**Технологично училище „Електронни системи“ към**

**Технически Университет - София**

**ДИПЛОМНА РАБОТА**

Тема: „Система за менажиране на Facebook страница”

|  |  |
| --- | --- |
| Дипломант: | Научен ръководител: |
| Иван Ивайлов Милев | Александър Чернаев |

СОФИЯ

2018

# Отзив на научния ръководител

Настоящата дипломна работа отговаря на изискванията в заданието и може да бъде допусната до защита.

|  |  |
| --- | --- |
|  | Ръководител:…………………………. |
|  | Александър Чернаев |

# УВОД

В днешно време човечеството се опитва да автоматизира всяко възможно действие. Още от индустриалната революция хората започват да бъдат заменяни от машините, което продължава и днешно време, само че от софтуер. Тази автоматизация позволява човешкото внимание да бъде фокусирано върху по-сложни и по-интересни проблеми пред, в много случаи, повтарящи се еднообразни действия. Освен това, автоматизацията увеличава продуктивността и качеството спрямо човешката работа, тъй като типично машините (софтуерът) не правят грешки и не се изморява. От другата страна идва и интернет със социалните мрежи, които позволяват неограничено общуване между хората по света. Те позволяват неограничено, а често и неконтролируемо дискутиране на всякакви теми, без ограничение дали съответната дискусия е на правилното място и с правилния речник. В общия случай хора и компании, които имат собствени страници в социалните мрежи биха желали да я поддържат без различни обидни коментари и дискусии сред постовете си. За тази цел често се назначават хора, които да отговарят за социалната страница на съответна фирма, но тяхната задача, с изключение на създаването на нови постове за страницата, е доста еднообразна, като трябва да следят коментарите и съответно да се стараят да поддържат позитивна среда в „общността“. Друга тяхна задача е да отговарят на въпроси, зададени от различни потребители по света, не рядко отговорът на техният въпрос вече е отговорен някъде в интернет и това също обезсмисля до някаква степен труда на назначената поддръжка. Тук идва и момента, в който да се замислим за автоматизация.

За целта е разработен Меринджей бот с две основни функции:

1. Да следи за недобронамерени коментари;
2. Да отговаря на задавани въпроси от хора.

Недобронамерената реч е проблем, с който големите компании се борят от не малко време. Първите похвати са се състояли в построяването на големи лексикони, които съдържат в себе си недобронамерени думи и изрази. Разбира се този похват не е много успешен, защото има твърде много думи, които могат да бъдат използвани непрекомерно, но те сами по себе си не носят отрицателния смисъл, който носи израза. След това се появяват похвати, базирани на машинното самообучение. За обучението на съответните класификатори се използват различни похвати за обработка на естествен език, които позволяват числова репрезентация на езика, с която компютърът може да работи.

Чатботовете са програми, които започват своето развитие още през 60-те години на 20 век. Първият чатбот е „ELIZA“. Идеята на повечето такива ботове е когато човек разговаря с тях да не може да разбере, че срещу него стои машина, а не човек. За да „разбере“ бота какво се изисква от него, като отговор се използват различни похвати за обработка на естествен език. При отговор на фактологични въпроси, един от основните похвати е да бъде намерен отговорът на съответния въпрос в голям ресурс от данни (интернет) и да бъде направена или извадка, или резюме от него като резултат.

Меринджей бот използва машинно самообучение за разпознаването на недобронамерена реч и за свой чатбот имплементира разпознаване на въпроси, и съответно отговарянето им чрез извадка.

# ПЪРВА ГЛАВА

## Съществуващи решения

### eMore

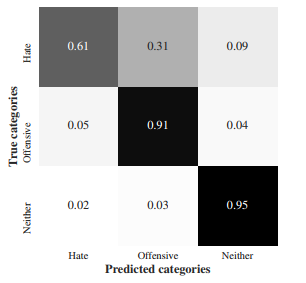
eMore е проект, който се старае да помогне с тестването и предаването на модел за онлайн класифициране на недобронамерена реч и престъпленията свързани с нея. Модела е базиран на напреднали системи за наблюдение и докладване, за да се създаде разбиране за теченията създавани в интернет мрежите и да позволят техният анализ на национално/европейско ниво и да подкрепят общата борба срещу грубите обиди на европейско/национално ниво.

Моделът ще включва най-важните категории на недобронамерени коментари, а именно полова омраза, расизъм, религиозна омраза, омраза свързана със сексуалната ориентация и инвалидност. [1]



### Automated Hate Speech Detection and the Problem of Offensive Language

Проектът е труд на Томас Давидсон, Дана Уармсли, Майкал Мейси и Ингмар Уебер относно автоматизирането на разпознаването на недобронамерена реч. В техният труд те приемат три основни варианта за едно изречение. То може да бъде или обида(hate) или офанзивно(offensive) или нормално тоест без да засяга никой в негативен аспект. Те не приемат обичайният подход, който досега е ползван, а именно лексикони и похвата в обработката на естествена реч „торба с думи“(bag of words). Те използват n-грами за репрезентация на изреченията, но освен тях добавят и допълнителни признаци освен самите изрази и думи. Тъй като труда им е фокусиран върху недобронамерената реч в социалните мрежи допълни признаци, които подбират са препратките към други сайтове, дължината на изреченията, това дали автора на съобщението е тагнал някой в него, броя на гласните, съгласните, анализ на това дали изречението е положително или отрицателно и резултат за четливостта му (Flesch-Kincaid Grade Level и Flesch Reading Ease). Използват данни генерирани от социалната мрежа Twitter. Резултатите им са забележителни и могат да се видят на фигура 1.1. [2]

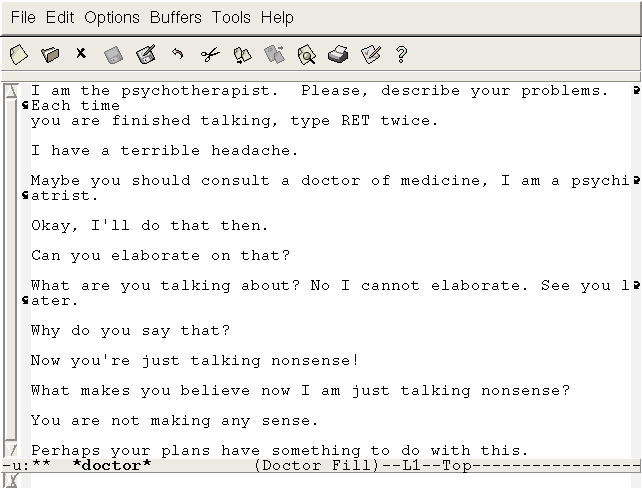
  
<https://aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/view/15665/14843>

Фигура 1.1 – резултати от модела на проекта за разпознаване на обидна и недобронамерена реч.

Hate –обида, Offensive- офанзивна реч и Neither нито едно от двете.

### ELIZA

ELIZA е програма за обработка на естествен език създадена през 60те години на 20ти век в MIT от Джозеф Уайзенбаум. Програмата е замислена с идеята да може да поддържа разговори между човек и машина. ELIZA разговаря посредством нагаждане на модели (pattern matching) и така дава илюзията на човек, че разбира съобщението, но не имплементира в себе си контекст на разговора. Програмата използва скриптове, които позволяват на ELIZA да обработва въведеното от човек и спрямо обработката да следва дадените правила и насоки на скрипта. Най-известният от тези скриптове е DOCTOR(доктор), който симулира психотерапевт като отговаря с въпроси, които не са директни. Така ELIZA е една от първите чатботове и е първият, който успява да премине през теста на Тюринг. [3]



### Watson

Watson е компютърна система за отговаряне на въпроси зададени с естествен език. Развит от екипа на IBM DeepQA проекта е ръководен от Дейвид Феручи. Watson е кръстен на първия изпълнителен директор на компанията - Томас Уотсън. Системата е създадена специално за да отговаряна на въпросите от теста на американското шоу Jeopardy! и през 2011 Watson участва срещу победителите Брат Ръттер и Кен Дженингс като печели първо място и награда от 1 милион долара.

По време на състезанието Watson имал достъп до 200 милиона страници информация, но не е бил свързан към интернет. Watson консистентно се справял по-добре от противниците си с изключение на няколко категории, за които е имал по-малко данни.

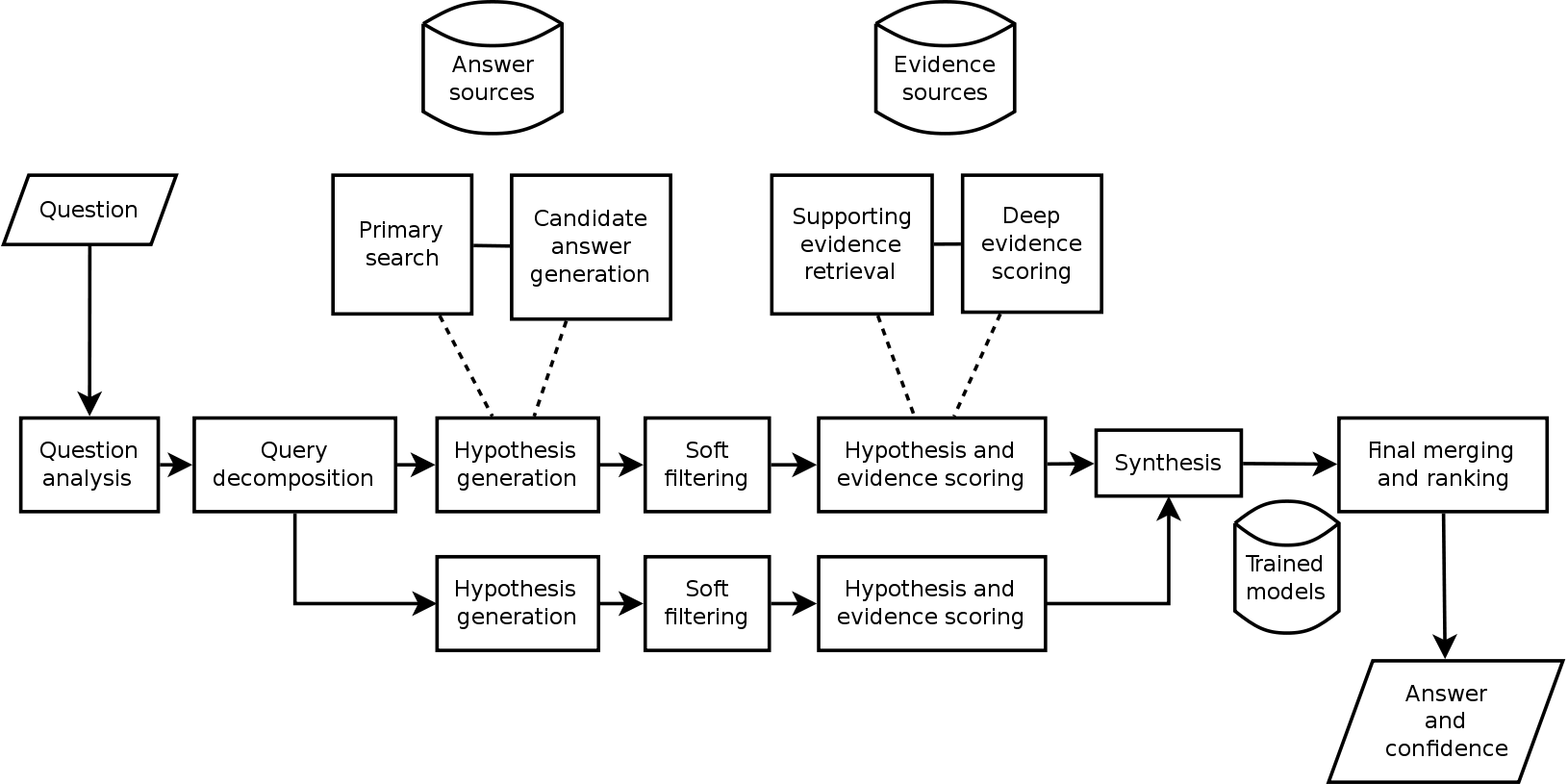
През 2013 IBM обявяват чe Watson е първата комерсиализирана система, която ще бъде използвана при вземането на решения относно лечението на рак на белите дробове.

