Изкуствен интелект за системи с големи данни

Курсова задача № 3: Анализ на чувства

Стефан Велев, 0MI3400521

1. Цел и задачи

Основната цел на курсовата задача е да се направи сравнение между точността на няколко модела за извършване на анализ на чувствата, поставени в конкретен домейн – комуникация чрез социалните мрежи и по-конкретно чрез съобщения в Tweeter. За постигането на тази цел, ще бъдат изпълнени следните задачи:

1. Намиране на данни, върху които ще се обучава модела, и тяхната обработка за по-лесното им разбиране от машина
2. Трениране на собствени модели чрез бернулиев наивен бейсов класификатор и логистична регресия върху конкретно избрано множество от данни, което е преминало през предварителна обработка
3. Дефиниране на метрики за оценка на моделите – точност (*accuracy*), класификационен доклад (*classification report*), матрица на объркване (*confusion matrix*), ROC крива (*ROC curve*)
4. Интегриране на готови модели с общо предназначение (*nltk – Sentiment Intensity Analyzer*) и такива специфични за домейна (*transformers – Twitter roBERTa Sentiment Analyzer*)
5. Извършване на оценка на моделите спрямо дефинираните метрики
6. Анализ и сравнение на получените резултати

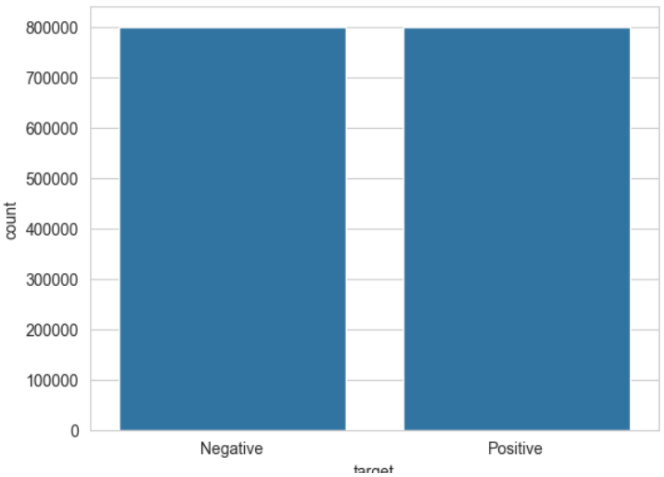
2. Проектиране

2.1 Модел на данните

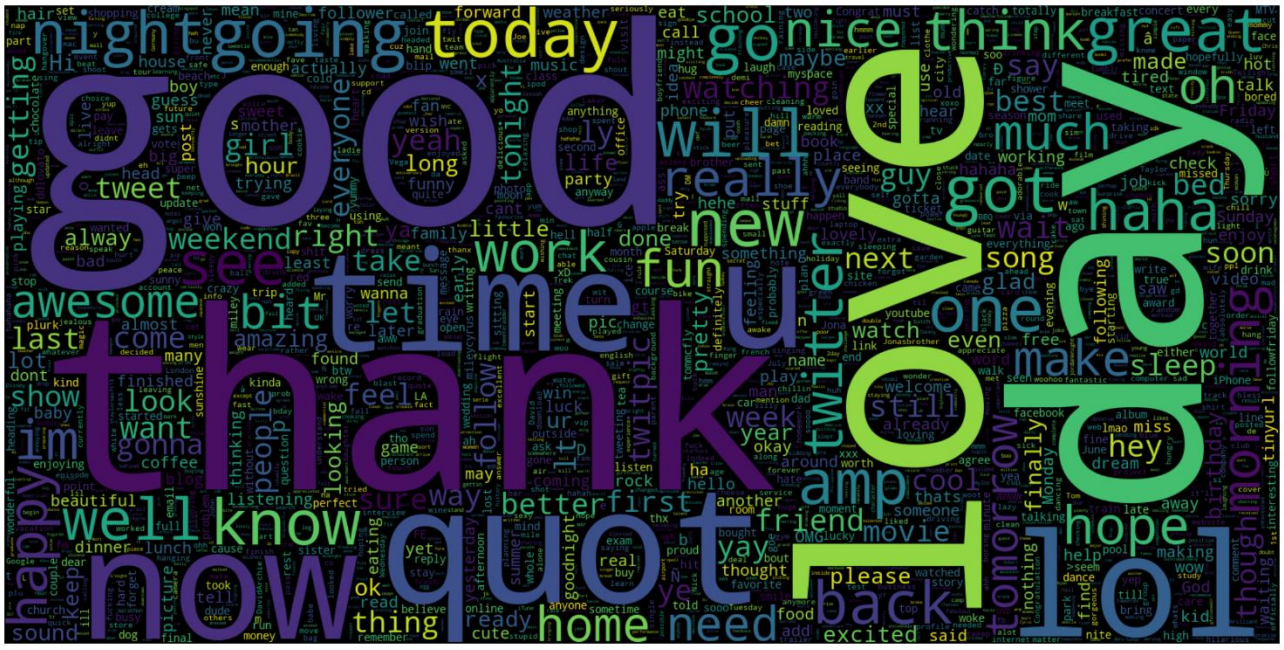
Данните, върху които ще бъдат тренирани конструираните модели, ще бъдат от областта на социалната мрежа *Twitter* – 1.6 милиона съобщения. Данните са изтеглени от *Kaggle* и имат следните полета:

- **target** (*int64*) – оценъчния заряд на съобщението (0 – негативни, 4 – положителни)
- **ids** (*int64*) – уникалният идентификатор на съобщението
- **date** (*object*) – датата на публикуване на съобщението
- **flag** (*object*) – заявката към съобщението
- **user** (*object*) – потребителят, публикувал съобщението
- **text** (*object*) – текстът на съобщението

