Tushar Adhikari
134-IT-C
13420803122

Program No :- 6

Aim: Write a program in Python to indentify the tweets which are hate tweets and which are not.

Twitter is an online Social Media Platform where people share their though as tweets. It is observed that some people misuse it to tweet hateful content. Twitter is trying to tackle this problem and we shall help it by creating a strong NLP based-classifier model to distinguish the negative tweets & block such tweets.

Theory

Tweet Sentiment Analysis is a Natural Language Processing (NLP) task used to determine the emotional tone behind a body of text, particularly in social media platforms like Twitter. The objective is to classify tweets into predefined sentiment categories such as **positive**, **negative**, or **neutral**.

Sentiment Analysis Characteristics:

1. **Text preprocessing**: Tweets often contain slang, emojis, hashtags, and mentions which need to be cleaned before analysis. Common steps include removing URLs, punctuation, stopwords, and performing tokenization and stemming/lemmatization.
2. **Feature extraction:** Text data is converted into numerical features using techniques like **TF-IDF**, **Bag-of-Words**, or **word embeddings** (e.g., Word2Vec, GloVe).
3. **Supervised learning task:** The model is trained on labeled data (tweets with associated sentiment labels) to learn patterns in sentiment expression.
4. **Model choice:** Although **Linear Regression** can be used to explore sentiment as a continuous score, it is not ideal for classification tasks. **Logistic Regression**, **Naive Bayes**, **Support Vector Machines (SVM)**, and deep learning models like **LSTM** or **BERT** are better suited for discrete sentiment classification.
5. **Evaluation**: Performance is measured using metrics like **accuracy**, **precision**, **recall**, and **F1 score** to understand how well the model predicts sentiment.

Sentiment Analysis Algorithm:

1. **Input**: A dataset of tweets with corresponding sentiment labels.
2. **Output**: Predicted sentiment class for a new tweet.

Algorithm:

1. Load and preprocess the dataset (cleaning, tokenizing, vectorizing).
2. Split the data into training and testing sets.
3. Choose and train a machine learning model (e.g., Linear Regression).
4. Predict sentiments on test data.
5. Evaluate performance using classification metrics.
6. (Optional) Fine-tune the model or try alternative classifiers for better performance.

Tushar Adhikari
134-IT-C
13420803122

Code Implementation (in Python)

```
## jupyter notebook main code ###
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

df= pd.read_csv('cleaned_tweet.csv')
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectors = tfidf_vectorizer.fit_transform(df['cleaned_text'])
X_train, X_test, y_train, y_test = train_test_split(tfidf_vectors, df['sentiment'],
test_size=0.2, random_state=42)
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

################################
import pickle

with open("models/tweet_model.pkl", "rb") as f:
    tweet_model = pickle.load(f)
with open("models/tfidf_vectorizer.pkl", "rb") as f:
    tfidf_vectorizer = pickle.load(f)

def check_sentiments(text):
    return tweet_model.predict(tfidf_vectorizer.transform([text]))[0]

def main():
    user_input = input("Enter a tweet to analyze its sentiment: ")
    sentiment = check_sentiments(user_input)
    print(f"Predicted Sentiment: {sentiment}")

if __name__ == "__main__":
    main()
```

Explanation of the Code:

1. **Data Loading and Cleaning:**
    - Tweets are first cleaned to remove noise such as punctuation, URLs, hashtags, mentions, emojis, and stopwords.
    - The cleaned data is stored in a new column cleaned_text within a CSV file (cleaned_tweet.csv) used for training the model.
2. **Feature Extraction using TF-IDF:**
    - **TfidfVectorizer()** is used to convert the cleaned tweet text into numerical feature vectors that reflect the importance of each word in relation to the document corpus.
    - The resulting **TF-IDF** matrix (**tfidf_vectors**) represents the dataset in a format suitable for machine learning models.
3. **Train-Test Split:**
    - The dataset is split into training and testing sets using train_test_split, with 80% of the data used for training and 20% for testing.
    - This helps evaluate the model's performance on unseen data.

