Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek

Računarstvo usluga i analiza podataka

SEMINARSKI RAD

„Raspoznavanje spam SMS poruka“

Domagoj Rojnić

Jakov Prpić

Danijel Dražetić

Osijek, 2021.

Sadržaj

1. Uvod

Zadatak ovog rada je izrada modela za raspoznavanje SPAM poruka od običnih poruka. Napravljena su dva modela, jedan model je izrađen koristeći biblioteku spaCy, dok je drugi izrađen pomoću biblioteke NLTK. U modelima uspoređujemo preciznost klasifikatora kao što su *Naive Bayes* i SVC *(eng. Support Vector Clustering).* Pomoću klasifikacije možemo riješiti probleme organiziranja podataka u kategorije prema zajedničkim svojstvima, temeljem podataka kojima je već utvrđeno pripadanje pojedinoj kategoriji.

Kod je pisan programskim jezikom Python u razvojnom okruženju *Spyder.* Klasifikatore je potrebno trenirati i testirati u većoj količini obrađenih poruka da bi dobili što vjerodostojnije rezultate. Rezultati su prikazani u obliku postotka te prikazuju preciznost određene metode klasifikacije.

2. Opis problema

Postupci klasifikacije se primjenjuju za različite svrhe, od prepoznavanja neželjene elektroničke pošte, razvrstavanja iste na kategorije poput "Primarno", "Promocije", "Društvene mreže". . . , određivanja rubrike članka (politika, sport, crna kronika. . .), analizu sentimenta i autorstva. Primjerice, moguće je provođenjem postupka klasifikacije odrediti jesu li komentari na neki proizvod, film ili članak pozitivni ili negativni, utvrditi je li neki tekst napisao muškarac ili žena, starija ili mlada osoba te utvrditi autora teksta.

U ovom radu su uspoređene preciznosti metoda klasifikacije dok razvrstavaju SPAM poruke te su odbrani najpovoljniji klasifikatori za ovaj posao.

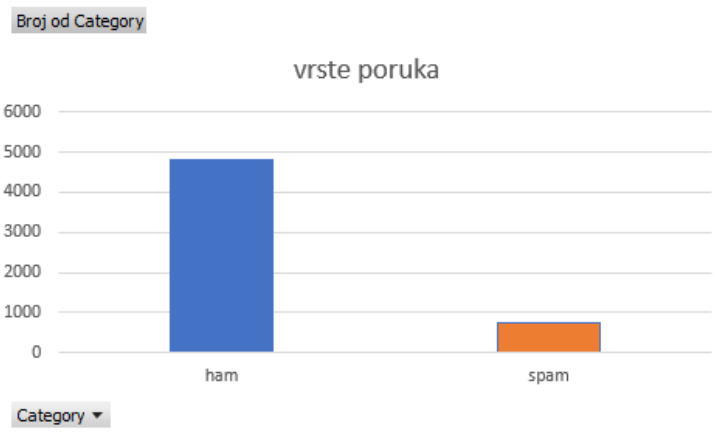
2.1 Korišteni podaci

Korišten je skup podataka koji se sastoji od kategorija i poruka. Kategorije se dijele na *''ham''* i *''spam''*, odnosno pokraj stupca poruke u stupcu kategorija vidimo vrstu poruke u istom redu. Skup podataka sadrži 5572 poruke koje služe za klasifikaciju. Ovaj skup podataka je preuzet sa stranice *Kaggle.* Na slici 1.1 vidljiv je isječak iz skupa podataka.

	Category	Message
1	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
2	ham	Ok lar... Joking wif u oni...
3	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
4	ham	U dun say so early hor... U c already then say...
5	ham	Nah I don't think he goes to usf, he lives around here though
6	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, Â£1.50 to rcv
7	ham	Even my brother is not like to speak with me. They treat me like aids patent.

Slika 2.1. Primjer poruka iz skupa podataka.

Od ukupno 5572 poruke, normalnih poruka ima 4825 (87%), dok je spam poruka manje, točnije 747 (13%). Na slici 2.2 vidljiva je grafička raspodjela poruka.



Slika 2.2. raspodjela poruka na grafikonu

2.2 Korišteni postupci strojnog učenja

Korištena je metoda za odvajanje podataka koja razdvaja podatke na test i trening podatke. Podatci se odvajaju kako bi model mogao na osnovu naučenog u trening podatcima pokušati razvrstavati poruke na *spam* i *ham* u test podatcima. Što je veći dio trening podataka to je veća vjerojatnost da je model više istreniran i da će imati veću preciznost.

U modelu su korišteni razni algoritmi klasifikacije da bi se usporedila preciznost među tim algoritmima. Korišteni algoritmi su: *"K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD Classifier", "Naive Bayes", "SVM Linear".*

Algoritam najbližih susjeda *(eng. K Nearest Neighbors)* jedna je od najjednostavnijih metoda strojnog učenja. Pripada diskriminativnom, neparametarskom, nelinearnom modelu nadziranog učenja, a određivanje klase nepoznatog dokumenta svodi se na određivanje većinske klase najbližih dokumenata iz skupa za učenje. Za udaljenost između podataka mogu se koristiti različite metrike. Neke od najpoznatijih su euklidska i Manhattan udaljenost, a izbor ovisi o problemu koji treba riješiti.