Watson е създаден, за да имплементира напреднала обработка на естествен език, извличане на информация, репрезентация на знания, автоматично разсъждение и машинно самообучение. [4]

Алгоритъма на работа на Watson може да бъде видян във Фигура 1.2.

The key difference between QA technology and document search is that document search takes a keyword query and returns a list of documents, ranked in order of relevance to the query (often based on popularity and page ranking), while QA technology takes a question expressed in natural language, seeks to understand it in much greater detail, and returns a precise answer to the question.

-Richard Craig



<https://en.wikipedia.org/wiki/Watson_(computer)#/media/File:DeepQA.svg>

Фигура 1.2

## Проучване

### Машинно самообучение

Машинното самообучение е науката, която позволява на компютрите да изпълняват дейност без да бъдат експлицитно програмирани за нея. Идеята на програмите използващи машинно самообучение е да решават даден проблем като се обучат на базата на примерни данни и/или опит. През последните десетилетия машинното самообучение е дало много нови възможности в лицето на самоуправляващи се коли, гласово разпознаване, по-ефективно търсене в интернет и много по-задълбочено разбиране за човешкия геном. Машинното самообучение е толкова широко разпространено, че ежедневно се сблъскваме с него без дори да разбираме. Много изследователи смятат, че това е най-добрият начин за достигане на изкуствен интелект на човешко ниво.

Много важно при МС е и това да бъде оценяван дадения модел, за да може да се следи дали промените по модела са полезни или не. Затова и е обичайна практика данните да се разделят на такива за трениране тестване и кръстосано валидиране. Като тези за трениране се ползват, за да се тренира модел. Този модел прави предположение върху данните за трениране и тези за тестове, като предложението се оценява и така се следи за прогрес. И данните за кръстосано валидиране се използват, за да се намали и види по-добре дали модела не е твърде генерален и предполага твърде общо в класификацията.

### Контролирано машинно самообучение (Supervised learning)

При контролираното машинно самообучение алгоритъмът се учи върху данни, на които вече е надписан даден резултат. То се дели на два основни типа: класификация и регресия. Като целта в края е да има модел, който на базата на входни признаци в случая на класификацията да може да каже към кой клас принадлежи съответния вход или в регресните проблеми да даде число, което седи зад съответния вход. Типични проблеми, които се решават с този вид МС са проверки за спам, пресмятане на големината на риба спрямо възраст и тегло, цена на къща спрямо множество нейни признаци. [5]

### Класификация

При класификацията данните принадлежат към два или повече класа и тъй като класификацията е контролирано МС то се иска да бъде научено от вече попълнените данни как да бъдат различавани няколко класа на база на избрани от нас признаци. Примерни проблеми, които се решават с класификация са разпознаването на ръкописни числа като тук идеята е от картинка на ръкописно число да бъде класифицирана цифрата, на която тя съответства. Друг начин да бъде представена класификацията е, че за разликата от регресията резултата тук винаги е дискретен като идеята е за n примерни подадени входа да се опитва да бъде класифицира/категоризира всеки един от тях с правилният му клас/категория. [5]

### Регресия

При регресията данните нямат точни/дискретни класове, а по-скоро техните данни са продължителни*.* Класически проблеми, които се разрешават с регресия това са цените на къщи, височината на човек и т.н. [5]

### Безнадзорно машинно самообучение (Unsupervised learning)

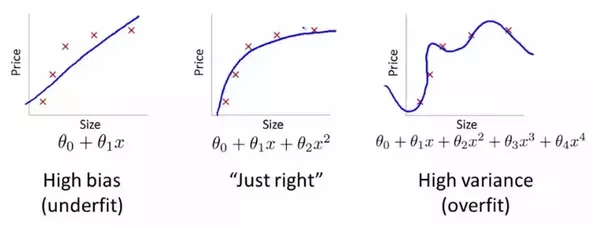
При този тип МС няма надписан резултат. Идеята на този тип МС е да открие групи от сходни примери в данните, което се нарича клъстеризация. Друг проблем, който се решава с не надзиравано МС е намаляването на дименсиите на данните от много дименсии до две или три с цел визуализация. [5]

### Оценяване на модела

Оценката в МС е едно от най-важните неща, защото тя дава реална метрика по която може да се следи развитието и работата на модел. Има два основни проблема при МС, които са Bias и Over fit. При Bias проблема е, че функцията на модела е твърде обща и това е причина за неточни резултати, а при over fit функцията е строго пригодена към данните за трениране и така, когато има нов пример преположението няма да бъде добър (Фигура 1.3).

Четирите метрики, които са използвани в дипломната работа са:

Първата е точността на модела, която дава колко процента от предположенията на модела са верни спрямо цялото количество данни.

  
<https://goo.gl/7yLLVT>   
Фигура 1.3 – графики на модели  
High Bias – много генерален, не достатъчно конкретен, Just right – най-добрият вариант, High variance (overfit) – твърде комплексен, много обвързан с данните за трениране.

Други важни метрики са precision(прецизност) и recall. Тези метрики са въведени поради факта, че е възможно при недостатъчно количество от данни при класификация да има добри резултати на точността въпреки, че реално не се разпознава някой клас както трябва и това ще си проличи при precision и recall метриките.

При всички формули tp са правилните позитивни примери, fp - неправилните позитивни примери, tn са истинските негативни примери и fn – не правилните негативни примери.

### Обработка на естествен език(Natural Language Processing)

Обработката на естествен език(ОЕЕ) е част от науката за изкуствен интелект, която се занимава с това да помага на компютрите да интерпретират естествения човешки език. ОЕЕ е комбинация от много науки включвайки: компютърните науки и компютърната лингвистика, за да доближи човешката комуникация с компютърните разбирания.

Въпреки че ОЕЕ не е нова наука технологиите използвани за ОЕЕ се развиват динамично благодарение на увеличилия се интерес към комуникацията човек-машина плюс многократно увеличилия се достъп до данни, мощната изчислителна мощ на днешните компютри и напреднали алгоритми.

За ОЕЕ има множество различни тактики и похвати, с които може да бъде интерпретиран човешкият език. От статистически методи до методи базирани на МС и различни алгоритмични похвати. Нужда от различни методи има, защото данните базирани на текст и разговори са крайно различни и всички биха могли да имат крайно различни приложения.

Обикновено задачите за ОЕЕ съдържат в себе се токенизация, парсване, лематизация, коернуване, откриване на частите на речта. Като цяло идеята на ОЕЕ е да приведе езика към елементарни кратки части и да разбере връзките помежду им като изследва как отделните части взаимодействат една с друга, така че да се получи израз, който носи в себе се смисъл.

На по-високо ниво в себе си ОЕЕ съдържа функционалности като: Категоризиране на съдържание, намиране на тема и моделиране, изваждане на ключови думи от текст, определяне на настроението на дадена информация(Sentiment analysis), говор към текст и обратното, резюмиране на документ и превод от един език на друг.

ОЕЕ винаги е съпровождана от анализ на текста, който брои, групира и категоризира думи, за да създаде структури, които да дадат значение на големите обеми от данни. Текстовият анализ се използва, за да може текстовите данни да бъдат визуализирани, филтрирани и дори използвани като вход за модели или статистически анализи. [6]

### Торба с думи(Bag of words)

Торба с думи(ТсД) е подход за изземане на признаци от текст, за да бъдат използвани за обучаване на МС модел.

Подходът е много прост и гъвкав и може да бъде използван без значение от типа документи, данни, на които се използва. ТсД представлява репрезентация на думите спрямо това колко пъти се срещат в документ. Всичко се свежда до две основни действия:

1. Създаване на речник с думите, които ще бъдат зачитани
2. Записване на броя срещания за всяка дума от речника в документ

Похвата е наречен торба с думи(Bag of words), защото при този похват не се пази подредбата на думите съответно и техния контекст. Този похват работи добре ако данните ни са със сходна тема, защото речникът няма да бъде твърде голям и думите(признаци), които се срещат рядко няма да бъдат твърде много. [7]

### Ngram-и

В компютърните лингвистики n-gram-ите са поредица от n на брой последователни части от даден текст или разговор. Частите може да са срички, гласни, букви думи и т.н. съответно според това от какво се нуждае приложението. Предимството им пред подхода на Торбата с думи е че дава контекст на думите. Реално Торбата с думи може да бъде представена и като Unigram т.е. N-gramи от една дума. [8]

### Коренуване (Stemming)

Различните документи типично съдържат в себе си повтарящи се думи, които обаче заради граматическите изисквания са в различни форми като: organize, organization, organizing въпреки че носят един и същ смисъл в общия случай. Освен това има думи, които са в различни форми(глагол, прилагателно, съществително) и отново смисъла, който различните форми носят е сходен. Идеята на коренуването е да намали тези разлики до минимум взимайки корените на думите в документите. Типично алгоритмите за коренуване на дума премахват наставките към думите оставяйки представките, защото често те показват дали съответната дума е позитивна или негативна(productive, unproductive). Най-популярният метод за коренуване на английски език е Porter stemmer. [9]

### Дърво на решението

Дърво на решението(Decision tree) е важен алгоритъм в МС, който съществува от дълго време. Алгоритъмът е базиран на надзираваното самообучение като е по-използван в класификационни проблеми. Алгоритъмът построява двоично дърво, което започва с бащин възел като той се разделя на два или повече възела, които съдържат условие и съответно могат отново да сочат към такива възли или към терминиращ възел, който дава крайния резултат на модела. [10]

### Произволна гора

Произволната гора(Random Forest) е алгоритъм, който може да работи както за класификационни проблеми така и за регресии. Произволната гора работи като обучава дървета на решението на различни части от данните като използва средно аритметичния резултат, за да увеличи точността и има контрол върху вероятността за over-fit. [11]

### Логистична регресия

В статистиката, логистичната регресия(Logistic regression) е модел за двоична класификация. Логистичната регресия е развита от Дейвид Кокс(David Cox) през 1958 година като двоичния модел се използва за намирането на клас, който базира решението си на един или повече признаци. [12]

### Маркиране на части от речта

Подобно на това, което се учи в училище – различаването на глаголи, прилагателни и т.н. в ОЕЕ една от типичните задачи е маркирането на частите на речта на едно изречение. В тези задачи думите се класифицират в следните класове: ADJ- adjective, ADP- ad position(on, of, at…), CONJ – conjunction (and, or, but, if while, ...), DET – determiner, article (the, a, some, most, every, which), NOUN (year, home, cost, time, Europe), NUM – numerical (1, 42, twenty-four, 1992, 13:30), PRT – participle (at, on, out, that …), PRON – pronoun (he, their, her, its, my, I …), VERB (go, stay, let, come),. – punctuation marks (., ?, !), X – друго. [13]

### Резюмиране на текст

В ОЕЕ има два основни похвата за резюмиране на текст: абстрактно и чрез извадка. Този използван в тази дипломна работа е чрез извадка като идеята на този метод е да избере изреченията, които носят в себе си най-много информация относно съответната тема. Абстрактният метод е по-близък до този, който хората използват а именно това да се направи резюме със собствени думи преразказвайки текста.[14]

# ВТОРА ГЛАВА

## Функционални изисквания

MerinjeiBot има три основни функционални изисквания. Първото от които е да може да разпознава коментари, които са недобронамерени и/или обидни. Второто изискване е да има чатбот, който да може да се прикачи към страница, на която сте администратор и да отговаря на въпроси посредством търсене на отговор в интернет. Третото последно изискване е системата да бъде интегрирана с Facebook. Интеграцията се състои в това потребителите посредством профилите си в социалната мрежа да могат да използват web интерфейса на MerinjeiBot като от него да могат да прикачат бота към коментарите на страницата си и да абонират чатбот към Messenger за страницата си. Освен това бота трябва да може да сканира коментарите на страницата и когато намери недобронамерен или обиден коментар да го изтрие. Чатбота трябва да може да се свърже към Messenger платформата за съответната страница и да отговаря на зададени от потребители въпроси.

