VISOKA ŠKOLA STRUKOVNIH STUDIJA ZA INFORMACIONE TEHNOLOGIJE

Veštačka inteligencija

Seminarski rad

Implementacija i uporedna analiza algoritama za detekciju e-mail spam poruka

Predmetni nastavnik:
prof. dr Aleksandar Simović

Student:
Mihailo Anđelić 284/24

Beograd

Januar, 2025.

SADRŽAJ

REZIME

Detekcija e-mail spam poruka predstavlja ključni izazov u savremenom informacionom društvu, jer svakodnevno dolazi do razmene miliona elektronskih poruka. Spam poruke ne samo da smanjuju produktivnost korisnika, već često sadrže zlonamerni sadržaj koji predstavlja ozbiljan bezbednosni rizik. U ovom radu analizirani su različiti algoritmi mašinskog učenja kako bi se identifikovao najefikasniji pristup za klasifikaciju poruka kao spam ili legitimnih (ham). Implementirani su Naivni Bajes, SVM, Decision Tree, KNN, Random Forest i Gradient Boosting algoritmi, a njihova efikasnost procenjena je korišćenjem metrika kao što su tačnost i preciznost. Eksperimenti su sprovedeni na standardnom dataset-u za detekciju spama, pri čemu su rezultati pokazali značajne razlike u performansama algoritama u zavisnosti od karakteristika podataka. Na kraju, izvršena je komparativna analiza performansi kako bi se utvrdio najefikasniji algoritam, uzimajući u obzir složenost implementacije i obrade podataka. Ovaj rad pruža smernice za izbor algoritama u realnim aplikacijama, doprinoseći unapređenju rešenja za filtriranje neželjenih poruka.

KLJUČNE REČI

Detekcija e-mail spama, mašinsko učenje, klasifikacija podataka, Naivni Bajes, SVM, Decision Tree, KNN, Random Forest, Gradient Boosting, komparativna analiza.

1. UVOD

E-mail komunikacija ima ključnu ulogu u savremenom digitalnom društvu, ali je suočena sa brojnim izazovima, među kojima spam poruke predstavljaju jedan od najznačajnijih. Spam poruke, koje uključuju neželjene ili nelegitimne e-mailove, često služe za reklamiranje, prevaru ili distribuciju zlonamernog softvera. Ovaj problem ne samo da negativno utiče na korisničko iskustvo, već izaziva i finansijske gubitke za organizacije zbog povećanih troškova filtriranja neželjenih poruka i potencijalnih bezbednosnih incidenata.

Razvoj efikasnih metoda za detekciju spama postao je ključna tema u oblasti mašinskog učenja. Algoritmi za klasifikaciju, poput Naivnog Bajesovog algoritma, SVM-a, Decision Tree, KNN-a, Random Forest-a i Gradient Boosting-a, nude različite pristupe rešavanju ovog problema, pri čemu svaki od njih ima svoje prednosti i ograničenja. Ovi algoritmi koriste osobine e-mail poruka, kao što su reči, frekvencija ključnih reči i druge metrike, kako bi klasifikovali poruke kao spam ili legitimne (ham).

Cilj ovog rada je istražiti efikasnost navedenih algoritama kroz implementaciju i eksperimentalnu analizu na stvarnom dataset-u. Korišćenjem ključnih performansi metrika kao što su tačnost i preciznost, rad pruža komparativnu analizu rezultata sa fokusom na identifikaciju najefikasnijeg algoritma. Takođe, razmatrani su aspekti kao što su brzina obrade, skalabilnost i složenost implementacije, kako bi se pružile sveobuhvatne preporuke za primenu u realnim scenarijima.

Ovaj rad doprinosi razumevanju primene različitih algoritama mašinskog učenja u kontekstu detekcije spama, omogućavajući bolji izbor algoritma u skladu sa specifičnim potrebama korisnika i resursima dostupnim za implementaciju.

2. ALGORITMI (NAČIN RADA I DOMEN PRIMENE)

2.1 NAIVE BAYES

Način rada

Algoritam koristi Bayesovu teoremu za procenu verovatnoće da određeni podatak pripada određenoj klasi. U kontekstu klasifikacije e-mailova, na primer, može analizirati učestalost određenih reči kako bi odredio da li je poruka spam ili legitimna. Iako pretpostavka o nezavisnosti atributa nije uvek tačna, Naivni Bajesov algoritam često daje zadovoljavajuće rezultate u praksi. Takođe, efikasan je u radu sa velikim količinama podataka i brzo se obučava.

Domen primene

Naivni Bajesov algoritam se široko koristi u klasifikaciji teksta, uključujući detekciju e-mail spama, analizu sentimenta, kategorizaciju dokumenata i prepoznavanje jezika. Njegova jednostavnost omogućava brzo donošenje odluka čak i na velikim skupovima podataka, što ga čini pogodnim za aplikacije u realnom vremenu. Međutim, njegova preciznost može biti smanjena kada atributi nisu zaista nezavisni, što može uticati na performanse u složenijim aplikacijama. Zbog svoje efikasnosti, koristi se i u filtriranju pretraga na internetu, sistemima za preporuke i detekciji zlonamernih aktivnosti [1].

2.2 SVM (SUPPORT VECTOR MACHINES)

Način rada

SVM funkcioniše tako što identifikuje hiper-ravan koja najbolje razdvaja klase u skupu podataka, maksimizirajući marginu, tj. razdaljinu između najbližih tačaka različitih klasa i same hiper-ravni. U slučajevima kada podaci nisu linearno razdvojivi, SVM koristi kernel funkcije za mapiranje podataka u viši dimenzionalni prostor gde se može pronaći linearna razdvajajuća hiper-ravan. Ova fleksibilnost omogućava SVM-u da efikasno rešava i linearne i nelinearne probleme klasifikacije.

Domen primene

SVM se široko koristi u raznim oblastima, uključujući klasifikaciju teksta (kategorizacija e-mailova kao spam ili ne-spam, analiza sentimenta u recenzijama), prepoznavanje obrazaca (prepoznavanje rukopisa, identifikacija lica na slikama), bioinformatiku (klasifikacija sekvenci gena ili proteina) i analizu finansijskih podataka (predviđanje kretanja cena akcija ili detekcija prevara). Zbog svoje sposobnosti da efikasno radi sa visokodimenzionalnim podacima i različitim tipovima kernela, SVM je popularan izbor za mnoge probleme klasifikacije i regresije [2].

2.3 DECISION TREE

Način rada

Algoritam stabla odluke započinje od korena i postepeno deli podatke na podskupove na osnovu atributa koji pružaju najveću informaciju za razdvajanje klasa. Ovi atributi se biraju korišćenjem mera kao što su Gini indeks ili entropija, koje kvantifikuju "čistoću" podataka u čvoru. Proces se nastavlja rekurzivno za svaki podskup, formirajući grane stabla, sve dok svi podaci u listu ne pripadaju istoj klasi ili dok se ne ispuni neki drugi kriterijum zaustavljanja, kao što je maksimalna dubina stabla. Ova hijerarhijska struktura omogućava donošenje odluka putem sekvencijalnih pitanja zasnovanih na atributima podataka.

Domen primene

Stabla odluke se široko primenjuju u oblastima kao što su finansije, za procenu kreditnog rizika, medicina, za dijagnostiku bolesti, marketing, za segmentaciju kupaca, i mnoge druge. Njihova popularnost proizlazi iz jednostavnosti interpretacije i vizualizacije, što olakšava objašnjavanje odluka donetih modelom. Međutim, sklona su pretreniravanju, posebno kada su previše duboka ili kada podaci sadrže šum. Zbog toga se često koriste tehnike kao što su orezivanje stabla ili ansambl metode poput Random Forest-a za poboljšanje performansi i generalizacije modela [3].

2.4 KNN (K-NEAREST NEIGHBORS)

Način rada

Kada se KNN koristi za klasifikaciju, algoritam funkcioniše na sledeći način:

1. **Izbor broja suseda (K):** Određuje se broj K, koji predstavlja broj najbližih suseda koji će se uzeti u obzir prilikom donošenja odluke.
2. **Izračunavanje udaljenosti:** Za novu tačku, izračunavaju se udaljenosti do svih tačaka u trening skupu podataka. Najčešće se koristi Euklidska udaljenost, ali mogu se koristiti i druge metrike.
3. **Identifikacija K najbližih suseda:** Na osnovu izračunatih udaljenosti, identifikuju se K najbliže tačke.
4. **Donošenje odluke:** Klasa nove tačke određuje se na osnovu većine klasa K najbližih suseda. U slučaju regresije, predikcija se vrši kao prosečna vrednost ciljne promenljive tih suseda.

Prednost KNN-a je u njegovoj jednostavnosti i efikasnosti za manje skupove podataka, jer ne zahteva prethodnu obuku modela. Međutim, za velike skupove podataka može biti spor, jer mora da izračuna udaljenost za svaku tačku u trening skupu prilikom klasifikacije nove tačke. Izbor vrednosti K takođe igra ključnu ulogu u performansama modela, jer previše mala ili previše velika vrednost K može dovesti do nezadovoljavajućih rezultata.

Domen primene

KNN se široko koristi u raznim aplikacijama, uključujući detekciju spama, prepoznavanje obrazaca, analizu slika, kao i u sistemima za preporuke i prepoznavanje karakteristika korisnika. Zbog svoje jednostavnosti i efikasnosti u klasifikaciji malih i srednje velikih skupova podataka, često se koristi u aplikacijama koje ne zahtevaju kompleksnu obuku modela. Međutim, za velike skupove podataka može biti spor, jer mora da izračuna udaljenost za svaku tačku u trening skupu prilikom klasifikacije nove tačke. Takođe, osetljiv je na "šum" u podacima, jer svi primeri imaju jednaku težinu, bez obzira na njihovu relevantnost za krajnju odluku [4].

2.5 RANDOM FOREST

Način rada

Random Forest je metoda zasnovana na kombinovanju više stabala odluke kako bi se poboljšala tačnost klasifikacije. Svako stablo u šumi koristi različite, slučajno odabrane podskupove podataka i selekciju atributa pri svakom koraku. Ovaj pristup smanjuje problem pretreniravanja, jer odluka nije zasnovana na jednom stablu, već na glasanju svih stabala u šumi. Klasifikacija je određena većinom glasova, što često dovodi do boljih rezultata od jednog stabla. Korišćenjem metrika poput Gini indeksa ili entropije, Random Forest optimizuje razdvajanje na svakom čvoru.

Domen primene

Random Forest je široko korišćen u mnogim industrijama, zahvaljujući svojoj robusnosti i stabilnim rezultatima, čak i u prisustvu šuma u podacima. Često se primenjuje u detekciji spama, analizi tržišta, biomedicinskim istraživanjima, predviđanju potrošnje i finansijskim analizama. Iako je precizan, algoritam zahteva značajnu količinu memorije i procesorskih resursa, posebno za velike dataset-ove, jer obuhvata treniranje velikog broja stabala. Takođe, može biti spor u fazi predviđanja kada broj stabala raste [5].

2.6 GRADIENT BOOSTING

Način rada

Gradient Boosting je tehnika koja koristi seriju slabih modela (najčešće stabala odluke) treniranih sekvencijalno, pri čemu svaki sledeći model ispravlja greške prethodnog. Algoritam se fokusira na podatke koje je prethodni model klasifikovao pogrešno, čime se postepeno smanjuju greške i poboljšava preciznost. Na svakom koraku koristi funkciju greške kako bi unapredio model. Popularne implementacije, poput XGBoost i LightGBM, koriste optimizovane metode za ubrzanje učenja i bolju obradu velikih dataset-ova.

