

Univerzitet u Sarajevu
Elektrotehnički fakultet
Odsjek za računarstvo i informatiku
Drugi ciklus studija

IMPLEMENTACIJA CHATBOTA ZAKORISNIČKU PODRŠKU

Dubinska analiza podataka

Članovi:

Imamović Nasiha (1988/18080)

Novalić Neira (1951/18112)

Rovčanin Nejra (1960/18480)

Rudalića Ema (1974/18555)

Sarajevo, juni 2023.

Sadržaj

Uvod	3
Motivacija	6
Pregled literature.....	7
Kreiranje seta podataka	8
Preprocesiranje podataka	12
Inženjering karakteristika	14
Implementacija algoritama	22
Random forest (Nasiha Imamović)	22
BERT i transformeri (Neira Novalić i Nejra Rovčanin)	27
Rekurentna neuronska mreža sa LSTM slojem (Rovčanin Nejra i Novalić Neira) ..	34
Konvolucijska neuronska mreža (CNN) (Rudalića Ema)	44
Zaključak.....	50

Uvod

U današnjem digitalnom dobu, komunikacija s korisnicima i klijentima postaje sve važnija za uspjeh kompanija. Tradicionalne metode komunikacije poput telefonskih poziva i e-pošte polako ustupaju mjesto novim, inovativnijim načinima interakcije. Jedna od najznačajnijih promjena u načinu na koji kompanije komuniciraju s korisnicima jest upotreba chatbota. Chatboti predstavljaju softverske programe koji su sposobni simulirati ljudsku komunikaciju. Oni su razvijeni kako bi korisnicima pružili brže i efikasne odgovore na njihova pitanja, podršku u rješavanju problema ili čak za obavljanje određenih zadataka. Razlozi za korištenje chatbota su mnogobrojni i ima ih sve više kompanija koje prepoznaju njihovu vrijednost. Prvo, chatboti pružaju korisnicima 24/7 dostupnost. Za razliku od tradicionalnih metoda komunikacije, chatboti mogu biti aktivni i odgovarati na upite korisnika bilo kad tokom dana ili noći. To omogućuje korisnicima da dobiju brže odgovore i podršku u bilo kojem trenutku, bez potrebe čekanja na radno vrijeme kompanije. Chatboti su izuzetno efikasni u obradi velikog broja upita istovremeno. Oni mogu pružiti istovremenu podršku velikom broju korisnika, čime smanjuju opterećenje za ljudske agente. Ova skalabilnost chatbota omogućuje poboljšava produktivnost i učinkovitost komunikacije s korisnicima. Pored ovih stavki oni mogu biti ekonomičan način komunikacije s korisnicima. Umjesto zapošljavanja velikog broja ljudskih agenata za pružanje podrške, kompanije mogu koristiti chatbote kako bi smanjile troškove. Uz sve prednosti koje chatboti pružaju, važno je istaknuti da oni ne zamjenjuju ljudsku interakciju, već je nadopunjuju. U slučajevima kada chatbot ne može riješiti složenije ili specifične upite, korisnici mogu biti preusmjereni na ljudske agente koji će pružiti daljnju podršku. Ova kombinacija ljudskih i virtualnih agenata omogućuje najbolje od oba svijeta - brzinu i efikasnost chatbota te ljudsku empatiju i razumijevanje.

Iz perspektive mašinskog učenja, chatboti se mogu podijeliti na nekoliko vrsta:

- Chatboti zasnovni na pravilima - Ovi chatboti se temelje na unaprijed definisanim pravilima i uzorcima. Razvijaju se na temelju skupa pravila koje programer određuje i odgovaraju na upite korisnika prema tim pravilima. Oni su ograničeni na skup predviđenih scenarija i ne mogu pružiti odgovore na složenije ili nepredvidive upite.
- Chatboti s korisničkim priručnikom - Ova vrsta chatbota koristi priručnike, dokumentaciju ili baze znanja kako bi pružila odgovore korisnicima. Algoritmi mašinskog učenja analiziraju upite korisnika i pronalaze relevantne informacije iz dostupnih izvora kako bi generisali odgovore. Ovi chatboti su korisni za pružanje specifičnih informacija i tehničke podrške.