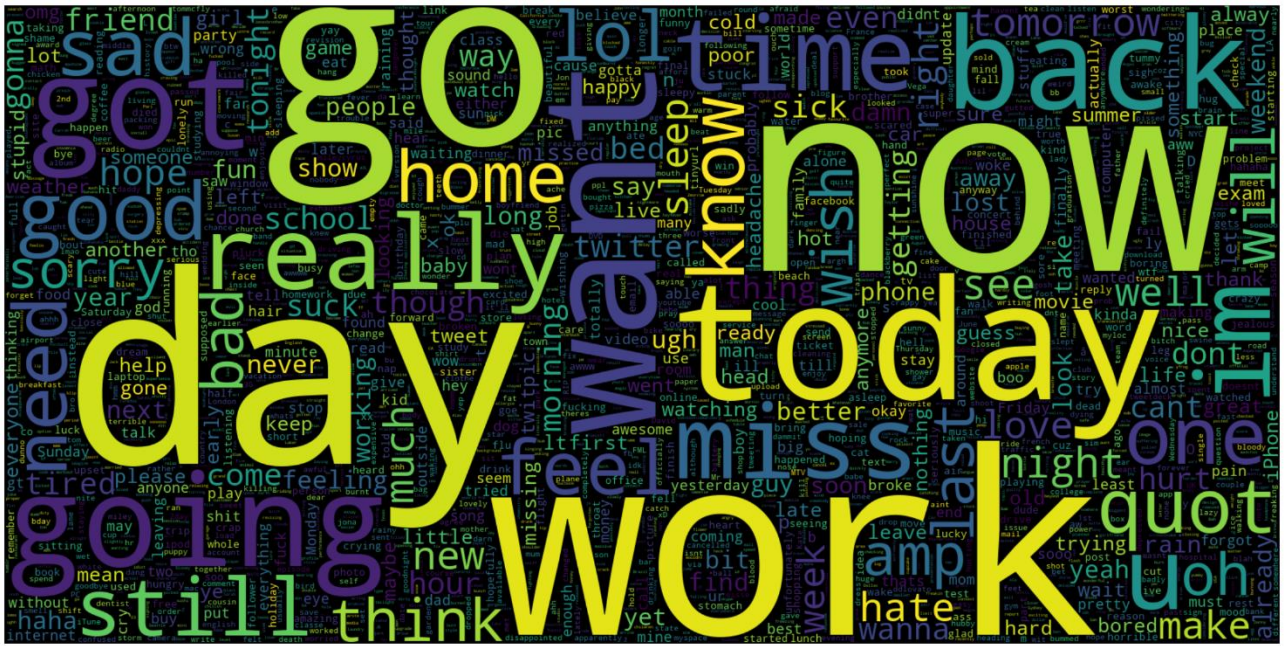
Може да се забележи, че отношението на положителните към отрицателните съобщения в избраното множество е в съотношение *50:50*, т.е. има *800 000* положителни съобщения и *800 000* отрицателни съобщения. Няма несъществуващи или непопълнени данни. Данните се намират в *CSV* файл с име *Tweets.csv* и големина 227 MB, който се намира в директория */data* към проекта.



Фигура 1: *Разпределение на съобщенията спрямо техния клас в избраното множество от данни*



Фигура 2: *Облак от думи на положителните съобщения, генериран в рамките на проекта*



Фигура 3: *Облак от думи на отрицателните съобщения, генериран в рамките на проекта*

2.2 Предварителна обработка на данните

2.2.1 Селекция на подходящите полета

Този процес от предварителната обработка на данни включва избирането на само тези атрибути и класове, които са релевантни за изследваната задача. В случая, за да извършим анализ на чувствата от съобщенията, ние ще използваме двете полета – '*text*' и '*target*':

```
data_df = df[['text', 'target']]
```

2.2.2 Трансформиране до подходящи стойности на полета

Този процес от предварителната обработка на данни включва трансформиране на стойност на дадено поле навсякъде, с цел замяната с по-удобна. Такъв е случаят и с разглежданото множество от данни, в което положителните оценки в колоната '*target*' са зададени със стойност 4, което би затруднило работата ни по-нататък, когато работим с вградени модели, които разчитат на бинарните стойности 0 и 1. За целта, правим замяна на всеки ред, в който стойността на класа е 4, т.е. е положително мнение:

```
data_df.loc[data_df['target'] == 4, 'target'] = 1
```

2.2.3 Сегрегация на данните

Този процес от предварителната обработка на данни включва разделяне на множеството от данните по подходящи критерии. В случая разделянето става по стойността на класовете, които са само два в даденото множество от данни:

```
data_negative_df = data_df[data_df['target'] == 0]
data_positive_df = data_df[data_df['target'] == 1]
```

2.2.4 Извадка от данните

Този процес от предварителната обработка на данни включва случаен избор на данните, които ще участват в обработването след това. Това се прави в случаите, когато предоставените данни са много и няма как да бъдат анализирани в цялост. В случая избираме половината от данните 400 000 положителни мнения и 400 000 отрицателни мнения:

```
data_positive_df = data_positive_df.sample(n = 400000, random_state = 42)
data_negative_df = data_negative_df.sample(n = 400000, random_state = 42)
```

2.2.5 Обединяване на данните

Този процес от предварителната обработка на данни включва обратното обединяване на няколко масива от данни, които преди това са били изрязани или разделени по друга причина. В случая положителните и негативните мнения се обединяват в една колекция, с която ще се работи навсякъде впоследствие:

```
merged_df = pd.concat([data_positive_df, data_negative_df])
```