Domen primene

Gradient Boosting se koristi u oblastima gde je potrebna visoka preciznost, kao što su finansijske analize, detekcija prevara, prepoznavanje obrazaca i detekcija spama. Često je izbor u takmičenjima u mašinskom učenju zbog svoje visoke tačnosti. Međutim, ovaj algoritam je računski intenzivan i može zahtevati značajno vreme za obuku, posebno na velikim dataset-ovima. Uspeh modela zavisi od pažljivog podešavanja hiperparametara, jer nepravilna podešavanja mogu loše uticati na performanse. XGBoost i LightGBM omogućavaju brži rad i veću skalabilnost [6].

3. ČIŠĆENJE PODATAKA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

```
df = pd.read_csv('spam.csv', encoding='ISO-8859-1')
```

```
df.sample(5)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
2977	ham	Yar lor... Keep raining non stop... Or u wan 2...	NaN	NaN	NaN
608	ham	Neva mind it's ok..	NaN	NaN	NaN
5373	ham	K I'll head out in a few mins, see you there	NaN	NaN	NaN
3846	spam	Fantasy Football is back on your TV. Go to Sky...	NaN	NaN	NaN
3962	ham	If you ask her or she say any please message.	NaN	NaN	NaN

```
df.shape
```

```
(5572, 5)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'> RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
#       Column   Non-Null Count Dtype

v1      5572 non-null    object
v2      5572 non-null    object
Unnamed: 2 50 non-null   object
Unnamed: 3 12 non-null   object
Unnamed: 4 6 non-null    object dtypes: object(5)
memory usage: 217.8+ KB
```

Ovde možemo da vidimo da poslednje tri kolone nemaju naziv i nemaju nikakvu svrhu. Zato je potrebno da uklonimo ove tri kolone iz našeg skupa podataka.

```
# dropping the last 3 columns
df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)
```

```
df.sample(5)
```

	v1	v2
367	spam	Here is your discount code RP176781. To stop f...
3856	ham	No! But we found a diff farm shop to buy some ...
10	ham	I'm gonna be home soon and i don't want to tal...
4122	ham	Cool, want me to go to kappa or should I meet ...
1465	spam	YOU 07801543489 are guaranteed the latests Nok...

```
# renaming the columns
df.rename(columns={'v1':'target','v2':'text'},inplace=True)
df.sample(5)
```

	target	text
5265	ham	Gud ni8.swt drms.take care
1574	ham	My sis is catching e show in e afternoon so i'...
5445	ham	And that's fine, I got enough bud to last most...
1737	ham	I cant pick the phone right now. Pls send a me...
1254	ham	What your plan for pongal?

Potrebno je kodirati ciljne kategorije Ham i Spam pomoću LabelEncoder() funkcije, tako da Ham bude 0, a Spam 1.

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
```

```
df['target'] = encoder.fit_transform(df['target'])
```

```
df.head()
```

	target	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...

```
df.isnull().sum()
#in this dataset, no duplicate values exist
```

```
target   0
text     0
dtype: int64
```

```
df.duplicated().sum()
```

```
403
```

U ovom skupu podataka postoji ukupno 403 duplikata koje je potrebno ukloniti.

```
df = df.drop_duplicates(keep='first')
#keeping only the first instance of a value
```

```
df.duplicated().sum()
```

```
0
```

4. ISTRAŽIVAČKA ANALIZA PODATAKA (EDA)

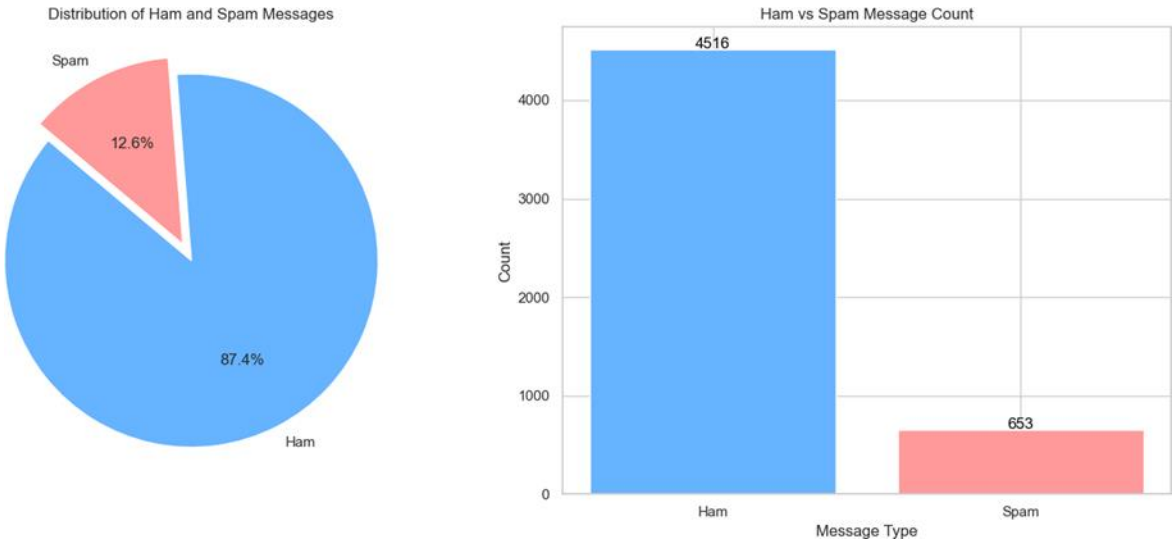
Exploratory Data Analysis (EDA) je pristup analizi skupova podataka kako bi se sumirale njihove glavne karakteristike, često uz pomoć statističkih grafika i drugih metoda vizualizacije podataka. EDA je ključni korak u procesu mašinskog učenja, jer pomaže naučnicima i analitičarima podataka da razumeju strukturu, obrasce i veze unutar podataka, što može uticati na naredne korake u procesu modelovanja.

Analiza 1: Provera neskladnosti u podacima

```
df['target'].value_counts()
#individual count of each category ham and spam
```

```
target
0       4516
1       653
Name: count, dtype: int64
```

```
# Define colors for the pie chart
colors = ['#66b3ff', '#ff9999']

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot the pie chart on the first subplot
axes[0].pie(
    df['target'].value_counts(),
    labels=['Ham', 'Spam'],
    autopct="%0.1f%%",  # Display percentages with one decimal
    colors=colors,
    explode=[0, 0.1],  # Separate the Spam segment for emphasis
    startangle=140     # Rotate the chart to start at a specific angle
)
axes[0].set_title('Distribution of Ham and Spam Messages')

# Plot the stacked column chart on the second subplot
counts = df['target'].value_counts()

axes[1].bar(['Ham', 'Spam'], counts, color=colors)

# Add data labels on top of each bar
for i, v in enumerate(counts):
    axes[1].text(i, v + 5, str(v), ha='center', fontsize=12, color='black')

# Add title and labels to the stacked column chart
axes[1].set_title('Ham vs Spam Message Count')
axes[1].set_ylabel('Count')
axes[1].set_xlabel('Message Type')

plt.tight_layout()
plt.show()
```



Slika 1. Vizualizacija podataka

Iz ove analize možemo da vidimo da je procenat legitimnih (ham) poruka u našem skupu podataka mnogo veći od procenta spam poruka. Tako da su podaci neuravnoteženi.

Analiza 2:

- Koliko karaktera sadrži poruka?
- Koliko reči sadrži poruka?
- Koliko rečenica sadrži poruka?

```
import nltk
```

```
nltk.download('punkt')
```

```
#Fetching no of characters in each mail
df['num_characters'] = df['text'].apply(len)
```

```
df.head()
```

	target	text	num_characters
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

```
#Fetching no of words in each mail
df['num_words'] = df['text'].apply(lambda x:len(nltk.word_tokenize(x)))
```

```
df.head()
```

	target	text	num_characters	num_words
0	0	Go until jurong point, crazy.. Available only ...	111	24
1	0	Ok lar... Joking wif u oni...	29	8
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37
3	0	U dun say so early hor... U c already then say...	49	13
4	0	Nah I don't think he goes to usf, he lives aro...	61	15

```
#Fetching no of sentences in each mail
df['num_sentences'] = df['text'].apply(lambda x:len(nltk.sent_tokenize(x)))
```

```
df.head()
```

	target	text	num_characters	num_words	num_sentences
0	0	Go until jurong point, crazy.. Available only ...	111	24	2
1	0	Ok lar... Joking wif u oni...	29	8	2
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2
3	0	U dun say so early hor... U c already then say...	49	13	1
4	0	Nah I don't think he goes to usf, he lives aro...	61	15	1

Statistika svih numeričkih vrednosti

```
df[['num_characters','num_words','num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	5169.000000	5169.000000	5169.000000
mean	78.977945	18.455794	1.965564
std	58.236293	13.324758	1.448541
min	2.000000	1.000000	1.000000
25%	36.000000	9.000000	1.000000
50%	60.000000	15.000000	1.000000
75%	117.000000	26.000000	2.000000
max	910.000000	220.000000	38.000000

Statistika "ham" poruka

```
df[df['target'] == 0][['num_characters','num_words','num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	4516.000000	4516.000000	4516.000000
mean	70.459256	17.123782	1.820195
std	56.358207	13.493970	1.383657
min	2.000000	1.000000	1.000000
25%	34.000000	8.000000	1.000000
50%	52.000000	13.000000	1.000000
75%	90.000000	22.000000	2.000000
max	910.000000	220.000000	38.000000

Statistika "spam" poruka

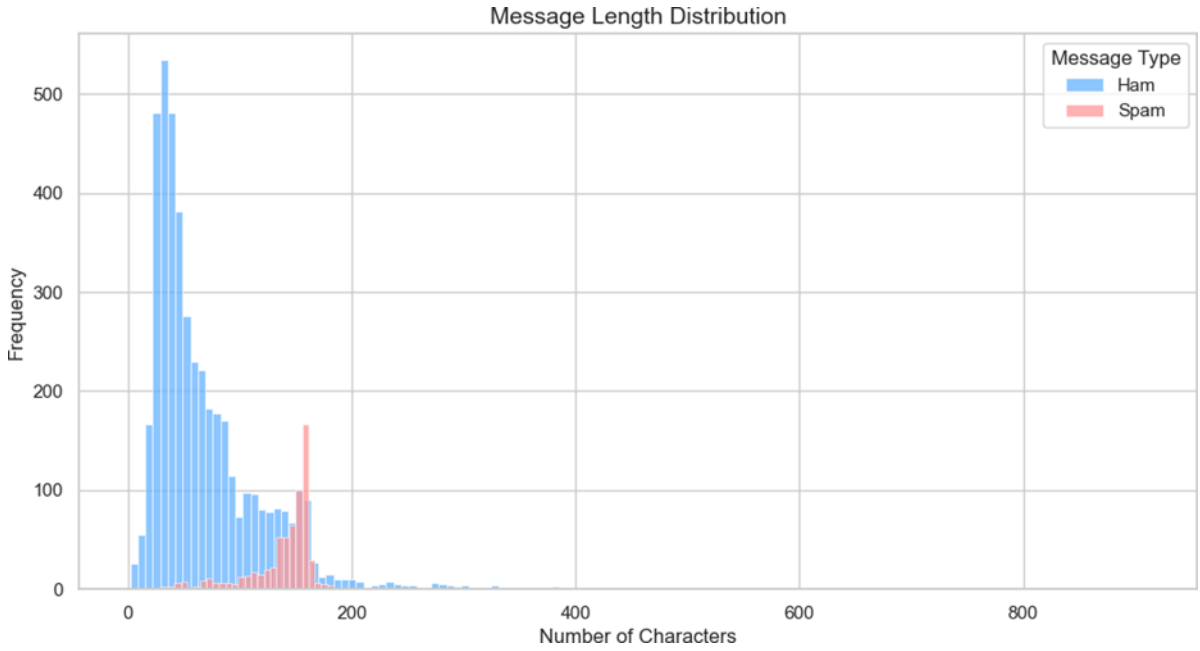
```
df[df['target'] == 1][['num_characters','num_words','num_sentences']].describe()
```

	num_characters	num_words	num_sentences
count	653.000000	653.000000	653.000000
mean	137.891271	27.667688	2.970904
std	30.137753	7.008418	1.488425
min	13.000000	2.000000	1.000000
25%	132.000000	25.000000	2.000000
50%	149.000000	29.000000	3.000000
75%	157.000000	32.000000	4.000000
max	224.000000	46.000000	9.000000