## Език за програмиране

Езикът за програмиране, който е избран за реализацията на дипломната работа е python. Python е избран, защото вече имах опит с него и освен това езикът е широко разпространен в средите на машинното самообучение, което предоставя много ресурси, библиотеки и голямо общество от хора, които го използват за решаване на проблеми подобни на този на дипломната работа. Освен това python е не малко използван и за Web приложения особено за по-малки каквото е MerinjeiBot. За Web приложението за front end технологиите, които са избрани стандартните JavaScript, CSS, HTML като за CSS е използвана библиотеката Bootstrap.

## Библиотеки

### Numpy

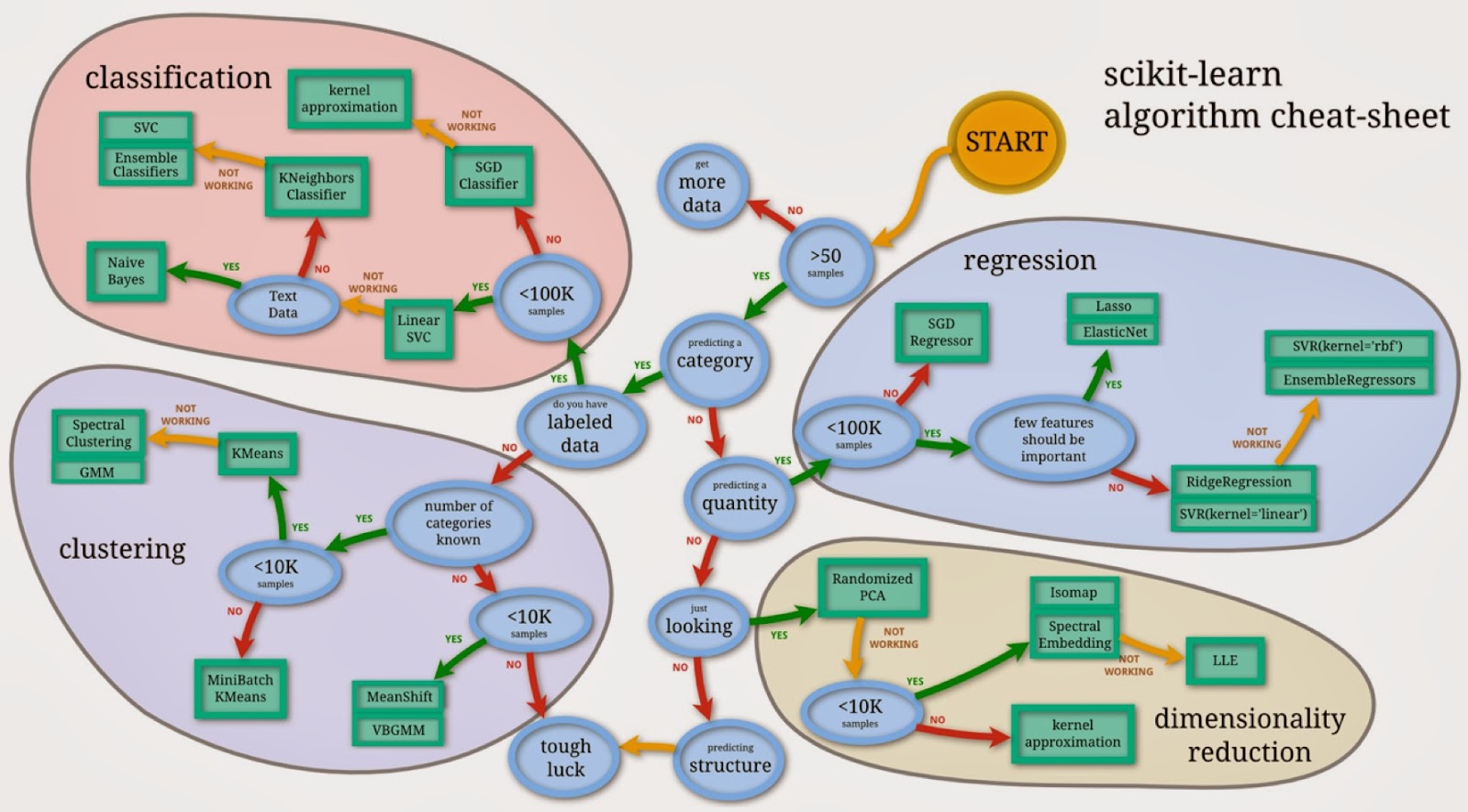
Numpy е фундаментален пакет за всеки програмен проект, който в себе си включва сериозни ресурсоемки изисквания към хардуера. Numpy е най-разпространената библиотека в научните среди на програмирането на python и в себе си съдържа много добра имплементация на матрици от високи дименсии. Векторизирани имплементации на множество функции върху матрици. Инструменти имплементирани на C/C++ и Fortran за максимална ефективност и скорост. Също има и инструменти за линейна алгебра, трансформации на матрици и добри възможности за произволни числа. Тази библиотека е избрана, защото е най-използваната, което предполага и най-много ресурси свързани с нея освен това sklearn изисква данните да бъдат във формат на матрица създадена от Numpy. Това се дължи на факта че машинното самообучение е изключително тежко от страна на изчисления и оптимизираните имплементации на Numpy матриците са изключителен бонус към операциите, които се извършват по време на обучение на модел. [16]

### NLTK

Библиотеката, която е избрана за ОЕЕ е NLTK. NLTK е една от водещите платформи за разработка на програми на python, които работят с данни от естествен език. Библиотеката предоставя лесни за използване интерфейси включвайки в себе си множество ресурси от данни. Библиотеката съдържа в себе си множество под-библиотеки за класификация, токенизация, коренуване, парсване, определяне на частите на речта, определяне на „емоцията“ и други. Основните причини за избора на тази библиотека е това, че тя е създадена за python и в себе си съдържа всичко, което би потрябвало в разработката на дипломната работа и далеч повече освен това е една от най-използваните, с което идват и множеството ресурси свързани с нея.[17]

### Sklearn

Sklearn е библиотека за МС. В себе си тя съдържа прости и ефективни инструменти за анализ на данни, класификация, регресия и т.н. Тя е базирана на numpy, което е изключително полезно, защото в дипломната работа са използвани numpy матрици като репрезентация на данните, които са узползвани. Избрах тази библиотека, защото има много добра документация с примери, които са написани достъпно и лесно може да се ориентира дори новобранец. Всички модели написани в библиотеката са написани от експерти в областта. Библиотеката е достатъчно богата на инструменти и обхваща повечето задачи свързани с машинното самообучение. Нейният широк обхват може да бъде видян на Фигура 2.1. [17]



<http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>

Фигура 2.1 - Алгоритмите, които библиотеката sklearn поддържа

### allauth

Allauth е Django базирана библиотека, която позволява на вашето приложение да използва автентикация от друго приложение, което е възможно с помощта на oAuth2. Тази библиотека е избрана след като бяха пробвани няколко други, с които имаше проблеми с тяхното настройване докато тази има добре описана и последователна документация, с която може лесно да се работи. Библиотеката е интегрирана много добре с Django софтуерната рамка, което дава възможност за добра и лесна персонализация без нуждата от писане на много допълнителен код. Allauth е интегрирана и със сесиите на Django, което позволява за лесно проследяване на това дали потребителя е автентициран.

## Софтуерни рамки

### Django

Django е Web рамка на python, която позволява изключително бързото и лесно разработване на Web приложения. Рамката е базирана на архитектурата MVT(MVC) и е устроена за възможно най-бързо реализиране на дадена идея. В себе си Django съдържа множество екстри свързани с обработката на често използваните в Web разработката задачи. Django е създаден, така че да бъде максимално защитен като в себе си имплементира защити против SQL Injection, cross-site scripting, cross-site request forgery и clickjacking. Освен всичко Django приложенията могат и да растат в мащаб и са доста гъвкави в отношения на голям трафик насочен към тях. Django бе избран, за рамка на която да бъде разработена дипломната работа по няколко причини. Първата от които е факта, че вече имах опит с Django. Други причини са, това че Django имплементира архитектурният модел MVC, с който съм добре запознат и имам опит. Предимства на Django пред други софтуерни рамки в същата категория като Flask и Pyramid са броя разработчици, които го използват и съответно материалите, които има за Django заедно с това идват и множество помощни библиотеки създадени за него. И последната причина, която не е фокус в дипломната работа е базата данни т.е. това, че Django използва по подразбиране релационна база данни с каквото имам най-много опит и се чувствам комфортно.

## Развойна среда

### Visual Studio Code

Visual Studio Code е текстови редактор с отворен код. Избрах да разработвам проекта на него тъй като в себе си текстовият редактор съдържа множество плъгини, които помогат при разработката на python приложение. След инсталацията на пакетите свързани с python той дава възможност за допълване на променливи, отиване при дефиницията на дадена функция или променлива. Редактора също дава предупреждения за лоши практики при писането на код. Други много удобни функционалности са вградения терминал, чрез който мога да пускам приложението и удобната лента с файлове и директории на отвеният проект както и функционален терминал, чрез който могат да бъдат извиквани с помощта на команди различни функционалности на редактора.

## Описание на алгоритъма на работа

Работата на ДР започва с абонирането на потребителя към коментарите и/или Messenger на своята страница във Facebook. След като вече програмата е свързана с webhook-ове към Facebook при поява на нов коментар той бива изпратен към Django сървъра, който има крайна точка(endpoint), която приема заявките изпратени от Facebook.