Stabla odlučivanja (eng. *Decision Tree*) predstavljaju neparametarski nadzirni način učenja koji se koristi za klasifikaciju i regresiju. Koriste skup pravila if-then koji je međusobno isključiv i iscrpan za klasifikaciju. Pravila se uče uzastopno koristeći podatke za učenje, jedno po jedno.

Slučajna šuma (*eng. Random Forest*) je općenit naziv za skupinu metoda koje se koriste kolekcijom stablastih klasifikatora pri čemu je skup nezavisnih slučajnih vektora jednake distribucije, a x ulazni vektorski uzorak. Svako stablo daje svoj glas za ulazni uzorak, a šuma određuje klasu uzorka na temelju većine glasova.

Unatoč imenu, logistička regresija (*eng. Logistic Regression*) je klasifikacijski model, ali se ovaj naziv koristi iz povijesnih razloga. Neki od problema na kojima se primjenjuje logistička regresija su spam filteri, online kupovina (koristi li netko ukradene podatke ili ne), procjena tumora (je li tumor zloćudan ili ne), vremenska prognoza...

Stohastički gradijentni *spust (engl. stochastic gradient descent)* je varijanta gradijentnog spusta kod koje se nakon svakog uzorka ažuriraju težine. Ovakva varijanta više oscilira te je zbog toga otpornija na zapinjanje u lokalne optimume. Također, kod učenja konvolucijskih mreža ovom varijantom, moguće je ubrzati postupak metodama drugog reda.

2.2.1 Naive Bayes

Naive Bayes klasifikator je algoritam koji koristi Bayesov teorem za razvrstavanje objekata. Bayesov teorem opisuje vjerojatnost da će se dogoditi nekakav događaj A ukoliko je događaj B istinit. Ovaj teorem je vrlo koristan i često korišten u području vjerojatnosti i statistike jer omogućuje proračun vjerojatnost nekog događaja u ovisnosti o prijašnjem znanju događaja koji bi mogli utjecati na taj događaj. Matematički se Bayesov teorem zapisuje na način opisan jednadžbom 2.1.

$$\boldsymbol{P(A|B) = \frac{P(B|A) * P(A)}{P(B)}}$$

Jednadžba 2.1. Bayesov teorem

Gdje je:

- P(A|B) – vjerojatnost događaja A ako se dogodio događaj B
- P(B|A) – vjerojatnost događaja B ako se dogodio događaj A
- P(A) – vjerojatnost događaja A
- P(B) – vjerojatnost događaja B

Uzmimo primjer da je događaj A klasifikacija poruka kao ham i događaj B da su značajke poruke z_i. Međutim problem se pojavljuje što će klasifikator s Bayesovim teoremom imati problema s klasifikacijom poruka ako se poredak riječi promjeni odnosno ukoliko ne postoji istreniran klasifikator nad svim mogućim slučajevima, tada ovaj oblik ne pomaže puno. Ono što se može napraviti je reći da se svaka riječ zasebno pojavljuje te na osnovu te „naivne“ pretpostavke izračunati vjerojatnost cijele rečenice, vidljivo na jednadžbi 2.2.

$$P(ham|značajke) = P(ham)\prod_{i=1}^{n} P(z_i|ham)$$

Jednadžba 2.2. Teorem naivnog Bayesa

Problem pri ovakvom pristupu je u tome što sama vjerojatnost događaja ham poruke ne znači ništa pošto u ovom slučaju ne vrijedi jednadžba 2.3.

$$P(spam|značajke) = 1 - P(ham|značajke)$$

Jednadžba 2.3. Vjerojatnost ne klasifikacije kao ham

što znači da će se morati istim postupkom izračunati i vjerojatnost za spam poruke koristeći jednadžbu 2.4 te ih usporediti i na osnovu toga čija je vjerojatnost veća donijeti odluku. Sada se javlja još jedan problem tokom ovog pristupa. Vjerojatnosti koja se dobije je najčešće vrlo mala te broj različit od 0 ponekad može biti tek na 10. ili 20. decimalnom mjestu. Ono što je moguće napravit je normalizirati pristup te dobiti vrijednost u intervalu [0,1] te reći da će se za vjerojatnost 0.5 i veće klasificirati značajke kao ham te ispod 0.5 kao spam.

$$P(ham|značajke) = \frac{P(ham)\prod_{i=1}^{n} P(z_i|ham)}{P(ham)\prod_{i=1}^{n} P(z_i|ham) + P(spam)\prod_{i=1}^{n} P(z_i|spam)}$$

Jednadžba 2.4. Normaliziran teorem naivnog Bayesa

2.2.2 Support Vector Machines (SVM)

Support Vector Machines je skup nadziranih metoda učenja koje se koriste za klasifkaciju, regresiju i vanjsko otkrivanje *(eng. outlier detection*). Cilj ove metode je odvajanje pozitivnih primjera od negativnih primjera s maksimalnom granicom, a granica je udaljenost od hiperravnine do najbližeg pozitivnog ili negativnog primjera, tj. cilj je pronaći hiperravninu u n-dimenzionalnom prostoru koji jasno razdvaja podatkovne točke. Ti primjeri nazivaju se potporni vektori *(eng*. *support vectors)*. Ovi algoritmi koriste teoriju minimalizacije strukturnog rizika.

