Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

КУРСОВ ПРОЕКТ

по

Интелигентни системи

спец. Софтуерно инженерство,
4 курс, зимен семестър, академ. година 2023/2024

Сравнение между модели за анализ на чувствата, общоцелеви и специфични за домейна, върху данни от Tweeter

Изготвил:
Стефан Димитров Велев,
Ф. Н.: 62537

Преподавател:
проф. д-р Иван Койчев

Декларация за липса на плагиатство:

- Плагиатство е да използваш, идеи, мнение или работа на друг, като претендираш, че са твои. Това е форма на преписване.
- Тази курсова работа е моя, като всички изречения, илюстрации и програми от други хора са изрично цитирани.
- Тази курсова работа или нейна версия не са представени в друг университет или друга учебна институция.
- Разбирам, че ако се установи плагиатство в работата ми ще получа оценка “Слаб”.

26.01.2024
гр. София

Подпис на студента:

СЪДЪРЖАНИЕ

СПИСЪК НА ФИГУРИТЕ

1. Увод

1.1 Мотивация

В последните години изкуственият интелект набира все по-голяма популярност. Науката, която изследва начините за това как да накараме компютрите да правят неща, които за хората изглеждат разумни, навлиза постепенно във всяка една сфера на живота. Такава е и обработката на естествени езици. Границите на тази наука са необозримо широки. Едно много интересно нейно приложение е анализът на чувства и мненията на потребителите. Да се направи такава оценка може да изглежда лесна задача за нас, но това не е така за един софтуер. Причината е, че за да вземат адекватно решение, алгоритмите трябва не само да бъдат обучени и да знаят перфектно съответния език, а също така и да могат да определят емоционалния заряд на неговите конструкции. Интересът, който тези въпроси, така поставени, предизвикват, е това, което допринася за избора на автора на тема за работа по курсовия проект.

1.2 Цел и задачи

Основната цел на курсовия проект е да се направи сравнение между точността на няколко модела за извършване на анализ на чувства, поставени в конкретен домейн – комуникация чрез социалните мрежи и по-конкретно чрез съобщения в *Tweeter*. За постигането на тази цел, ще бъдат изпълнени следните конкретни 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. Преглед на подходите за извършване на анализ на чувства [1][5][6][7][10][14][16][17][18]

Задачите, свързани с обработката на естествен език, стават все по-предизвикателни в наши дни. Интересите към теорията, която стои зад естествените езици е нещо, което покрай процесите по цифровизация и дигитализация възвръща популярност. Когато даден език е майчин за нас, ние не можем да видим и осъзнаем колко сложен е наистина.

Обработката на естествен език е интердисциплинарна наука, която има сечение с компютърните науки и лингвистиката. Тя цели да даде способност на компютрите да поддържат и манипулират човешки езици. Включва обработване на набори от текстови данни, основавайки се на подходи базирани на правила и вероятностни такива за машинно самообучение.

Подходите базирани на правила използват множество създадени от човек правила, които да помогнат да се идентифицира субективност и да се извърши оценка на мнението. Тези правила обикновено са прераснали в конкретни известни техники в тази област. Такива са:

- **Разделяне на текста на графични думи и пунктуационни знаци** (*Tokenisation*)
- **Стеминг** (*Stemming*) – съкращаване на думата до нейния корен, което намалява размера на речника от думи в документа и подобрява резултатите, особено при по-малко множества от данни
- **Лематизация** (*Lemmatisation*) – групиране на различните части на една дума в лема
- **Идентифициране на „шумови“ думи** (*Stopwords*) – премахване на думи, които не носят важно значение (цифри, знаци, местоимения, предлози и др.)
- **Филтриране на думи** (*Filtering Tokens*) – филтриране на думите по различни критерии (напр. дължина на думата)
- **Трансформиране на думи** (*Transforming Tokens*) – напр. превръщане на главните букви в малки
- **Автоматичен морфологичен анализ** (*Part-Of-Speech-Tagging*) – маркиране на частите на речта в рамките на изречението напр. съществителни, глаголи, наречия, съюзи и др.
- **Автоматичен синтактичен анализ** (*Parsing*) – генериране на дърво на връзките между думите в рамките на изречението (синтактично дърво)
- **Многозначност** (*Word Sense Disambiguation*) – техника за отстраняване на многозначността на думите или фразите

Подходите, базирани на вероятности или още наречени **автоматични**, разчитат на методи за машинно самообучение за извършване на самите анализи. Универсално това включва процес на обучение, в който моделът се учи да асоциира определен вход (текст) със съответния изход, въз основа на т.нар. тренировъчно множество, използвано за обучение, и процес на предсказване (тестване) на модела, в който се генерират отговори на невиждани досега от модела входове, с цел изчисляване на точност. Последното се осъществява върху т.нар. тестово множество. Известни класификационни алгоритми в това отношение са наивния Бейсов класификатор, логистичната регресия, машина с поддържащи вектори, както и методи за дълбоко самообучение.