При получаване на информацията от Facebook се изваждат самият коментар както и идентификаторите на коментара и страницата, от която идва. Коментарът се взима и обработва като обработката му се състои в извличането на признаци от него т.е. разделянето му на uni, bi, tri-gram-и, добавяне на uni-, bi-, tri-gram-и на частите на речта на коментара, броя на гласни, съгласни, броя на всички букви, броя споменати хора, точкуването му спрямо Flesch-Kincaid Grade и Flesch Reading Ease. След това класификатора за недобронамерена реч го оценява и ако коментара бъде сметнат за недобронамерен бива изтрит посредством GraphAPI на Facebook с Delete заявка.

Поради промяна в правилата и условията на Facebook алгоритъма на работа за коментарите е променен. Тъй като според новите права и условия Facebook не разрешават свободното използване на Facebook Page Webhook-ове дори ако потребителите са тестери на приложението. За това е разработен втори вариант, който прави polling на Facebook страница на всеки 5 секунди и така сканира страницата. Това решение е активно докато на приложението не му бъде позволено да използва Webhook за страница във Facebook.(Фигура 2.3)

Когато MerinjeiBot получи съобщение от Messenger той пъро отново го обработва като взима unigram или създава торба с думи(Bag of words) от частите на речта му и с класификатора на въпроси се опитва да предположи типа на въпроса като ако не е достатъчно сигурен какъв е въпроса то не го смята за такъв и посредством GraphAPI на Facebook връща съобщение, че не е разбрал какво точно го питат. Ако обаче класифицира въпроса, то ако е от тип класа ABBR – абревиатура, DESC – описание, PROCEDURE – процес, LOC – локация или NUM – числова стойност започва процес по намирането на отговор. Първо се прави изватка от въпроса, с която се прави търсене в StackOverflow посредством отворения им API. След като са намерени подобни на нашия въпрос се взимат първите три въпроса с техните най-добри отговори и върху тях се прави кратко резюме до 50 думи и отговорът се връща чрез GrapgAPI. Pipeline-а на алгоритъма на работа е описан в фигура 2.2

  
Фигура 2.2 - Описание на алгоритъма на работа

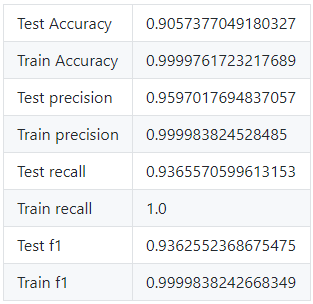


Фигура 2.3

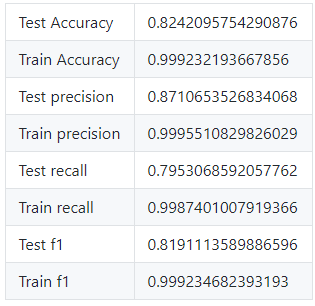
# ТРЕТА ГЛАВА

## Класификация на недобронамерена реч

В началото разработката започна с данни от ревюта от Amazon като те бяха разделени на две категории- позитивни и негативни. Тъй като данните са представени като торба с думи(Bag of words) това бе и подхода, който бе използван. В самото начало се изполозваха всички думи като признаци, които бяха около 300 000 думи. Това бавеше много модела и освен това много от признаците бяха неизползваеми поради твърде рядкото им срещане. Затова бе приложено коренуване и премахване на думите, които не носят никаква смислова стойност(stopwords) и така признаците бяха сведени до около 25 000. Върху тези признаци бяха тренирани множество модели, от които Произволната гора показа най-добри резултати. (Таблица 3.1)

  
Таблица 3.1

Гореописаните резултати изглеждад добри, но те все още не предполагат дали дадена реч е обидна. За момента това, което те описват е дали дадено ревю е позитивно или не. Затова се наложи добавянето на данни, които съдържат в себе си недобронамерена реч. Данните изполозвани за целта са от twitter като те в себе си съдържат 25 000 примера, от които 4000 омразни и 8000 недобронамерени обиди, расизъм и т.н. след добавянето им нямаше голяма промяна в резултатите, но при мои тествания забелязах, че моделът не разпознава добре недобронамерената реч. Добрите резултати се дължат на малкия процент от данни, които всъщност са от данните на twitter около 1/3, от които 20% участват в тест данните. Модела смята всичко за обиди при тестове от мен на ръка. Таблица 3.2 показва резултата от метриките на най-добре справящия се модел – Произволна Гора(RandomForest).

  
Таблица 3.2

Следващата стъпка бе добавянето на признаците от данните на twitter. Всички думи са коренувани и така се получават около 30000 уникални думи или признаци. Те бяха толкова много поради факта че се включваха и имената на различните потребители отбелязани в съобщенията след премахването им признаците намалияват до около 6700. След обединяването им с признаците от данните на амазон се получават 48000 признака. След обучаване на модел и премахване на тези, които модела не използва ефективно остават около 13000 признака.

Пъвро бяха обучени множество класификатори с данни само от twitter и резултатите на хартия не изглеждат зле, но въпреки това трудно се получа ват примери, които модела да определя за позитивни особено в кратки изречения съдържащи в себе си по-малко от 6, 7 думи. Резултатите са представени в (Таблица 3.3)

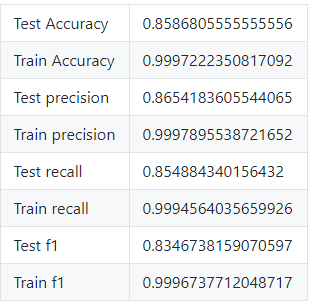
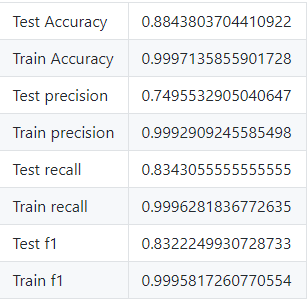


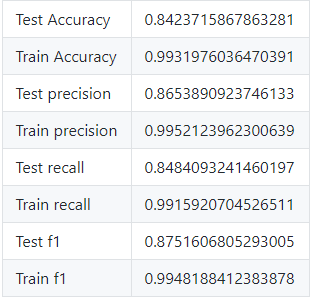
Таблица 3.3

След горните незадоволителни резултати последва обучение на модела със смесените признаци от амазон и twitter данните. Този път резултатие не са толкова добри, колкото при горният резултат (Таблица 3.4), но при проба на ръка се справя по-добре, като проблема с твърде кратките изречения все още не е разрешен. Както се вижда и от резултата, точността е малко по-малка. Според мен това се дължи на факта, че модела предполага негативни примери много „по-лесно“ от колкото позитивни.

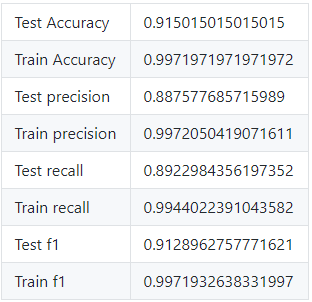
  
Таблица 3.4

Следващата логична стъпка предвид факта, че модела има проблем с късите изречения бе това да бъде намалено влиянието на размера на коментара като за тази цел е използван TF IDF. Това, което прави term frequency–inverse document frequency е да дава оценка или числова репрезентация на различните думи спрямо това колко често те се срещат в документа като това намаля влиянието на това колко дълго е едно изречение. Съответно най-висок резултат имат най-срещаните думи, а тези които не се срещат остават с 0.

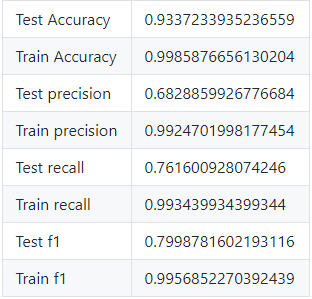
Приложен е TF IDF векторизатор от библиотеката sklearn директно върху вече направената Торба с думи(Bag of words). Резултатите се подобряват спрямо миналите тестове освен това и при лични тестове вече модела се справя по-добре, но все още имаше примери, при които бърка. Като модела е доста сигурен в избора си с процент от около 90% сигурност. (Таблица 3.5)

  
Таблица 3.5

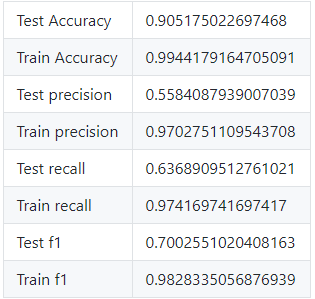
С идеята, че за реално позитивните изречения нямат значение е пробвано обучение на класификатора, отново с TF IDF ,да се тренира само на данните от twitter като се получават много добри резултати (Таблица 3.6). Въпреки добрите резултати при тестове клаисификатора винаги счита коментара/изречението за негативно. Това се дължи на небалансираните данни, които в себе си съдържат 8000 не негативни изречения и 17000 негативни.

  
Таблица 3.6

Следващата логична стъпка бе смяната на подхода от Тобра с думи към ngram-и като това вече дава и контекста на всяка дума. Първо бе опитано с Trigrams (три последователни думи). Резултатите не са много добри, но са добро начало (Таблица 3.7). Голям проблем бе факта, че този подход изисква много RAM поради факта, че имаше един милион признака. За това бе направено проучване в интернет за различни подходи с ngram-и и бе намерен документ, в който бе описан подход чрез, който се получават добри резултати за намиране на недобронамерена и обидна реч.[2]

  
Таблица 3.7

Следвайки проучването, бе добавено към вече съществуващите ngram-и и частите на речта на всяко съобщение като те също се добавят като ngram-и. Освент това има и допълнителни признаци като броя на гласни, съгласни, hashtags(хаш тагове), тагвани хора, броя на букви, дали изречението е позитивно или отрицателно с помощта на sentiment analysis, метод от библиотеката NLTK, който е базиран на Vader sentiment analysis, който подход работи с лексикони, колко четимо е дадено изречение според Flesch-Kincaid Grade и Flesch Reading Ease точкуване. (Таблица 3.8).

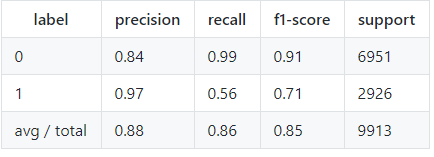
  
Таблица 3.8

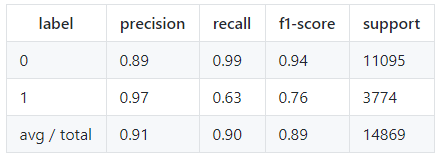
Тъй като при прилагане на обикновените формули за accuracy, recall и F1 score се базират само върху положителните примери се наложи смяна на методите с метода от sklearn:

classification\_report(test\_labels\_pred, test\_labels\_true)

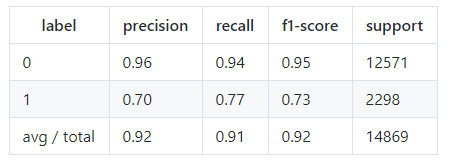
Така с него могат да бъдат видяни резултатите и за позитивните и за негативните примери във формата на таблица 3.9 за тестовите данни (таблица 3.10) и за данните, на които е трениран модела.