Algoritmi bazirani na *SVM* upotrebljavaju se za klasifikaciju nekoliko primjera u dvije različite klase. Ovaj algoritam nalazi hiperravninu između te dvije klase tako da odvajanje granica između te dvije klase postaje maksimalno. Klasifikacija testnog primjera ovisi o strani hiperravnine, odnosno strani gdje se testni primjer nalazi. Ulazne značajke mogu se preslikati i u prostor visokih dimenzija, ali u tom slučaju, za smanjenje računskih troškova obuke i postupka ispitivanja u prostoru visokih dimenzija, koriste se neke funkcije kernela. Parametar regularizacije koristi se u slučaju neodvojivih primjera treninga. Zadana vrijednost ovog parametra smatra se 1. *SVM* ima nekoliko metoda pomoću kojih se može izvesti klasifikacija: To su *SVC, NuSVC* i *LinearSVC.*

Primjena SVC

SVC je linearni binarni klasifikator koji može rješavati nelinearne probleme pomoću jezgrenog trika.

Klasifikacija pomoću algoritma stroja potpornih vektora ima mnoštvo primjena. Samo neke od njih su: klasifikacija slika, kategorizacija teksta, primjene u bioinformatici, medicini, fizici i mnogim drugim područjima.

Kod klasifikacije slika, SVC se odlikuje visokom preciznosti. Može se koristiti za npr. prepoznavanje lica tako da klasificira dijelove slike koji pripadaju licu u jednu grupu, a ostale dijelove u drugu. Također, može se koristiti za prepoznavanje pisanog teksta i znamenaka. U medicini su primjene poput prepoznavanja zloćudnih tumora.

Kod kategorizacije teksta, *SVC* se može naučiti na skupu za treniranje tako da svrstava tekst u različite klase poput web-stranica, članaka, e-pošte i drugih. Može se koristiti i da npr. razvrstava članke u kategorije poput sporta, politike, crne kronike, oglasa...

U bioinformatici, jedan od važnih problema je *engl. protein remote homology detection*. Upravo *SVC* postiže najbolje rezultate u rješavanju tog problema. Također, SVC ima i primjenu u problemima klasifikacije u fizici visokih energija tj. fizici elementarnih cestica.

3. Opis programskog rješenja

3.1 Izrada spam filtra u Pythonu

U ovom zadatku izradit ćemo spam filtar odnosno filtar neželjenih elektroničkih poruka. Program će svaki predani e-mail označiti sa spam (ukoliko je on neželjen) ili sa ham (sve ostale). Za izradu koristiti ćemo programski jezik Python uz razvojno okruženje *Anacodna* te skup podataka koji smo preuzeli sa stranice *Kaggle*. Nadalje su navedene neke korištene glavne biblioteke u Pythonu:

- Matplotlib.pyplot - za crtanje grafike
- Numpy - za rad s matricama i poljima
- Sklearn.metrics - za izračun matrice zbunjenosti
- Pandas – za analizu podataka
- Nltk (Natual Language Toolkit) - biblioteka za obradu prirodnog jezika
- Spacy - biblioteka za obradu prirodnog jezika

Uspoređivanje algoritama je izvršeno na dva različita načina. Prvi način je sa bibliotekom za obradu prirodnog jezika spaCy, dok je drugi način sa bibliotekom NLTK.

3.1.1 spaCy

SpaCy je biblioteka otvorenog koda za obradu prirodnog jezika u pythonu. Davanje riječi / teksta kao parametra spaCy-ovom nlp modelu rezultira dokumentom koji sadrži svoje tokene i vektore. *Word Embeddings* koriste se za izračunavanje sličnosti između zadanih riječi / tekstova. Na primjer, riječi poput "Sea", "Ocean", "Water" imat će slične vektorske vrijednosti u usporedbi s riječima poput "Train" ili "Notebook". Drugi primjer je oduzimanje vektora riječi "Queen" iz vektora riječi "King", a zatim dodavanje rezultata riječi "man" rezultirat će riječju "woman". Ti se vektori mogu koristiti kao značajke za modele strojnog učenja, a zatim predviđati sentiment / klasu teksta.