Обикновено анализът на чувствата различава три вида емоции – положителни, неутрални и отрицателни. Може да бъде приложен на отделно изречение или част от него, както и на различни видове текстови документи. Съществуват редица форми, зависещи от задачата и целта, която се преследва. Някои от тях са следните:

- **Оценъчен анализ на мнението** (*Fine-grained sentiment analysis*) – използват се категорни стойности за оценка на различните нива на емоция
  Пр.: 5-степенен рейтинг – от много позитивно до много негативно
- **Изследване на емоциите** (*Emotion detection*) – позволява оценка на още едно ниво по-навътре, различавайки различни емоции като щастие, гняв, тъга и др. За да се направи такъв вид оценка често се разчита на предварително създадени лексикони или на по-сложни алгоритми за машинно самообучение
- **Аспектно базиран анализ на чувствата** (*Aspect-based sentiment analysis*) – изследва кои черти, характеристики определят мнението, от кой аспект се интересуваме
  Пр.: Животът на батерията на тази камера е твърде малък.

– **Анализ на намеренията/подбудите** (*Intent-based sentiment analysis*) – прави оценка на намеренията, тона, отношението

Една опростена универсална техника е т.нар. опростен (*bag-of-words – n-grams*) метод, който обикновено включва групиране на думите в n-торки и намиране на броя на срещанията на положителни и отрицателни комбинации в рамките на дадено изречение или текст и вземане на решение въз онова на това кои от тях надделяват. Тази техника си има своите недостатъци, тъй като невинаги контекста е достатъчно широк и има достатъчно множество от данни (думи).

Съществуват основно три подхода за извършване на анализ на чувствата – такива от машинното самообучение, чрез лексикони и хибриден вариант. Нека да обърнем внимание на всеки един от тях.

Подходите от машинното самообучение се разделят на два вида сами по себе си – на учене с учител (*supervised learning*) и без учител (*unsupervised learning*). При машинното самообучение с учител се използват две множества от данни – тренировъчни (обучителни) данни и тестови данни. Тренировъчните данни включват входни данни (характеристики) и очаквани резултати (класове) и служат за учител по отношение на тестовите данни. Алгоритмите за машинно самообучение изграждат модел за предсказване на нови данни, използвайки тренировъчните данни и тествайки модела върху тестовите данни. При машинното самообучение без учител (*unsupervised learning*) алгоритъмът има за задача да открие моделите и структурите в данните, без да разполага с очакваните резултати (класовете). Някои от методите, които най-често се използват в машинното самообучение с учител за извличане на ключови думи и фрази (атрибути), където съставянето на множество от ефективни атрибути оказва влияние върху класификацията на мненията, са представени в таблицата:

Таблица 1: *Методи за машинно самообучение с учител при работа с модели за анализ на чувствата* [1]

Техники за избор на атрибути	Описание на техниката
Двоично представяне на термините в текста (*Binary Presence*)	Наличие на термин (дума или фраза), с двоично представена стойност на атрибутите, в който записите показват дали един термин се появява (заема стойност 1) или не (заема стойност 0)
Абсолютна честота на термин (*Term Occurrences*)	Колко пъти терминът се появява в съответния документ като абсолютна стойност
Относителна честота на термин (*Term Frequency*)	Относителна честота на термин в документ като отношение между колко пъти в думата се появява в съответния документ и общия брой на думите в документа: $\mathrm{TF} = \frac{n_w}{n}$
TF-IDF претегляне (*Term Frequency-Inverse Document Frequency*)	Техника за калкулиране на теглото на думите или фразите в текста: $TF - IDF = \frac{n_w}{n} \log_2 \frac{N}{N_w}$ където N е общият брой на документите, а N_w е броят на документите, които съдържат думата w
Автоматичен морфологичен анализ (*Part-Of-Speech Tagging*)	Прилагането на автоматичен морфологичен анализ обикновено е свързано с наблюдението, че най-често характеристиките на продуктите и услугите се изразяват със съществителни думи и фрази, а прилагателните и наречията са носители на емоционален заряд и като такива участват в определянето на полярността. Използвайки тази техника, всяка дума в текста се

	идентифицира като част на речта и се извличат само съществителни (в анализа на ниво аспект) и прилагателни и наречия, в зависимост от поставената задача

Под понятието термин (*term*) се разбира една дума или фраза състояща се от няколко думи, които са извлечени директно от текстовия корпус в определена предметна област чрез средства и методи на обработката на естествен език. *TF-IDF* стойностите се използват за създаване на векторни представяне на документите. Всеки компонент на вектора съответства на *TF-IDF* стойността на определена дума в корпуса. Термини, които не се срещат в даден документ, приемат стойност нула. Този вид представяне на текста се нарича векторен пространствен модел, който се използва в процеса на извличане на информация за по-лесната класификация впоследствие. Всяко измерение на пространството съответства на термин от речника на текстовия корпус.

В прилагането на подходите за машинно самообучение с учител в контекста на анализ на чувствата се използват различни видове класификатори: наивен Бейсов класификатор, метод на поддържащите вектори, логистична регресия, дървета на решения и др. За да бъдат използвани повечето от тях, задачата за определяне на мнението на потребители се свежда до класификационна, т.е. класът приема краен брой стойности – в случая 0 (за отрицателно мнение) и 1 (за позитивно мнение). Възможни са и други представяния като такива, включващи неутрално мнение и др.