Histogrami za statističke podatke

```
import seaborn as sns
```

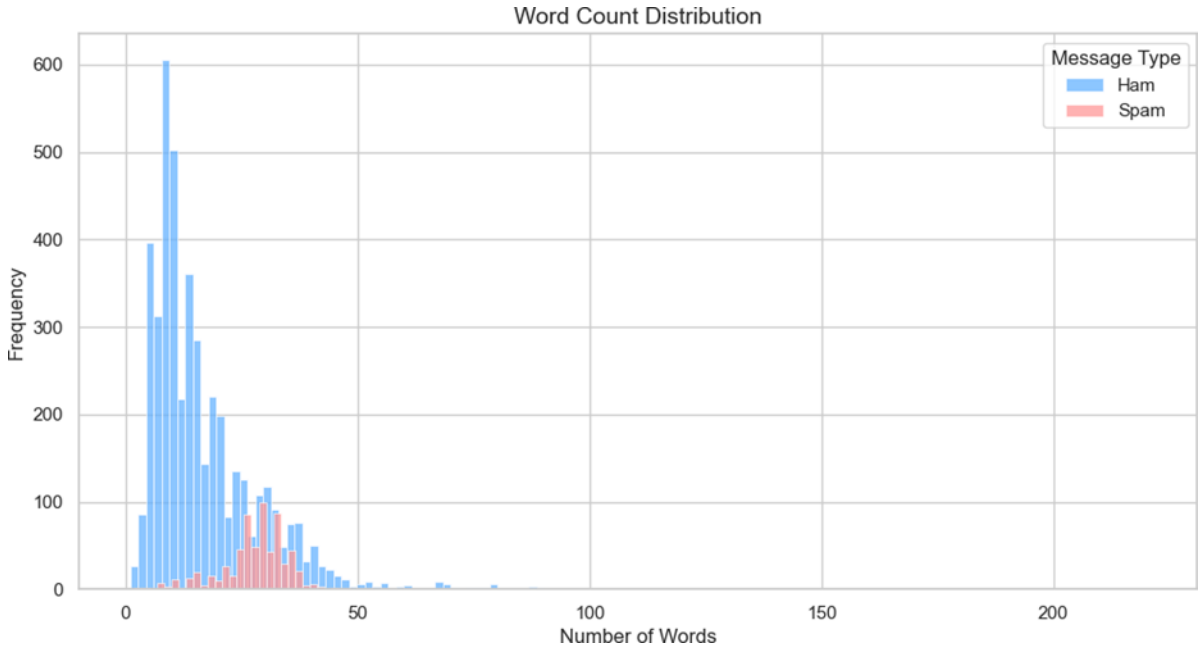
```
# Set figure size
plt.figure(figsize=(12, 6))

# Plot histogram for 'Ham' messages
sns.histplot(
    df[df['target'] == 0]['num_characters'],
    color='#66b3ff',
    label='Ham'
)

# Plot histogram for 'Spam' messages
sns.histplot(
    df[df['target'] == 1]['num_characters'],
    color='#ff9999',
    label='Spam'
)

# Add a title and legend
plt.title('Message Length Distribution', fontsize=14)
plt.xlabel('Number of Characters')
plt.ylabel('Frequency')
plt.legend(title='Message Type')
plt.show()
```



Slika 2. Vizualizacija dužine poruke i broja karaktera

Plavi segment predstavlja "ham" poruke, dok crveni segment označava spam poruke. U proseku, "ham" poruke sadrže veći broj karaktera po poruci u poređenju sa spam porukama, koje su, u većini slučajeva, primetno kraće.

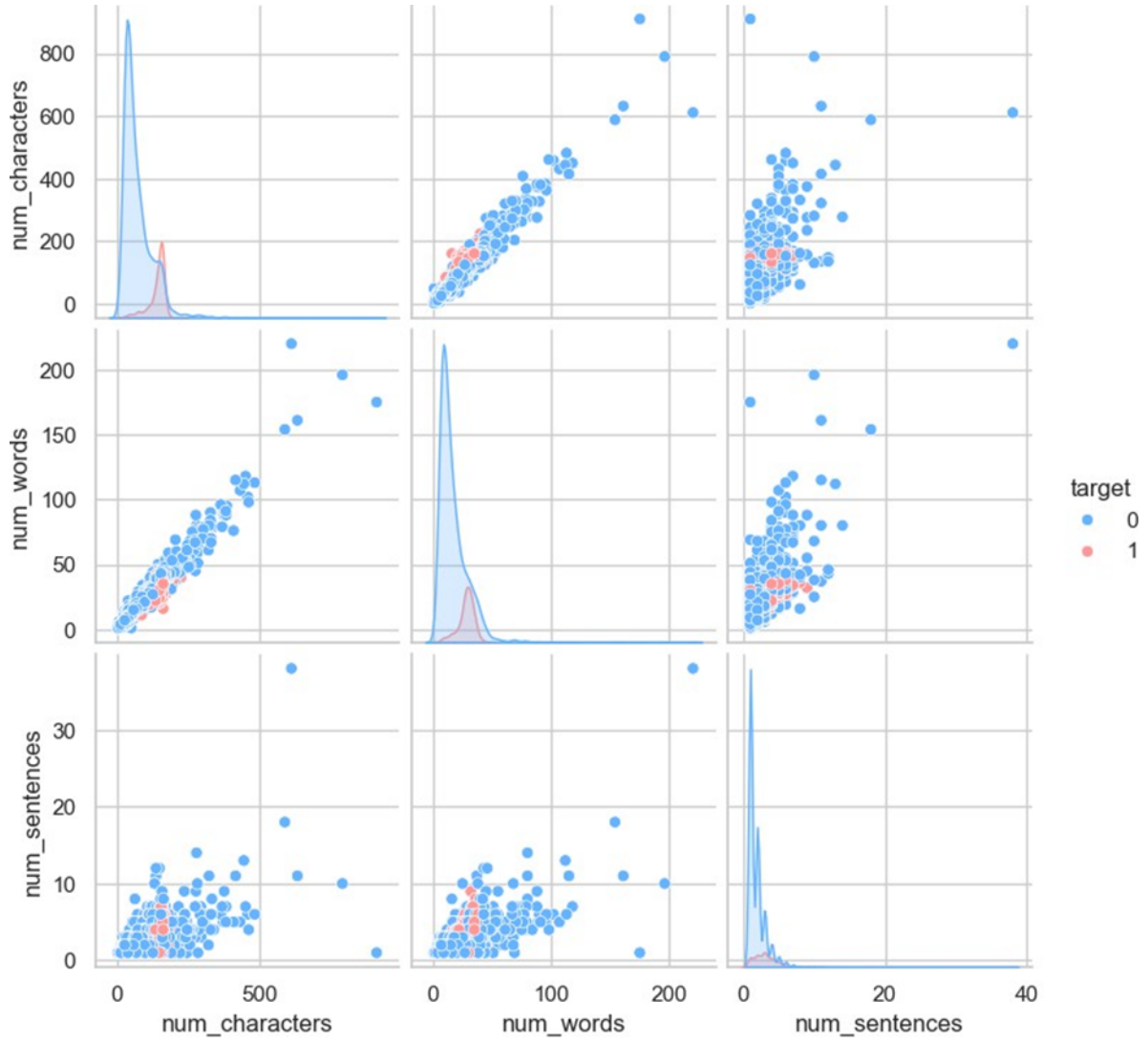
```
# Set figure size
plt.figure(figsize=(12, 6))

# Plot histogram for 'Ham' messages
sns.histplot(
    df[df['target'] == 0]['num_words'],
    color='#66b3ff',
    label='Ham'
)

# Plot histogram for 'Spam' messages
sns.histplot(
    df[df['target'] == 1]['num_words'],
    color='#ff9999',
    label='Spam'
)

# Add a title and legend
plt.title('Word Count Distribution', fontsize=14)
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.legend(title='Message Type')
plt.show()
```



Slika 3. Vizualizacija broja reči u poruci

Primetno je da "ham" poruke sadrže značajno veći broj reči u poređenju sa spam porukama.

```
# Define a custom color palette
custom_palette = {0: '#66b3ff', 1: '#ff9999'}

# Create a pairplot with the custom palette
sns.pairplot(df, hue='target', palette=custom_palette)

# Display the plot
plt.show()
```



Slika 4. Vizualizacija ekstremih vrednosti

Možemo primetiti da naš skup podataka obuhvata prisustvo ekstremnih vrednosti.

```
numeric_df = df.select_dtypes(include=['float64', 'int64'])
custom_palette = sns.color_palette("icefire", as_cmap=True)
sns.heatmap(numeric_df.corr(), annot=True, cmap=custom_palette)
plt.show()
```

Slika 5. Heatmap

5. OBRADA PODATAKA

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import string

nltk.download('stopwords')
nltk.download('punkt')
```

```
ps = PorterStemmer()
```

```
def transform_text(text):
    text = text.lower()
    text = nltk.word_tokenize(text)

    y = []
    for i in text:
        if i.isalnum():
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        if i not in stopwords.words('english') and i not in string.punctuation:
            y.append(i)

    text = y[:]
    y.clear()

    for i in text:
        y.append(ps.stem(i))


    return " ".join(y)
```

U mašinskom učenju, stop reči predstavljaju uobičajene reči koje se često uklanjaju iz tekstualnih podataka tokom faze njihove obrade. Ove reči se obično ne smatraju značajnim za prenošenje ključnih informacija o sadržaju teksta. U obradi prirodnog jezika (NLP), primeri stop reči uključuju reči poput „the“, „and“, „is“, „in“ i slične.

```
transform_text("I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.")
```

```
'gon na home soon want talk stuff anymor tonight k cri enough today'
```

Ovde možemo primetiti da su stop reči uklonjene i da su sve reči konvertovane u mala slova, što predstavlja transformaciju koja je bila neophodna za dalju obradu.