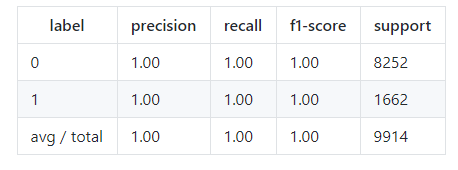
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

  
Таблица 3.9

  
Таблица 3.10

Въпреки добрите резултати все още се наблюдават не малко премери, при които класификатора прави грешки и не ги смята за обидни. За това бяха добавени лексикон и относно частите на речта също е приложен TF IDF тъй като до този момент не е добавен. Тези промени оказват положителен резултат върху резултата на класификатора. Резултатите за тестовите данни са в таблица 3.11, а за тренираните в 3.12.

  
Таблица 3.11

  
Таблица 3.12

### Обработка на данните

#### Използвани признаци

За да бъдат постигнати тези резултати а използвани признаците: частите на речта за всяко съобщение (tweet), думите на всяко съобщение (tweet) представени като unigram, bigram и trigram трансформирани с TF IDF, броя на споменатите интернет адреси(URL), броя на споменатите потребители, броя на хаштагове, броя на думите, броя на term-овете, броя на уникалните term-ове, броя на гласни, броя на буквите, броя на всички символи, средния брой на гласни на дума, точките по Flesch-Kincaid grade level с формулата:

fkra\_score = (0.39 \* number\_of\_words) + ( 11.8 \* number\_of\_syllables / number\_of\_words) - 15.59

точките от Flesch reading ease с формулата:

fre\_score = 206.835 - (1.015 \* number\_of\_words) - (84.6 \* number\_of\_syllables / number\_of\_words)

За ngram-ите на думите е използван TfidfVectorizer от sklearn по следния начин:

TfidfVectorizer(use\_idf=True,

tokenizer=PreprocessHateData.tokenize, stop\_words=nltk.corpus.stopwords.words('english'),

decode\_error='replace',

min\_df=5,

max\_df=0.75,

max\_features=10000,

smooth\_idf=False,

norm=None,

ngram\_range=(1, 3),

token\_pattern=r'\b\w+\b')

като както се вижда се използва метода tokenize, за да се {\displaystyle {\text{Recall}}={\frac {tp}{tp+fn}}\,}токенизират съответните изречения(специалното на този метод е че използва SnowballStemmer, за да коренува думите).

def tokenize(tweet: str) -> list:

stemmer = SnowballStemmer("english")

tokens = tweet.split()

tokens = [stemmer.stem(w) for w in tokens]

return tokens

Параметъра stop\_words представлява списък, в който са записани думи, които да не бъдат зачитани при генерирането на ngram-и. В случая се използват списък от nltk, който съдържа в себе си 179 думи в себе си за английски език.

Decode\_error параметъра се използва в случаи, при които данните ни съдържат символ чийто енкодинг не е подходящ, по подразбиране опцията генерира изключение от тип UnicodeDecodeError.

Следващите два параметъра са свързани с речника, който се генерира от TfidfVectorizer-а като първият е min\_df, който описва минималния брой срещания на дадена дума или ngram-а, за да бъде добавена в речника в случая това е поне 5 пъти. Може да се даде и число с плаваща запетая(float) и така вече параметъра ще бъде отчитан процентно(числото е от 0.0 до 1.0).

Max\_df е съответно максималният брой на срещания всичко над този брой няма да бъде добавени в речника. В случаят параметърът е процентен и всичко, което се среща в над 75% от примерите няма да бъде добавяно към речника. Отново може да бъде използвано цяло число и тогава лимитацията ще е от грубия брой срещания. В случаят избрах 75% по интуиция, защото всичко над това не би носило информация, а по-скоро е често използвана дума.

max\_features е параметърът, който ограничава броя признаци, които векторизаторът ще даде. Ако параметърът не е инициализиран и е None то тогава ще бъдат взети всички признаци. В случая се взимат 10000 като се взимат тези с най-висока срещаемост измежду данните. Ограничението е направено, за да може класификатора да може да се обучава достатъчно бързо и типично ако се вземат всички признаци по-голямата част от тях няма да оказват никакво влияние върху решението на модела.

smoot\_idf по подразбиране е True, което ще добави по единица към тежестите като симулира добавянето на един допълнителен документ, който в себе си съдържа точно по един път всеки признак.

Ngram\_range – приема tuple, като това е обхват, с който ще генерира ngram-и в случая ще бъдат генерирани unigram, bigram и trigram-и.

token\_pattern – подава се регулярен израз, който да бъде използван за разделянето на думите от корпуса. В случая регулярният израз намира думите със символи от a-z, A-Z, 0-9 и \_.

За генерирането на частите на речта отново е използван TfidfVectorizer. Отново се използват и unigram, bigram и trigram-и. За разлика от обикновените думи тук няма stopwords както и tokenizer, защото няма нужда от коренуване или каквито и да е специални действия, единствено се изисква разделянето на думите по идентация. Друга разлика е и че този път се взимат само 5000 признака. Важно е да се отбележи, че преди да се маркират частите на речта се извършва преработка на съобщенията като се премахват URL-и, имена на потребители и хаштагове(#).

def replace\_mentions\_urls(self, tweet, replace\_url='', replace\_mention='', replace\_hashtag=''):

url\_regex = re.compile(

r'(?:http|ftp)s?://'

r'(?:(?:[A-Z0-9](?:[A-Z0-9-]{0,61}[A-Z0-9])?\.)+(?:[A-Z]{2,6}\.?|[A-Z0-9-]{2,}\.?)|'

r'localhost|'

r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})'

r'(?::\d+)?'

r'(?:/?|[/?]\S+)', re.IGNORECASE)

mentions\_regex = re.compile(r'@\w+')

hashtag\_regex = re.compile(r'#\w+')

tweet = replace\_url.join(re.split(url\_regex, tweet))

tweet = replace\_mention.join(re.split(mentions\_regex, tweet))

tweet = replace\_hashtag.join(re.split(hashtag\_regex, tweet))

tweet = ' '.join(re.split(r'\s+', tweet))

return tweet

Тук по-специален е регулярния израз, с който се разпознават линкове. Другите са по-прости като се състоят от @ и # съответно последвани от, който и да е символ за дума. Този за линкове търси стринг, който започва съответно с http/s или ftp/s, след което е последван от израз съдържащ в себе си букви и цифри т.е. домейн като проверява и за портове. Вторият параметър посочва това да не се гледа за вида на буквите т.е. дали са малки или главни.

За тази класификация e използван Логистична регресия (LogisticRegression) от sklearn. Като изборът му е базиран на обучението на множество класификатори и прегледа на техните резултати. За да бъде подбрани най-добрите параметри за Логистичната регресия се използва GridSearchCV, който пробва всички дадени му параметри и дава най-добрият класификатор с неговите най-добри параметри.

param\_grid\_l2 = {

'C': [0.001, 0.01, 1, 10, 100, 1000],

'penalty': ['l2'],

'dual': [True, False],

'fit\_intercept': [True, False],

'warm\_start': [True, False],

'n\_jobs': [-1]

}

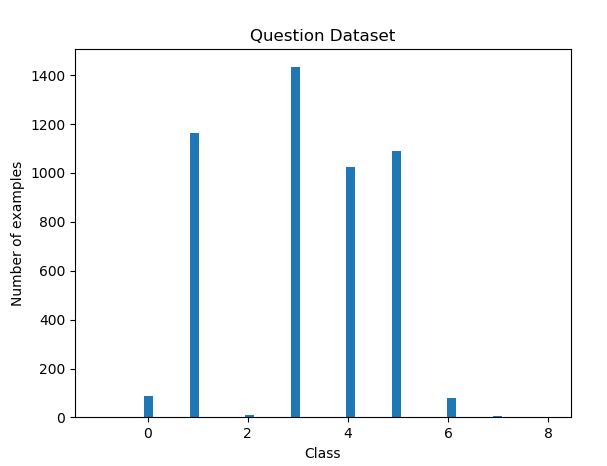
clf = GridSearchCV(logistic\_regression, param\_grid\_l2)

## Чатбот

### Класификация на въпроси

Изискването към чат бота е да може да отговоря на зададени му от потребителя въпроси, като за източник на информация използва интернет или по-конкретно в случая StackOverflow.

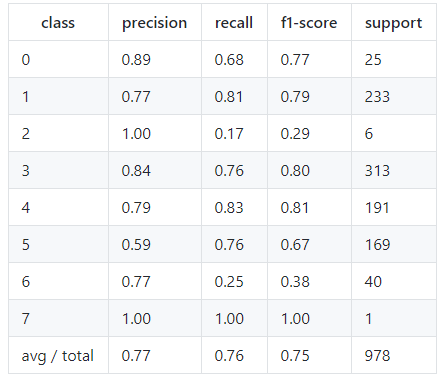
Похвата, с който бе започнато е първо да бъде направена проверка за типа на отговора, който въпроса очаква от съответните шест класа ABBR – абревиатура, DESC – описание, PROCEDURE – процес, HUM – човек/фирма, LOC – локация, NUM – числова стойност. За данни съм използвал два източника единият с 5500 резултата от различните типове, а другият с 1500 взети заедно около 7000 примера (Фигура 3.1 - Разпределение на данните).

Фиг.3.12 – класовете са съответно: 0-ABBR, 1-DESC, 2-PROCEDURE,

3-HUM, 4-LOC, 5-NUM.

#### Признаци

В началото за признаци бе използвана торба с думи(bag of words) от самите думи като преди да бъдат добавени към торбата биват коренувани, но тъй като въпросите, които има в данните са с много различен контекст и се получават множество признаци, от които няма полза, а и също така няма въпроси от естеството на тези в StackOverflow затова сметнах за по-коректно да бъдат използвани частите на речта за признаци отново представени като торба с думи (bag of words). Изборът бе базиран на това, че въпросите типично имат строго структура и често съдържат сходни части на речта като например върпосителна дума . За модел бе избрана произволната гора на база на експерименти върху множество модели, от които произволната гора(Random Forest) даде най-добър резултат. С този похват са постигнати задоволителни резултати в Таблица 3.11 на данните за тест и Таблица 3.12 на данните за трениране. И в двете таблица класовете отговарят съответно на: 0 - ABBR, 1 - DESC, 2 - PROCEDURE, 3 - PERSON, 4 - LOCATION, 5 - NUMBER, 6 - ORGANIZATION, 7 – CAUSALITY

   
Таблица 3.11

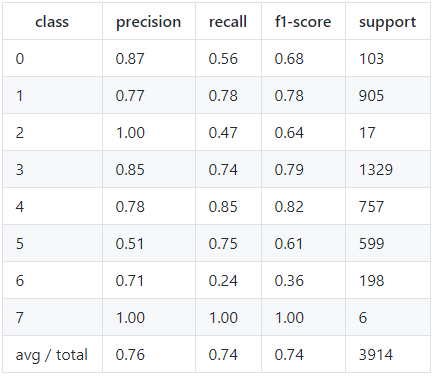


Таблица 3.12

#### Приложение на класификатора на въпроси

За отговор се използват ресурси от Stack Overflow за достъпването им е описано по-надолу във главата с Facebook интеграция. Тук е засегнат въпроса за това как се използва класификаторът на въпроси и как се прави резюмирането на отговор. Класификатора на въпроси е използван за две основни неща в ДР като първото, от които е да определи дали дадено съобщение е въпрос или не. Това се постига като се използва сигурността на модела в предположението му за типа на въпроса при сигурност в решението под 30% въпроса е сметнат за невалиден. Второто нещо, за което се използва модела е при извикването на StackOverflow API, при което се връщат множество въпроси и техните отговори. За това кои въпроси ще бъдат избрани се взимат трите въпроса от същия тип на зададения от потребителя, като се сравняват посредством модела за класификация на въпроси.