2.2.6 Трансформиране на данните

2.2.6.1 Преминаване към малки букви

Трансформацията към малки букви е една от основните стъпки преди множеството от съобщения да бъде подадено на алгоритъм за самообучение. Това е така, защото много често началната главна буква не предава различно значение на думите. Изключение от това има при езиците, в които главните букви се използват за изписване на съществителни имена като немския и др. езици. В случая, множеството, което изследваме, е на английски език и преминаването към малки букви е удачна практика:

```
merged_df['text'] = merged_df['text'].str.lower()
```

2.2.6.2 Премахване на „шумови“ думи

Премахването на шумовите думи е също много ефективна техника, която се извършва в началните етапи, с цел заличаване на думи, които не носят смислово значение и емоционален заряд. Това са най-вече части от граматични конструкции като спомагателни глаголи, определителен и неопределителен член, съюзи, предлози и др. В случая множеството от „шумови“ думи може да бъде взето от библиотеката *nltk* и с функция да бъдат почистени от тях съобщенията:

```
def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in
stopwords_set])
merged_df['text'] = merged_df['text'].apply(remove_stopwords)
```

2.2.6.3 Премахване на URL адреси

Премахването на URL адреси е важна стъпка, когато данните, които ще обработваме, имат пряка връзка със света на Интернет. В случая, съобщенията от социална мрежа с доста голяма вероятност ще съдържат множество такива записи. Чрез регулярен израз съответните редове могат да бъдат идентифицирани, а самите адреси – премахнати:

```
def remove_URLs(text):
    return re.sub(r'(www\.[^\s]+|https?://[^\s]+)', '', text).strip()
merged_df['text'] = merged_df['text'].apply(remove_URLs)
```

2.2.6.4 Премахване на пунктуация

Премахването на пунктуацията е следващ етап от трансформирането на данни. В голяма част от случаите пунктуацията не би помогнала на един класификационен модел, освен ако той не е специално обучен да разчита на нея или да изследва по-сложни връзки чрез нея. За премахване на пунктуацията в случая е използвана следната функция:

```
def remove_punctuations(text):
    translator = str.maketrans('', '', punctuation_list)
    return text.translate(translator)
merged_df['text'] = merged_df['text'].apply(remove_punctuations)
```

2.2.6.5 Премахване на повтарящи се символи

Често в комуникацията в социалните мрежи се набляга на някои думи, с цел да се вкара емоция в текстова форма. Един от начините това да стане е като се използват повтарящи символи, обикновено в края на думата. За да могат моделите да разпознаят, че става въпрос за една и съща дума, то повтарящите символи е добре да бъдат премахнати. В случая за улеснение са премахнати всички повтарящи се един след друг символи, независимо къде се намират, защото е възможно даден потребител несъзнателно да повтори даден символ:

```
def remove_repeated_characters(text):
    return re.sub(r'(.)\1+', r'\1', text)
merged_df['text'] = merged_df['text'].apply(remove_repeated_characters)
```

2.2.6.6 Премахване на числа

Друга важна стъпка преди подаване на данните към даден модел е премахването на числата. В повечето случаи информацията под формата на числа не носи полза за алгоритъма, тъй като той не разполага с обективна оценка за тях, т.е. не може да определи кое е много и кое малко в различните ситуации и контекст, освен ако не е предварително обучен на това. В случая това важи и за настоящия проект:

```
def remove_numbers(text):
    return re.sub('[0-9]+', '', text)
merged_df['text'] = merged_df['text'].apply(remove_numbers)
```

2.2.6.7 Разделяне на текста на графични думи

След като данните са почистени от информацията, която не би могла да ни донесе емоционален заряд, се преминава към разделяне на текстовете на отделни единици. Процесът по *tokenization* включва именно разделянето на отделни думи. За да се осъществи това, обикновено се разчита на готови инструменти, които са част от библиотеките, свързани с обработката на естествен език:

```
tokenizer = RegexpTokenizer(r'\w+')
merged_df['text'] = merged_df['text'].apply(tokenizer.tokenize)
```

2.2.6.8 Стеминг

Съкращаването на отделните думи до техния корен е следващият етап от обработката. Тук се включва премахването на формите за множествено число, формите за окончанието на думите в различните глаголни времена, както и други особености. Отново тази техника се поддържа от инструментите за обработка на естествен език:

```
st = nltk.PorterStemmer()
def stemming(text):
    return [st.stem(word) for word in text]
merged_df['text'] = merged_df['text'].apply(stemming)
```

2.2.6.9 Лематизация

Лематизацията е процес, сходен на разгледания в предишната точка. Разликата тук е, че думите преминават в основната си форма, тази която обикновено стои в речниците на даден естествен език. Обработката се осъществява по подобен начин:

```
wnlm = WordNetLemmatizer()
def lemmatization(text):
    return [wnlm.lemmatize(word) for word in text]
merged_df['text'] = merged_df['text'].apply(lemmatization)
```