- Chatboti s nadziranim učenjem - Ovi chatboti koriste nadzirano učenje kako bi razumjeli i generisali odgovore na temelju skupa podataka koji sadrže primjere pitanja i odgovora. Programer obučava chatbota koristeći označene primjere podataka, a zatim se algoritmi mašinskog učenja koriste za modeliranje veza između pitanja i odgovora. Chatboti s nadziranim učenjem mogu pružiti bolje rezultate u obradi složenih upita i mogu se prilagoditi promjenjivim scenarijima.
- Generativni chatboti - Ova vrsta chatbota koristi tehnike generativnog modeliranja kako bi generirala odgovore. Koriste se duboki modeli generativnih neuronskih mreža (npr. rekurentne neuronske mreže) kako bi naučili generisati odgovore na temelju ulaznih podataka. Generativni chatboti mogu stvarati ljudima prirodne odgovore ali mogu biti izazovni za upravljanje kontekstom i osiguravanje tačnosti odgovora.
- Hibridni chatboti - Ovi chatboti kombinuju različite pristupe i tehnike kako bi pružili najbolje moguće iskustvo korisnicima. Oni mogu koristiti kombinaciju pravilno zasnovanih pristupa, chatbota s korisničkim priručnikom i mašinskog učenja kako bi pružili raznolik i fleksibilnu podršku korisnicima.

Zanimljive činjenice o chatbotima:

- 64% poduzeća vjeruje kako će chatbot njihovim klijentima pružiti bolje iskustvo korisničke usluge (Statista). Iskustvo klijenata koji koriste Valori pokazalo je kako je ta pretpostavka istinita.
- Chatbot se najčešće koristi za prodaju (41%), korisničku podršku (37%) i marketing (17%) (Intercom, 2019). S obzirom na brojne mogućnosti, može se prilagoditi tačno prema potrebama poslovanja.
- Djelatnosti nekretnina (28%), putovanja (16%), obrazovanja (14%), zdravstva (10%) i finansija (5%) su top 5 tržišta koja imaju najveći postotak zarade od chatbota. (Chatbots Life)
- 1,4 milijarde ljudi koristi chatbote (Chatbot.com). To je trenutno 18% svjetske populacije
- Korištenjem chatbota, pozivi, razgovori i e-mail upiti u organizaciji smanjuju se za 70% (Gartner). Jedan od najvećih benefita chatbota je mogućnost brzog pružanja tražene informacije ili upućivanje korisnika do iste. Jednom kada korisnik dobije odgovor na pitanje, nema potrebe ponovno zvati ili slati mail S obzirom na puno

benefita koje kompanije mogu ostvariti korištenjem chatbota ali i različitih implementacija i vrsta istih u nastavku ovog rada bit će obrađene različite metode mašinskog učenja korištene za kreiranje chatbota koji će simulirati korisničku podršku.

Motivacija

Kao što je u uvodnom poglavlju navedenu, izučavanje i kreiranja chatbota je izuzetno zanimljiva oblast. Pisanje rada na temu chatbota nam omogućava proširivanje našeg znanja o ovoj tehnologiji i njenim mogućnostima te zahtijeva dublje razumjevanje njihove ulogu, primjene i uticaja na različite industrije. Također, podstaknuti smo na analizu relevantne literature i istraživanja. Kroz ovaj proces, možemo biti informisani o najnovijim trendovima, metodologijama i primjerima koji se odnose na chatbote a s obzirom da je ovo izuzetno aktuelna oblast, to nam omogućava da ostanemo u toku s najnovijim informacijama i da steknemo dublje razumijevanje polja analiziranjem različitih algoritma mašinskog učenja koji se koriste za implementaciju.