Наивният Бейсов класификатор е вероятностен алгоритъм за машинно самообучение, който се базира на теоремата на Бейс. Използва се в класификационни задачи и е много подходящ като начално средство за анализ и когато нямаме друга особена информация в това отношение. Нарича се наивен, защото се прави допускането, че характеристиките са независими при условие даден клас. Това се прави с цел опростяване на формулите и сметките, което се оказва доста полезно при по-големи обеми от данни. Когато се прави предсказание за дадена нова инстанция, класификаторът изчислява апостериорната вероятност за всеки от класовете по отделно, при условие наблюдението. Класът, за който се получава най-голяма апостериорна вероятност, се задава като предположение за клас на даденото наблюдение.

Метод на поддържащите вектори е мощен алгоритъм за машинно самообучение с учител, който се използва както за класификация, така и за регресия. Основната му цел е да намери хиперплоскост в пространство с висока размерност, която най-добре разделя данните от различните класове. Избира се такава хиперплоскост, че да се максимизира разстоянието между класовете, т.е. разстоянието между хиперплоскостта и най-близката точка от всеки клас. Поддържащите вектори са тези точки от данните, които лежат най-близо до границата за вземане на решение (хиперплоскостта) и оказват най-голямо влияние върху нейната позиция.

Логистичната регресия е статистически метод за машинно самообучение, който се използва основно за задачи за бинарна класификация. Той цели да моделира вероятността за принадлежност на данни към определен клас. Логистичната регресия използва логистична функция (сигмоид) за преобразуване на комбинацията от линейни характеристики (*features*) в интервала [0, 1]. Това преобразуване позволява интерпретация на резултата като вероятност за принадлежност към даден клас.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Моделът се обучава чрез оптимизация на параметрите, така че логистичната функция да приближава стойности близки до 1 за положителни примери и близки до 0 – за отрицателни.

Обикновено се използва метод на максималната правдоподобност за оценка на параметрите. След трениране моделът за използва за класификация на нови данни. Определя се прагова стойност, която обикновено е 0.5 и ако предвидената вероятност е над нея, то тя се определя за положителна (1), а ако е по-малка от 0.5 – за отрицателна (0).

Вторият подход за анализ на чувства използва т.нар. **лексикони** (речници с предварително зададена оценка на думите). Процесът на събиране на думи, които оформят речника, започва с ръчно събиране на малък набор от думи, съдържащи положителен и отрицателен заряд, който се увеличава чрез търсене на техни синоними, антоними в други речници. Новооткритите думи се добавят в речника и започва следващата итерация. Итеративният процес на търсене спира едва когато не могат да бъдат открити нови думи. Основен недостатък на този подход е невъзможността да се намерят думи, характерни за определен домейн или контекст, както и да се направи разграничение по отношение на някои думи. Това е така, защото много често една и съща дума има различен емоционален заряд в зависимост от контекста. Това създава трудност дадената дума да бъде добавена в един такъв лексикон, защото оценката, която носи, не може да бъде категоризирана само и единствено на база на думата. Такъв пример може да се даде с думата „тих“, която може да има отрицателен смисъл по отношение на телефон (със значение, че не се чува добре отсрещния в разговора), както и положителен смисъл по отношение на автомобил (със значение, че двигателят не е шумен). В такива случай се разчита на двойките <оценъчна дума, обект на оценка> като контекст на мнението. Настоящият проект има за цел да изследва как именно това поставяне в определен домейн се отнася по отношение на точността на алгоритмите за анализ на чувствата.

Хибридните подходи, които съчетават подходите за машинно самообучение, заедно с базираните на лексикони, са едни от най-прилаганите, особено на ниво аспект. Анализът на мнението на потребителите е специфична задача от извличането на знания от текст, в която се използват техники и методи за извличане на закономерности от данни и обработване на естествения език. Затова повечето средства за анализ на текст и данни предоставят възможност и за анализ на мнението на потребителите. Такава е библиотеката *nltk* на *Python,* която предлага такъв инструмент с общо предназначение. Съществуват и предварително тренирани модели, които в повечето случаи са специално създадени за даден домейн. В настоящия проект ще бъдат използвани комбинация от всички варианти за постигане на по-добро и по-обективно сравнение на резултатите.