2.3 Предварителни стъпки при работа с модели

2.3.1 Разделяне на тренировъчно и тестово множество

За да може да се оцени точността на даден модел съществуват редица въпроси – върху всички или част от наличните данни моделът да се обучи; ако се обучи на всички данни, измерването на точността отново върху същите данни ли да става; ако не, как да разберем новите данни в какъв истински клас попадат, кой може да ни даде обективна оценка. На тези въпроси отговор дава процесът по кръстосано валидиране на данни – техника, широко позната и използвана в повечето алгоритми за машинно самообучение. Наличните данни се разделят в дадено съотношение, обикновено 80:20, 90:10, 95:5, съответно в тренировъчно множество и тестово множество. Тренировъчното множество е това, върху което моделът се обучава, а чрез тестовото множество оценяваме неговата точност, като го викаме на всеки един запис по отделно и сравняваме дали предсказаната от него стойност отговаря на истинската. Поради известността на тази техника за работа с данните съществуват готови решения, които извършват това разделение за нас, както сме използвали в случая:

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1,
random_state = 36104664)
```

2.3.2 TF-IDF векторизация

Превръщането и работата с числа е много по-удобна за алгоритмите вместо обработката на масиви от текстови данни. Още повече, по този начин се засилва влиянието на

по-често срещаните думи, които много вероятно да носят емоционалния заряд на даденото съобщение. За осъществяване на конвертирането използваме готовата библиотека в *Python*:

```
vectoriser = TfidfVectorizer(ngram_range=(1,3), max_features = 800000)
vectoriser.fit(X_train)
X_train = vectoriser.transform(X_train)
X_test = vectoriser.transform(X_test)
```

2.3.3 Метрики за оценка на моделите

2.3.3.1 Матрица на неточностите (Confusion Matrix)

При матрицата на неточностите се съпоставят определените от класификатора и действителните стойности по четири метрики:

- **Истински позитивни** (*True Positive*) – обекти, класифицирани като положителни и в действителност са в положителния клас
- **Лъжливи позитивни** (*False Positive, Type I Error*) – обекти, класифицирани като положителни, но в действителност са в отрицателния клас
- **Истински негативни** (*True Negative*) – обекти, класифицирани като отрицателни и в действителност са в отрицателния клас
- **Лъжливи негативни** (*False Negative, Type II Error*) – обекти, класифицирани като отрицателни, но в действителност са в положителния клас

```
def plot_confusion_matrix(y_pred, title):
    cf_matrix = confusion_matrix(y_test, y_pred)
    categories = ['Negative', 'Positive']
    group_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in
    cf_matrix.flatten() / np.sum(cf_matrix)]
    labels = [f'{v1}n{v2}' for v1, v2 in zip(group_names, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues', fmt = '', xticklabels
    = categories, yticklabels = categories)
    plt.xlabel("Predicted values", fontdict = {'size': 14}, labelpad = 10)
    plt.ylabel("Actual values", fontdict = {'size': 14}, labelpad = 10)
    plt.title("Confusion Matrix: " + title, fontdict = {'size': 18}, pad = 20)
    plt.show()
```

2.3.3.2 Обща точност (Overall Accuracy Score)

Общата точност е показател, който ни дава информация за това каква част от всички случаи са правилно класифицирани от модела. Тя е една от най-често използваните метрики при оценка на работата на класификатор. Изчислява се като общият брой правилно класифицирани обекти се разделя на всички случаи:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

```
print(f'Accuracy Score: {accuracy_score(y_test, y_pred)}')
```

2.3.3.3 Класификационен доклад (Classification Report)

Класификационния доклад е метод на *scikit-learn*, с който автоматично се генерира текстов доклад, който включва основни метрики за оценка на даден модел сред които:

- **прецизност** (*precision*) – отношението между истинските позитивни предсказания към всички предсказани позитивни. Тази метрика измерва точността на позитивните предсказания:

$$Precision = \frac{TP}{TP + FP}$$

– **пълнота** (*recall, TPR, Sensitivity*) – отношението между истинските позитивни предсказания към всички реални позитивни. Тази метрика измерва способността на модела да улавя всички позитивни инстанции:

$$Recall = \frac{TP}{TP + FN}$$

– **F1 оценка** (*F1 Score*) – средно хармоничното на прецизността и пълнотата. Тази метрика измерва баланса между горните две метрики:

$$F1 - Score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

– **Поддръжка** (*Support*) – броя на срещанията на класа в дадено множество данни

```
def print_classification_report(y_pred, title):
    report = classification_report(y_test, y_pred)
    print("Classification Report: " + title)
    print(report)
```

2.3.3.4 ROC-крива (Receiver Operating Characteristic curve)

Тази крива е един от най-известните начини за оценка на модел за бинарна класификация. При нея се съпоставят **TPR** (*True Positive Rate*) и **FPR** (*False Positive Rate*) при различни прагове (*threshold*). Правата y = x (второстепенния диагонал) изобразява случайно взет класификатор. Колкото повече се отдалечава от нея кривата на определен модел и се доближава до горния ляв ъгъл на графиката, толкова по-добър е той. Площта под кривата (*area under curve – AUC*) определя до каква степен класификаторът може да различи единия от другия клас при разпределяне на обектите. Стойностите на площта под кривата са в интервала между 0 и 1. Ако *AUC* е 1, тогава моделът е определил всички случаи правилно.