Pregled literature

Neki od relevantnih radova koji se bave konkretnim metodama mašinskog učenja koje se koriste za kreiranje chatbota su:

1. *Sequence to Sequence Learning with Neural Networks* - Ovaj rad predstavlja osnovni model za mašinsko prevođenje temeljen na rekurentnim neuronskim mrežama (RNN) i encoderdecoder arhitekturi. Taj model može se koristiti i za generiranje odgovora u chatbotima
2. *Attention is All You Need* - Ovaj rad predstavlja Transformer model, koji koristi mehanizam pažnje kako bi poboljšao performanse u generiranju sekvenci. Transformer se može primijeniti u chatbotima kako bi se poboljšala sposobnost razumijevanja konteksta i generisanja relevantnih odgovora.
3. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* - Ovaj rad predstavlja BERT (Bidirectional Encoder Representations from Transformers), model koji je treniran na velikom korpusu teksta i koji postiže izvanredne rezultate u zadacima razumijevanja jezika. BERT se može primijeniti u chatbotima kako bi se poboljšalo razumijevanje korisničkih upita i generisanje prirodnijih odgovora.
4. *Deep Reinforcement Learning for Dialogue Generation* - Ovaj rad istražuje primjenu dubokog ojačavanja učenja (deep reinforcement learning) za generisanje dijaloga. Koristi se model temeljen na dubokim neuronskim mrežama koji uči optimalnu strategiju generisanja odgovora na temelju povratne informacije iz okoline.
5. *Transfer Learning for Conversational Agents* - Ovaj rad istražuje primjenu prenosivog učenja (transfer learning) u kontekstu agenata. Autori analiziraju različite pristupe i tehnike prenosivog učenja za poboljšanje performansi chatbota u različitim domenama i scena.
6. *Smart College Chatbot using ML and Python* - Fokus rada je istraživanje načina kako se umjetna inteligencija koristi za poboljšanje iskustva korisnika, s posebnim osvrtom na razvoj chatbota kao kanala za distribuciju informacija. Koristi se WordNet za odabir najbliže podudarajućeg odgovora na temelju unosa korisnika, a zatim se odabire odgovor iz predefiniranog skupa izjava. Generisanje odgovora temelji se na znanju, dok WordNet preuzima odgovornost za logiku koja se koristi. Chatbot je također baziran na AIML jeziku, koji je vrsta Extensible Markup Language (XML). Korištenjem više logičkih adaptera odabire se jedan odgovor iz

skupa odgovora koje je chatbot konfiguriran da koristi. Obrada informacija uključuje ugrađivanje riječi, pri čemu se svaka riječ mapira na vektor.

Kreiranje seta podataka

Pretraživanjem dostupne literature na internetu, ustanovljeno je da nema dataset-ova koji su prilagođeni temi chatbota u ulozi korisničke podrške. Analizom je ustanovljeno da su podaci za korisničku podršku često osjetljivi te da se kompanije koje koriste ovaj resurs često ne osjećaju ugodno da podijele informacije vezane za njih. Iz tog razloga odlučile smo uz pomoć dostupnih materijala i alata kreirati početni dataset koji obuhvata neka osnovna pitanja koja se odnose na podršku korisnicima prilikom online kupovine. Datoteka koja predstavlja skup ovih podataka je kreirana u json formatu te sadrži uređene trojke ključ-pattern-odgovor. Ključ predstavlja namjeru korisnika u komunikaciji, pattern predstavlja listu riječi koja opisuje tu namjenu te odgovor je povratna informacija chatbota na korisnički upit.

```
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": [
        "Hello",
        "Hi",
        "Hey",
        "Good morning",
        "Good afternoon",
        "Good evening"
      ],
      "responses": [
        "Hi there, how can I help you today?",
        "Hello! How can I assist you?",
        "Hi! How can I help you today?",
        "Hey there, how can I assist you?",
        "Good [morning/afternoon/evening]! How may I help you?",
        "Hello, how may I assist you today?"
      ]
    },
    {
      "tag": "thanks",
      "patterns": [
        "Thank you",
        "Thanks",