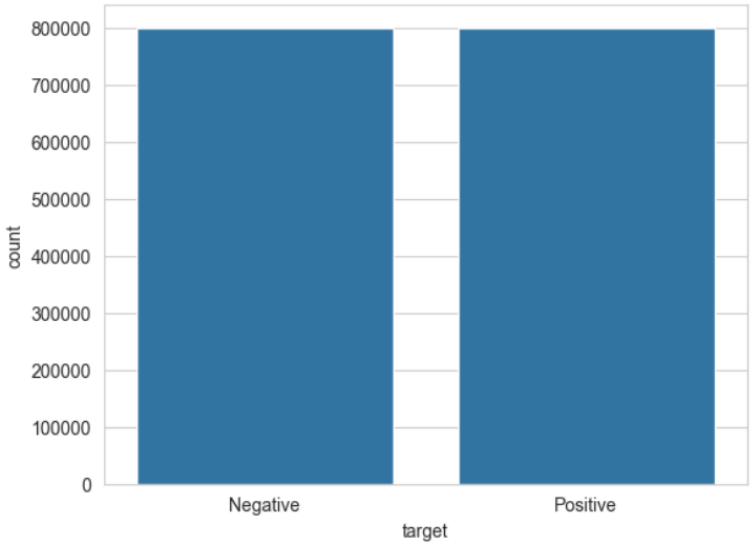
3. Проектиране [11][13]

3.1 Модел на данните

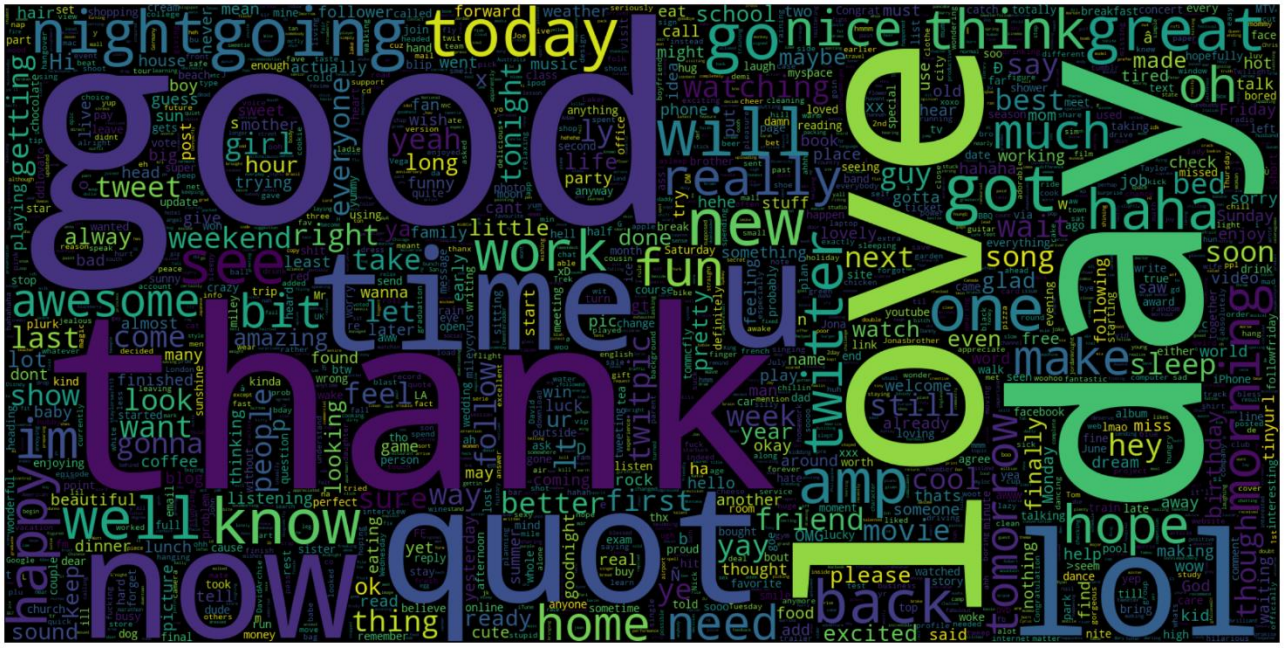
Данните, върху които ще бъдат тренирани конструираните от автора модели за настоящия проект, ще бъдат от областта на социалната мрежа *Twitter* – 1.6 милиона съобщения. Данните са изтеглени от *Kaggle* и имат следните полета:

- **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

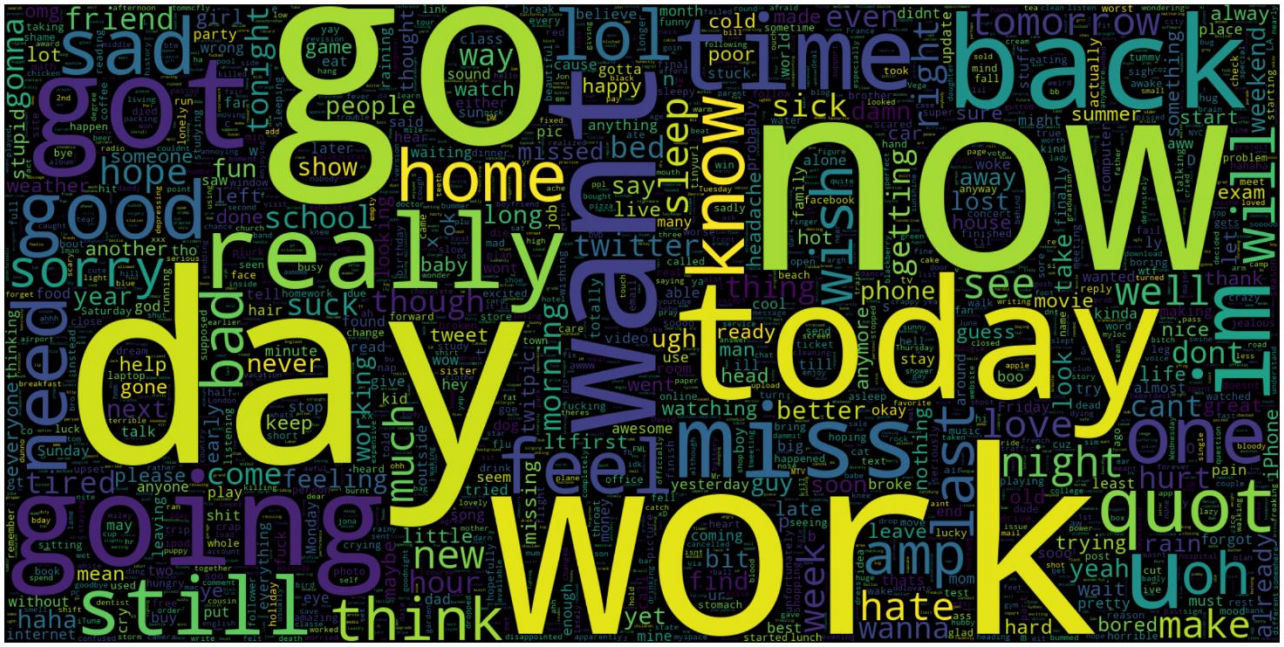
Може да се забележи, че отношението на положителните към отрицателните съобщения в избраното множество е в съотношение *50:50*, т.е. има *800 000* положителни съобщения и *800 000* отрицателни съобщения. Няма несъществуващи или непопълнени данни. Данните се намират в *CSV* файл с име *Tweets.csv* и големина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

3.2 Предварителна обработка на данните

3.2.1 Селекция на подходящи 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[['text', 'target']]
```

3.2.2 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['target'] = data['target'].replace(4, 1)
```

3.2.3 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 = data[data['target'] == 0]
data_positive = data[data['target'] == 1]
```

3.2.4 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 = data_positive.sample(n = 400000, random_state = 42)
data_negative = data_negative.sample(n = 400000, random_state = 42)
```

3.2.5 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:

```
dataset = pd.concat([data_positive, data_negative])
```

3.2.6 Трансформиране на данните

3.2.6.1 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:

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

3.2.6.2 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])
dataset['text'] = dataset['text'].apply(lambda text: remove_stopwords(text))
```

3.2.6.3 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('((www.[^s]+)|(https?://[^s]+))', ' ', text)
dataset['text'] = dataset['text'].apply(lambda text: remove_URLs(text))
```

3.2.6.4 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)
dataset['text'] = dataset['text'].apply(lambda text: remove_punctuations(text))
```

3.2.6.5 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)
dataset['text'] = dataset['text'].apply(lambda text:
remove_repeated_characters(text))
```

3.2.6.6 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)
dataset['text'] = dataset['text'].apply(lambda text: remove_numbers(text))
```

3.2.6.7 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+')
dataset['text'] = dataset['text'].apply(tokenizer.tokenize)
```

3.2.6.8 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]
dataset['text'] = dataset['text'].apply(lambda text : stemming(text))
```

3.2.6.9 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]
dataset['text'] = dataset['text'].apply(lambda text: lemmatization(text))
```

3.3 Предварителни стъпки при работа с модели

3.3.1 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)
```

3.3.2 TF-IDF векторизация

Както разгледахме по-рано, 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)
```

3.3.3 Метрики за оценка на моделите

3.3.3.1 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()
```

3.3.3.2 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:

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

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

3.3.3.3 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:

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

– **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)
```

3.3.3.4 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()
```

4. Реализация [2][3][4][8][9][12][15]

4.1 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 намерени в началото на кода към проекта.

4.2 Реализация на моделите

4.2.1 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")
```

4.2.2 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")
```

4.3 Вграждане на предварително тренирани модели

4.3.1 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)
```

4.3.2 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:

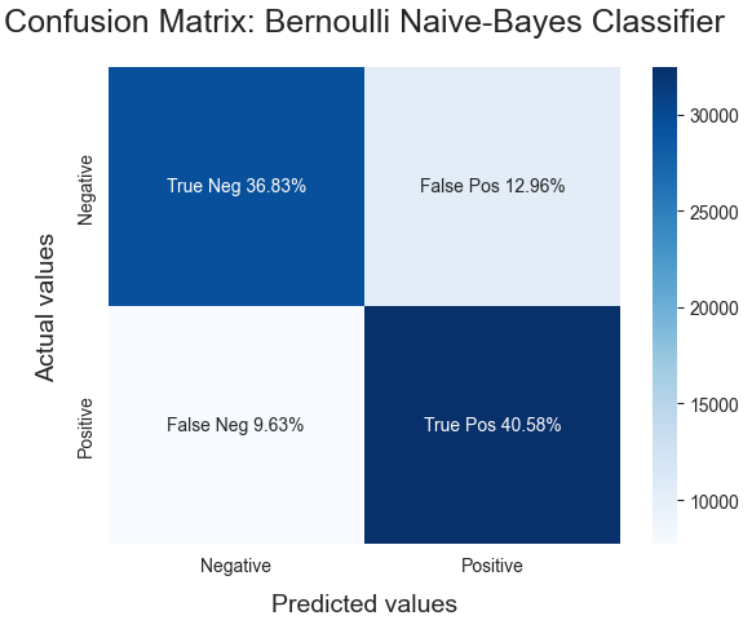
```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
def assess_sentiment_roberta(example, threshold = 0.5):
    encoded_text = tokenizer(example, return_tensors = 'pt')
    output = 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)
```