```
def plot_roc_curve(y_pred, title):
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=1,
    label='ROC curve (AUC = %0.2f)' % roc_auc)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve: ' + title)
    plt.legend(loc="lower right")
    plt.show()
```

3. Реализация

3.1 Използвани технологии, платформи и библиотеки

За реализация на проекта е използван езикът за програмиране *Python*, версия 3.11. Кодът по проекта е придружен с обяснения и получени резултати стъпка по стъпка в *Jupyter Notebook*. Използваните библиотеки са:

– **re** – за работа с регулярни изрази, манипулация на текст и др.
– **string** – за работа със стрингове, тяхното манипулиране, форматиране и др.
– **numpy** – за работа с големи масиви, матрици, математ. функции и др.
– **pandas** – за работа и манипулиране и анализиране на данни
– **seaborn** – за визуализация на статистически данни чрез различни графики

- **matplotlib** – създаване на статични и интерактивни визуализации на данни
- **nltk** – инструменти за обработка на ест. език, вкл. за анализ на чувствата
- **sklearn** – множество алгоритми за класификация, регресия и др.
- **scipy** – за оптимизация, интеграция на научни изчислителни задачи
- **tqdm** – за следене на прогреса чрез визуализация на съответна лента
- **transformers** – предоставя тренирани модели за различни цели

Конкретните версии за работя с тях могат да бъдат намерени в *requirements.txt* файл, който се намира в директорията на проекта. Използваните класове от горепосочените библиотеки могат да бъдат видени в началото на кода към проекта.

3.2 Реализация на моделите

3.2.1 Бернулиев наивен Бейсов класификатор

Бернулиевият наивен Бейсов класификатор е наивен Бейсов класификатор, който се използва за класификация на бинарни данни, т.е. заемащи само две стойности – 0 или 1. Важно е да се отбележи допускането, че характеристиките са независими едни от други при условие даден клас. Това допускане правим и за съобщенията в нашия домейн. Обикновено се използва за идентифициране на спам, класификация на текст, анализ на чувствата и др. За имплементиране на модела сме използвали готовия модел от *scikit-learn*. Подадените на модела данни са в *TF-IDF* векторен формат:

```
BNBmodel = BernoulliNB()
BNBmodel.fit(X_train, y_train)
evaluate_model(BNBmodel, "Bernoulli Naive-Bayes Classifier")
```

3.2.2 Логистична регресия

Логистичната регресия е класификационен алгоритъм за машинно самообучение, който е много подходящ в нашия случай, тъй като работи добре за бинарни данни. Предимствата на подхода са, че е лесен за изчисление, с по-малък риск от преспецифициране и подходящ за изходна конфигурация. В случая използваме готовия модел от библиотеката *scikit-learn*. Параметърът *C* в него означава силата на регуляризацията – ако е твърде малък, алгоритъмът може да схожда бързо, ако е много голям – да се нуждае от повече итерации, за да намери оптималното решение. Параметърът *n_jobs* определя броя ядра, които да бъдат използвани по време на тренирането на модела. Ако стойността е -1, това означава, че ще бъдат използвани всички ядра на дадената машина. Подадените на модела данни отново са в *TF-IDF* векторен формат за по-лесна и точна обработка:

```
LRmodel = LogisticRegression(C = 2, max_iter = 10000, n_jobs = -1)
LRmodel.fit(X_train, y_train)
evaluate_model(LRmodel, "Logistic Regression")
```

3.3 Вграждане на предварително тренирани модели

3.3.1 Модел с общо предназначение от nltk

За дефинираните в миналата точка метрики за оценка, ще сравним горните модели с вече имплементирани и тренирани такива. Ще използваме готовия модел за анализ на чувствата, който идва от библиотеката *nltk* – *SentimentIntensityAnalyzer*. Библиотеката и модула са широко-известни. Базирани са на *VADER* модела, който е подход за анализ на чувства, базиран на правила и *bag-of-words* подхода. Вероятностните предположения в резултата се връщат под формата на речник, който съдържа четири различни ключа – *pos*, *neu*, *neg* и *compound*. За оценка на модела използваме *compound* стойността, която е нормализирана оценка въз основа на останалите три. Тя може да бъде между -1 и 1, като в случаите, когато стойността е положителна, оценката също е положителна и когато стойността е отрицателна,

оценката на съответното съобщение е отрицателна. Тук данните подаваме в оригиналния текстови формат, тъй като подадени в обработената съкратена форма, биха довели до по-нисък резултат на алгоритъма, тъй като евентуалните вътрешни проверки са заложени стандартно в него:

```
def assess_sentiment_nltk(text):
    sid = SentimentIntensityAnalyzer()
    sentiment_score = sid.polarity_scores(text)['compound']
    return 1 if sentiment_score >= 0 else 0