Biblioteke u programskom jeziku Python se uključuju s ključnom riječi „import“. Uključit ćemo biblioteke pandas pomoću koje ćemo vršiti analizu i čitanje iz skupa podataka, spacy koja služi za obradu prirodnog jezika te numpy koju ćemo koristiti za rad s matricama. Programski kod 3.1. učitava skup podataka pomoću biblioteke pandas te ispisuje prvih pet poruka iz skupa podataka. Također se učitava spaCy model pomoću kojeg se računaju vektori riječi.

```
import pandas as pd
import spacy
import numpy as np

data = pd.read_csv("spam_data.csv")

nlp = spacy.load("en_core_web_lg")

# disabling other pipes that aren't needed
with nlp.disable_pipes():
    doc_vectors = np.array([nlp(text).vector for text in data.Message])

print(f"{data.head(5)}\n\ndoc_vectors shape: {doc_vectors.shape}")
```

Programski kod 3.1. Učitavanje podataka

Ispis prvih pet poruka:

```
Category                                                Message
0      ham  Go until jurong point, crazy.. Available only ...
1      ham                      Ok lar... Joking wif u oni...
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
doc_vectors shape: (5572, 300)
```

Pomoću slijedećih linija koda u programskom kodu 3.2., skup podataka se razdvaja na trening podatke i test podatke. Nakon raspodjele podataka, budući da je y_train lista, ona će i dalje držati indekse izvorne y liste u rasponu od [0-5572], iako je njegova veličina nakon raspodjele 3900. Budući da *Stratified K-Fold* uzima (X_train, y_train) kao skupove za razdvajanje, indeksi liste y_train se moraju resetirati, tako da nema vrijednosti koje "nedostaju" zbog razlike u indeksima.

```
from sklearn.model_selection import train_test_split

X=doc_vectors
y=data.Category
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=388,
                                                    stratify=y,
shuffle=True)
y_train.reset_index(drop=True,inplace=True)

print(f"y list shape: {y.shape} \ny_train list shape: {y_train.shape}")
```

Programski kod 3.2. Podjela podataka

```
y list shape: (5572,)
y_train list shape: (3900,)
```

Kako bi izabrali najbolji algoritam, testirati ćemo 4 algoritma: *SVC, LogisticRegression, Decision Trees i MLPClassifier.*

Kako bi se osiguralo da se koriste najbolji parametri modela za ovaj skup podataka, hiperparametri su podešeni s *TuneSklearn's TuneSearchCV.*

Postavljanje modela vidljiv je na prikazu programskog koda 3.3.

```
# Importing models
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

svc = SVC(C= 10, gamma= 0.1, kernel= 'rbf').fit(X_train,y_train)

logreg =
LogisticRegression(solver='liblinear',random_state=388).fit(X_train,y_train
)

rfc = RandomForestClassifier(bootstrap=False,
max_depth=None,min_samples_leaf=1,min_samples_split=2,n_estimators=200).fit
(X_train,y_train)

mlp = MLPClassifier(activation = 'relu', solver = 'adam',
learning_rate='adaptive', alpha = 0.0001,

hidden_layer_sizes=(100,100,100)).fit(X_train,y_train)
```

Programski Kod 3.3. Inicijalizacija modela

Nakon što su postavljeni modeli, slijedeće će se prikazati evaluacija modela.

Prva metoda evaluacije je metoda *model.score ().* Budući da ta metoda nije dovoljno pouzdana, evaluacija će također uključivati *Cross validation s f1 micro i f1 macro* metodama bodovanja i *Stratified K-Fold.* Zbog neravnoteže skupa podataka (87% - 13%), naglasak će biti na *f1 macro* rezultatu i *Stratified K-Foldu.*

Razlika između *f1 micro i f1 macro* rezultata je u tome što *f1 mikro* daje jednaku važnost svakom uzorku, dok *f1 makro* daje jednaku važnost svakoj klasi, što čini *f1 makro* informativnijim kada se radi o neuravnoteženom skupu podataka.

Budući da u *Cross validation* metodi nije obećano da će svaki *fold* imati uravnoteženi dio svake klase, kako bi se izbjegla slučajnost, koristi se *Stratified K-Fold*. *Stratified K-Fold* je *Cross validation* provjera u kojoj svaki *fold* ima jednaku raspodjelu klasa, tako da neće doći do netočnosti ako se "posreći" s odabranim *foldovima*. Navedene metode su raspisane u programskom kodu 3.4.

```
# Model evaluation
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

models=[svc,logreg,rfc,mlp]

for model in models:

    print(f"------------Model: {model}------------")
    print(f"Model score: {model.score(X_test,y_test)*100:.3f}%")

    skf = StratifiedKFold(n_splits=10)
    skf_splits = skf.get_n_splits(X_train, y_train)
```

```
    skf_scores=[]

    for train_index, test_index in skf.split(X_train, y_train):
        X_train_fold , X_test_fold = X_train[train_index],
X_train[test_index]
        y_train_fold, y_test_fold = y_train[train_index],
y_train[test_index]
        model.fit(X_train_fold, y_train_fold)
        skf_scores.append(model.score(X_test_fold,y_test_fold))

    cv_scores_macro = cross_val_score(model, X, y, cv=skf_splits,
scoring='f1_macro')
    cv_scores_micro = cross_val_score(model, X, y, cv=skf_splits,
scoring='f1_micro')

    print(f"Cross-validation f1-macro score: {np.mean(cv_scores_macro) *
100:.3f}%")
    print(f"Cross-validation f1-micro score: {np.mean(cv_scores_micro) *
100:.3f}%")
    print(f"Stratified k-fold score: {np.mean(skf_scores) * 100:.3f}%\n")
```