4.4 Получени резултати

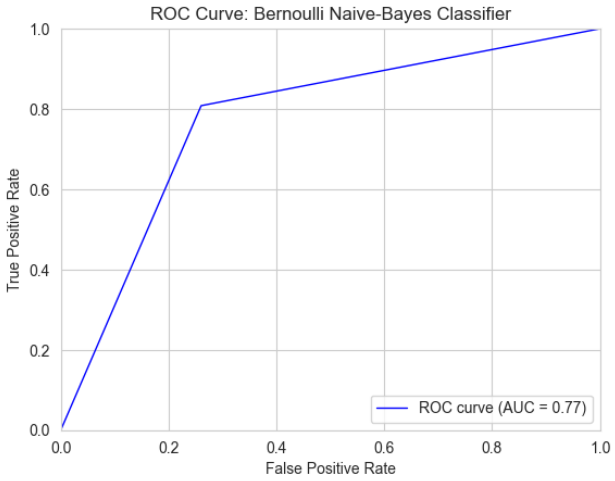
Получените резултати по определените метрики за оценка за **ръчно дефинирания модел с Бернулиевия наивен Бейсов класификатор** са следните:

```
Accuracy Score: 0.7740875
Classification Report: Bernoulli Naive-Bayes Classifier
              precision    recall  f1-score   support

           0       0.79      0.74      0.77     39834
           1       0.76      0.81      0.78     40166

    accuracy                           0.77     80000
   macro avg       0.78      0.77      0.77     80000
weighted avg       0.78      0.77      0.77     80000
```

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



Фигура 4: *Матрица на неточностите за модела за анализ на чувства с Бернулиев наивен Бейсов класификатор*

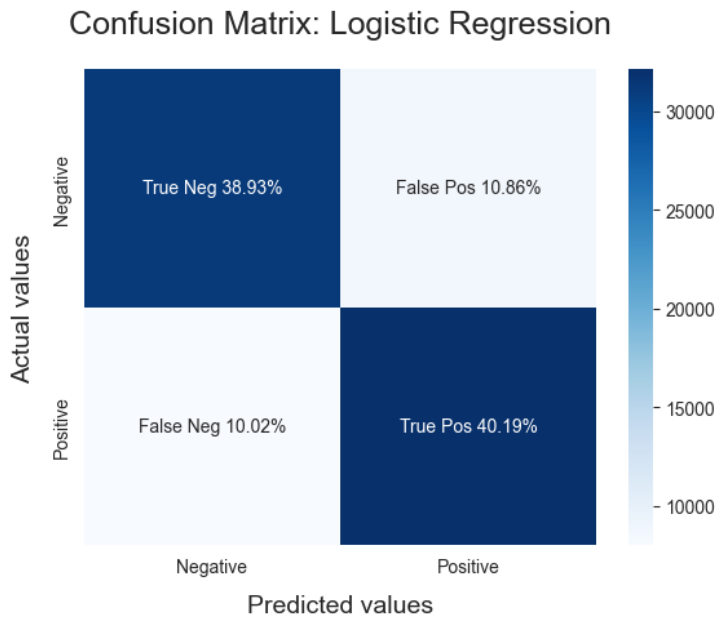


Фигура 6: *ROC-крива за модела за мализ на чувства с Бернулиев наивен Бейсов класификатор*