y_pred_nltk = []
for text in tqdm(X_test_as_text, desc="Processing texts"):
    y_pred_nltk.append(assess_sentiment_nltk(text))
evaluate_model(None, "NLTK Sentiment Analyser", y_pred_nltk)
```

3.3.2 Модел, специфичен за домейна на социалните мрежи

За сравнение ще използваме още един предварително трениран модел от *Hugging Face*, който е изграден на основата на т.нар. трансформираща архитектура – конкретно върху модела *BERT Sentiment Analysis*. Той използва модел, основан на дълбоко обучение, трениран на огромни масиви от данни. Избраният модел в нашия случай е трениран на множество от съобщения в социалната мрежа *Twitter*. Характерно за тези видове езикови модели е, че те имат по-високо разбиране на семантиката и контекста в сравнение с други модели основани на рекурентните невронни мрежи или стандартните опростени подходи. Вероятностната оценка, която ни връща модела, е в три категории – неутрална, негативна и позитивна. За да преобразуваме данните към стойностите 0 и 1, извършваме сравнение с т.нар. прагова стойност, която в нашия случай и по подразбиране е 0.5, т.е. сравнението е следното: ако оценката за положително мнение е по-голяма или равна на 0.5, то съобщението се определя като положително, ако не – като отрицателно. Отново данните се подават в оригиналния формат без предварителната обработка, която извършихме в началото, отново от същите съображения:

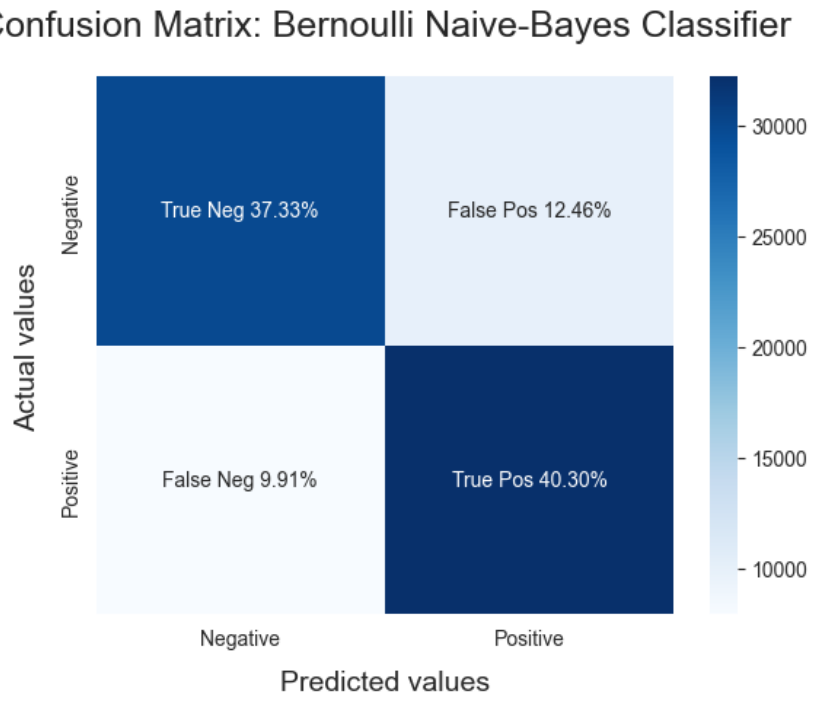
```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment-latest"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
def assess_sentiment_roberta(example, threshold = 0.5):
    encoded_text = tokenizer(example, return_tensors = 'pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    compound_score = scores[2]
    sentiment_label = 1 if compound_score >= threshold else 0
    return sentiment_label
y_pred_roberta = []
for text in tqdm(X_test_as_text, desc="Processing texts"):
    y_pred_roberta.append(assess_sentiment_roberta(text))

evaluate_model(None, "Twitter roBERTa Sentiment Analyser", y_pred_roberta)
```

3.4 Получени резултати

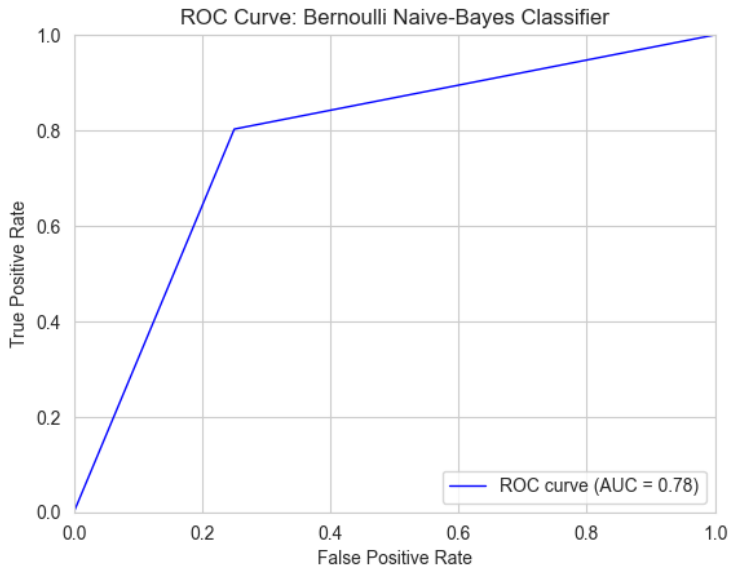
Получените резултати по определените метрики за оценка на **дефинирания модел с Бернулиевия наивен класификатор** са следните:

```
Accuracy Score: 0.7763
Classification Report: Bernoulli Naive-Bayes Classifier
              precision    recall  f1-score   support

           0       0.79      0.75      0.77     39834
           1       0.76      0.80      0.78     40166