#### Изваждане на ключови думи за търсене

Подхода за извадка на ключови думи за използване, който бе предприет е доста прост тъй като самият филтър за търсене на StackOverflow е много добър и вероятността да бъде подобрен е минимална затова единственото, което се прави като обработка на въпроса изпратен от потребителя е да се премахне въпросителната дума както и някои думи които не носят информация като глаголът съм (to be), пунктуационни знаци, както и определителни местоимения determiners. В последствие се обаче се оказва, че това реално не помага на търсенето и се наложи избора на друг подход за изваждане на ключови думи. След кратко проучване на различни алгоритми за търсене на ключови думи бе избран Rake(Rapid Automatic Keyword Extraction), защото е един от малкото алгоритми, които могат да работят с малки изречения, а не с цели текстове. След като алгоритъмът е приложен на изречението се взимат ключовите думи и с тях се търси в StackOverflow.

#### Резюмиране на отговор

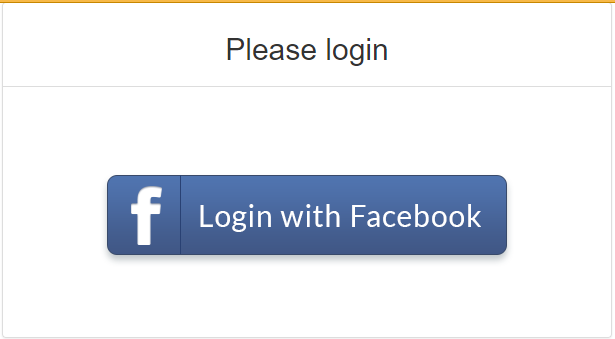
За резюмиране на отговор е използвана функцията от genism библиотеката summarize. Преди отговорът да бъде даден на метода за резюмиране той бива преработен. Обработката му се състои в две основни функционалности. Първата от които е премахването на html тагове от текста, за да бъде нормален текста само с изречения от отговора също така бива и премахнат кодът от отговора.

Функцията gensim () е базирана на PageRank алгоритъма оценява важността на всяко едно изречение и взима тези, които имат най-висок резултат. Алгоритъмът PageRank е разработен от Google и е използван за оценяване на различните интернет страници като така сравнява търсенето на потребителя с релевантността на различните страници в интернет. В дипломната работа взима най-добрите изречения докато не надвиши 50 символа резюмираното съобщение.

## Интеграция с Facebook

Интеграцията на Facebook започна първо с автентикацията на потребителите. За целта е използвана библиотеката allauth, която предоставя готов middleware както и готови функции за вход посредством Django template-ния език. Настройките на библиотеката са много прости състоят се в това да се посочи URL, на който след успешен вход потребителят да бъде препратен. Разбира се автентикацията не би била възможна без създаването на приложение във Facebook. След създаването на такова приложение във Facebook се добавят нужните му функционалности в случая на MerinjeiBot трябва да бъдат активирани Facebook Login, който се ползва за автениткиция, Messenger за чат бота и Webhooks, които се използват за абониране на приложението към коментарите на страница за следене на недобронамерена реч и за изпращане на Messenger съобщения към MerinjeiBot, за да бъдат отговорени ако са въпроси. Също от модула за управление на Facebook приложението e добавена и опцията за вградената ОЕЕ(NLP) относно Messenger съобщения, който може да класифицира в следните класове: Greetings (English only), Тhanks(English only), Bye (English only), Date/Time, Amount of money, Phone number, Email address, Distance, Duration, Quantity, Temperature, от които за момента се използват Greetings(поздрави за посрещане-„Hi, Hello, Good morning“), Thanks(съобщения за благодаря) и Bye(съобщения за довиждане) съответно, за да може бота да бъде по-дружелюбен.

MerinjeiBot има три web екрана. Първият е този за вход фигура 3.2, чието единствено предназначение е да се автентикираш с профила си от Facebook посредством гореописаните методи и библиотеки.



Фигура 3.2

Вторият екран е контролният, от който потребителят може да „закачи“ към своята страница съответно защита и проверка на коментарите на страницата и/или чат бот за messenger. (Фигура 3.3)

Третият екран предоставя изтритите съобщения от коментарите на съответна страница. (Фигура 3.4)

За да се популира тази страница се правят първо две заявки към Facebook. Първата от които е за взимането на данни на потребителя в случая профилна снимка и име. Това става с заявка към:

https://graph.facebook.com/v2.11/me?fields=picture,name&access\_toke='+ access\_token

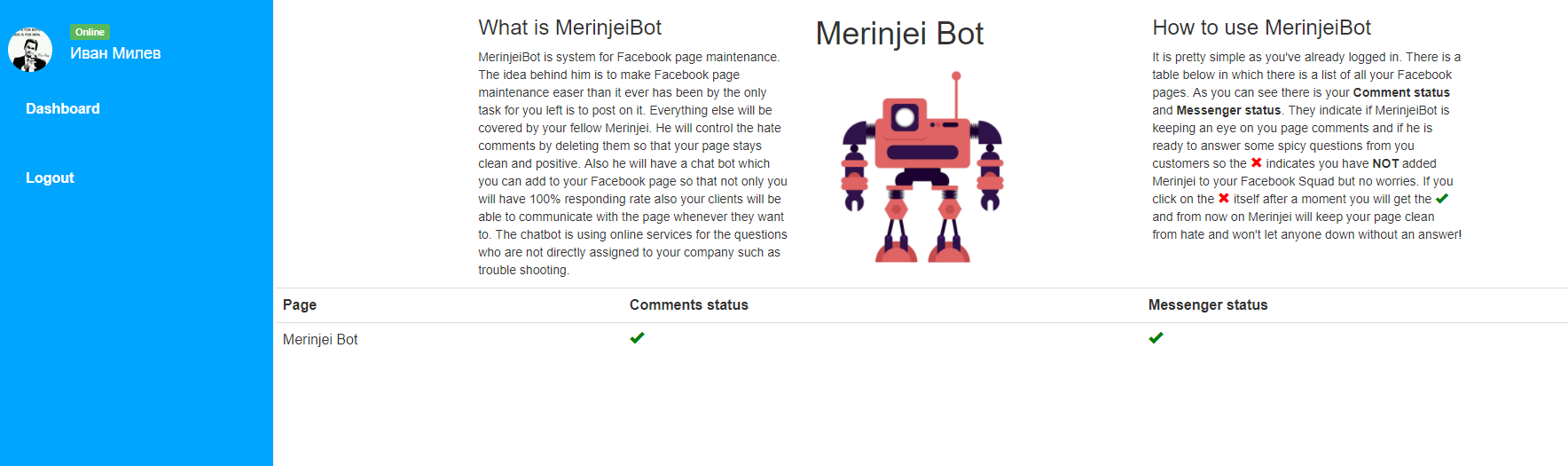
В заявката GET параметъра fields посочва кои данни за потребителя искаме в случая това са неговото име и снимка. Access\_token е жетона за достъп на потребителя, който Facebook е дал при автентикацията.

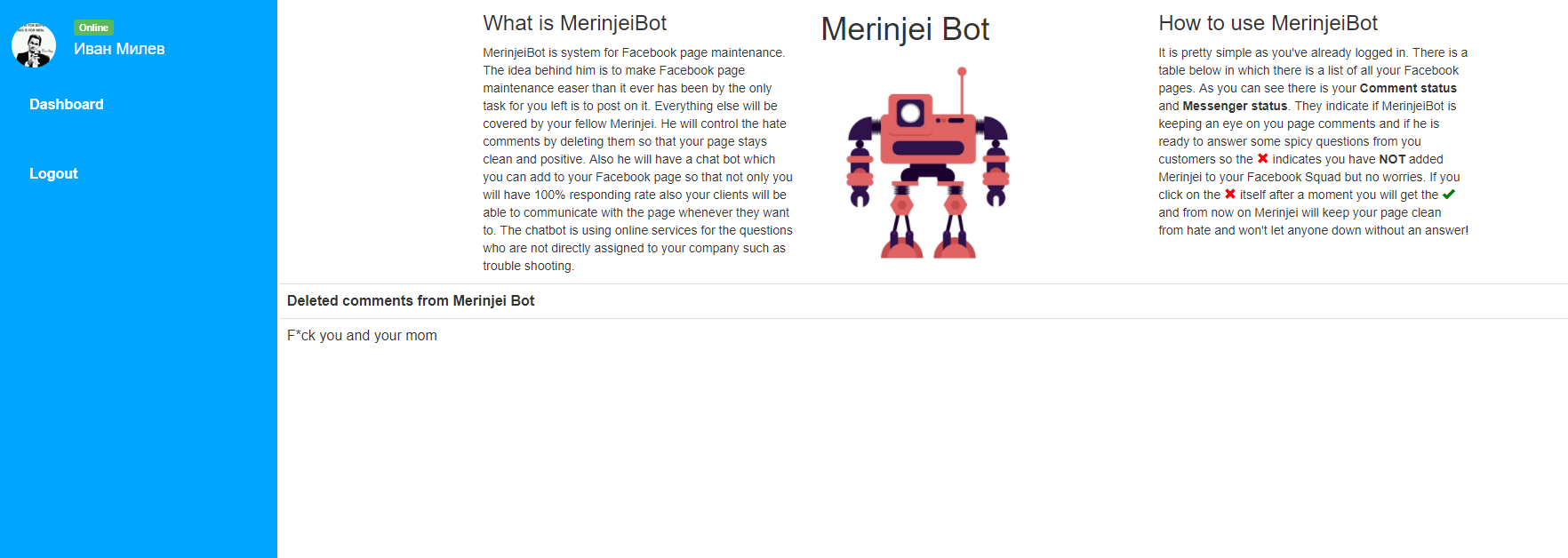
След тази заявка се популира и таблицата с потребителските страници. За целта отново се прави заявка към Facebook:

https://graph.facebook.com/v2.11/me/accounts?type=page&access\_token=' +

access\_token

В заявката GET параметъра type дава видовете профили на съответния потребител, в случая това са страниците му. Access\_token е жетона за достъп на потребителя, който Facebook е продставил при автентикацията на потребителя. Тази заявка връща идентификатора на страницата, нейното име и абонаментите към нея относно Webhook-ове. Като от тези данни биват използвани името за таблицата, която потребителя на MerinjeiBot вижда както и от абонаментите тези, които са за коментарите т.е. feed и за Messenger messaging т.е. за съобщения. Идентификатора и жетона за достъп биват запазени в базата данни на приложението като идентификатор на таблицата се използва идентификатора на страницата. Това се прави, защото когато получаваме заявка от Webhook на Facebook приложението няма достъп до конкретния потребител, който е админ на страницата и съответно до неговия жетон за достъп, което прави невъзможно правенето на заявка за жетон за достъп на страницата. Освен това тъй като прекратяването на абонамент към Facebook страница е към цялата страница, а не конкретно към някое поле, затова се налога записването в базата данни на това към кои полета е свързан потребителя и премахването на абонамента да представлява проверка в сървъра преди обработката на получена заявка от Facebook.