```
df['text'][10]
```

```
"I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today."
```

```
from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()
ps.stem('loving')
```

```
'love'
```

```
df['transformed_text'] = df['text'].apply(transform_text)
```

```
df.head()
```

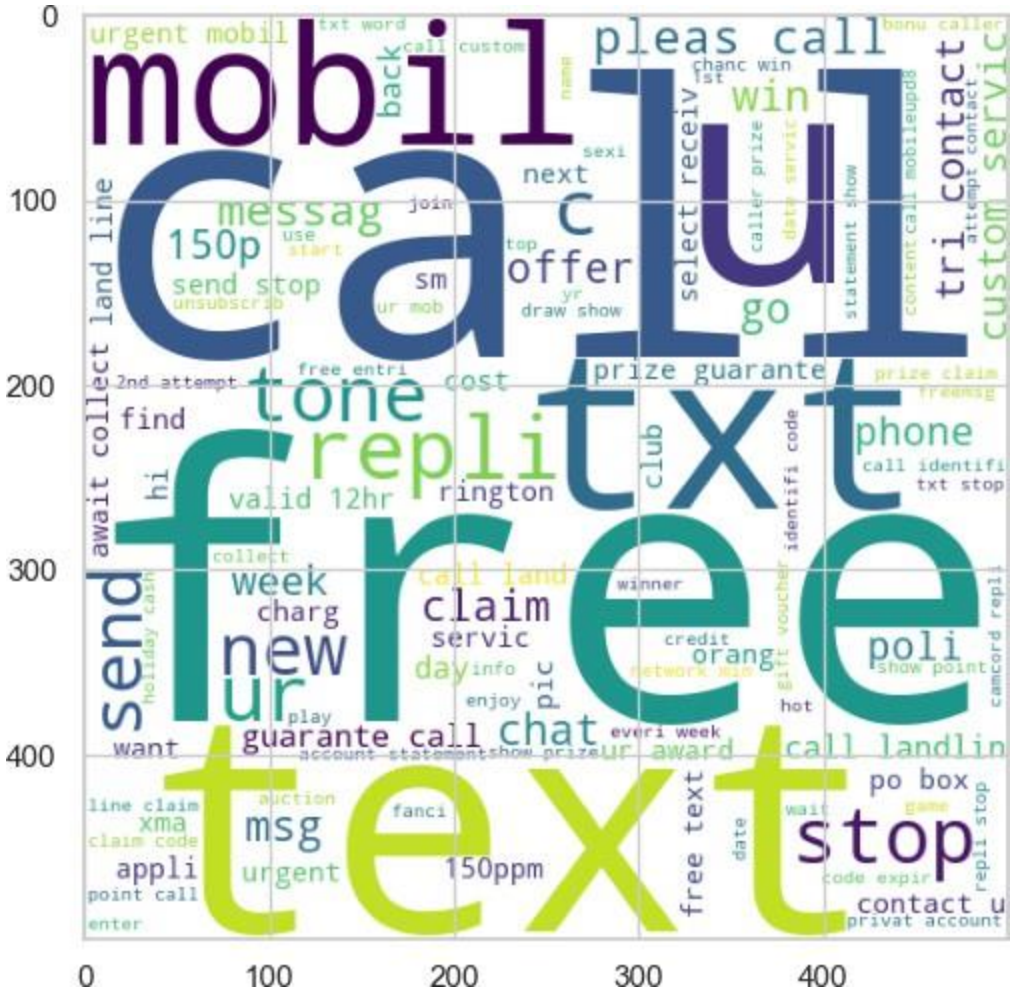
	target	text	num_characters	num_words	num_sentences	transformed_text
0	0	Go until jurong point, crazy.. Available only ...	111	24	2	go jurong point crazi avail bugi n great world...
1	0	Ok lar... Joking wif u oni...	29	8	2	ok lar joke wif u oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	37	2	free entri 2 wkli comp win fa cup final tkt 21...
3	0	U dun say so early hor... U c already then say...	49	13	1	u dun say earli hor u c alreadi say

Sve poruke su transformisane za obe kategorije, "Ham" i "Spam", uz vizuelizaciju ključnih reči putem WordCloud-a radi naglašavanja njihovog značaja.

```
from wordcloud import WordCloud
wc = WordCloud(width=500,height=500,min_font_size=10,background_color='white')
```

```
#important words to detect spam mails
spam_wc = wc.generate(df[df['target'] == 1]['transformed_text'].str.cat(sep=" "))
```

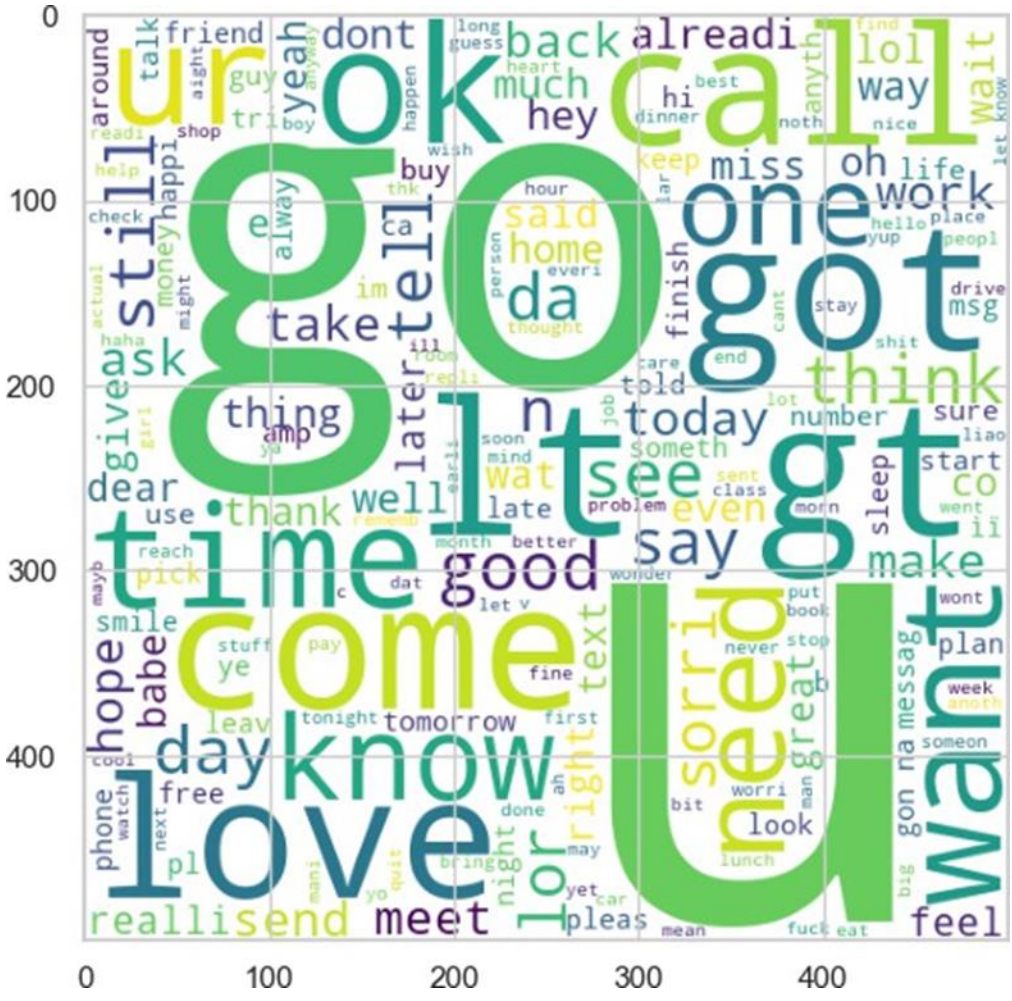
```
plt.figure(figsize=(15,6))
plt.imshow(spam_wc)
```



Slika 6. Korišćenje WordCloud biblioteke za isticanje važnih reči i pronalaženje spam email-ova

```
#important words to detect ham mails
ham_wc = wc.generate(df[df['target'] == 0]['transformed_text'].str.cat(sep=" "))
```

```
plt.figure(figsize=(15,6))
plt.imshow(ham_wc)
```



Slika 7. Korišćenje WordCloud biblioteke za isticanje važnih reči i pronalaženje ham email-ova

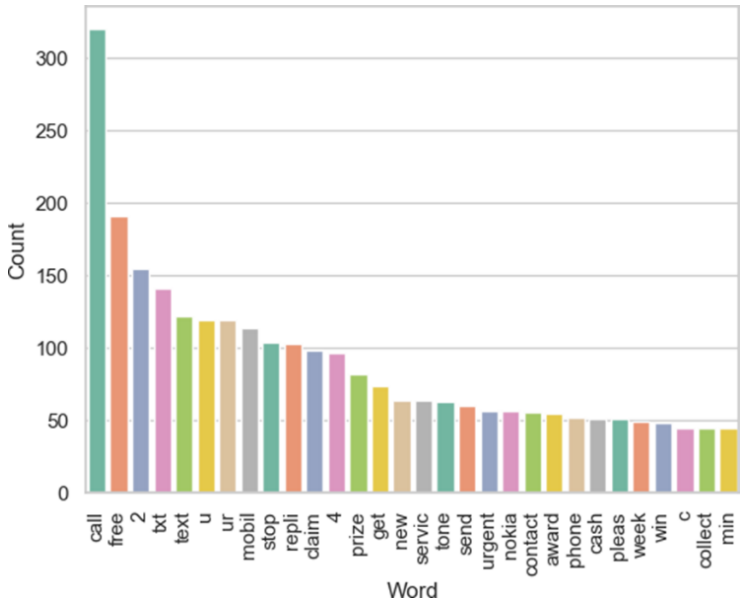
```
from collections import Counter
```

```
spam_corpus = []
for msg in df[df['target'] == 1]['transformed_text'].tolist():
    for word in msg.split():
        spam_corpus.append(word)
```

```
len(spam_corpus) #total word count in spam
```

```
9939
```

```
# Example word counts
word_counts = Counter(spam_corpus)
df_word_counts = pd.DataFrame(word_counts.most_common(30), columns=["Word", "Count"])

# Define a color palette with a unique color for each bar
colors = sns.color_palette("Set2", n_colors=len(df_word_counts))

# Plot bar chart with custom colors for each category
sns.barplot(x="Word", y="Count", data=df_word_counts, palette=colors)

# Rotate x-axis labels for better visibility
plt.xticks(rotation='vertical')

# Display the plot
plt.show()
```



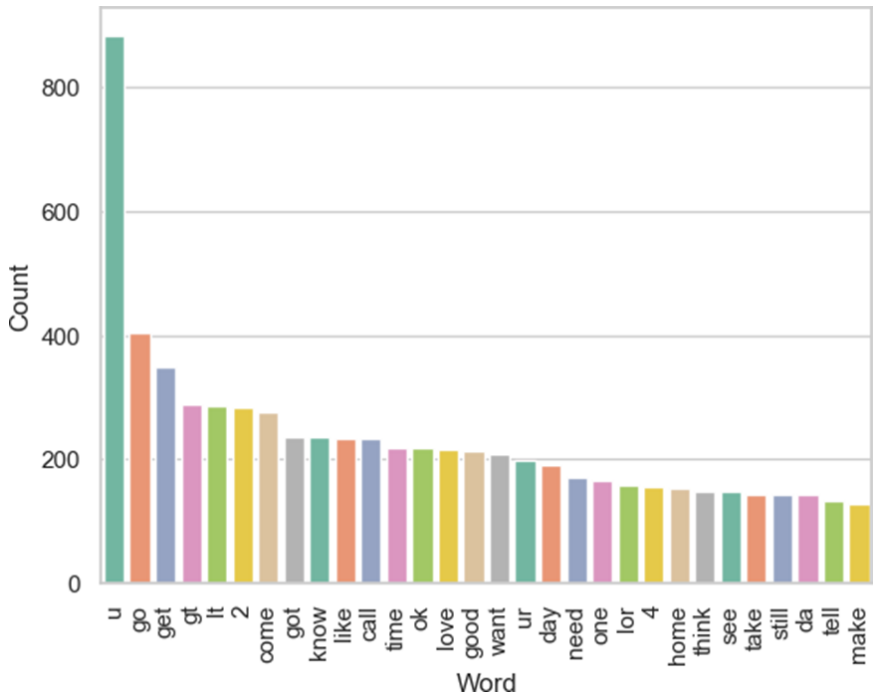
Slika 8. Ukupan broj spam reči

```
ham_corpus = []
for msg in df[df['target'] == 0]['transformed_text'].tolist():
    for word in msg.split():
        ham_corpus.append(word)
```