    accuracy                           0.78     80000
   macro avg       0.78      0.78      0.78     80000
weighted avg       0.78      0.78      0.78     80000
```

Фигура 4: *Обща точност и класификационен доклад за модела за анализ на чувства с Бернулиев наивен Бейсов класификатор*



Фигура 5: *Матрица на неточностите за модела за анализ на чувства с Бернулиев наивен Бейсов класификатор*

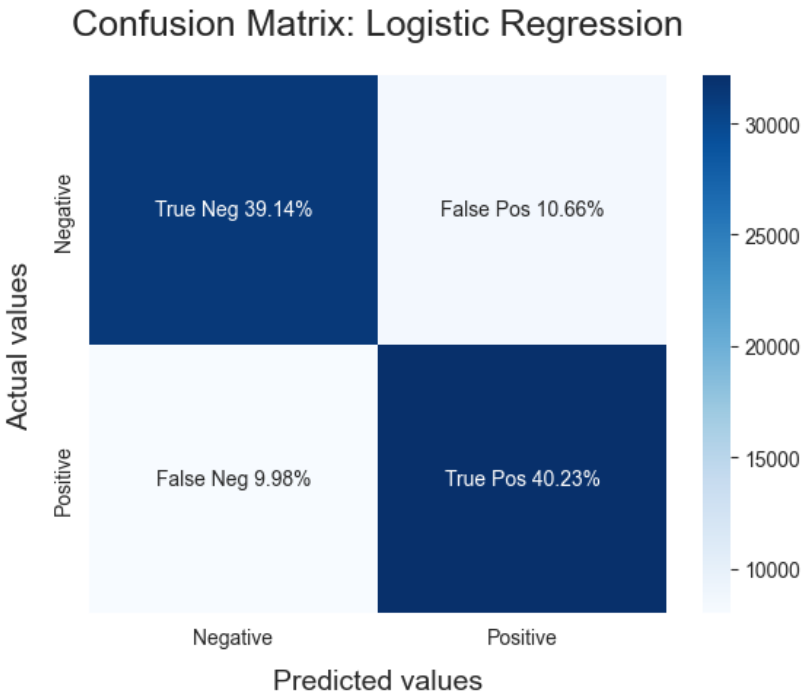


Фигура 6: *ROC-крива за модела за анализ на чувства с Бернулиев наивен Бейсов класификатор*

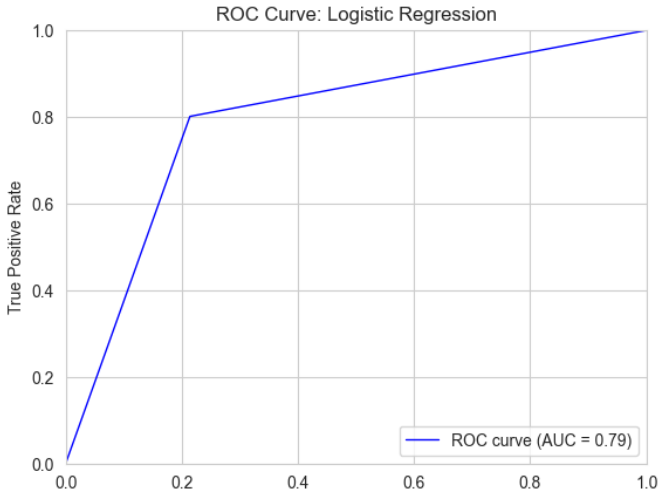
Получените резултати по определените метрики за оценка на **дефинирания модел с логистична регресия** са следните:

```
Accuracy Score: 0.7936625
Classification Report: Logistic Regression
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     39834
           1       0.79      0.80      0.80     40166

    accuracy                           0.79     80000
   macro avg       0.79      0.79      0.79     80000
weighted avg       0.79      0.79      0.79     80000
```

***Фигура 7**: Обща точност и класификационен доклад за модела за анализ на чувства с логистична регресия*



***Фигура 8:** Матрица на неточностите за модела за анализ на чувства с логистична регресия*

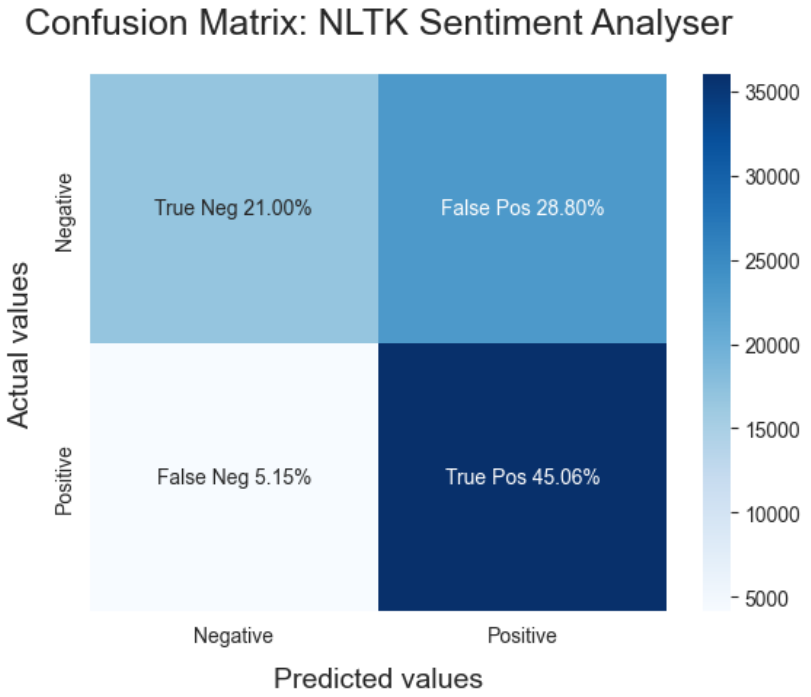


***Фигура 9:** ROC-крива за модела за анализ на чувства с логистична регресия*

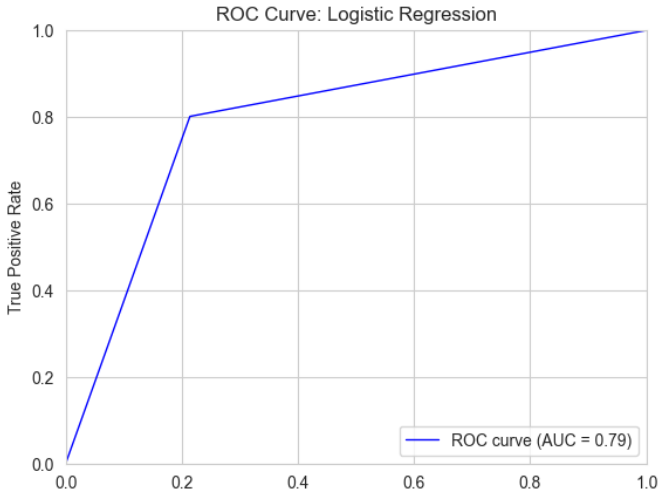
Получените резултати по определените метрики за оценка на **предварително тренирания модел за анализ на чувства с общо предназначение nltk** са следните:

```
Accuracy Score: 0.6605375
Classification Report: NLTK Sentiment Analyser
              precision    recall  f1-score   support

           0       0.80      0.42      0.55     39834
           1       0.61      0.90      0.73     40166

    accuracy                           0.66     80000
   macro avg       0.71      0.66      0.64     80000
weighted avg       0.71      0.66      0.64     80000
```

***Фигура 10**: Обща точност и класификационен доклад за модела за анализ на чувства с общо предназначение от NLTK*



***Фигура 11:** Матрица на неточностите за модела за анализ на чувства с общо предназначение от NLTK*



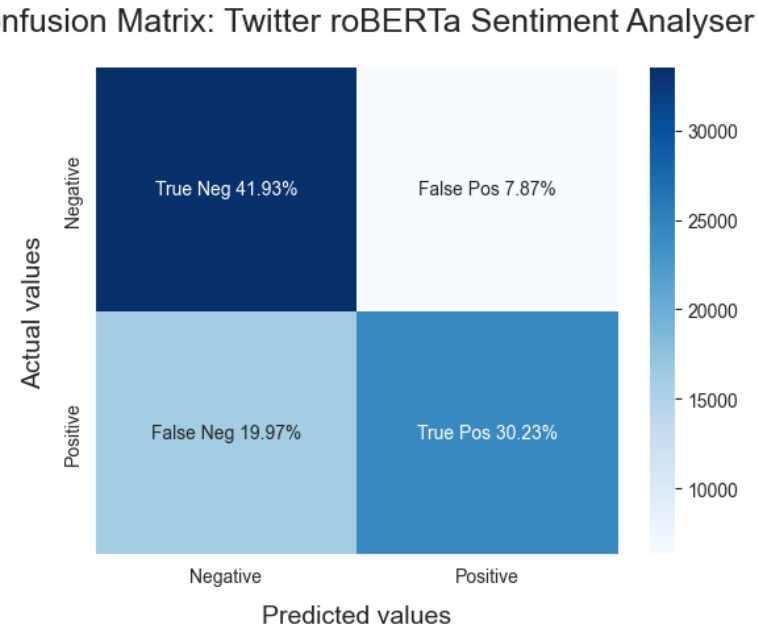
***Фигура 12:** ROC-крива за модела за анализ на чувства с общо предназначение от NLTK*

Получените резултати по определените метрики за оценка на **предварително тренирания модел за анализ на чувствата за конкретния домейн roBERTa Sentiment Analysis** са следните:

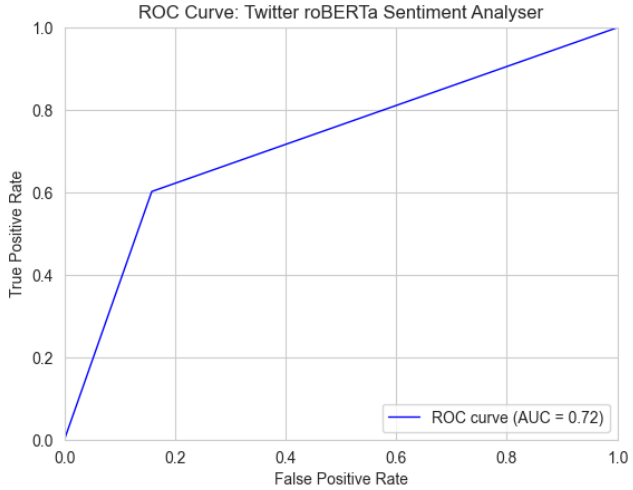
Accuracy Score: 0.7216Accuracy Score: 0.7216
Classification Report: Twitter roBERTa Sentiment Analyser

	precision	recall	f1-score	support
0	0.68	0.84	0.75	39834
1	0.79	0.60	0.68	40166
accuracy			0.72	80000
macro avg	0.74	0.72	0.72	80000
weighted avg	0.74	0.72	0.72	80000

***Фигура 13**: Обща точност и класификационен доклад за модела за анализ на чувства с конкретен домейн roBERTa*



***Фигура 14:** Матрица на неточностите за модела за анализ на чувства с конкретен домейн roBERTa*



***Фигура 15:** ROC-крива за модела за анализ на чувства с конкретен домейн roBERTa*

4. Заключение

Целта на курсовата задача беше да се направи сравнение между моделите за анализ на чувствата на мнения на потребители, в случая съобщения в конкретен домейн (*Twitter*) и за такива с общо предназначение. Получените резултати показват, че с най-добри показатели е **ръчно тренираният модел, използващ логистична регресия**. Това се дължи на няколко фактора, сред които внимателната предварителна подготовка на данните преди те да бъдат подадени на модела, трансформацията им с *TF-IDF* вектор, както и модела на логистичната регресия, който е изключително подходящ за решаване на бинарни проблеми. Получените по-ниски резултати за тренираните предварително модели доказват факта, че те няма как да се държат еднакво добре във всяка една сфера от живота. Дори моделът *roBERTa*, за който се твърди, че е трениран на съобщения от социалната мрежа *Twitter*, не води до значително по-добри резултати. Една от възможните причини е неяснотата на готовите модели по отношение на това какъв е бил процесът на трениране и за каква структура на данните е най-подходящ.

Получените резултати доказват силата на специфицираните за даден домейн модели за анализ на чувствата. Възможно разширение на задачата би било тестване с други множества от данни – такива отново от социалните мрежи, за да се сравнят резултатите, както и от други сфери, за които анализът на чувствата е важно приложение.