 Фигура 3.3



Фигура 3.4

След като в таблицата са поставени вашите групи и съответно това дали сте се свързали към коментарите и/или Messenger-а на вашата страница. Ако потребителят иска да се свърже с тях то натиска върху червения хикс.

При „закачане“ към коментарите на потребителската страница първо се изпраща асинхронна заявка към сървъра посредством AJAX от JS, който очаква събитие от тип натискане върху съответното хиксче или тикче. След което вече сървъра започва със сканиране на всички коментари в страницата. Това се осъществява посредством GET заявка към:

'https://graph.facebook.com/v2.11/' + page\_id + '/posts?access\_token='+ access\_token.

Като на мястото на page\_id се поставя id на страницата, за която сканираме, а access\_token е жетон за достъп предоставен от Facebook за потребителя след като той се е автентикирал пред MerinjeiBot.

Всички коментари се групират и оценяват от модела за недобронамерена реч и тези, които са оценени негативно биват премахнати от вашата страница посредством DELETE заявка към:

response = requests.delete('https://graph.facebook.com/v2.11/' + comment['id'] + '?access\_token=' +

access\_token)

като comment\_id е id-то(идентификатора) на съответния коментар, който трябва да бъде изтрит, но този път жетона за достъп е този на страницата, а не на потребителя като жетонът трябва да има права за премахване на коментари.

След като първоначалното сканиране е свършило MerinjeiWeb се закача към Webhook, с който да слуша за промени на стената(feed) на страницата. Това става посредством POST заявка към:

'https://graph.facebook.com/v2.11/' + page\_id + '/subscriptions'

Като зад page\_id стои id-то на страницата, за която приложението трябва да се прикачи и се подават POST параметри на заявката в следния формат:

data = {

'object': 'page',

'callback\_url': COMMENTS\_CALLBACK,

'fields': ['feed'],

'verify\_token': VERIFY\_TOKEN,

'access\_token': access\_token,

}

В данните за callback\_url се дава крайната точка, на която Facebook Webhook-а да изпраща посредством POST заявки обновленията, за които му е казано във fields полето в случая това е стената(feed-a) на страницата. Access\_token е жетона за достъп, който в случая е този за страницата, който в себе си съдържа права за редакции по коментарите и съобщенията в страницата. Verify\_token е стринг избран от нас, с който се автентицира абонамента. След изпращането на POST заявката за абонамент от Facebook се изпраща GET заявка, в чиито параметри се съдържа verify\_token-а и при съответствие се връща резултат със статус код 200 и параметърът от получената заявка в hub.challenge. След този момент вече се получават коментари и промени във feed-a на страницата на крайната точка посочена като callback\_url в абонамента посредством POST заявка от Facebook. При нов коментар първо се проверява състоянието на абонамента в базата данни и ако то е активно отново той се оценява и ако се сметне за недобронамерен се изтрива по горе описания начин(както е при сканирането на страницата), ако абонамента не е активен просто не се обработва получената заявка от Facebook.

Поради промени в правилата и условията на Facebook през май 2018 става невъзможно използването на Webhook за коментарите, защото те са забранени освен за одобрени приложения. Затова бе имплементиран друг начин за сканиране на страницата. Новата имплементация използва гореописаното сканиране на страницата като то се прави на всеки 5 секунди.

Другата опция е закачането на MerinjeiBot към Messenger, което се състои в абонирането на бота към Webhook на Messenger. За да се абонира MerinjeiBot изпраща отново POST заявка към:

'https://graph.facebook.com/v2.11/' + page\_id + '/subscriptions'

както и при коментарите page\_id е id-то на страницата разликата тук е в POST параметъра fields като този път вместо feed се дава messages, за да може бота да получава съобщенията от Messenger. Освен това жетона за достъп е съставен от id-то на приложението и тайния код на приложението(App secret).

data = {

'object': 'page',

'callback\_url': MESSENGER\_CALLBACK,

'fields': ['messages'],

'verify\_token': VERIFY\_TOKEN,

'access\_token': access\_token,

}

Тук обаче трябва да се активира и Facebook NLP тъй като се използва при генерирането на отговор. Това става посредством POST заявка към:

'https://graph.facebook.com/v2.11/me/nlp\_configs?nlp\_enabled=true'

като POST параметри се дава само жетон за достъп на страницата.

Интересно е как се връща съответният резултат или отговор. След получаване на заявка от Facebook съответно за Messenger съобщение първо се проверява дали абонамента е активен, ако не е съобщението не се обработва. Ако абонамента е активен първата стъпка на бота е да използва вграденият от Facebook NLP и при появата на клас от тип Greetings, Bye, Thanks да отговори с готови съобщения от типа на поздрави или благодарност. Ако случаят не е такъв то се проверява дали даденото съобщение е въпрос посредством класификатора за въпроси и при убеденост на модела над 30% за типа той се смята за въпрос. Ако не се връща съобщение на потребителя, че ботът не го е разбрал. Ако обаче е въпрос се прави извадка на ключовите думи и с тях се прави търсене в StackOverflow посредством техния публичен API(Приложно-програмен интерфейс) като се прави GET заявка към:

“https://api.stackexchange.com/2.2/search/advanced?order=desc&sort=votes&title=” + user\_question + "&site=stackoverflow"

Зад user\_question са ключовите думи, с които се търси в StackOverflow, като се търси по title(заглавие или въпрос в контекста на StackOverflow). Резултата, който се връща е подреден по гласове на въпросите като започва от тези с най-много. Резултата се съхранява в OrderedDict т.е. речник, който запазва реда на JSON обекта, който е получен от StackOverlow като отговор(response). След което се взимат първите три въпроса, чиито заглавия (title , „въпросът зададен от потребител на StackOverflow“) е същия тип като този зададен на MerinjeiBot и на базата на тях се прави резюмиран отговор до 50 символа. Отговорите се взимат след като от получените резултати за въпроси се взимат id-тата на техните одобрени отговори и след това се прави GET заявка към:

'https://api.stackexchange.com/2.2/answers/' + answers\_id +

'?order=desc&sort=votes&site=stackoverflow&filter=withbody'

като answers\_id е id-то на отговора на съответния въпрос като този път се добавя GET параметъра filter със стойност withbody, за да може отговорът да бъде върнат с тялото си, което бива парснато от html към обикновен текст.

Ако не бъдат намерени три въпроса отговарящи на гореописаните условия се взимат трите Stack Overflow въпроса, които имат най-висок рейтинг и на тях се прави резюме отново до 50 символа. Отговорът се връща на потребителя посредством Graph API(Приложно-програмен интерфейс) като се прави POST заявка към:

"https://graph.facebook.com/v2.6/me/messages?access\_token=" + access\_token

Тук жетона за достъп е на страницата. Самият отговор се съдържа в POST параметрите които се дават на заявката:

data = {

"messaging\_type": "RESPONSE",

"recipient": {"id": recipient},

"message": {"text": answer}

}

В данните се посочва типа на съобщението в случая отговор („RESPONSE“), id-то на потребителя към който трябва да бъде изпратено съобщението и отговора, който му изпращаме.

Ако потребителят иска да се откаже от абонамента си то той просто трябва да натисне зеленото тикче. При натискането му събитието ще бъде прихванато от JS и ще бъде направена AJAX заявка към Django сървъра, която ще извика метод, който взима от заявката идентификатора на страницата и променя състоянието на таблицата за Webhook абонаменти на съответната страница като неактивна и заявки получени от този абонамент вече няма да бъдат обработвани.

# ЧЕТВЪРТА ГЛАВА

## Инсталация на MerinjeiBot

За да инсталирате MerinjeBot първо ще ви трябва Anaconda за версия на Python 3.6.3, която може да намерите и свалите тук:

<https://www.anaconda.com/download/>

След инсталацията на Anaconda трябва да бъдат свалени пакетите нужни на приложението, за да работи, което може да направите от основната директория на MerinjeiBot с командата:

> conda env create --file conda\_environment.yml

Последната стъпка, която трябва да изпълните е да си свалите ngrok, за да може да използвате Facebook API. Ngrok може да бъде свален за Linux, Windows, Mac OS X от <https://ngrok.com>.

Другият вариант за инсталация на пакетите е нужен пакетният мениджър pip. След като е инсталиран пакетният мениджър инсталацията става с една команда:

pip install –r pip\_requirements.txt

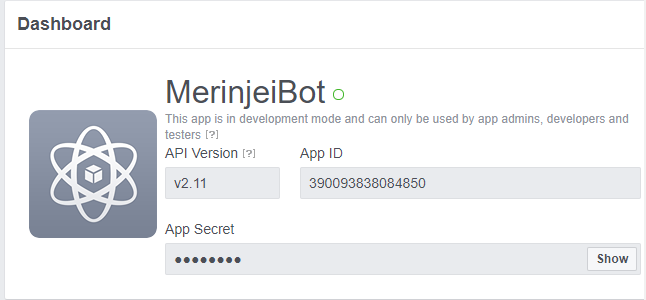
Програмният код за MerinjeiBot може да бъде намерен на приложения диск към книжното тяло както и на:

<https://github.com/ivanmilevtues/merinjei_bot>

## Настройка на Facebook за работа с MerinjeiBot

За да може MerinjeiBot да работи с Facebook трябва да създадете собствено приложение в <https://developers.facebook.com>. След като го създадете трябва да добавите три продукта към него: Facebook Login, Messenger, Webhooks.

След като сте добавили трите приложения трябва да намерите от контролното табло на приложението ви App Id и App secret(, след което да копирате съответните стойности и да ги приложите в merinjei\_bot/merinjeiweb/CONSTANTS.py на съответните променливи със съответващите им имена.

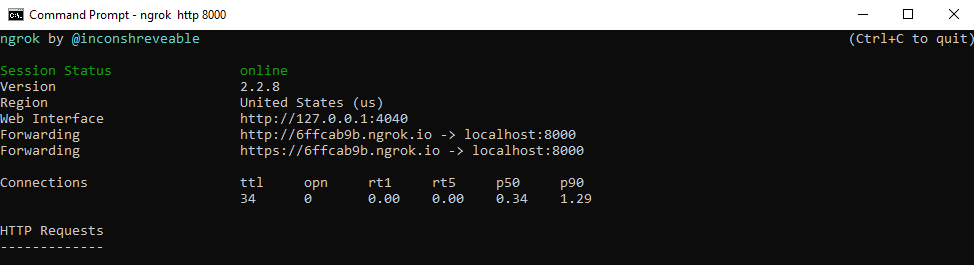


Фигура 4.1

След като сте извършили тези стъпки трябва да инсталирате ngrok и да ползвате следната команда в CMD и/или в терминал:

ngrok http 8000

така ngrok ще ви даде domain, който ще бъде пренасочва http заявки към порт 8000 на вашата машина, на който порт слуша и Django. След което ще ви излезе статусния екран на ngrok Фигура 4.2.