Programski kod 3.4.. Evaluacija modela

```
------------Model: SVC(C=10, gamma=0.1)------------
Model score: 98.266%
Cross-validation f1-macro score: 96.961%
Cross-validation f1-micro score: 98.618%
Stratified k-fold score: 98.667%

------------Model: RandomForestClassifier(bootstrap=False, n_estimators=200)------------
Model score: 97.010%
Cross-validation f1-macro score: 94.587%
Cross-validation f1-micro score: 97.649%
Stratified k-fold score: 97.718%

------------Model: MLPClassifier(alpha=0.05, hidden_layer_sizes=(100, 100, 100))------------
Model score: 97.907%
Cross-validation f1-macro score: 95.909%
Cross-validation f1-micro score: 98.044%
Stratified k-fold score: 98.333%

------------Model: MLPClassifier(hidden_layer_sizes=(100, 100, 100), learning_rate='adaptive')------------
Model score: 97.608%
Cross-validation f1-macro score: 96.452%
Cross-validation f1-micro score: 98.331%
Stratified k-fold score: 98.590%
```

Slika 3.1. Evaluacija modela

Vidljivo na slici 3.1, SVC algoritam je najprecizniji dok je MPLClassifier na drugom mjestu. Osim preciznosti, SVC je učinkovitiji od ostalih algoritama.

RandomForestClassifier se pokazao najsporijim dok je MLPClassifier na drugom mjestu.

3.1.2 NLTK

Drugi način izrade filtera je pomoću biblioteke NLTK. *Natural Language Toolkit*, poznatiji pod nazivom NLTK je niz biblioteka i programa za simboličko i statističko obrađivanje pomoću programskog jezika Pythona. NLTK uključuje grafičko prikazane ogledne primjerke te je popraćen

opsežnom dokumentacijom i uključuje knjigu objašnjenja fundamentalnih načela iza obrađivačkih zadataka podržanih od strane ovih alata

NLTK je pretežito namijenjen učenju o računalnoj obradi prirodnog jezika ili pak za istraživanja u obradi prirodnog jezika i sličnih bliskih struka poput empirijske lingvistike, kognitivnih znanosti, umjetne inteligencije, vađenju informacija (iz dokumenata) i strojnom učenju.

Prije početka izrade spam filtera, potrebno je preuzeti model za rastavljanje riječi (engl. Tokenizer model) i korpus nebitnih riječi. Preuzimanja se rade na način da se pozove naredba nltk.download('punkt') – model za rastavljanje riječi te nltk.download('stopwords') – korpus nebitnih riječi. Nakon preuzimanja uključuju se potrebne biblioteke, vidljivo u programskom kodu 3.5.

```
import numpy as np
import pandas as pd
import nltk
import sys
import sklearn

from sklearn.preprocessing import LabelEncoder
nltk.download('stopwords')
from nltk.corpus import stopwords
nltk.download('punkt')
from nltk.tokenize import word_tokenize
```

Programski kod 3.5. Preuzimanja i učitavanje biblioteka

Funkcija find_features pronalazi značajke koje se nalaze u poruci i uspoređuje ih sa značajkama iz liste word_features. Funkcija u programskom kodu 3.6. vraća listu značajki prikupljenih iz poruke.

```
def find_features(message, word_features):
    words = word_tokenize(message)
    features = {}
    for word in word_features:
        features[word] = (word in words)
    return features
```

Programski kod 3.6. Funkcija find_features

U programskom kodu 3.7. obavlja se učitavanje skupa podataka (liste poruka) pomoću biblioteke pandas funkcijom pd.readtable() te se deklarira broj uzoraka pojedine klase.

```
df = pd.read_table('SMSSpamCollection', header = None, encoding = 'utf-8')
classes = df[0]
```

Programski kod 3.7. Učitavanje skupa podataka

U programskom kodu 3.8. Klasama se pridružuju binarne vrijednosti (ham = 0, spam = 1) te se sprema sms poruka.

```
encode = LabelEncoder()
y = encode.fit_transform(classes)

text_messages = df[1]
```

Programski kod 3.8. Dodjela klasa

U programskom kodu 3.9. se vršila obrada poruke kao što su: uklanjanje punktacija, više razmaka između riječi smanjeno na jedan razmak, sva slova pretvorena u mala slova, itd… Metodom stop_words iz biblioteke nltk se uklanjaju sve riječi koje ne mijenjaju značenje u rečenici.

```
# regularni izrazi da zamijenimo email, brojeve, brojeve telefona, url,
simbole sa rijecima
processed = text_messages.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
'emailaddress')
processed = processed.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-
Z]{2,3}(/\S*)?$', 'webaddress')
processed = processed.str.replace(r'£|\$', 'moneysymb')
processed = processed.str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-
]?[\d]{4}$', 'phonenumbr')
processed = processed.str.replace(r'\d+(\.\d+)?', 'numbr')

# uklanjanje punktacija
processed = processed.str.replace(r'[^\w\d\s]', ' ')

# vise razmaka izmedju rijeci u jedan razmak
processed = processed.str.replace(r'\s+', ' ')

# uklanjanje razmaka na pocetku i kraju poruke
processed = processed.str.replace(r'^\s+|\s+?$', '')

# sva slova u mala slova
processed = processed.str.lower()

#uklanjanje stop rijeci - rijeci koje ne mijenjaju znacenje recenice
stop_words = set(stopwords.words('english'))
processed = processed.apply(lambda x: ' '.join(term for term in x.split()
if term not in stop_words))

ps = nltk.PorterStemmer()
processed = processed.apply(lambda x: ' '.join(ps.stem(term) for term in
x.split()))
```