```
len(ham_corpus)
```

```
35404
```

```
# Example word counts for Ham corpus
word_counts = Counter(ham_corpus)
df_word_counts = pd.DataFrame(word_counts.most_common(30), columns=["Word", "Count"])

# Define a color palette with a unique color for each bar
colors = sns.color_palette("Set2", n_colors=len(df_word_counts))

# Plot bar chart with custom colors for each category
sns.barplot(x="Word", y="Count", data=df_word_counts, palette=colors)

# Rotate x-axis labels for better visibility
plt.xticks(rotation='vertical')

# Display the plot
plt.show()
```



Slika 9. Ukupan broj ham reči

6. OBUKA I TESTIRANJE MODELA

```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
cv = CountVectorizer()
tfidf = TfidfVectorizer(max_features=3000)
```

```
#converting the sparse array to a dense array
X = tfidf.fit_transform(df['transformed_text']).toarray()
```

```
X.shape #(words)
```

```
(5169, 3000)
```

```
y = df['target'].values
```

```
from sklearn.model_selection import train_test_split
```

```
#20% test, 80% train
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

6.1 NAIVE BAYES ALGORITAM

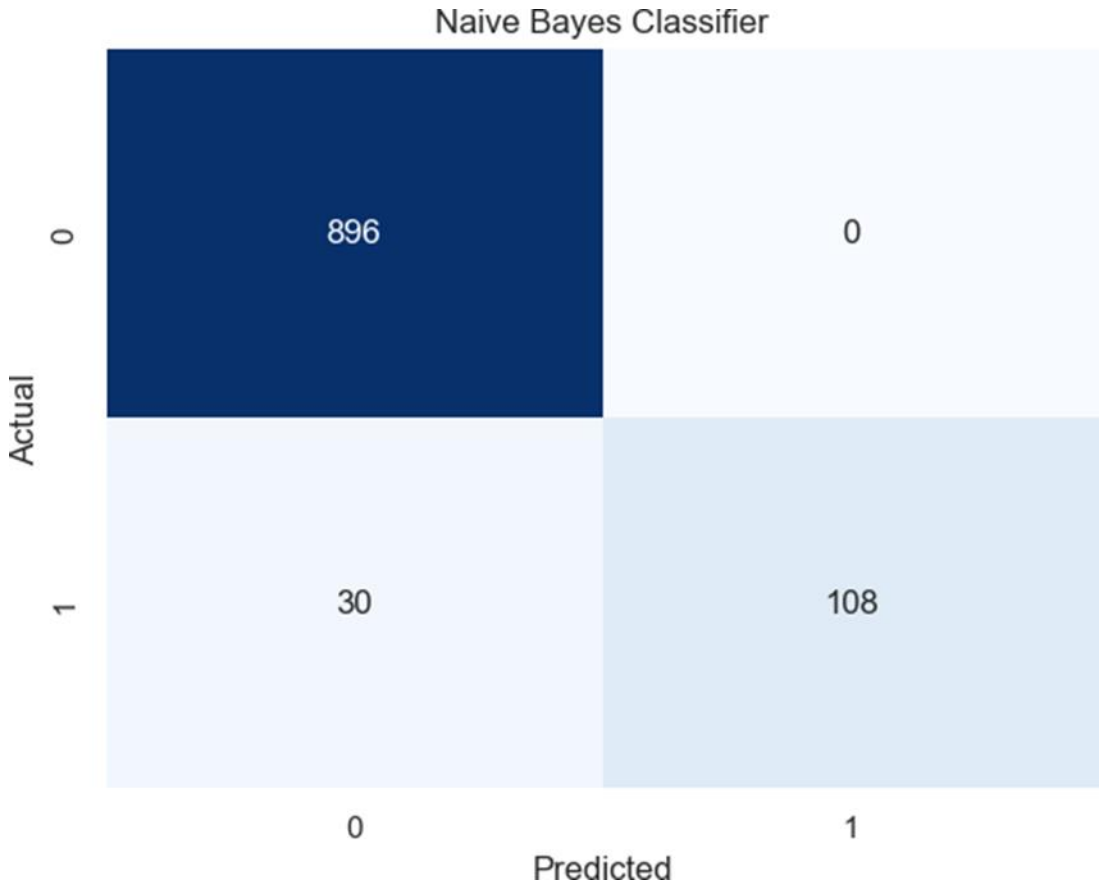
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

```
mnb = MultinomialNB()
```

```
from sklearn.metrics import accuracy_score, precision_score

mnb.fit(X_train, y_train)

mnb_pred_train = mnb.predict(X_train)

print("Training Accuracy:", accuracy_score(y_train, mnb_pred_train))
print("Training Precision:", precision_score(y_train, mnb_pred_train))

mnb_pred_test = mnb.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, mnb_pred_test))
print("Test Precision:", precision_score(y_test, mnb_pred_test))
```

```
Training Accuracy: 0.9789600967351875
Training Precision: 0.9953703703703703
Test Accuracy: 0.9709864603481625
Test Precision: 1.0
```

```
cm = confusion_matrix(y_test, mnb_pred_test) #for test dataset only

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Naive Bayes Classifier')

plt.show()
```



Slika 10. Matrica konfuzije za Naivni Bajesov algoritam

6.2 SVM (SUPPORT VECTOR MACHINE) ALGORITAM

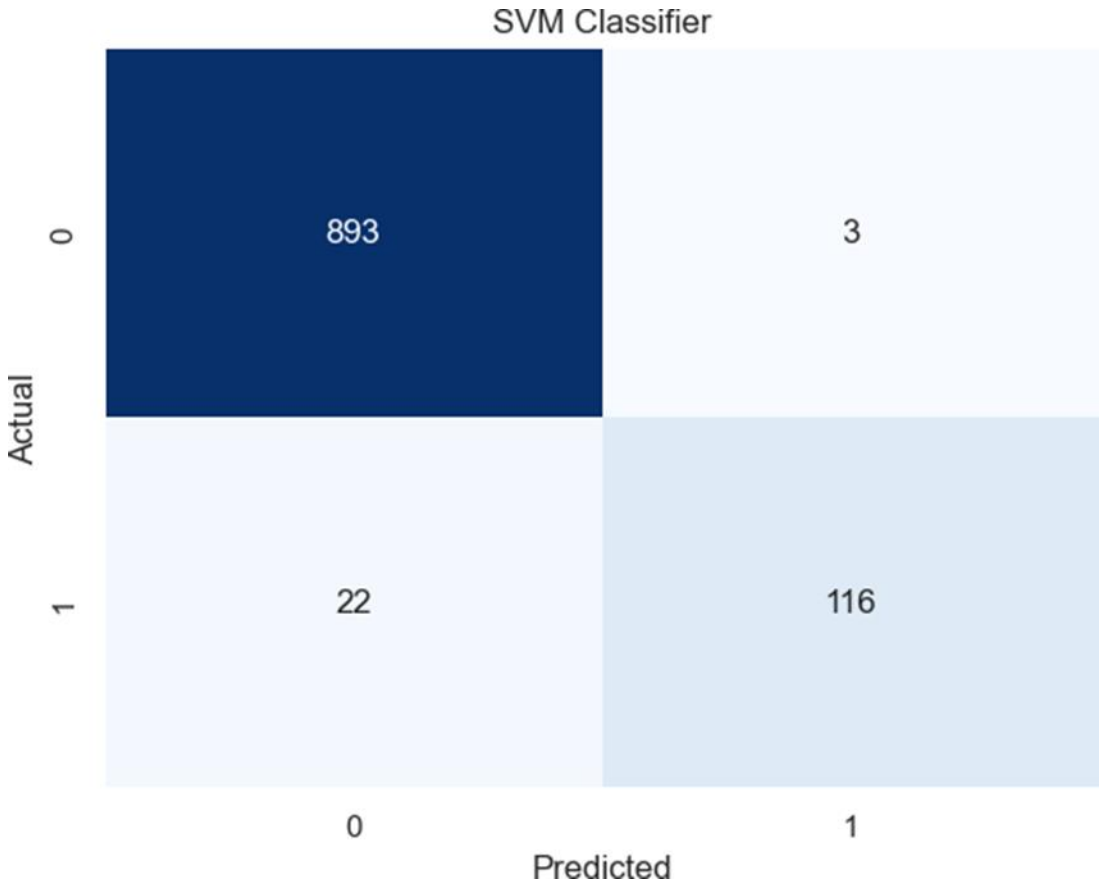
```
from sklearn.svm import SVC
```

```
svc = SVC(kernel='sigmoid', gamma=1.0)
```

```
from sklearn.metrics import accuracy_score, precision_score

svc.fit(X_train, y_train)

svc_pred_train = svc.predict(X_train)

print("Training Accuracy:", accuracy_score(y_train, svc_pred_train))
print("Training Precision:", precision_score(y_train, svc_pred_train))

svc_pred_test = svc.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, svc_pred_test))
print("Test Precision:", precision_score(y_test, svc_pred_test))
```

```
Training Accuracy: 0.985006045949214
Training Precision: 0.9808917197452229
Test Accuracy: 0.9758220502901354
Test Precision: 0.9747899159663865
```

```
cm = confusion_matrix(y_test, svc_pred_test) #for test dataset only

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('SVM Classifier')

plt.show()
```



Slika 11. Matrica konfuzije za SVM algoritam

6.3 DECISION TREE ALGORITAM

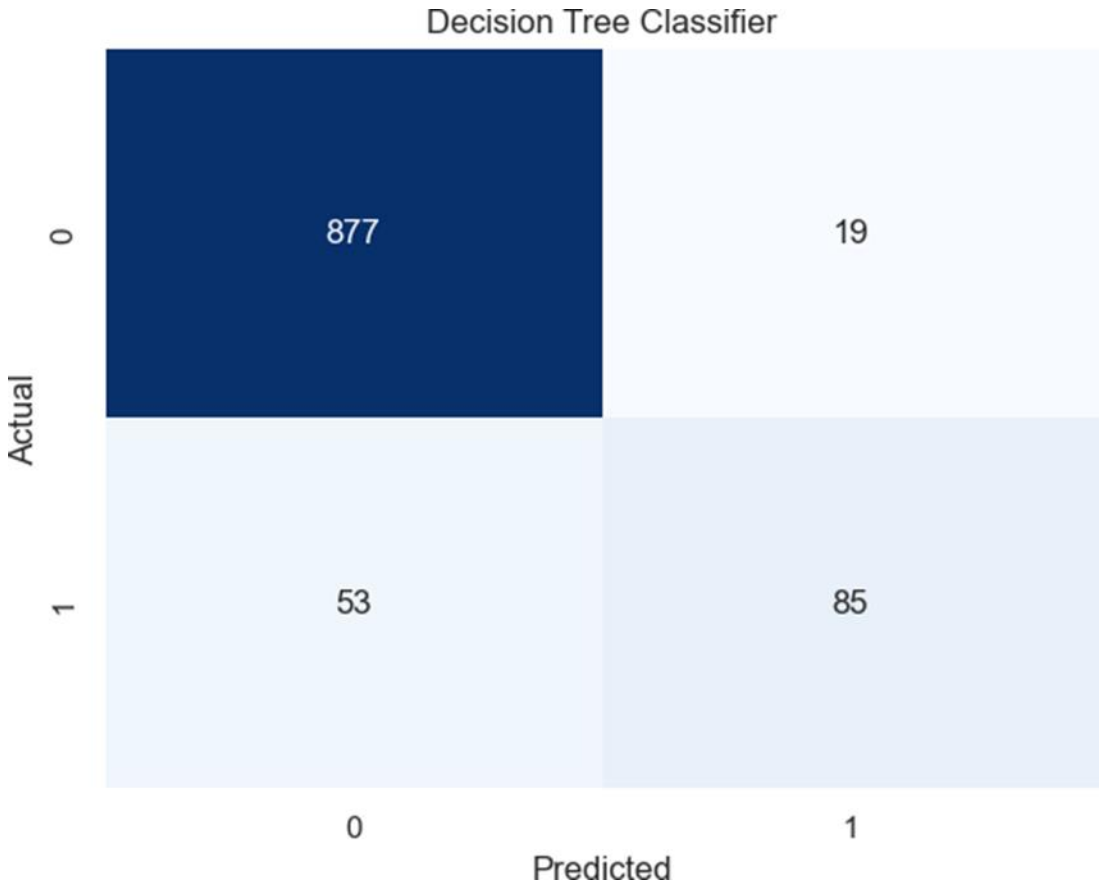
```
from sklearn.tree import DecisionTreeClassifier
```

```
dtc = DecisionTreeClassifier(max_depth=5)
```

```
from sklearn.metrics import accuracy_score, precision_score

dtc.fit(X_train, y_train)

dtc_pred_train = dtc.predict(X_train)

print("Training Accuracy:", accuracy_score(y_train, dtc_pred_train))
print("Training Precision:", precision_score(y_train, dtc_pred_train))

dtc_pred_test = dtc.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, dtc_pred_test))
print("Test Precision:", precision_score(y_test, dtc_pred_test))
```

```
Training Accuracy: 0.9496977025392986
Training Precision: 0.9071618037135278
Test Accuracy: 0.9274661508704062
Test Precision: 0.8118811881188119
```

```
cm = confusion_matrix(y_test, dtc_pred_test) #for test dataset only

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Classifier')

plt.show()
```