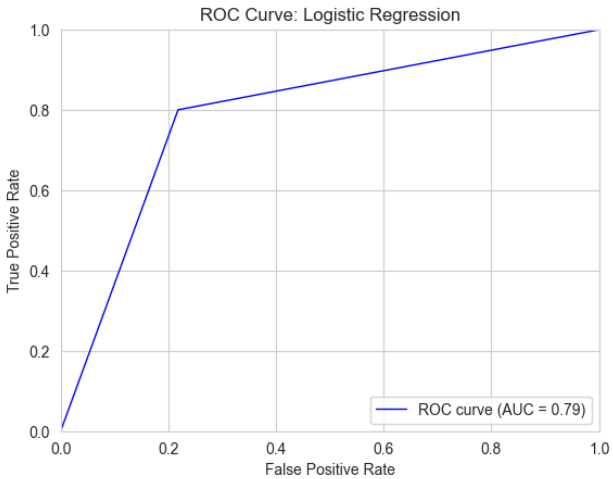
Получените резултати по определените метрики за оценка за **ръчно дефинирания модел с логистична регресия** са следните:

```
Accuracy Score: 0.7912
Classification Report: Logistic Regression
              precision    recall  f1-score   support

           0       0.80      0.78      0.79     39834
           1       0.79      0.80      0.79     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 5: *Матрица на неточностите за модела за анализ на чувства с логистична регресия*

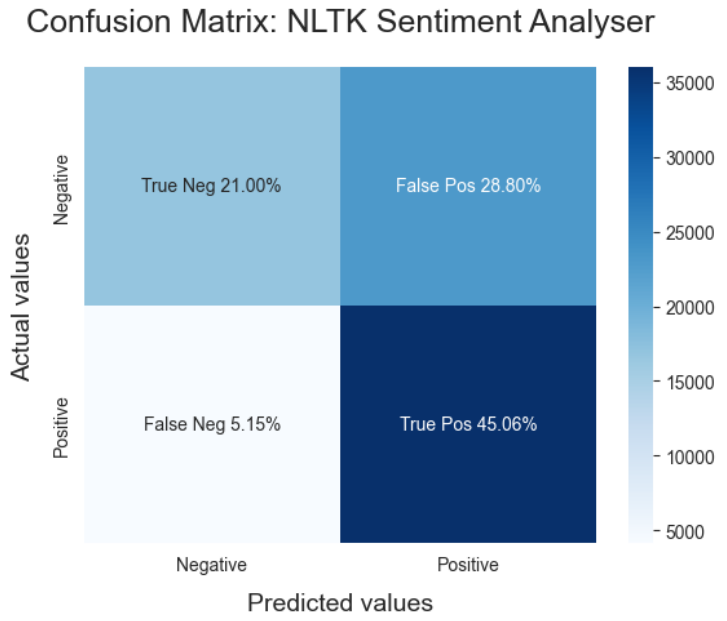


Фигура 9: *ROC-крива за модела за мализ на чувства с логистична регресия*

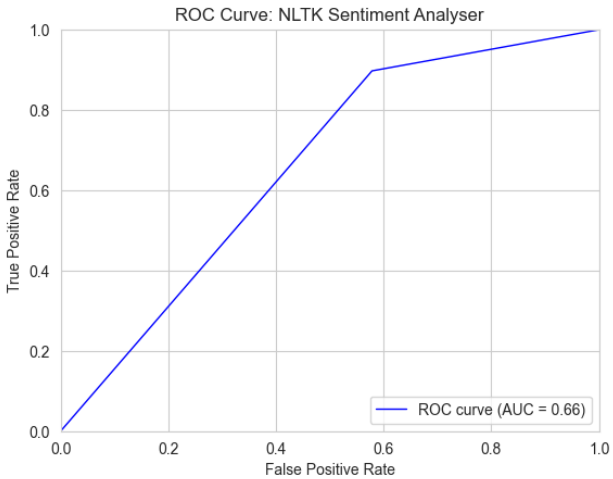
Получените резултати по определените метрики за оценка за **предварително тренирания модел за анализ на чувствата с общо предназначение *nltk*** са следните:

```
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 6: *Матрица на неточностите за модела за анализ на чувства с общо предназначение от NLTK*

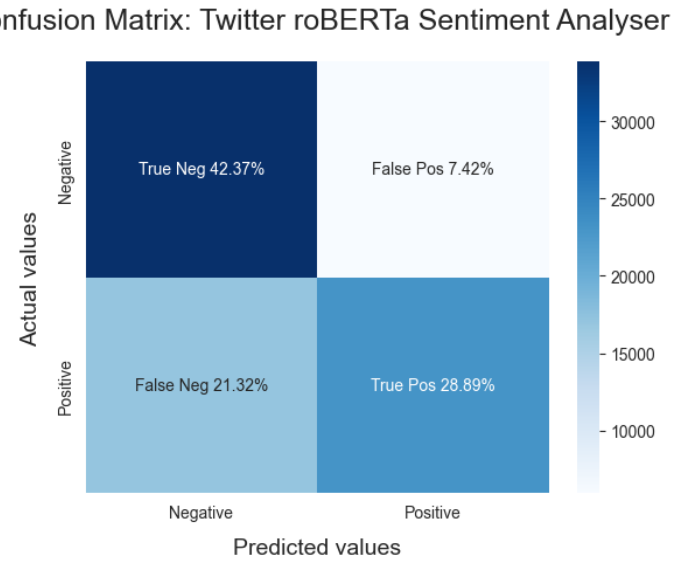


Фигура 12: *ROC-крива за модела за мализ на чувства с общо предназначение от NLTK*