Programski kod 3.9. Obrada poruka

U programskom kodu 3.10. Inicijaliziramo listu all_words u koju ćemo spremiti sve riječi nakon što se je izvršila obrada nad porukama. Dohvaćanje značajki se izvodi na način da se uzme 1500 najčešćih riječi iz liste all_words. Bibliotekom random se nasumično izmiješaju poruke.

Poziva se funkcija find_features() da bi se pronašle značajke u poruci te se značajke spremaju u listu feature_sets.

```
all_words = []
for message in processed:
    words = word_tokenize(message)
    for word in words:
        all_words.append(word)

all_words = nltk.FreqDist(all_words)

#make features
word_features = list(i[0] for i in all_words.most_common(1500))

messages = list(zip(processed, y))

seed = 1
np.random.seed = seed
```

```
np.random.shuffle(messages)

# find_features za svaki SMS
feature_sets = [(find_features(text, word_features), label) for (text,
label) in messages]
```

Programski kod 3.10. Pronalazak značajki

U programskom kodu 3.11. se odvija raspodjela skupa podataka na test i trening podatke u omjeru 25%-75%. Učitavamo sve algoritme kojima želimo usporediti preciznost. Modeli koje uspoređujemo su: "K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD Classifier", "Naive Bayes" i "SVM Linear".

```
from sklearn import model_selection
# split the data into training and testing datasets
training, testing = model_selection.train_test_split(feature_sets,
test_size = 0.25, random_state=seed)

# We can use sklearn algorithms in NLTK
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC

# Define models to train
names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic
Regression", "SGD Classifier",
         "Naive Bayes", "SVM Linear"]

classifiers = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    LogisticRegression(),
    SGDClassifier(max_iter = 100),
    MultinomialNB(),
    SVC(kernel = 'linear')
]
```

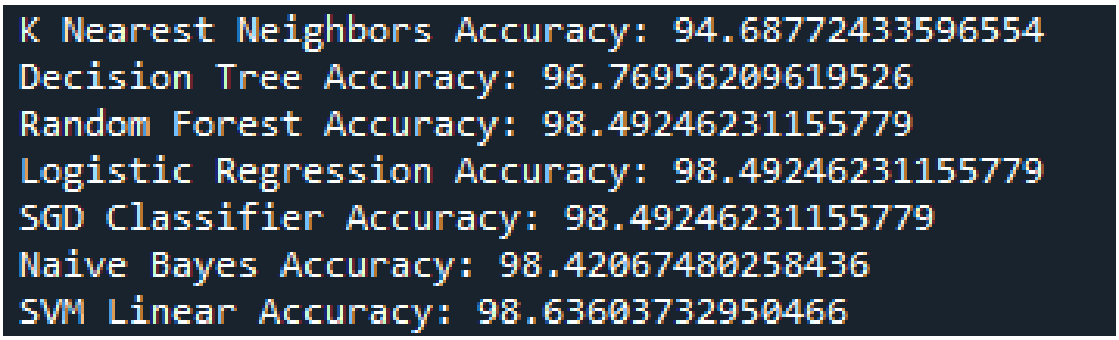
Programski kod 3.11. Učitavanje algoritama

Nakon postavljanja modela, programskim kodom 3.12. se evaluiraju modeli. Rezultati evaluacije vidljivi su na slici 3.2.

```
models = zip(names, classifiers)

for name, model in models:
    nltk_model = SklearnClassifier(model)
    nltk_model.train(training)
    accuracy = nltk.classify.accuracy(nltk_model, testing)*100
    print("{} Accuracy: {}".format(name, accuracy))
```

Programski kod 3.12. Evaluacija modela



Slika 3.2. evaluacija modela s NLTK bibliotekom

Na slici 3.2. vidljivo je da je SVM Linear opet najpreciznji algoritam za prepoznavanje SPAM poruka, dok K Nearest Neighbors i Decision Tree Accuracy imaju dva najlošija rezultata.

Voting classifier

Voting classifier je metoda klasifikacije koja koristi više klasifikatora za predviđanje. Vrlo je primjenjiv u situacijama kada je inženjer strojnog učenja zbunjen oko toga koju klasifikacijsku metodu koristiti. Stoga, koristeći predviđanja iz više klasifikatora, *Voting classifier* izrađuje predviđanja na temelju najčešćeg.