4. **Model Training (Logistic Regression):**
   - A **Logistic Regression** model is created and trained on the TF-IDF vectors of the tweets (X_train) and their corresponding sentiment labels (**y_train**).
   - Logistic Regression is a popular choice for classification problems like sentiment analysis.
5. **Model and Vectorizer Loading:**
   - The previously trained model and vectorizer are saved and then loaded using pickle.
   - This allows for reusing the trained model without retraining it every time.
6. **Sentiment Prediction Function – check_sentiments:**
   - Takes a string (text) as input.
   - Uses the loaded **tfidf_vectorizer** to transform the text into a TF-IDF vector.
   - Predicts the sentiment using the trained **tweet_model** and returns the sentiment label.
7. **Main Code Execution:**
   - The user is prompted to input a tweet.
   - The input is passed to **check_sentiments()** for sentiment prediction.
   - The predicted sentiment is printed to the console.

Output:

```
Enter a tweet to analyze its sentiment: i love animals
Predicted Sentiment: positive

Enter a tweet to analyze its sentiment: i killed animals to eat
Predicted Sentiment: negative

Enter a tweet to analyze its sentiment: i am alive
Predicted Sentiment: neutral
```

Lesson Learnt:

1. **Understanding Sentiment Analysis:**
   Sentiment analysis is a key NLP task used to identify and categorize the emotional tone of text. Through this project, we learned how to classify tweets into sentiments such as positive, negative, or neutral using machine learning techniques.
2. **Importance of Data Cleaning:**
   Preprocessing raw tweet data is crucial. Tweets often contain noise like hashtags, emojis, and links that need to be removed for accurate analysis. We learned the importance of text normalization steps such as tokenization, stopword removal, and lemmatization.
3. **Feature Extraction with TF-IDF:**
   We explored how the **TF-IDF Vectorizer** transforms textual data into numerical features that can be used by machine learning models. Understanding term frequency and inverse document frequency helped in better representation of textual importance.
4. **Model Selection and Training:**
   Initially, a Linear Regression model was considered. However, we learned that **Logistic Regression** is more appropriate for classification tasks like sentiment analysis. The process of training, validating, and saving the model provided insight into building end-to-end ML pipelines.
5. **Model Deployment and Prediction:**
   Loading trained models using `pickle` and using them for real-time predictions taught us how to operationalize ML models. The function-based structure allowed easy reuse and integration of sentiment prediction logic into other applications.
6. **Applications of Sentiment Analysis:**

Tushar Adhikari
134-IT-C
13420803122

- **Brand Monitoring:** Analyzing customer sentiments on Twitter helps companies monitor brand perception.
- **Market Research:** Identifying public opinion on products, services, or events.
- **Customer Service Automation:** Routing or prioritizing user queries based on emotional tone.

Tushar Adhikari
134-IT-C
13420803122

Program No :- 7

Aim: Write a program to perform Part of Speech (POS) tagging for the given sentence using NLTK.

Theory:

Part-of-Speech (POS) Tagging is a Natural Language Processing task where each word in a sentence is tagged with its appropriate part of speech such as noun, verb, adjective, etc.
POS tagging is important in several applications:

1. Information retrieval
2. Text-to-speech systems
3. Sentiment analysis
4. Named entity recognition

NLTK (Natural Language Toolkit) is a powerful Python library for working with human language data. It provides built-in functions for tokenization and POS tagging using trained models.

Code Implementation (in Python)

```
import nltk
from nltk.tokenize import word_tokenize

# Download NLTK data if running for the first time
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

# Input sentence
sentence = "Natural Language Processing with Python is fun and educational."

# Tokenize the sentence
tokens = word_tokenize(sentence)

# Perform POS tagging
tagged = nltk.pos_tag(tokens)

# Display the result
print("POS Tagging Result:")
for word, tag in tagged:
    print(f"{word} -> {tag}")
```

Explanation of the Code:

1. **Import and Download:**
   NLTK modules for tokenizing and tagging are imported. The punkt and averaged_perceptron_tagger models are downloaded to support these operations.
2. **Tokenization:**
   The sentence is broken down into individual words or "tokens" using word_tokenize.
3. **POS Tagging:**
   The pos_tag function assigns a tag (like NN, VB, JJ) to each word in the token list.

