СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

КУРСОВ ПРОЕКТ
ПО СИСТЕМИ, ОСНОВАНИ НА ЗНАНИЯ

Тема:

Класификация на настроенията в социалните мрежи

Студенти:

Людмила Пулова, фак. №: 1MI0700175
Десислава Добрева, фак. №: 0MI0700129

София, януари 2025 г.

Дисциплина „Системи, основани на знания“
Курсов проект на тема „Класификация на настроенията в социалните мрежи“

Съдържание:

1. Формулировка на задачата

Целта на проекта е да се създаде модел за класификация на настроенията в публикации от социални мрежи. Текстовете трябва да бъдат анализирани и категоризирани като положителни, отрицателни или неутрални. Използван е набор от данни Sentiment Dataset от платформата Kaggle. Освен това, е разработено приложение с графичен интерфейс, което позволява въвеждане на текст и получаване на моментална класификация.

2. Използвани алгоритми

За задачата са използвани следните алгоритми за класификация:

Naive Bayes

Алгоритъмът Naive Bayes се основава на теоремата на Байес с предположението за условна независимост между характеристиките. Това означава, че всяка дума в текста се разглежда като независима от останалите, което значително улеснява изчисленията. Този метод е бърз и ефективен при текстова класификация, но може да бъде по-малко точен при сложни зависимости между думите.

Основни предимства:

- Лесен за имплементация и много бърз.
- Добре се справя с малки и средно големи набори от данни.

Ограничения:

- Предположението за независимост често не отразява реалността.

Логистична регресия

Логистичната регресия е статистически модел, използван за класификация на бинарни и многокласови данни. Тя изчислява вероятността дадена проба да принадлежи към определен клас, като използва логистична функция. При текстова класификация, логистичната регресия работи добре в комбинация с TF-IDF векторизация.

Основни предимства:

- Добра производителност при линейно разделими данни.
- Подходяща за големи набори от данни.

Ограничения:

- Ограничена е в ситуации, когато данните не са линейно разделими.

SVM (Support Vector Machine)

SVM е мощен алгоритъм, който се стреми да намери оптимална хиперплоскост, разделяща различните класове. Използва се линейно ядро за задачата, което е подходящо за текстова класификация. SVM е известен със своята способност да се справя добре с високоизмерни данни.

Основни предимства:

- Много точен при подходящо избрани параметри.
- Ефективен при голям брой характеристики (например текст).

Ограничения:

- По-бавен за трениране, особено при големи набори от данни.

Random Forest

Random Forest е ансамблов метод, който комбинира множество решаващи дървета за вземане на решение. Всеки клас се предсказва от отделно дърво, а финалната класификация се базира на гласуване. Методът е устойчив на overfitting ("свръхнаучаване") и добре се справя с неструктурирани данни.

Основни предимства:

- Висока устойчивост на шум и отклонения.
- Подходящ за сложни задачи с множество характеристики.

Ограничения:

- Може да бъде бавен при големи набори от данни

Общ подход към използването на алгоритмите:

- **Предварителна обработка на текста:** Включва почистване от ненужни символи, премахване на стоп думи и използване на TF-IDF за векторизация.
- **Обучение на моделите:** Всеки от моделите е обучен върху един и същ тренировъчен набор, за да се осигури справедливо сравнение.

- **Оценка на представянето:** За сравнение на моделите са използвани метрики като точност (accuracy), F1-score и визуализация на матрици на объркване.

3. Описание на програмната реализация:

Програмата е разделена на два основни модула:

Модул за обучение на модели (sentiment_model_training.py) :

1. Зареждане и обработка на данни: Данните се зареждат от CSV файлове и преминават през обработка, която включва:

- Премахване на празни стойности.
- Почистване на текст чрез премахване на линкове, хаштагове и специални символи.

```
# Зареждане на данни
train_file = 'train.csv'
test_file = 'test.csv'

train_data = pd.read_csv(train_file, encoding='latin1')
test_data = pd.read_csv(test_file, encoding='latin1')

# Прочистване на данни
train_data.dropna(subset=['text', 'sentiment'], inplace=True)
test_data.dropna(subset=['text', 'sentiment'], inplace=True)

train_data['sentiment'] = train_data['sentiment'].map({'positive': 1,
'negative': -1, 'neutral': 0})
test_data['sentiment'] = test_data['sentiment'].map({'positive': 1,
'negative': -1, 'neutral': 0})

# Почистване на текст

def clean_text(text):
    text = re.sub(r"http\S+|www\S+|https\S+", '', text,
flags=re.MULTILINE)  # Премахване на линкове
    text = re.sub(r'\@\w+|\#', '', text)  # Премахване на хаштагове и
споменавания
    text = re.sub(r'[^\w\s]', '', text)  # Премахване на пунктуация
    text = re.sub(r'\s+', ' ', text)  # Премахване на допълнителни
интервали
    return text.strip().lower()
```

2. Разделяне на данни: Данните се разделят на тренировъчен и тестов сет за осигуряване на справедлива оценка на моделите.

```
train_data['text'] = train_data['text'].apply(clean_text)
test_data['text'] = test_data['text'].apply(clean_text)

# Разделяне на данните
X_train = train_data['text']
y_train = train_data['sentiment']
X_test = test_data['text']
y_test = test_data['sentiment']
```

3. Векторизация: Текстът се преобразува в числов формат с помощта на TF-IDF, който извлича важните характеристики на текста, като отчита честотата на думите.

```
# Векторизация
vectorizer = TfidfVectorizer(max_features=3000, ngram_range=(1, 2),
stop_words='english', sublinear_tf=True)
X_train_vect = vectorizer.fit_transform(X_train)
X_test_vect = vectorizer.transform(X_test)
```

4. Обучение на модели: Обучават се четири модела (Naive Bayes, Logistic Regression, SVM, Random Forest), които се сравняват по точност.

```
# Инициализация на модели
models = {
    "Naive Bayes": MultinomialNB(),
    "Logistic Regression": LogisticRegression(max_iter=1000,
class_weight='balanced'),
    "SVM": SVC(kernel='linear', class_weight='balanced', probability=True),
    "Random Forest": RandomForestClassifier(class_weight='balanced',
random_state=42, n_jobs=-1)
}

# Резултати от модели
results = {}
best_model = None
best_accuracy = 0

for name, model in models.items():
    print(f"\nОбучение и тестване на модел: {name}")
    model.fit(X_train_vect, y_train)
    predictions = model.predict(X_test_vect)
    accuracy = accuracy_score(y_test, predictions)
    results[name] = accuracy
```

```
    print(f"\n{name} Classification Report:\n")
    print(classification_report(y_test, predictions, zero_division=0))

    # Визуализация на матрицата на объркване
    ConfusionMatrixDisplay.from_predictions(
        y_test, predictions, display_labels=['Negative', 'Neutral',
'Positive']
    )
    plt.title(f"Confusion Matrix: {name}")  # Заглавие на графиката
    plt.gcf().canvas.manager.set_window_title(f"Confusion Matrix: {name}")  #
Заглавие на прозореца
    plt.show()

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = model

# Резултати от всички модели
print("\nРезултати от всички модели:")
for model_name, acc in results.items():
    print(f"{model_name}: {acc:.4f}")
```

6. **Запазване на модели:** Най-добрият модел и векторизаторът се запазват в файлове с помощта на библиотеката joblib.

```
# Запазване на най-добрия модел и векторизатора
print("\nЗапазване на най-добрия модел...")
model_path = os.path.join(save_directory, 'best_sentiment_model.pkl')
vectorizer_path = os.path.join(save_directory, 'vectorizer.pkl')

joblib.dump(best_model, model_path)
joblib.dump(vectorizer, vectorizer_path)
```

Модул за графичен интерфейс (sentiment_analyzer_app.py) :