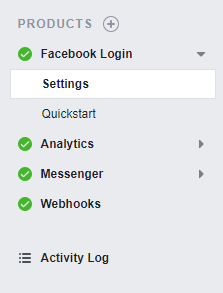


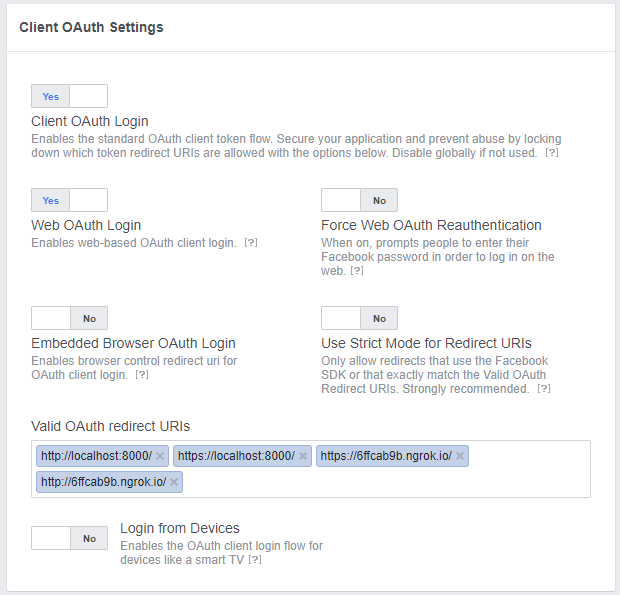
Фигура 4.2

От тук трябва да копирате https URL-а на Forwarding и да го поставите на DOMAIN променливата в merinjei\_bot/merinjeiweb/CONSTANTS.py. Освен това трябва да вземете частта от URL-a, която съдържа domain name-а и да я поставите в merinjei\_bot/merinjeiweb/merinjeiweb/setting.py като част от списъка ALLOWED\_HOSTS в случая трябва да се добави 6ffcab9b.ngrock.io и списъка трябва да изглежда съответно

ALLOWED\_HOSTS = ['localhost', '197d7a0d.ngrok.io']

Последната стъпка при настройката на бота за работа с Facebook е да се навигирате посредством лявото меню на приложението ви създадено на <https://developers.facebook.com> до Products/Facebook Login Фигура 4.3. В него трябва да се уверите, че слайдарите за Client OAuth Login и Web OAuth са позволени и да добавите URL адресите на приложението си в Valid OAuth redirect URIs(Фигура 4.4). Важно е да не се забравят и тези, които използват http не само тези с https.

  
Фигура 4.3

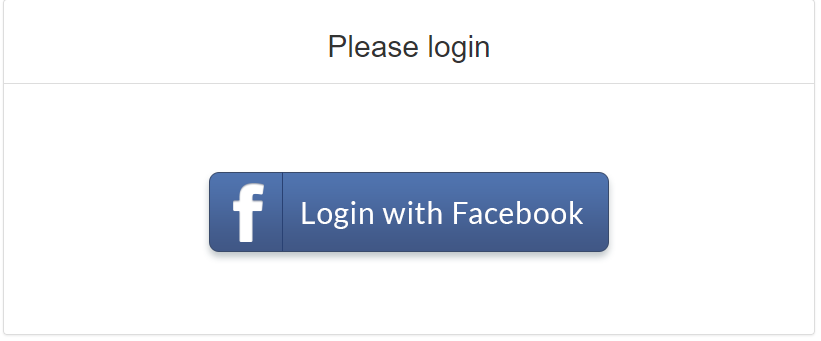
  
Фигура 4.4

## Използване на MerinjeiBot от потребителя

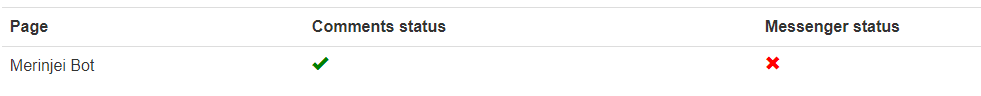
Потребители ще могат да използват приложението на следният URL:

https://merinjei-bot.herokuapp.com

За да използва MerinjeiBot потребителят трябва да се логне в него чрез Facebook. Фигура 4.5.

Фиг. 4.5

Пред него ще се появи неговият бордови екран, на който може да видим списък с страниците, на които потребителят е админ Фигура 4.6. От таблицата потребителят може да добави MerinjeiBot посредством клик върху червените хиксове. Червените хиксове означават, че MerinjeiBot не е свързан към страницата, а зелените тикчета, че е. След като бота вече е свързан той е абониран за страницата ще започне да следи коментарите и да трие тези, които сметне за недобронамерени. Както и да отговаря на хората, които му пишат.

Фиг. 4.6

# Заключение

Разработената дипломна работа разглежда различни методи и подходи в обработката на естествен език като по-конкретно в класификацията на недобронамерена и/или обидна реч както и обработката на въпроси и тяхната класификация. Дипломната работа е изпълнила всички изисквания. Системата може да разпознава недобронамерена и обидна реч с точност от 88% и съответно при разпознаване на недобронамерен коментар на страница към, която бота е абониран да го изтрие. Също така системата съдържа в себе си чат бот, който може да отговаря на въпроси посредством резюмиране на отговори от StackOverflow на подобни въпроси като за целта използва класификатор на типове въпроси, който е с точност от 77%, а за резюмирането се използва метод на извличане.

Могат да бъдат до усъвършенствани класификаторът на недобронамерена реч както и този на типове въпроси. Това може да бъде реализирано посредством опити с други модели и добавянето на нови признаци за данните. Друг аспект върху, който може да бъде постигнат прогрес е резюмирането на отговор като подходът за резюмиране бъде сменен от метод чрез извличане към абстрактен. Абстрактният метод ще предостави по лесен за четене текст както и по-смислен, защото наподобява човешкото резюмиране. Освен това могат да бъдат добавени и функционалности като извличане и/или създаване на Facebook постове от даден сайт следейки от него за новости, отговаряне на зададени въпроси и като цяло на коментари.

# Библиография

[1] Monitoring and reporting online hate speech in Europe,   
<https://www.emoreproject.eu/about-project/>

[2] Automated Hate Speech Detection and the Problem of Offensive Language <https://aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/view/15665/14843>

[3] Alan Tuning at 100,   
<https://news.harvard.edu/gazette/story/2012/09/alan-turing-at-100/>

[4] The DeepQA Research Team IBM

<https://researcher.watson.ibm.com/researcher/view_group_pubs.php?grp=2099>

[5] Machine learning: the problem setting,   
<http://scikit-learn.org/stable/tutorial/basic/tutorial.html#machine-learning-the-problem-setting>

[6] NLP methods and applications,

<https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.html>

[7] A Gentle Introduction to the Bag-of-Words Model,

<https://machinelearningmastery.com/gentle-introduction-bag-words-model/>

[8] Syntactic clustering of the Web,

<https://www.sciencedirect.com/science/article/pii/S0169755297000317?via%3Dihub>

[9] Stemming and lemmatization,   
<https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>

[10] A Complete Tutorial on Tree Based Modeling from Scratch, <https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/>

[11] Random Forest Classifier,   
<http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

[12] Logistic Regression Tutorial for Machine Learning,

<https://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/>

[13] Categorizing and Tagging Words,

<http://www.nltk.org/book/ch05.html>

[14] Text Summarization in Python: Extractive vs. Abstractive techniques revisited,  
<https://rare-technologies.com/text-summarization-in-python-extractive-vs-abstractive-techniques-revisited/>

[15] NumPy,   
<http://www.numpy.org>

[16] NLTK,

<http://www.nltk.org>

[17] Six reasons why I recommend scikit-learn,   
<https://www.oreilly.com/ideas/six-reasons-why-i-recommend-scikit-learn>

# СЪДЪРЖАНИЕ

[Отзив на научния ръководител 3](#_Toc508133822)

[УВОД 4](#_Toc508133823)

[1 ПЪРВА ГЛАВА 6](#_Toc508133824)

[1.1 Съществуващи решения 6](#_Toc508133825)

[1.1.1 eMore 6](#_Toc508133826)

[1.1.2 Automated Hate Speech Detection and the Problem of Offensive Language 6](#_Toc508133827)

[1.1.3 ELIZA 8](#_Toc508133828)

[1.1.4 Watson 9](#_Toc508133829)

[1.2 Проучване 10](#_Toc508133830)

[1.2.1 Машинно самообучение 10](#_Toc508133831)

[1.2.2 Контролирано машинно самообучение (Supervised learning) 11](#_Toc508133832)

[1.2.3 Класификация 11](#_Toc508133833)

[1.2.4 Регресия 12](#_Toc508133834)

[1.2.5 Безнадзорно машинно самообучение (Unsupervised learning) 12](#_Toc508133835)

[1.2.6 Оценяване на модела 12](#_Toc508133836)

[1.2.7 Обработка на естествен език(Natural Language Processing) 14](#_Toc508133837)

[1.2.8 Торба с думи(Bag of words) 15](#_Toc508133838)

[1.2.9 Ngram-и 15](#_Toc508133839)

[1.2.10 Коренуване (Stemming) 15](#_Toc508133840)

[1.2.11 Дърво на решението 16](#_Toc508133841)

[1.2.12 Произволна гора 16](#_Toc508133842)

[1.2.13 Логистична регресия 16](#_Toc508133843)

[1.2.14 Маркиране на части от речта 17](#_Toc508133844)

[1.2.15 Резюмиране на текст 17](#_Toc508133845)

[2 ВТОРА ГЛАВА 18](#_Toc508133846)

[2.1 Функционални изисквания 18](#_Toc508133847)

[2.2 Език за програмиране 18](#_Toc508133848)

[2.3 Библиотеки 19](#_Toc508133849)

[2.3.1 Numpy 19](#_Toc508133850)

[2.3.2 NLTK 19](#_Toc508133851)

[2.3.3 Sklearn 20](#_Toc508133852)

[2.3.4 allauth 20](#_Toc508133853)

[2.4 Софтуерни рамки 21](#_Toc508133854)

[2.4.1 Django 21](#_Toc508133855)

[2.5 Развойна среда 22](#_Toc508133856)

[2.5.1 Visual Studio Code 22](#_Toc508133857)

[2.6 Описание на алгоритъма на работа 22](#_Toc508133858)

[3 ТРЕТА ГЛАВА 26](#_Toc508133859)

[3.1 Класификация на недобронамерена реч 26](#_Toc508133860)

[3.1.1 Обработка на данните 33](#_Toc508133861)

[3.1.1.1 Използвани признаци 33](#_Toc508133862)

[3.2 Чатбот 37](#_Toc508133863)

[3.2.1 Класификация на въпроси 38](#_Toc508133864)

[3.2.1.1 Признаци 39](#_Toc508133865)

[3.2.1.2 Приложение на класификатора на въпроси 40](#_Toc508133866)

[3.2.1.3 Изваждане на ключови думи за търсене 41](#_Toc508133867)

[3.2.1.4 Резюмиране на отговор 41](#_Toc508133868)

[3.3 Интеграция с Facebook 42](#_Toc508133869)

[4 ЧЕТВЪРТА ГЛАВА 51](#_Toc508133870)

[4.1 Инсталация на MerinjeiBot 51](#_Toc508133871)

[4.2 Настройка на Facebook за работа с MerinjeiBot 51](#_Toc508133872)

[4.3 Използване на MerinjeiBot от потребителя 54](#_Toc508133873)

[5 Заключение 56](#_Toc508133874)

[Библиография 57](#_Toc508133875)

[СЪДЪРЖАНИЕ 59](#_Toc508133876)