# SENTIMENT ANALYSIS OF MOVIE REVIEWS

## CODE….

import nltk import random from nltk.corpus import movie_reviews, stopwords from nltk.classify import NaiveBayesClassifier from nltk.classify.util import accuracy from nltk.stem import WordNetLemmatizer from nltk.tokenize import RegexpTokenizer import matplotlib.pyplot as plt from io import BytesIO import base64 from sklearn.metrics import confusion_matrix import seaborn as sns from wordcloud import WordCloud # ✅ Download required resources nltk.download('movie_reviews') nltk.download('stopwords') nltk.download('wordnet') # ✅ Setup stop_words = set(stopwords.words("english")) lemmatizer = WordNetLemmatizer() tokenizer = RegexpTokenizer(r'\w+') # ✅ avoids 'punkt' error # ✅ Text preprocessing def preprocess(words): words = [w.lower() for w in words if w.isalpha()] words = [w for w in words if w not in stop_words] words = [lemmatizer.lemmatize(w) for w in words] return words # ✅ Load and preprocess the dataset documents = [(preprocess(movie_reviews.words(fileid)), category) for category in movie_reviews.categories() for fileid in movie_reviews.fileids(category)] random.shuffle(documents) # ✅ Create word features all_words = nltk.FreqDist(word for doc, _ in documents for word in doc) word_features = list(all_words)[:2000] def document_features(document): words = set(document) return {f'contains({word})': (word in words) for word in word_features} # ✅ Feature sets featuresets = [(document_features(doc), category) for (doc, category) in documents] # ✅ Train and test split train_set = featuresets[100:] test_set = featuresets[:100] # ✅ Train classifier classifier = NaiveBayesClassifier.train(train_set) # ✅ Accuracy and top features print("="*60) print("🎯 Sentiment Analysis on Movie Reviews") print("="*60) print(f"✅ Model Accuracy: {accuracy(classifier, test_set) * 100:.2f}%\n") print("📌 Top Informative Features:") classifier.show_most_informative_features(10) # ✅ Custom input for prediction print("\n📝 Test on Custom Review") print("="*60) sample = input("👉 Enter your movie review: ") # ✅ Predict sentiment tokens = tokenizer.tokenize(sample) cleaned = preprocess(tokens) features = document_features(cleaned) prediction = classifier.classify(features) # ✅ Output result print(f"\n🔍 Predicted Sentiment: {'Positive 😊' if prediction == 'pos' else 'Negative 😞'}") # ✅ Graphs showing overall sentiment stats (positive vs negative) def show_sentiment_graphs(): pos_count = 120 # Simulate 120 positive reviews neg_count = 80 # Simulate 80 negative reviews # Pie chart labels = ['Positive', 'Negative'] values = [pos_count, neg_count] colors = ['#2ecc71', '#e74c3c'] plt.figure(figsize=(6, 6)) plt.pie(values, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140) plt.title("Sentiment Distribution (Pie Chart)") plt.show() # Bar chart 1: Positive sentiment over time (simulated) months = ['Jan', 'Feb', 'Mar', 'Apr', 'May'] pos_counts = [30, 25, 40, 60, 120] # Simulated data plt.figure(figsize=(8, 6)) plt.bar(months, pos_counts, color='#2ecc71') plt.title('Positive

Sentiment Counts Over Time') plt.xlabel('Month') plt.ylabel('Count') plt.show() # Bar chart 2: Negative sentiment over time (simulated) neg_counts = [20, 25, 30, 50, 80] # Simulated data plt.figure(figsize=(8, 6)) plt.bar(months, neg_counts, color='#e74c3c') plt.title('Negative Sentiment Counts Over Time') plt.xlabel('Month') plt.ylabel('Count') plt.show() # Call the function to display graphs show_sentiment_graphs() # ✅ Word Cloud for Positive and Negative Reviews def generate_word_cloud(): pos_reviews = [doc for doc, category in documents if category == 'pos'] neg_reviews = [doc for doc, category in documents if category == 'neg'] pos_words = [word for review in pos_reviews for word in review] neg_words = [word for review in neg_reviews for word in review] # Positive Word Cloud pos_wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(pos_words)) plt.figure(figsize=(8, 6)) plt.imshow(pos_wordcloud, interpolation='bilinear') plt.title("Positive Review Word Cloud") plt.axis('off') plt.show() # Negative Word Cloud neg_wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(neg_words)) plt.figure(figsize=(8, 6)) plt.imshow(neg_wordcloud, interpolation='bilinear') plt.title("Negative Review Word Cloud") plt.axis('off') plt.show() # Generate Word Clouds generate_word_cloud() # ✅ Confusion Matrix for Classifier Performance def show_confusion_matrix(): # Test data: Predictions and actual results actual = [category for _, category in test_set] predicted = [classifier.classify(features) for features, _ in test_set] # Create confusion matrix cm = confusion_matrix(actual, predicted, labels=['pos', 'neg']) # Plot Confusion Matrix using seaborn heatmap plt.figure(figsize=(6, 6)) sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Positive', 'Negative'], yticklabels=['Positive', 'Negative']) plt.title("Confusion Matrix") plt.xlabel("Predicted") plt.ylabel("Actual") plt.show() # Show Confusion Matrix show_confusion_matrix() # The movie was fantastic! The plot was engaging and the acting was great. #This movie was terrible. The plot was confusing and the characters were dull. #I absolutely loved the storyline and the acting was brilliant!