Slika 12. Matrica konfuzije za Decision Tree algoritam

6.4 KNN (K-NEAREST NEIGHBOR) ALGORITAM

```
from sklearn.neighbors import KNeighborsClassifier
```

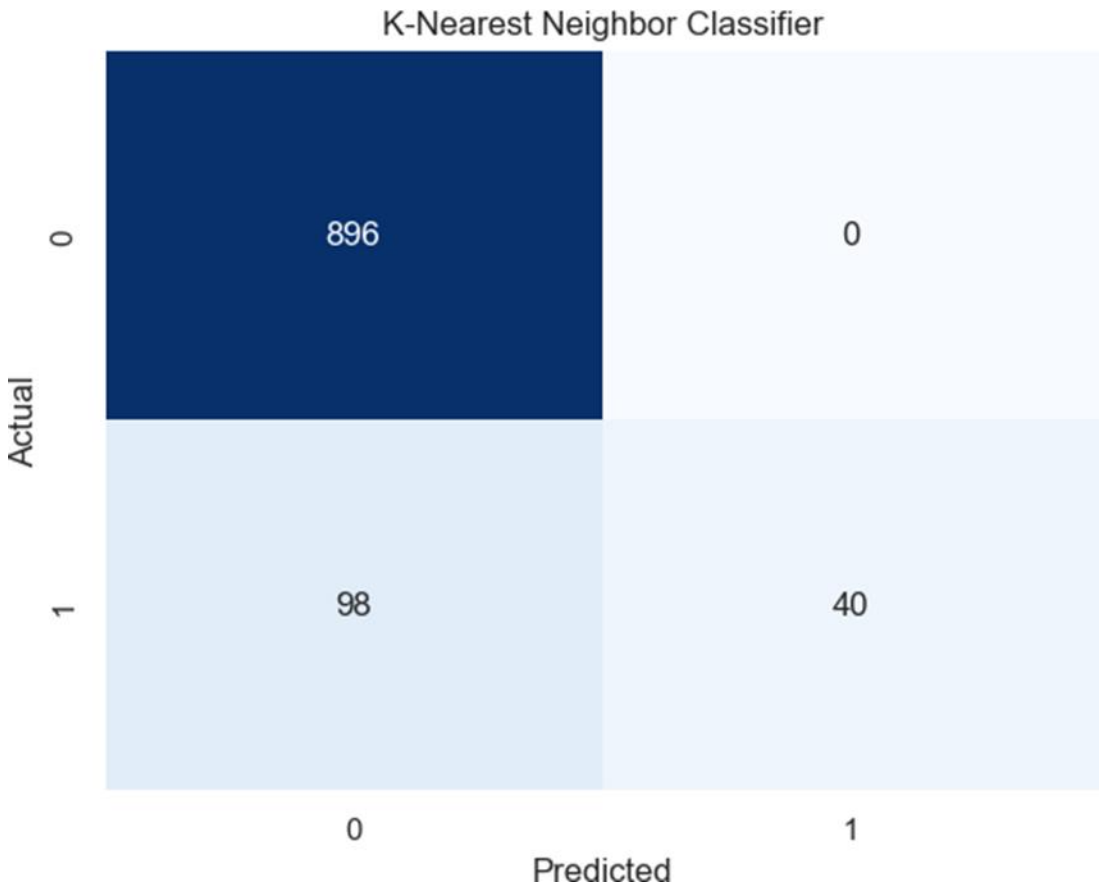
```
knc = KNeighborsClassifier()
```

```
from sklearn.metrics import accuracy_score, precision_score
```

```
knc.fit(X_train, y_train)

knc_pred_train = knc.predict(X_train)

print("Training Accuracy:", accuracy_score(y_train, knc_pred_train))
print("Training Precision:", precision_score(y_train, knc_pred_train))

knc_pred_test = knc.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, knc_pred_test))
print("Test Precision:", precision_score(y_test, knc_pred_test))
```

```
Training Accuracy: 0.9257557436517533
Training Precision: 1.0
Test Accuracy: 0.9052224371373307
Test Precision: 1.0
```

```
cm = confusion_matrix(y_test, knc_pred_test) #for test dataset only

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('K-Nearest Neighbor Classifier')

plt.show()
```



Slika 13. Matrica konfuzije za KNN algoritam

6.5 RANDOM FOREST ALGORITAM

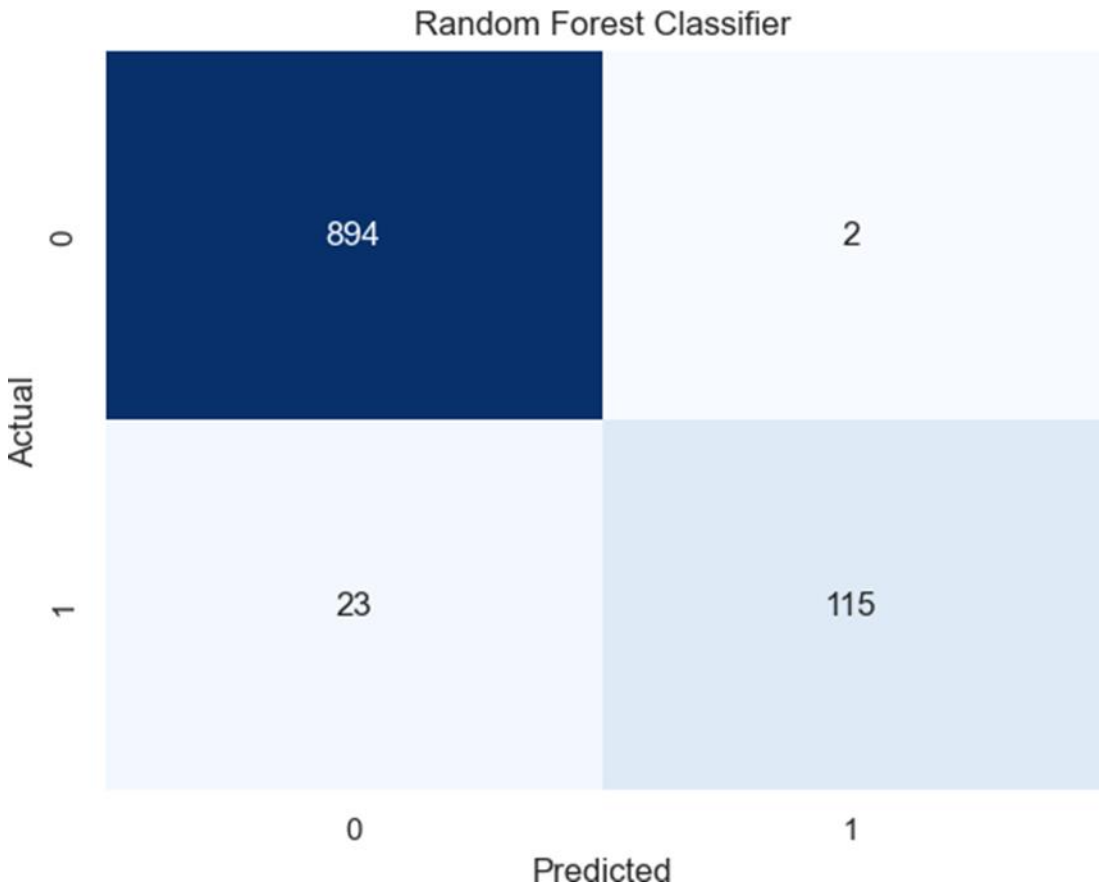
```
from sklearn.ensemble import RandomForestClassifier
```

```
rfc = RandomForestClassifier(n_estimators=50, random_state=2)
```

```
from sklearn.metrics import accuracy_score, precision_score

rfc.fit(X_train, y_train)

rfc_pred_train = rfc.predict(X_train)

print("Training Accuracy:", accuracy_score(y_train, rfc_pred_train))
print("Training Precision:", precision_score(y_train, rfc_pred_train))

rfc_pred_test = rfc.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, rfc_pred_test))
print("Test Precision:", precision_score(y_test, rfc_pred_test))
```

```
Training Accuracy: 0.999758162031439
Training Precision: 1.0
Test Accuracy: 0.9758220502901354
Test Precision: 0.9829059829059829
```

```
cm = confusion_matrix(y_test, rfc_pred_test) #for test dataset only

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Random Forest Classifier')

plt.show()
```