Voting classifier je model strojnog učenja koji trenira na skupu brojnih modela i predviđa izlaz (klasu) na temelju njihove najveće vjerojatnosti odabrane klase kao rezultata. Jednostavno objedinjuje rezultate svakog klasifikatora koji je u skupu i predviđa izlaznu klasu na temelju većine glasova. Ideja je da umjesto stvaranja zasebnih namjenskih modela i pronalaženja preciznosti za svaki od njih, stvorimo jedinstveni model koji trenira prema tim modelima i predviđa izlaz na temelju njihove kombinirane većine glasova za svaku izlaznu klasu.

U programskom kodu 3.13. se svi modeli osim K Neighbors i Decision Tree udružuju u VotingClassifier. Na slici 3.2. K Neighbors i Decision Tree su se pokazali previše netočni pa nisu uvršteni u zajednički model.

```
from sklearn.ensemble import VotingClassifier

names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic
Regression", "SGD Classifier",
         "Naive Bayes", "SVM Linear"]

classifiers = [
    RandomForestClassifier(),
    LogisticRegression(),
    SGDClassifier(max_iter = 100),
    MultinomialNB(),
    SVC(kernel = 'linear')
]
models = list(zip(names, classifiers))

nltk_ensemble = SklearnClassifier(VotingClassifier(estimators = models,
voting = 'hard', n_jobs = -1))
nltk_ensemble.train(training)
accuracy = nltk.classify.accuracy(nltk_model, testing)*100
print("Voting Classifier: Accuracy: {}".format(accuracy))
```

Programski kod 3.13. Udruživanje modela u ensemble

Na slici 3.3 vidljiv je rezultat preciznosti Voting Classifiera koji izgleda prilično uspješno sa 98.636%.

```
Voting Classifier: Accuracy: 98.63603732950466
```

Slika 3.3. Preciznost Voting Classifiera

Matrica zabune je matrica u kojoj svaki redak predstavlja predviđenu klasu dok stupac predstavlja stvarnu klasu. Matrica zabune je uvijek simetrična odnosno ima jednak broj stupaca koliko ima redaka a taj broj je jednak broju klasa. U ovom radu radit će se binarna klasifikacija odnosno imat ćemo dvije klase te za takvu klasifikaciju matrica zabune izgleda kao tablica ispod.

Programski kod 3.14. prikazuje ispis matrice zabune za model Voting Classifier koja je vidljiva na slici 3.4.

```
# make class label prediction for testing set
from sklearn.metrics import classification_report, confusion_matrix

txt_features, labels = zip(*testing)

prediction = nltk_ensemble.classify_many(txt_features)

#print a confusion matrix and a classification report
print(classification_report(labels, prediction))

pd.DataFrame(
    confusion_matrix(labels, prediction),
    index = [['actual', 'actual'], ['ham', 'spam']],
    columns = [['predicted', 'predicted'], ['ham', 'spam']])
```

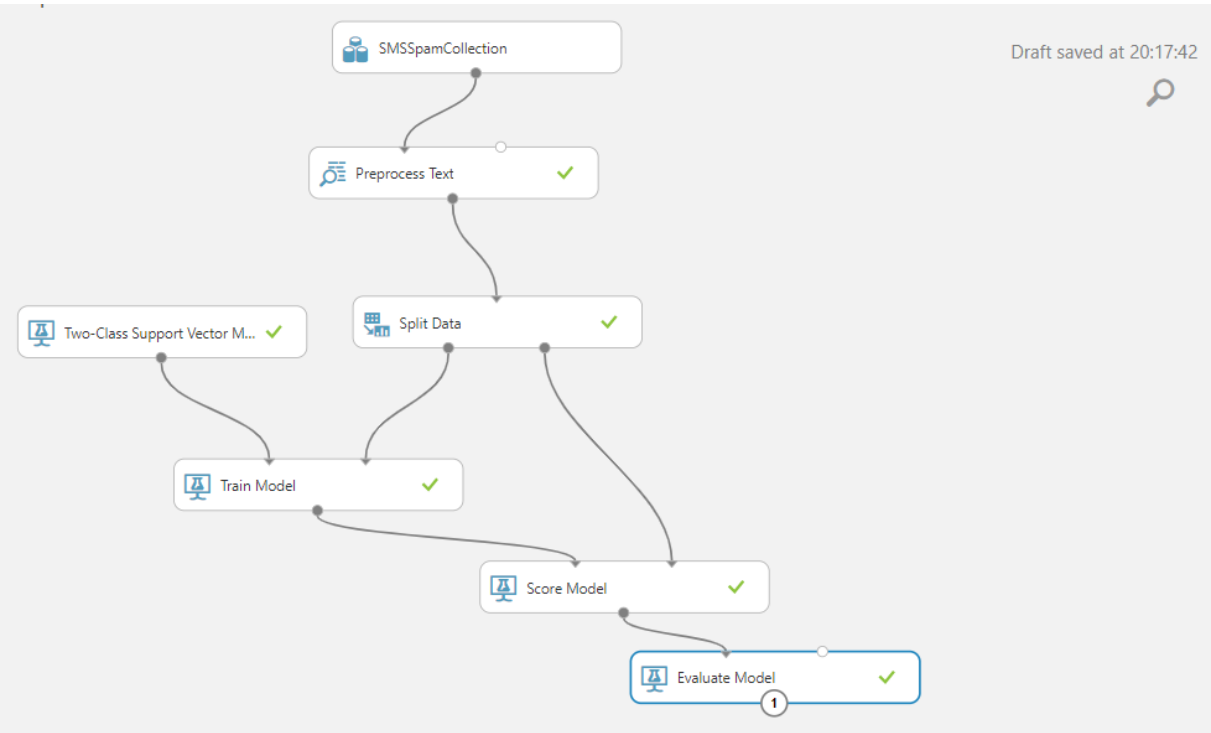
Programski kod 3.14. Udruživanje modela u ensemble