- **Основен интерфейс:** Създаден с помощта на библиотеката tkinter, интерфейсът включва текстово поле за въвеждане на текст, бутон за анализ и етикет за показване на резултата.

```
# Създаване на главния прозорец
root = tk.Tk()
root.title("Класификация на настроенията")  # Заглавие на прозореца
root.resizable(False, False)  # Забраняване на разширяване и смаляване

# Стайлинг на интерфейса
style = ttk.Style()
style.configure('TFrame', background='#f0f0f0')  # Цвят на фона на рамката
```

```
style.configure('TButton', font=('Arial', 10), background='#e1e1e1',
width=20)  # Стил на бутона
style.configure('TLabel', background='#f0f0f0', font=('Arial', 12))  # Стил
на етикета
style.configure('TScrolledText', font=('Consolas', 10))  # Стил на
текстовото поле

# Създаване на рамка за въвеждане на текст
frame = ttk.Frame(root, padding=20)
frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))
frame.columnconfigure(0, weight=1)  # Указване, че колоната се разтяга
равномерно

# Текстово поле за въвеждане на текст
input_textbox = scrolledtext.ScrolledText(frame, width=60, height=10,
font=('Consolas', 12))
input_textbox.grid(row=0, column=0, columnspan=2, pady=10, padx=5)

# Бутон за предсказване на настроението
predict_button = ttk.Button(frame, text="Анализирай",
command=predict_sentiment)
predict_button.grid(row=1, column=0, pady=10, sticky=tk.N)  # Центриране на
бутона над етикета

# Етикет за показване на резултата
result_label = ttk.Label(frame, text="Настроение: Неизвестно")
result_label.grid(row=2, column=0, pady=10, sticky=tk.N)  # Подравняване на
етикета под бутона
```

- **Обработка на въведен текст:** Въведеният текст се обработва по същия начин, както при обучението на моделите – чрез почистване и векторизация.

```
# Дефиниция на функцията за почистване на текст
# Премахване на линкове, хаштагове и специални символи
def clean_text(text):
    text = re.sub(r"http\S+|www\S+|https\S+", '', text,
flags=re.MULTILINE)  # Премахване на URL адреси
    text = re.sub(r'\@\w+|\#', '', text)  # Премахване на хаштагове и @
    text = re.sub(r'[^\w\s]', '', text)  # Премахване на специални символи
    text = re.sub(r'\s+', ' ', text)  # Премахване на излишни интервали
    return text.strip().lower()  # Връщане на изчистен текст в малки букви
```

- **Класификация на настроението:** Зареденият модел предсказва настроението (положително, неутрално или отрицателно) и го визуализира в интерфейса.

```
# Функция за предсказване на настроението
def predict_sentiment():
```

```
    text = input_textbox.get("1.0", tk.END).strip()  # Вземане на въведения
текст
    if text:  # Проверка дали е въведен текст
        cleaned_text = clean_text(text)  # Почистване на текста
        vect_text = vectorizer.transform([cleaned_text])  # Векторизиране
на текста
        prediction = model.predict(vect_text)  # Предсказване на
настроението
        sentiment = {1: 'Положително', 0: 'Неутрално', -1:
'Отрицателно'}  # Карта на настроенията
        result_label.config(text=f"Настроение:
{sentiment[prediction[0]]}")  # Показване на резултата
    else:
        result_label.config(text="Моля, въведете текст.")  # Съобщение за
липсващ текст
```

Тази структура позволява лесно допълване и модифициране на системата при необходимост.

4. Примери, илюстриращи работата на програмната система

Примерни входове и изходи:

Кратки примери:

> Въведен текст: "I love this!"
> Изход: "Настроение: Положително"
>
> Въведен текст: "This is terrible..."
> Изход: "Настроение: Отрицателно"
>
> Въведен текст: "The sky is blue."
> Изход: "Настроение: Неутрално"

Примери под формата на публикации:

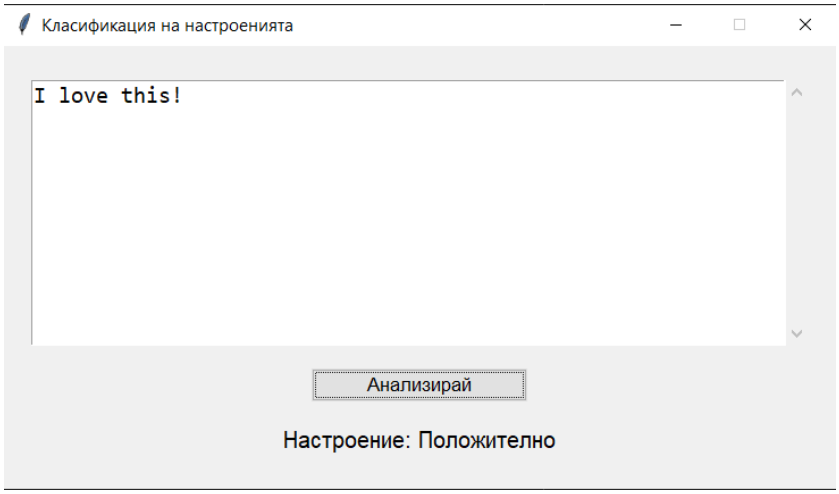
> Въведен текст: "The free fillin’ app on my iPod is so fun, I can’t stop playing it. I haven’t been this hooked on something in a long time. Highly recommend giving it a try if you need some entertainment!"
> Изход: "Настроение: Положеително"