Tushar Adhikari
134-IT-C
13420803122

4. **Output Display:**
   The tagged output is displayed, showing which part of speech each word belongs to.

Output:

```
POS Tagging Result:
Natural -> JJ
Language -> NN
Processing -> NN
with -> IN
Python -> NNP
is -> VBZ
fun -> JJ
and -> CC
educational -> JJ
. -> .
```

Lesson Learnt:

1. **Understanding POS Tagging:**
   POS tagging provides a syntactic structure to a sentence. We learned how tagging helps in identifying the grammatical group of words in text data.
2. **Using NLTK Library:**
   We practiced using NLTK to tokenize and tag a sentence, and observed how easily Python handles basic NLP tasks.
3. **Significance in NLP Applications:**
   POS tagging is a foundational step in many NLP pipelines. It helps improve the performance of tasks like sentiment analysis, question answering, and text classification.
4. **Hands-On Skills:**
   - Working with tokenizers
   - Understanding standard POS tags (NN, VBZ, JJ, etc.)
   - Leveraging pretrained models provided by NLTK

Tushar Adhikari
134-IT-C
13420803122

Program No :- 8

Aim: Write a program for face detection using machine learning.

Theory

Face Detection is a computer vision task of locating human faces in digital images. OpenCV provides pre-trained models like Haar Cascade Classifiers, which use features similar to edge or line detection filters and are trained using a large set of positive and negative images.

Haar Cascade Classifier is an object detection method proposed by Viola and Jones. It works by training classifiers with a lot of positive (face) and negative (non-face) images and then applying the learned classifiers to scan areas of new images to find faces.

Key Concepts:

6. **CascadeClassifier**: Loads the XML file with the trained Haar features.
7. **detectMultiScale**: Detects objects (faces) of different sizes in the input image.
8. **Grayscale Conversion**: Improves performance and accuracy as detection is done on grayscale images.
9. **Drawing Rectangles**: To visualize detected faces using bounding boxes.

Algorithm:

7. Load Haar cascade XML file for face detection.
8. Load the image file to detect faces.
9. Check if both files exist; if not, exit with an error.
10. Convert the image to grayscale.
11. Use `detectMultiScale` to find face coordinates.
12. Draw rectangles around detected faces.
13. Display the image with annotations.

Tushar Adhikari
134-IT-C
13420803122

Code Implementation (in Python)