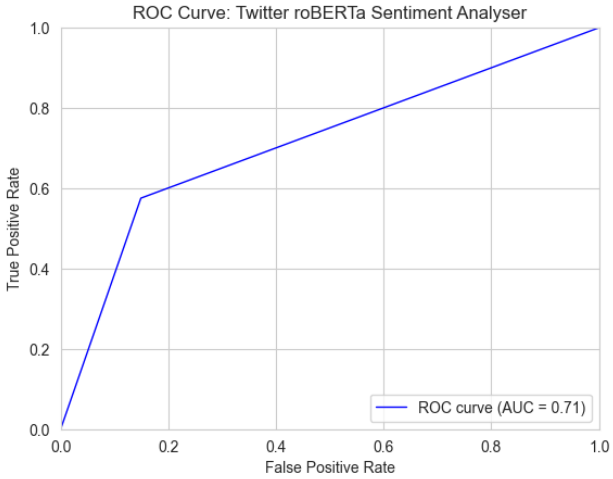
Получените резултати по определените метрики за оценка за **предварително тренирания модел за анализ на чувствата за конкретния домейн *roBERTa Sentiment Analysis*** са следните:

```
Accuracy Score: 0.712625
Classification Report: Twitter roBERTa Sentiment Analyser
              precision    recall  f1-score   support

           0       0.67      0.85      0.75     39834
           1       0.80      0.58      0.67     40166

    accuracy                           0.71     80000
   macro avg       0.73      0.71      0.71     80000
weighted avg       0.73      0.71      0.71     80000
```

Фигура 7: *Обща точност и класификационен доклад за модела за анализ на чувства с конкретен домейн roBERTa*



Фигура 9: *Матрица на неточностите за модела за анализ на чувства с конкретен домейн roBERTa*



Фигура 8: *ROC-крива за модела за мализ на чувства с конкретен домейн roBERTa*

5. Заключение

Целта на проекта 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.

6. Използвана литература

[1] **Йорданова**, Станимира, **Стефанова**, Камелия. *Извличане на знания от неструктурирани данни чрез анализ на мнението на потребители*. Сп.: Икономически и социални алтернативи [онлайн]. Бр. 1, 2017. [Прегледан 11.01.2024]. Достъпно от: https://www.unwe.bg/uploads/Alternatives/Stanimira_Alternativi%20br_1_2017_B-2.pdf

[2] **Христова**, Десислава. *Machine Learning: Метрики за оценка на класификационни модели* [онлайн]. [Прегледан 11.01.2024]. Достъпно от: https://expert-bg.org/blog/machine-learning-metriki-za-oczenka-na-klasifikaczionni-modeli/

[3] **BERT: Documentation** [online]. [Viewed 11.01.2024]. Available from: https://huggingface.co/docs/transformers/model_doc/bert

[4] **Classification: ROC Curve and AUC** [online]. [Viewed 11.01.2024]. Available from: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[5] **Damerau**, Fred, **Indurhkya**, Nitin. *Handbook of Natural Language Processing*. CRC Press, 23.02.2010. Ch. 26. ISBN: 9781420085921

[6] **Dhavale**, Shravani. *Top 4 Types of Sentiment Analysis* [online]. [Published 20.01.2013] [Viewed 11.01.2024]. Available from: https://www.nitorinfotech.com/blog/top-4-types-of-sentiment-analysis/

[7] **Eisenstein**, Jacob. *Introduction to Natural Language Processing*. The MIT Press,13.11.2018. Ch. 4. ISBN: 9780262042840

[8] **Goyal**, Gunjan. *Twitter Senitment Analysis Using Python | Introduction & Techniques* [online]. [Updated 03.12.2023]. [Viewed 11.01.2024]. Available from: https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/

[9] **Matplotlib: Documentation** [online]. [Viewed 11.01.2024]. Available from: https://matplotlib.org/

[10] **MonkeyLearn: Natural Language Processing (NLP): What is it & How does it Work?** [online]. [Viewed: 11.01.2024]. Available from: https://monkeylearn.com/natural-language-processing/

[11] **MonkeyLearn: Sentiment Analysis: A Definitive Guide** [online]. [Viewed 11.01.2024]. Available from: https://monkeylearn.com/sentiment-analysis/

[12] **NLTK: Documentation** [online]. [Viewed 11.01.2024]. Available from: https://www.nltk.org/index.html

[13] **Sentiment140 dataset with 1.6 million tweets** [online]. [Viewed 11.01.2024]. Available from: https://www.kaggle.com/datasets/kazanova/sentiment140/data

[14] **Socher**, Richard, **Perelygin** Alex, **Chuang**, Jason, et al. *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank* [online]. Stanford University, Stanford. [Viewed 11.01.2024]. Available from: https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf

[15] **VaderSentiment: Documentation** [online]. [Viewed 11.01.2024]. Available from: https://vadersentiment.readthedocs.io/en/latest/

[16] **Wikipedia: Logistic Regression** [online]. [Viewed 11.01.2024]. Available from: https://en.wikipedia.org/wiki/Logistic_regression

[17] **Wikipedia: Natural Language Processing** [online]. [Viewed 11.01.2024]. Available from: https://en.wikipedia.org/wiki/Natural_language_processing

[18] **Wikipedia: Sentiment Analysis** [online]. [Viewed 11.01.2024]. Available from: https://en.wikipedia.org/wiki/Sentiment_analysis