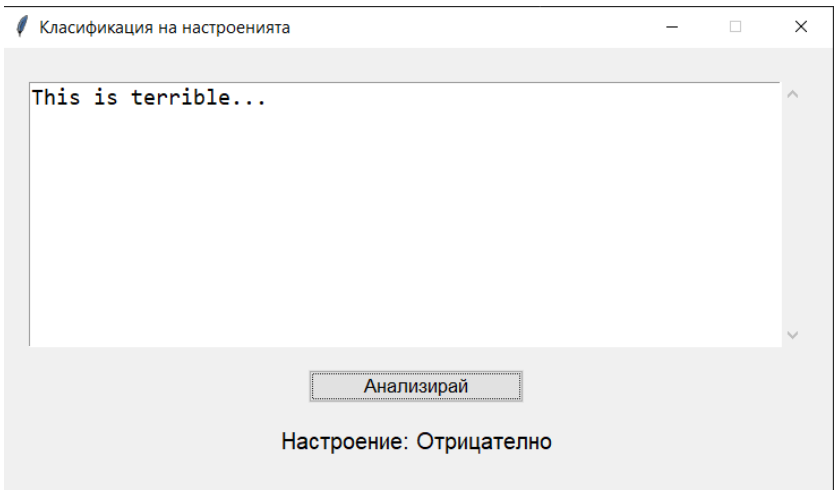
Въведен текст: "My Sharpie is running DANGERously low on ink. Just my luck! Every time I need something to work, it fails on me. Can’t anything go right for once?"
Изход: "Настроение: Отрицателно"

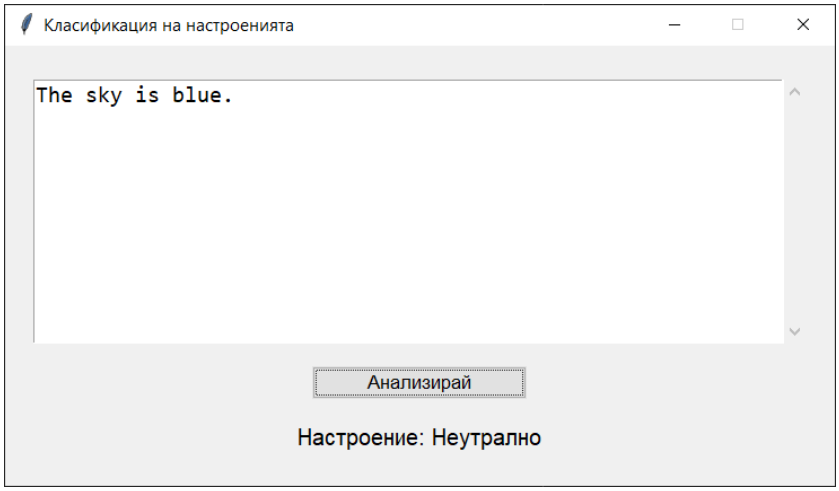
Въведен текст "I’d have responded, if I were going. Plans just didn’t work out this time. It’s okay, though; maybe next time something better will come up. For now, I’ll just stay in and relax."
Изход: "Настроение: Неутрално"

Снимки на графичния интерфейс:

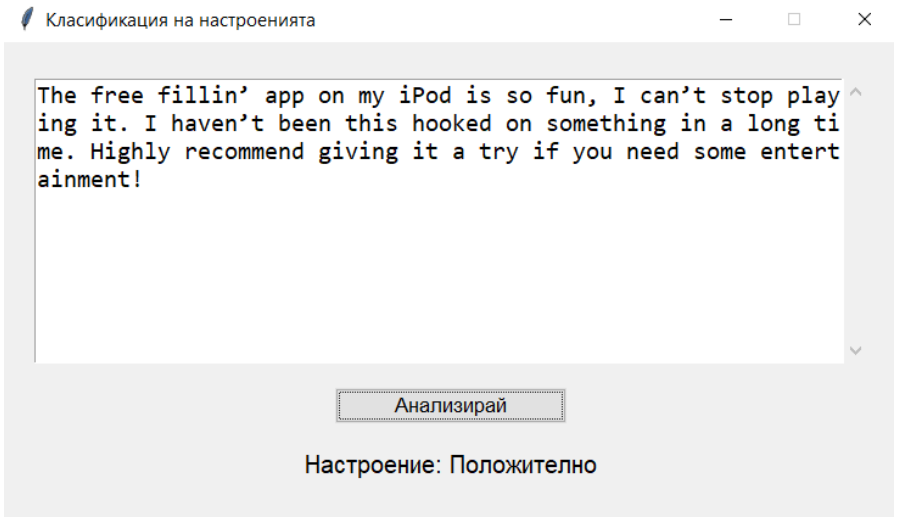
Кратки примери:

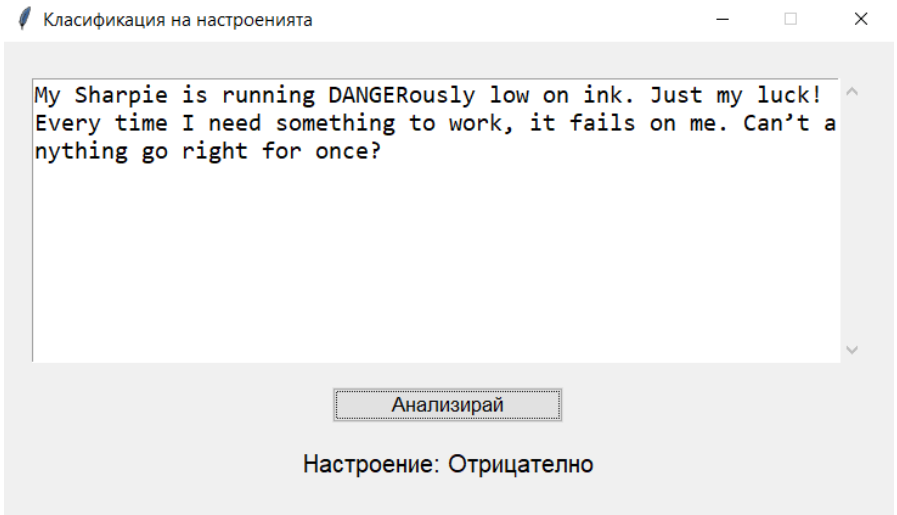


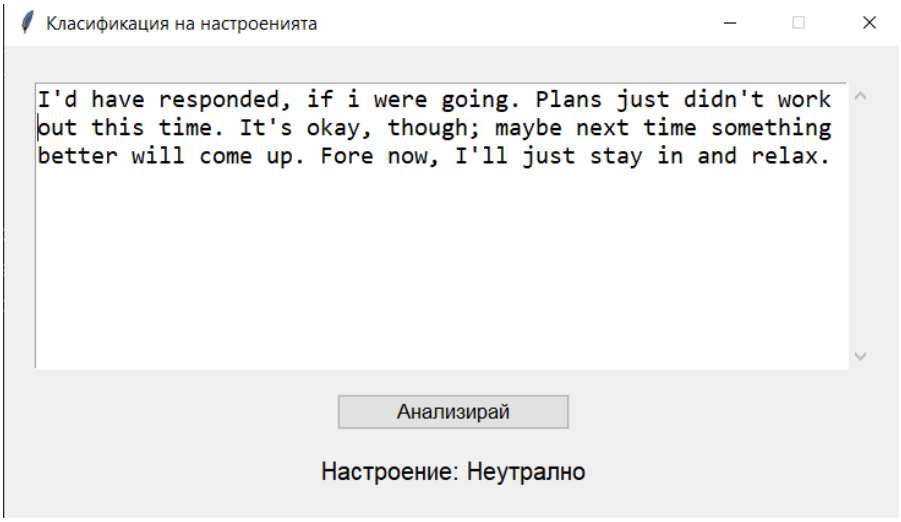




Примери под формата на публикации:







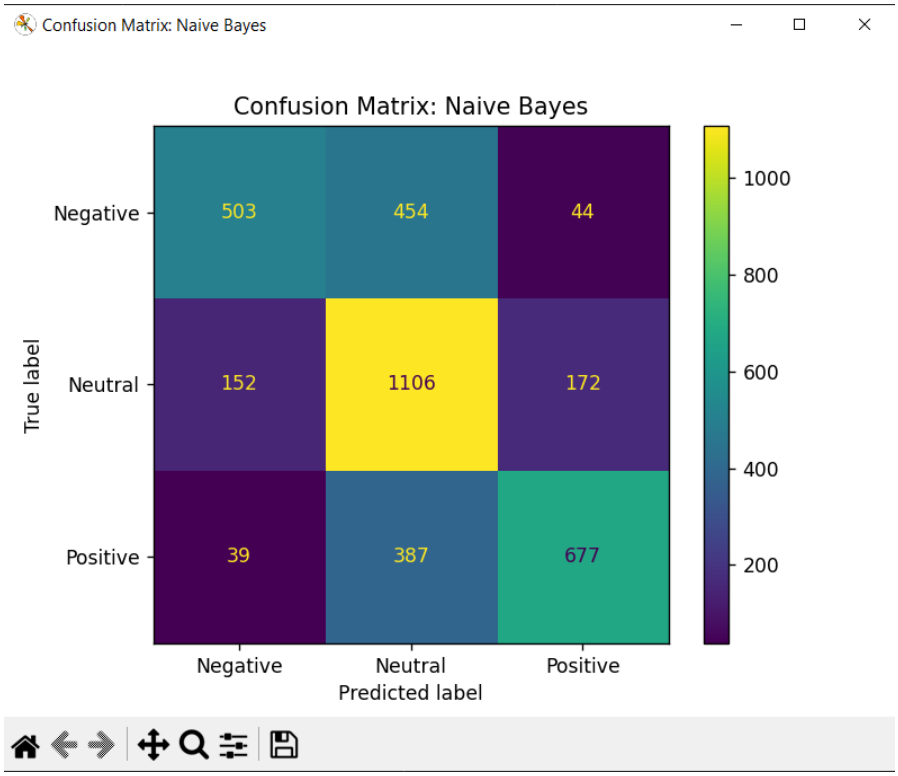
Резултати от обучението:

Confusion Matrix (Матрица на объркване) е инструмент за визуализация, който показва как моделът за класификация се справя с предсказването на класовете. Тя сравнява действителните стойности на класовете с предсказаните стойности.

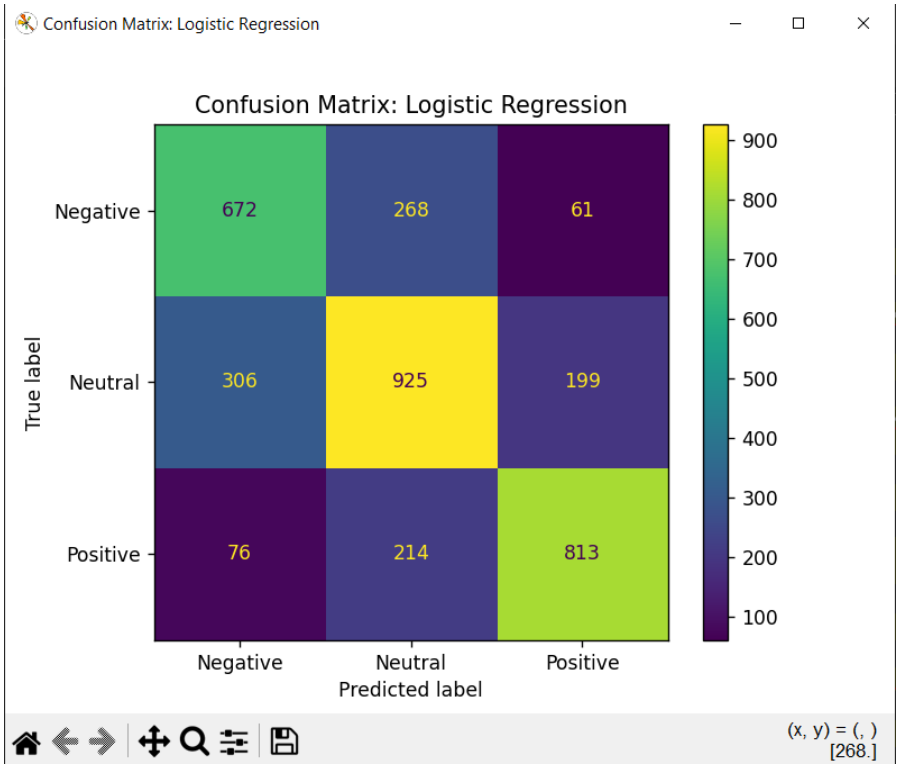
Тя е квадратна таблица, където:

- Редовете показват **действителните класове** (реални стойности).
- Колоните показват **предсказаните класове**.

Модел Naive Bayes

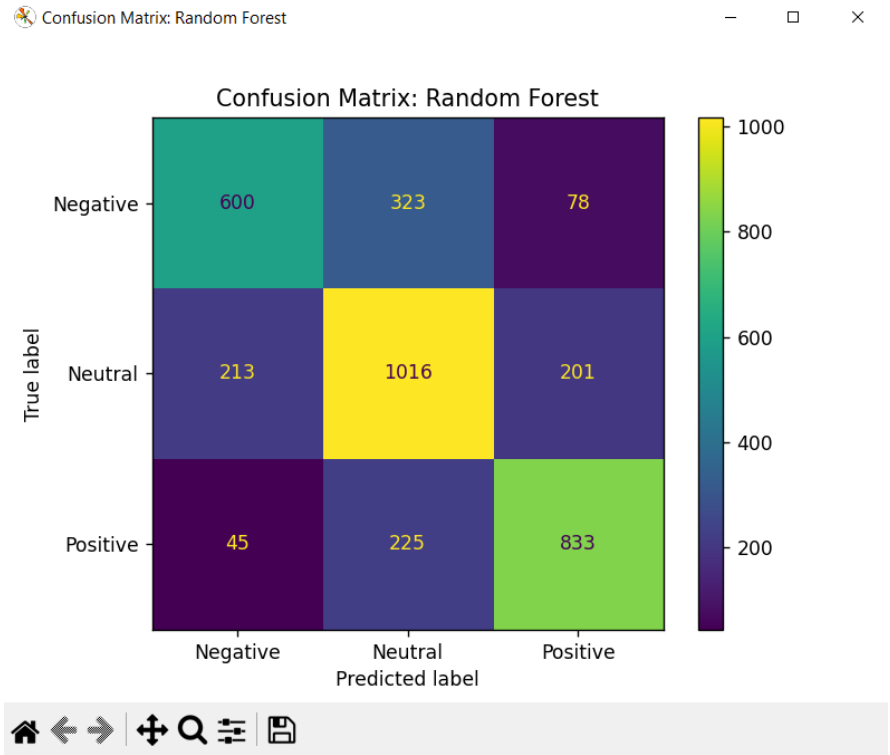


Модел Logistic Regression



Модел SVM

Модел Random Forest



Classification report (Доклад за класификация):

1. Precision (Точност):

Точността измерва какъв процент от класификациите, които моделът е предсказал като положителни (или конкретен клас), наистина са коректни.

2. Recall (Чувствителност):

Чувствителността измерва какъв процент от реално положителните примери (или друг клас) моделът успешно е идентифицирал. Показва какъв процент от всички истински стойности за даден клас са открити от модела.

3. F1-score:

F1-score е хармоничното средно между точността (Precision) и чувствителността (Recall). Тя балансира между тях, особено когато е важно да се отчитат както FP (грешно положителни), така и FN (грешно отрицателни). Полезно е, когато данните са небалансирани.

4. Support:

Support показва броя на реалните примери за всеки клас в набора за тест или валидация.

5. Accuracy (Точност на модела):

Точността измерва общия процент на коректните предсказания спрямо всички предсказания. Процент от всички предсказания, които са правилни.

6. Macro Average (macro avg):

Macro avg изчислява средната стойност на дадена метрика (например precision, recall или f1-score) за всички класове, като третира всички класове с еднаква тежест.

7. Weighted Average (weighted avg):

Weighted avg също изчислява средната стойност на дадена метрика, но отчита размера на всеки клас (брой примери в класа). Така по-големите класове оказват по-голямо влияние върху резултата.