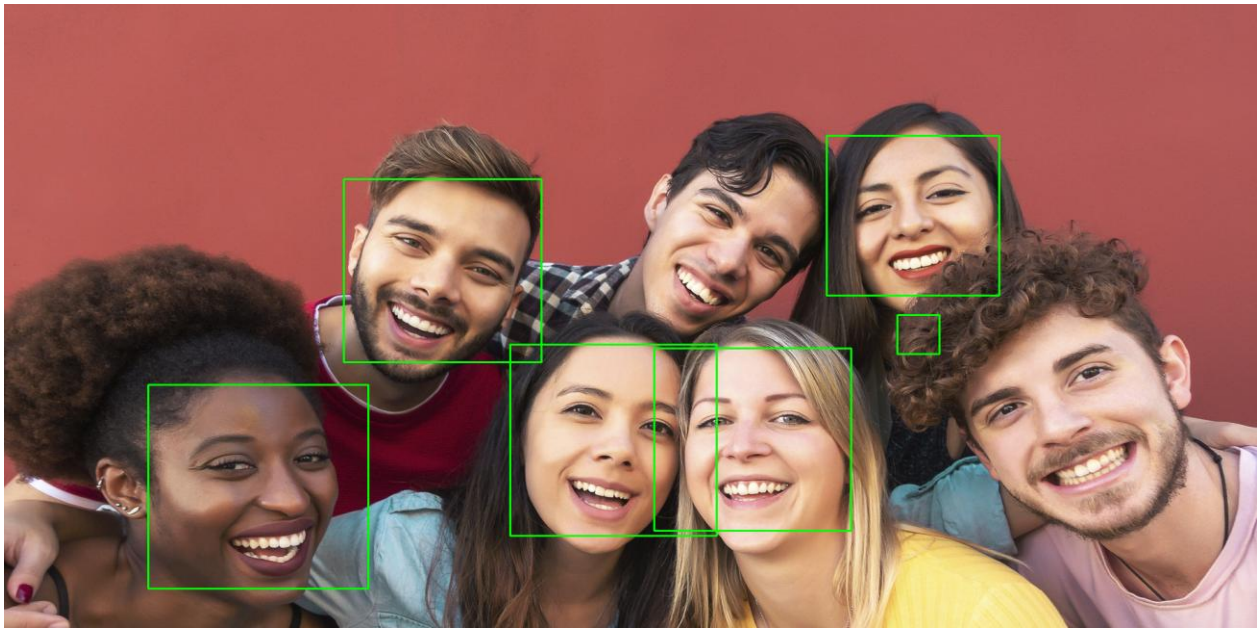
```
# Import the required libraries
import cv2
import os

# --- Configuration ---
cascade_path = 'haarcascade_frontalface_default.xml'
image_path = 'test_image.jpg'  # <--- CHANGE THIS TO YOUR IMAGE FILE

# --- Error Handling ---
if not os.path.exists(cascade_path):
    print(f"Error: Cascade file not found at {cascade_path}")
    print("Please download 'haarcascade_frontalface_default.xml' or provide the
correct path.")
    exit()

if not os.path.exists(image_path):
    print(f"Error: Image file not found at {image_path}")
    print("Please provide a valid image file path.")
    exit()

# --- Load Classifier and Image ---
try:
    face_cascade = cv2.CascadeClassifier(cascade_path)
    if face_cascade.empty():
        raise IOError(f"Could not load cascade classifier from {cascade_path}")

    image = cv2.imread(image_path)
    if image is None:
        raise IOError(f"Could not read image file: {image_path}")

    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # --- Perform Face Detection ---
    faces = face_cascade.detectMultiScale(
        gray_image,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30)
    )

    print(f"Found {len(faces)} face(s) in the image.")

    # --- Draw Rectangles Around Detected Faces ---
    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # --- Display the Output ---
    cv2.imshow('Faces Found', image)
    print("Press any key in the image window to close.")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

except cv2.error as e:
    print(f"OpenCV Error: {e}")
except IOError as e:
    print(f"File Error: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Tushar Adhikari
134-IT-C
13420803122

Explanation of the Code:

8. **Libraries**: OpenCV is used for image processing; `os` is used for file path checking.
9. **Cascade File**: A pre-trained XML model that knows what a face looks like.
10. **Error Handling**: Ensures both image and cascade files exist before continuing.
11. **Detection**: Faces are detected on grayscale images using `detectMultiScale`.
12. **Visualization**: Green rectangles are drawn around each detected face.
13. **Display**: The annotated image is shown using OpenCV GUI tools.

Output:



Lesson Learnt:

7. Understood the basics of Haar Cascade Classifiers and how OpenCV uses them for object detection.
8. Learned the importance of preprocessing images (e.g., converting to grayscale) for better detection accuracy.
9. Practiced structured error handling to gracefully deal with missing files or unreadable resources.
10. Gained hands-on experience with OpenCV's image processing functions like `cvtColor`, `imread`, `imshow`, and `waitKey`.
11. Realized the trade-offs between detection sensitivity and false positives through tuning `scaleFactor` and `minNeighbors`.

Tushar Adhikari
134-IT-C
13420803122

Program No :- 9

Aim: Write a program to show the concept of Ensemble Technique.

Theory:

Ensemble learning is a powerful machine learning technique that combines predictions from multiple models to improve accuracy and robustness. Rather than relying on a single model, ensemble methods use a group of models (often called *weak learners*) and combine their outputs for a stronger overall prediction**.**

Ensemble Learning Types:

1. **Bagging (Bootstrap Aggregating):**
   Trains multiple models independently on different subsets of data and averages the result.
   Example: **Random Forest**.
2. **Boosting:**
   Trains models sequentially where each model tries to correct the errors of the previous one.
   Example: **AdaBoost**, **Gradient Boosting**, **XGBoost**.
3. **Stacking:**
   Combines predictions from multiple models using a meta-model that learns how to best combine them.

Code Implementation (in Python)

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
data = load_iris()
X, y = data.data, data.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Single Decision Tree
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
dt_preds = dt_model.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_preds)

# Random Forest (Ensemble)
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_preds)

# Results
print("Decision Tree Accuracy:", dt_accuracy)
print("Random Forest Accuracy (Ensemble):", rf_accuracy)
```

