

```
tweets = fetch_tweets("Bitcoin", 100)
print(tweets[:5])  # Show first 5 tweets
```

python

```
import nltk
import string
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

Download required NLTK resources

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

Initialize lemmatizer

```
lemmatizer = WordNetLemmatizer()
```

Function to preprocess text

def preprocess_text(text):

 # Lowercase the text

 text = text.lower()

 # Remove punctuation

 text = ''.join([char for char in text if char not in string.punctuation])

 # Tokenize text

 tokens = word_tokenize(text)

 # Remove stopwords and lemmatize the words

 stop_words = set(stopwords.words("english"))

 tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]

 return ' '.join(tokens)

Preprocess the fetched tweets

processed_tweets = [preprocess_text(tweet) for tweet in tweets]

print(processed_tweets[:5]) # Show first 5 processed tweets

python

from sklearn.feature_extraction.text import TfidfVectorizer

Initialize TF-IDF vectorizer

vectorizer = TfidfVectorizer(max_features=5000)

Convert text data to numeric vectors (TF-IDF representation)

X = vectorizer.fit_transform(processed_tweets).toarray()

print(X.shape) # Print the shape of the matrix

``

python

import random

Generate random sentiment labels for the tweets (1 for positive, 0 for negative)

In practice, you should have actual sentiment labels

y = [random.choice([0, 1]) for _ in range(len(processed_tweets))]

Print first few labels

print(y[:5])

Step 6: Train the Model

python

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, accuracy_score

Split the data into training and test sets

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

Initialize RandomForest model

model = RandomForestClassifier(n_estimators=100, random_state=42)

Train the model

model.fit(X_train, y_train)

Make predictions on the test set

y_pred = model.predict(X_test)

Evaluate the model

print("Accuracy: ", accuracy_score(y_test, y_pred))

print("Classification Report: \n", classification_report(y_test, y_pred))

python

Example of a new tweet to analyze

new_tweet = "Bitcoin is the future of currency!"

Preprocess the new tweet

processed_new_tweet = preprocess_text(new_tweet)

Convert to vectorized format

vectorized_new_tweet = vectorizer.transform([processed_new_tweet]).toarray()

Predict sentiment (0: Negative, 1: Positive)

prediction = model.predict(vectorized_new_tweet)

```
if prediction == 1:
    print("Sentiment: Positive")
else:
    print("Sentiment: Negative")
```

```python
import matplotlib.pyplot as plt

Plot sentiment distribution
sentiment_counts = {0: y.count(0), 1: y.count(1)}
plt.bar(sentiment_counts.keys(), sentiment_counts.values(), color=['red', 'green'])
plt.title("Sentiment Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.xticks([0, 1], ['Negative', 'Positive'])
plt.show()
```

TASK 2 CREDIT SCORING MODEL

Build a credit scoring model to predict the credtiworthiness of applicants.Use features like income debet and credit histroy.

python

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report

import matplotlib.pyplot as plt

Create a synthetic dataset

data = {

'Income': [50000, 60000, 45000, 80000, 30000, 55000, 70000, 40000, 85000, 120000],

'Debt': [2000, 5000, 8000, 1000, 15000, 3000, 2000, 7000, 1000, 500],

'Credit_History': [1, 1, 0, 1, 0, 1, 1, 0, 1, 1], # 1: Good Credit History, 0: Bad Credit History

'Creditworthy': [1, 1, 0, 1, 0, 1, 1, 0, 1, 1] # 1: Creditworthy, 0: Not Creditworthy

}

df = pd.DataFrame(data)

Show the first few rows of the dataset

print(df.head())

The dataset looks like this:

Income	Debt	Credit_History	Creditworthy
50000	2000	1	1

60000	5000	1	1
45000	8000	0	0
80000	1000	1	1
30000	15000	0	0

Step 3: Preprocessing the Data

We will split the data into features (X) and target (y), then scale the features using StandardScaler for better model performance.

```python
Split the data into features (X) and target (y)
X = df[['Income', 'Debt', 'Credit_History']]  # Features
y = df['Creditworthy']  # Target
Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Scale the features for better performance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 4: Train the Model

We'll use a Random Forest Classifier to train the model. You can also try other models like Logistic Regression or Support Vector Machine (SVM).

python

Initialize the Random Forest model

model = RandomForestClassifier(n_estimators=100, random_state=42)

Train the model on the training data

model.fit(X_train, y_train)

Predict on the test data

y_pred = model.predict(X_test)

Step 5: Evaluate the Model

Now we will evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

python

Evaluate the model

print("Accuracy: ", accuracy_score(y_test, y_pred))

print("Classification Report: \n", classification_report(y_test, y_pred))

Confusion Matrix (Optional)

from sklearn.metrics import confusion_matrix

import seaborn as sns

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Creditworthy', 'Creditworthy'], yticklabels=['Not Creditworthy', 'Creditworthy'])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()

Step 6: Predict on New Data (Example)

Finally, you can use the trained model to predict the creditworthiness of new applicants.

python

Example of new applicants' data for prediction

new_applicants = pd.DataFrame({

'Income': [70000, 30000],

'Debt': [3000, 10000],

'Credit_History': [1, 0]

})

Scale the new applicants' data

new_applicants_scaled = scaler.transform(new_applicants)

```
Predict creditworthiness (0: Not Creditworthy, 1: Creditworthy)
predictions = model.predict(new_applicants_scaled)

Show the results
for i, prediction in enumerate(predictions):
    print(f"Applicant {i+1}: {'Creditworthy' if prediction == 1 else 'Not
Creditworthy'}")
```