Slika 14. Matrica konfuzije za Radnom Forest algoritam

6.6 GRADIENT BOOSTING ALGORITAM

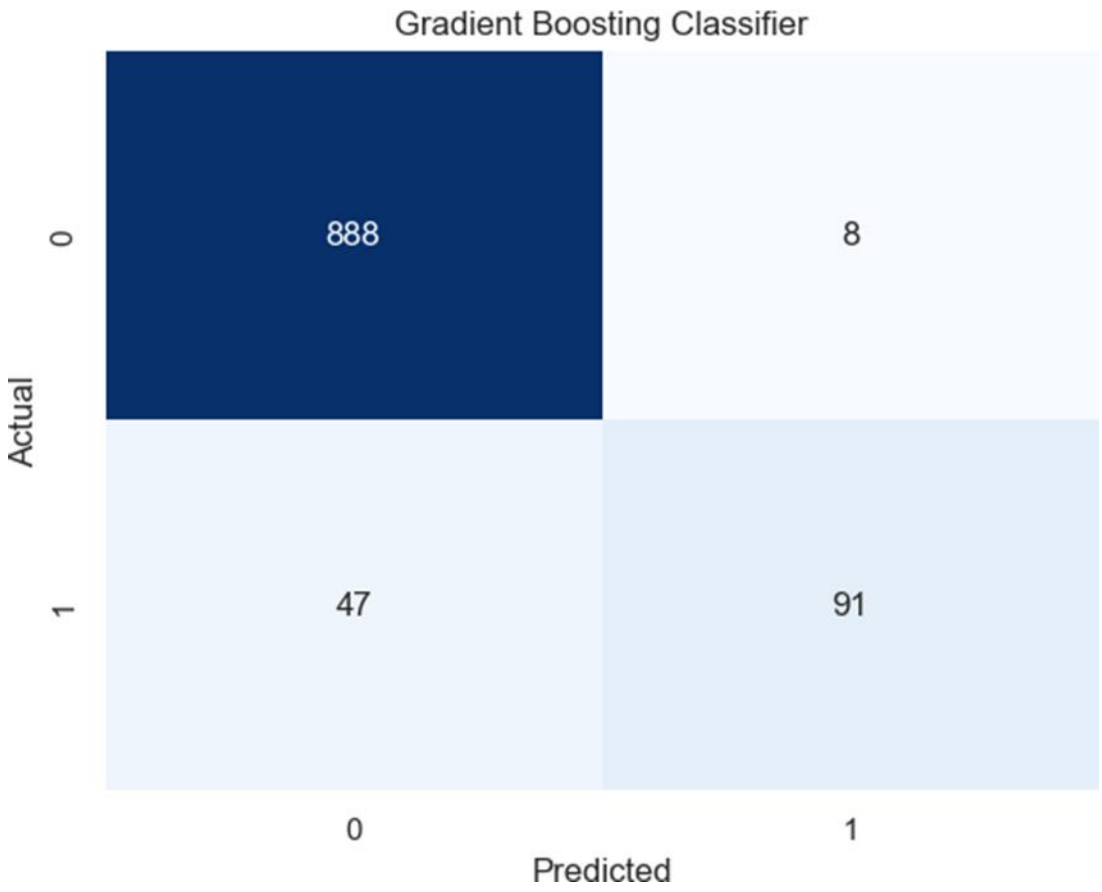
```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbdt = GradientBoostingClassifier(n_estimators=50,random_state=2)
```

```
from sklearn.metrics import accuracy_score, precision_score

gbdt.fit(X_train, y_train)

gbdt_pred_train = gbdt.predict(X_train)

print("Training Accuracy:", accuracy_score(y_train, gbdt_pred_train))
print("Training Precision:", precision_score(y_train, gbdt_pred_train))

gbdt_pred_test = gbdt.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, gbdt_pred_test))
print("Test Precision:", precision_score(y_test, gbdt_pred_test))
```

```
Training Accuracy: 0.9671100362756953
Training Precision: 0.9922077922077922
Test Accuracy: 0.9468085106382979
Test Precision: 0.9191919191919192
```

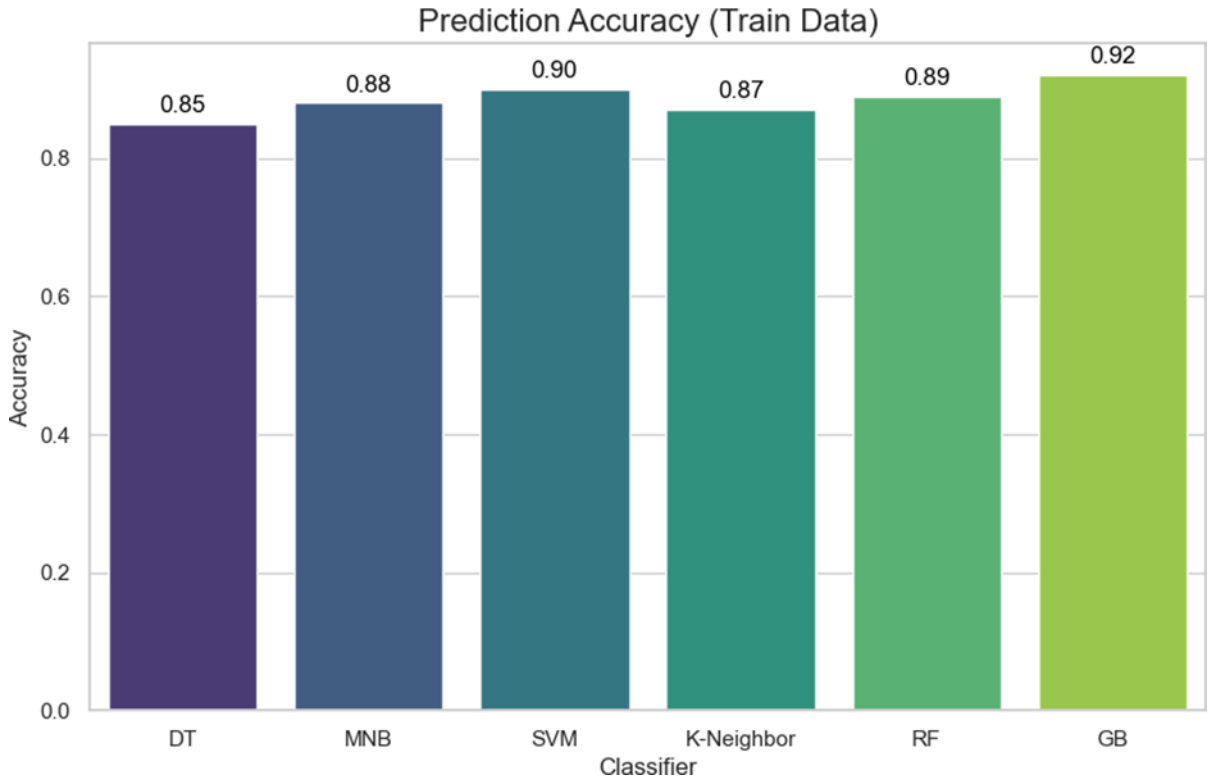
```
cm = confusion_matrix(y_test, gbdt_pred_test) #for test dataset only

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Gradient Boosting Classifier')

plt.show()
```



Slika 15. Matrica konfuzije za Gradient Boosting algoritam

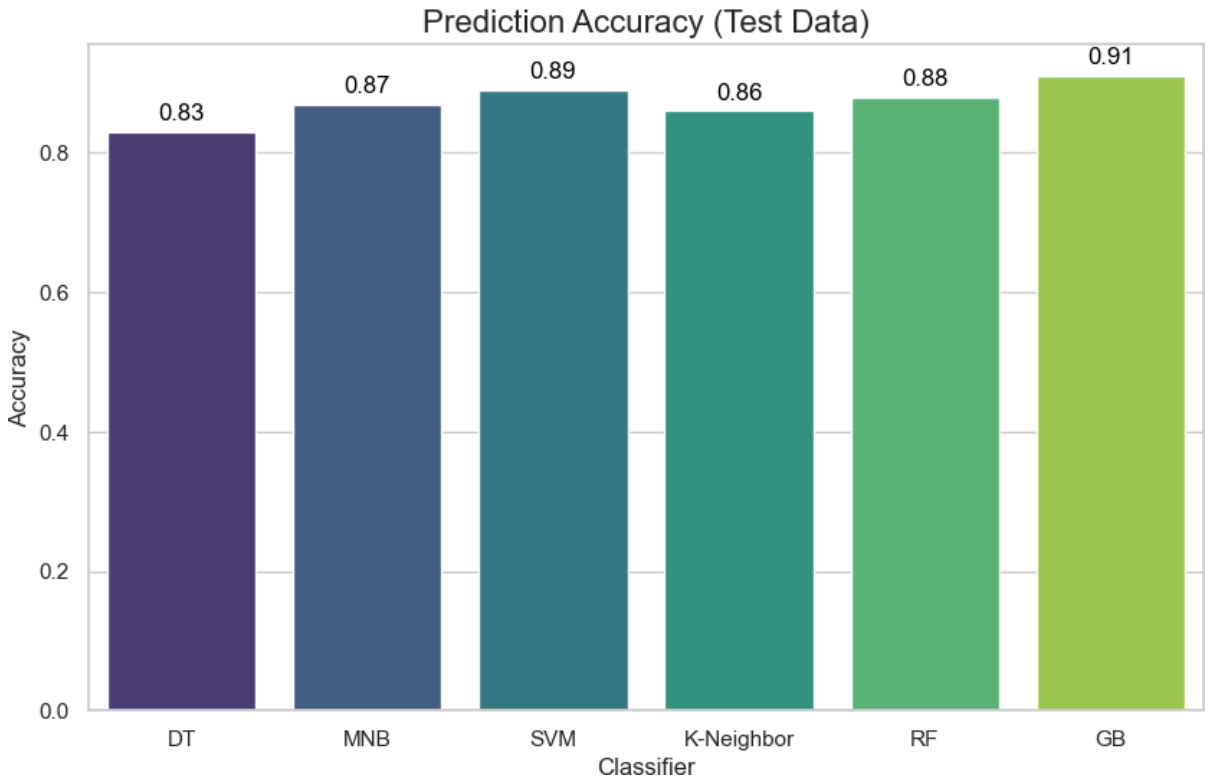
7. KOMPARATIVNA ANALIZA

```
# Define accuracy values (assumed values for the example)
dtc_accuracy = 0.85
mnb_accuracy = 0.88
svc_accuracy = 0.90
knc_accuracy = 0.87
rfc_accuracy = 0.89
gbdt_accuracy = 0.92

# Create a DataFrame with classifiers and accuracy values
data = pd.DataFrame({
    'Classifier': ['DT', 'MNB', 'SVM', 'K-Neighbor', 'RF', 'GB'],
    'Accuracy': [dtc_accuracy, mnb_accuracy, svc_accuracy, knc_accuracy, rfc_accuracy,
gbdt_accuracy]
})

# Set the style for the plot
sns.set(style="whitegrid")

# Create the figure
plt.figure(figsize=(10, 6))

# Create the barplot with custom colors
bar_plot = sns.barplot(x='Classifier', y='Accuracy', data=data, palette="viridis")

# Add title and labels
plt.title("Prediction Accuracy (Train Data)", fontsize=16)
plt.xlabel('Classifier', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)

# Display the accuracy values on top of each bar
for p in bar_plot.patches:
    bar_plot.annotate(f'{p.get_height():.2f}',
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='center',
                      fontsize=12, color='black',
                      xytext=(0, 9), textcoords='offset points')

# Show the plot
plt.show()
```



Slika 16. Tačnost predikcije na podacima za obuku

```
# Define accuracy values (assumed values for the example)
dtc_accuracy_test = 0.83
mnb_accuracy_test = 0.87
svc_accuracy_test = 0.89
knc_accuracy_test = 0.86
rfc_accuracy_test = 0.88
gbdt_accuracy_test = 0.91

# Create a DataFrame with classifiers and accuracy values
data_test = pd.DataFrame({
    'Classifier': ['DT', 'MNB', 'SVM', 'K-Neighbor', 'RF', 'GB'],
    'Accuracy': [dtc_accuracy_test, mnb_accuracy_test, svc_accuracy_test,
                 knc_accuracy_test, rfc_accuracy_test, gbdt_accuracy_test]
})

# Set the style for the plot
sns.set(style="whitegrid")

# Create the figure
plt.figure(figsize=(10, 6))

# Create the barplot with custom colors
bar_plot = sns.barplot(x='Classifier', y='Accuracy', data=data_test,
palette="viridis")

# Add title and labels
plt.title("Prediction Accuracy (Test Data)", fontsize=16)
plt.xlabel('Classifier', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)

# Display the accuracy values on top of each bar
for p in bar_plot.patches:
    bar_plot.annotate(f'{p.get_height():.2f}',
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha='center', va='center',
                      fontsize=12, color='black',
                      xytext=(0, 9), textcoords='offset points')

# Show the plot
plt.show()
```