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1204
1	0.99	0.92	0.95	189
accuracy			0.99	1393
macro avg	0.99	0.96	0.97	1393
weighted avg	0.99	0.99	0.99	1393

Slika 3.4. matrica zabune za model Voting Classifier

3.2 Model strojnog učenja

Prema slikama 3.1. i 3.4. zaključeno je kako SVM algoritam najbolje obavlja NLP posao za SMS SPAM klasifikaciju u oba spaCy i NLTK pristupima. Iz razloga što u Azureu ne postoji VotingClassifier, odabrani algoritam učenja je SVM. Zbog brzine procesiranja poruka, korišten je spaCy način obrade podataka, tj. pretvaranje SMS poruke u vektor. Nakon pretvorbe, SVM lakoćom klasificira danu poruku. Izgrađeni model kao web servis prikazan je slikom 3.5.



Slika 3.5. Korišteni model

3.3 Način korištenja API-ja

Dobiveni API kod koristi se u Python skripti programskim kodom 3.15. unutar koje se zadaju podaci data, tj. SMS poruke koje se žele klasificirati. Kao povratnu vrijednost, response vraća predviđenu klasu i vjerojatnost da je predani string klasificiran kao spam.

```
body = str.encode(json.dumps(data))

url =
'https://ussouthcentral.services.azureml.net/workspaces/0c025ac58964406d8f4
1281fc76f62f6/services/eaa7db383038462dbe1c03efd94433ae/execute?api-
version=2.0&details=true'
api_key =
'PQXe70W/u2kCqguhih1yE2g7JDLhgfUnshr7nOdMM52s/MsEqNTKm7MpDABHqQc0DEmNewk3Z7
dwiVfiE3c7Uw=='
headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+
api_key)}

req = urllib.request.Request(url, body, headers)

try:
    response = urllib.request.urlopen(req)
    result = response.read()
    print(result)
```

```
except urllib.error.HTTPError as error:
    print("The request failed with status code: " + str(error.code))
    print(error.info())
    print(json.loads(error.read()))
```

Programski kod 3.15.

3.4 Klijentska aplikacija

Aplikacija napravljenog rješenja je Python skripta. Predavanje poruka "Hey, see you tomorrow?" i "URGENT! You have won a 1 week FREE membership in our £100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18" vraća rezultat vidljiv na slici 3.6.

Takva skripta se može pozivati na velikom broju poruka te ih lakoćom klasificirati.

```
Probability 1. message is spam: 0.0007747949566%
Probability 2. message is spam: 0.9999999403953%
```

Slika 3.6. Rezultat predavanja poruka

4. Zaključak

Klasifikacija SMS Spam poruka područje je *Natural Language Processinga*. Dobivene SMS poruke se uz pomoć *spaCy* biblioteke pretvaraju u vektore, što pretvara ovaj problem u posao koji se lakoćom obavlja. Različitost obilježenih *ham* i *spam* poruka računa se kutem između njihovih vektora. Klasifikacija takvih podataka može velik broj algoritama dobro obaviti, no zbog svog binarnog načina rada, SVM se ispostavio najefikasnijim u oba slučaja korištenja *NLTK* biblioteke i *spaCy* biblioteke. S ciljem povećanja preciznosti, u Pythonu je moguće koristiti i *VotingClassifier* koji temelji svoja predviđanja na glasu većine, tj. više modela. Iz razloga što *VotingClassifier* ne postoji na *Azureu*, korišten je SVM za kreiranje Web servisa koji preko svog API-ja obavlja klasifikaciju predanih poruka. Kao povratnu vrijednost Web servis vraća predviđenu klasu i vjerojatnost da je predana poruka spam. Pomoću izgrađenog Web servisa dobivanje predikcije poruka je brzo i ne mora se znati pozadinska logika koja se izvršava pa se ovaj model može koristiti kao dio različitih aplikacija koje rukuju porukama u kojima postoji vjerojatnost da je određena poruka spam.

5. Poveznice i literatura

Programskom je rješenju moguće pristupiti preko:

Programsko rješenje na GitLabu
ML model

[1] https://scikit-learn.org/stable/modules/naive_bayes.html
[2] https://www.geeksforgeeks.org/naive-bayes-classifiers/
[3] https://scikit-learn.org/stable/modules/svm.html
[4] https://www.geeksforgeeks.org/ml-voting-classifier-using-sklearn/
[5] https://en.wikipedia.org/wiki/Confusion_matrix
[6] https://docs.microsoft.com/en-us/azure/machine-learning/classic/deploy-a-machine-learning-web-service