Системата е обучена и тествана с четири различни модела за машинно обучение: Naive Bayes, Logistic Regression, SVM и Random Forest. Резултатите са сравнени чрез метрики като точност (accuracy), precision, recall, f1-score, и визуализиране на матрицата на объркване (confusion matrix).

Обучение и тестване:

Naive Bayes:

- Точност: 64.69%
- Силни страни: Простота и бързина.
- Ограничения: По-ниска точност при по-сложни текстове.

```
Обучение и тестване на модел: Naive Bayes

Naive Bayes Classification Report:

              precision    recall  f1-score   support

          -1       0.72      0.50      0.59      1001
           0       0.57      0.77      0.66      1430
           1       0.76      0.61      0.68      1103

    accuracy                           0.65      3534
   macro avg       0.68      0.63      0.64      3534
weighted avg       0.67      0.65      0.64      3534
```

Logistic Regression:

- Точност: 68.19%
- Силни страни: Балансираност между точност и скорост.
- Ограничения: По-ниска производителност при сложни класове.

```
Обучение и тестване на модел: Logistic Regression

Logistic Regression Classification Report:

              precision    recall  f1-score   support

          -1       0.64      0.67      0.65      1001
           0       0.66      0.65      0.65      1430
           1       0.76      0.74      0.75      1103

    accuracy                           0.68      3534
   macro avg       0.68      0.69      0.68      3534
weighted avg       0.68      0.68      0.68      3534
```

SVM (Support Vector Machines):

- Точност: 68.36%
- Силни страни: Висока точност при данни с ясна граница между класовете.
- Ограничения: По-бавно обучение при големи набори данни.

```
Обучение и тестване на модел: SVM

SVM Classification Report:

              precision    recall  f1-score   support

          -1       0.64      0.66      0.65      1001
           0       0.65      0.67      0.66      1430
           1       0.77      0.72      0.75      1103

    accuracy                           0.68      3534
   macro avg       0.69      0.68      0.69      3534
weighted avg       0.69      0.68      0.68      3534
```

Random Forest:

- Точност: 69.30%
- Силни страни: Най-висока точност и устойчивост на дисбалансирани данни.
- Ограничения: По-бавен процес на прогнозиране.

```
Обучение и тестване на модел: Random Forest

Random Forest Classification Report:

              precision    recall  f1-score   support

          -1       0.70      0.60      0.65      1001
           0       0.65      0.71      0.68      1430
           1       0.75      0.76      0.75      1103

    accuracy                           0.69      3534
   macro avg       0.70      0.69      0.69      3534
weighted avg       0.69      0.69      0.69      3534
```

Най-добър модел:

```
Резултати от всички модели:
Naive Bayes: 0.6469
Logistic Regression: 0.6819
SVM: 0.6836
Random Forest: 0.6930
```

След обучение и тестване, моделът Random Forest показа най-добри резултати с точност 69.30%. Този модел и съответният векторизатор (TfidfVectorizer) бяха запазени за бъдеща употреба.

Заключение

Системата успешно изпълнява задачата за класификация на настроения. Най-добрият модел (Random Forest) е запазен и може да се използва за реално време анализ на текстове. Accuracy от 69.30% демонстрира надеждност при анализ на нови и разнообразни текстови данни.

5. Литература

- "An Empirical Analysis of Naïve Bayes, SVM, Logistic Regression and Random Forest to Spot False Information in Real-World Networks" https://ieeexplore.ieee.org/document/9862430 (Дата на достъп: януари 2025 г.)
- "Comparing Naïve Bayes and SVM for Text Classification" https://www.baeldung.com/cs/naive-bayes-vs-svm (Дата на достъп: януари 2025 г.)
- Sentiment Dataset – https://www.kaggle.com/datasets/abhi8923shriv/sentiment-analysis-dataset?resource=download&select=train.csv (Дата на достъп: януари 2025 г.).
- TfidfVectorizer Documentation. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (Дата на достъп: януари 2025 г.)
- Matplotlib Documentation. (n.d.). https://matplotlib.org/ (Дата на достъп: януари 2025 г.)
- Tkinter Documentation – Tkinter https://docs.python.org/3/library/tkinter.html (Дата на достъп: януари 2025 г.)