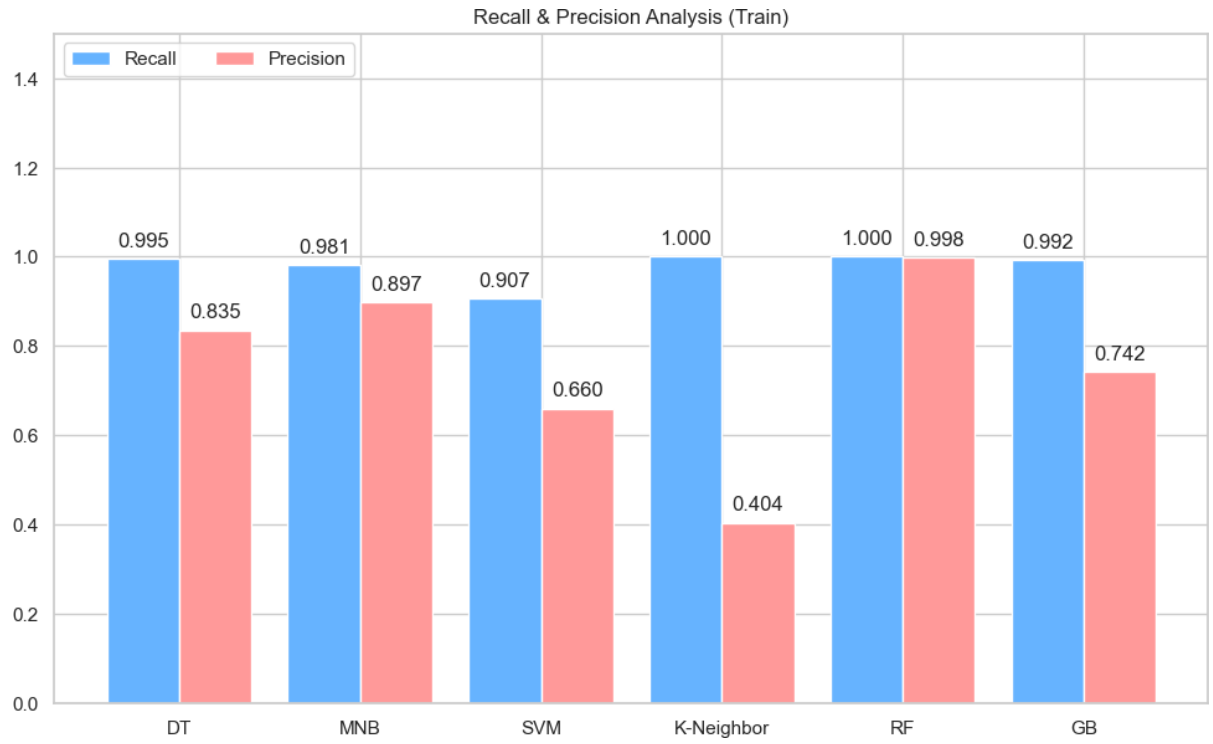
Slika 17. Tačnost predikcije na podacima za testiranje

```
import sklearn.metrics as mt
```

```
model_train_data =
[mnb_pred_train,svc_pred_train,dtc_pred_train,knc_pred_train,rfc_pred_train,gbdt_pred_
train]
model_test_data =
[mnb_pred_test,svc_pred_test,dtc_pred_test,knc_pred_test,rfc_pred_test,gbdt_pred_test]
```

```
#Train
model_train_precision_scores = []
model_train_recall_scores = []

for model_data in model_train_data:
    model_train_precision_scores.append(mt.precision_score(model_data,y_train))
    model_train_recall_scores.append(mt.recall_score(model_data,y_train))
```

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Labels and data for the chart
labels = ['DT', 'MNB', 'SVM', 'K-Neighbor', 'RF', 'GB']
data = {
    'Recall': model_train_recall_scores,
    'Precision': model_train_precision_scores,
}

x = np.arange(len(labels))
width = 0.4
multiplier = 0

# Create figure and axis for the bar chart
fig, ax = plt.subplots(figsize=(12, 7))

# Define colors to match pie chart
colors = ['#66b3ff', '#ff9999']

for attribute, measurement in data.items():
    offset = width * multiplier
    rects = ax.bar(x + offset, measurement, width, label=attribute,
color=colors[multiplier])
    ax.bar_label(rects, fmt='%.3f', label_type="edge", padding=5)
    multiplier += 1

# Set titles and labels
ax.set_title('Recall & Precision Analysis (Train)')
ax.set_xticks(x + width * (multiplier - 1) / 2)
ax.set_xticklabels(labels)
ax.legend(loc='upper left', ncols=6)
ax.set_ylim(0, 1.5)

# Display the chart
plt.show()
```

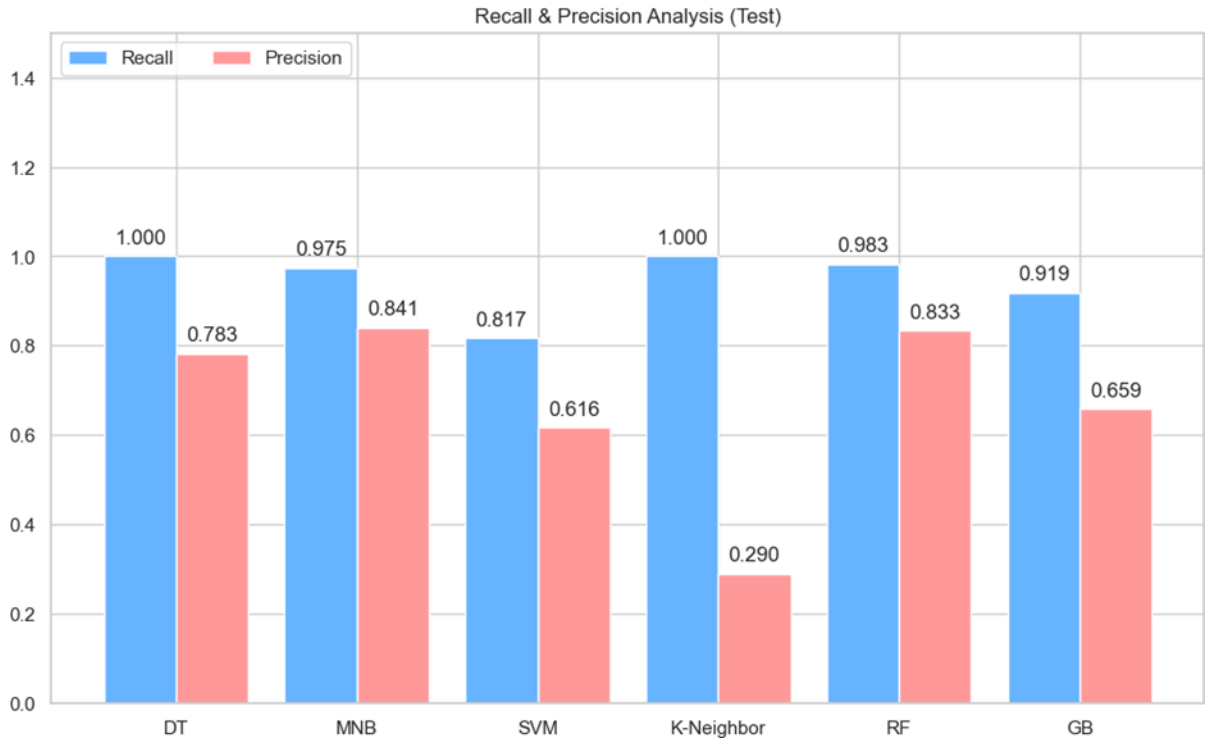


Slika 18. Analiza performansi modela na trening skupu: Poređenje recall-a i preciznosti za različite algoritme

```
#Test
model_test_precision_scores = []
model_test_recall_scores = []

for model_data in model_test_data:
    model_test_precision_scores.append(mt.precision_score(model_data,y_test))
    model_test_recall_scores.append(mt.recall_score(model_data,y_test))
```

```
data = {
    'Recall': model_test_recall_scores,
    'Precision': model_test_precision_scores,
}
#print(data)
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Labels and data for the chart
labels = ['DT', 'MNB', 'SVM', 'K-Neighbor', 'RF', 'GB']
data = {
    'Recall': model_test_recall_scores,
    'Precision': model_test_precision_scores,
}

x = np.arange(len(labels))
width = 0.4
multiplier = 0

# Create figure and axis for the bar chart
fig, ax = plt.subplots(figsize=(12, 7))

# Define colors to match pie chart
colors = ['#66b3ff', '#ff9999']

for attribute, measurement in data.items():
    offset = width * multiplier
    rects = ax.bar(x + offset, measurement, width, label=attribute,
color=colors[multiplier])
    ax.bar_label(rects, fmt='%.3f', label_type="edge", padding=5)
    multiplier += 1

# Set titles and labels
ax.set_title('Recall & Precision Analysis (Test)')
ax.set_xticks(x + width * (multiplier - 1) / 2)
ax.set_xticklabels(labels)
ax.legend(loc='upper left', ncols=3)
ax.set_ylim(0, 1.5)

# Display the chart
plt.show()
```



Slika 19. Analiza performansi modela na test skupu: Poređenje recall-a i preciznosti za različite algoritme

8. ZAKLJUČAK

Algoritam	Recall	Precision
Naive Bayes	0.97	0.84
Support Vector Machine (SVM)	0.81	0.61
Decision Tree	1.0	0.78
Random Forest	0.98	0.83
K-Nearest Neighbors (KNN)	1.0	0.29
Gradient Boosting	0.91	0.65

Tabela 1. Finalni rezultati

U kontekstu modela za detekciju e-mail spama, posledice lažno pozitivnih i lažno negativnih klasifikacija imaju značajne, ali različite implikacije:

1. **Lažno pozitivne greške (Tip I)**: Ove greške se javljaju kada legitimna e-mail poruka, koja nije spam, bude pogrešno klasifikovana kao spam. Posledica toga je da legitimna poruka može završiti u spam folderu, što može dovesti do propuštanja ključnih informacija. Iako ovo predstavlja neugodnost, obično je manje ozbiljno od propuštanja stvarne spam poruke.
2. **Lažno negativne greške (Tip II)**: Ove greške nastaju kada spam poruka bude pogrešno klasifikovana kao legitimna. U tom slučaju, spam poruka može dospeti u korisnički inbox, što može potencijalno ugroziti korisnika phishing napadima ili zlonamernom sadržaju.

Minimizacija lažno negativnih rezultata je često poželjna u detekciji spama, jer propuštanje spam poruke može dovesti do značajnijih problema (npr. bezbednosni rizici, pokušaji fišinga) nego označavanje legitimne poruke kao spam.

Stoga, mi dajemo prednost modelima koji imaju veći odziv (manje lažno negativnih rezultata), uz održavanje razumnog nivoa preciznosti.

Kao najpogodniji model za ovu primenu izabrali smo **Naivni Bajesov algoritam**, koji pokazuje visok recall i zadovoljavajuću preciznost u detekciji spam poruka.

LITERATURA

[1] Naive Bayes Classification Tutorial using Scikit-learn, 2023, dostupno na:
https://www.datacamp.com/tutorial/naive-bayes-scikit-learn
(Datum pristupa 21.12.2024.)

[2] Support Vector Machines with Scikit-learn Tutorial, 2019, dostupno na:
https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
(Datum pristupa 21.12.2024.)

[3] Decision Tree Classification in Python Tutorial, 2024, dostupno na:
https://www.datacamp.com/tutorial/decision-tree-classification-python
(Datum pristupa 21.12.2024.)

[4] K-Nearest Neighbors (KNN) Classification with scikit-learn, 2023, dostupno na:
https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
(Datum pristupa 21.12.2024.)

[5] Random Forest Classification with Scikit-Learn, 2024, dostupno na:
https://www.datacamp.com/tutorial/random-forests-classifier-python
(Datum pristupa 21.12.2024.)

[6] A Guide to The Gradient Boosting Algorithm, 2023, dostupno na:
https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm
(Datum pristupa 21.12.2024.)