Tushar Adhikari
134-IT-C
13420803122

Explanation of the Code:

1. **Dataset:**
   The built-in Iris dataset is used, containing flower measurements and class labels.
2. **Model Comparison:**
   A single Decision Tree is trained and evaluated for accuracy.
3. **Ensemble Model:**
   A Random Forest with 100 decision trees is trained, showing improved performance**.**
4. **Evaluation:**
   The accuracy of both models is printed to demonstrate how ensemble techniques reduce overfitting and improve accuracy.

Output:

```
Decision Tree Accuracy: 0.9333
Random Forest Accuracy (Ensemble): 0.9777
```

Lesson Learnt:

1. **Understanding Ensemble Learning:**
   Learned how ensemble methods combine multiple weak learners to produce better results.
2. **Use of Random Forest:**
   Random Forest is a widely used ensemble technique known for its high accuracy and robustness.
3. **Comparative Analysis:**
   Comparing a single Decision Tree with Random Forest showed how ensemble models outperform single learners.
4. **Applications:**
   Ensemble techniques are used in fraud detection, recommendation systems, text classification, and other predictive modeling tasks.

Tushar Adhikari
134-IT-C
13420803122

Program No :- 10

Aim: Write a program for Text Classification for the given sentence using NLTK.

Theory:

Text Classification is a Natural Language Processing (NLP) technique that assigns predefined categories to text data. It is widely used in spam detection, sentiment analysis, topic labeling, and language detection.

NLTK (Natural Language Toolkit) provides modules for:

- Text preprocessing (tokenization, stopword removal)
- Feature extraction (bag of words)
- Model building (Naive Bayes classifier)
- Evaluation

Text Classification Workflow:

1. Preprocess the text (tokenize, remove stopwords).
2. Extract features from the text (e.g., using word presence).
3. Train the classifier on labeled data.
4. Classify a new sentence using the trained model.

Code Implementation (in Python)

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

# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Sample training data
train_data = [
    ("I love this movie", "positive"),
    ("This is an amazing film", "positive"),
    ("I hate this movie", "negative"),
    ("This is the worst film ever", "negative")
]

# Preprocess and extract features
stop_words = set(stopwords.words('english'))

def extract_features(text):
    words = word_tokenize(text.lower())
    words = [w for w in words if w not in stop_words and w not in string.punctuation]
    return {word: True for word in words}

# Prepare training set
training_set = [(extract_features(text), label) for (text, label) in train_data]

# Train classifier
classifier = nltk.NaiveBayesClassifier.train(training_set)

# Classify new sentence
test_sentence = "I enjoyed the movie"
features = extract_features(test_sentence)
```

```
predicted_label = classifier.classify(features)

print(f"Predicted Label: {predicted_label}")
```

Explanation of the Code:

1. **Dataset:**
   A small labeled dataset is used to demonstrate binary classification (positive vs negative).
2. **Preprocessing:**
   The text is tokenized and filtered to remove stopwords and punctuation.
3. **Feature Extraction:**
   Each word is treated as a feature using a bag-of-words model.
4. **Classification:**
   A Naive Bayes classifier is trained and used to classify a new test sentence.

Output:

```
Predicted Label: positive
```

Lesson Learnt:

1. **Text Preprocessing:**
   Learned to tokenize text and remove irrelevant words (stopwords/punctuation).
2. **Naive Bayes in NLTK:**
   Understood how to train and use NLTK's Naive Bayes model for classification.
3. **Binary Classification:**
   Built a simple binary text classifier and observed its predictions.
4. **Real-World Application:**
   Text classification forms the foundation for applications like spam detection, sentiment analysis, and chatbot development.