```



```
"Thanks a lot",
"Thank you so much",
"Appreciate it"
],
"responses": [
"You're welcome!",
"No problem!",
"Glad to help!",
"Anytime!",
"My pleasure!",
"Not a problem!",
"It was my pleasure to assist you!"
]
},
{
"tag": "goodbye",
"patterns": [
"Goodbye",
"Bye",
"See you later",
"Talk to you soon"
],
"responses": [
"Goodbye, have a great day!",
"Bye, take care!",
"See you later!",
"Talk to you soon!",
"Have a great day!"
]
},
{
"tag": "order",
"patterns": [
"How do I place an order?",
"What is the process for ordering?",
"Can you help me with placing an order?",
"How long does it take to receive an order?",
"What is the status of my order?",
"I have a problem with my order"
],
"responses": [
"To place an order, please visit our website and select the items you wish to purchase. The standard delivery time is 3-5 business days. You can check the status of your order by logging into your account on our website. If you have any issues with your order, please contact our customer support team by phone
```

```
or email.",
"Certainly, to place an order you can call our toll-free number or visit our
website. Delivery usually takes 5-7 business days. If you need to check the
status of your order, you can log into your account on our website. If you have
any problems with your order, please contact our customer support team by phone
or email.",
"If you need assistance with placing an order, we're happy to help. You can
reach out to us through our website's live chat or by phone and we'll guide you
through the process. The estimated delivery time for orders is 7-10 business
days. If you have any issues with your order, please let us know and we'll do
our best to help you out."
]
},
{
  "tag": "payment",
  "patterns": [
    "How can I pay for my order?",
    "What payment methods do you accept?",
    "Is it safe to make a payment on your website?",
    "I have a problem with my payment",
    "When will my payment be processed?"
  ],
  "responses": [
    "We accept credit cards, debit cards, PayPal, and Apple Pay. Our payment
process is secure and all transactions are encrypted. Your payment will be
processed within 24 hours. If you have any problems with your payment, please
contact customer support via email or phone.",
    "You can pay for your order using credit cards, debit cards, PayPal, or Apple
Pay. Our payment process is secure and your information is encrypted. Your
payment will be processed within 24 hours. If you encounter any issues with
your payment, please contact customer support via email or phone.",
    "Our payment options include credit cards, debit cards, PayPal, and Apple Pay,
all of which are secure and encrypted. We usually process payments within 24
hours. If you have any concerns or issues with your payment, please contact
customer support via email or phone, and we'll help you out."
  ]
},
{
  "tag": "returns",
  "patterns": [
    "What is your return policy?",
    "How do I return an item?",
    "Can I get a refund?",
    "I received a damaged item",
    "I received the wrong item"
```

```
],
"responses": [
  "Our return policy allows you to return items within 30 days. To initiate a return, please fill out our online return form and follow the instructions provided. If you are eligible for a refund, it will be processed within 5-7 business days. If you received a damaged or incorrect item, please contact our customer support team at [email/phone number] and we'll assist you.",
  "For returns, please initiate the process within 14 days and follow these instructions: pack the item securely in its original packaging, complete the return form provided, and attach the shipping label. Once we receive the item, we'll process your refund within 3-5 business days. If you received a damaged or incorrect item, please contact our customer support team at [email/phone number] for assistance.",
  "If you would like to return an item, please do so within 30 days using our online return form and following the provided instructions. Refunds are usually processed within 7-10 business days. In case you received a damaged or incorrect item, please contact our customer support team at [email/phone number] and we'll take care of it for you."
],
},
{
  "tag": "support",
  "patterns": [
    "How can I contact customer support?",
    "I need help with an issue",
    "I have a question",
    "Can you help me with something?"
  ],
  "responses": [
    "Our customer support team is available 24/7 and can be reached by phone, email, or live chat. We're happy to help you with any issues or questions you may have.",
    "If you need assistance, our customer support team is available during business hours (9am-5pm PST). You can reach us by phone or email and we'll do our best to help you.",
    "For any questions or issues, please don't hesitate to contact our customer support team. We're available 24/7 and can be reached by phone, email, or live chat.",
    "If you have any questions or need help with anything, our customer support team is here for you. You can reach us 24/7 by phone, email, or live chat."
  ]
}
]
```

U nastavku rada bit će izvršena priprema ovih podataka za obradu istih u različitim metodama mašinskog učenja.

Preprocesiranje podataka

Za obradu prirodnog jezika u Pythonu korištena je NLTK biblioteka. Cilj koda je učitavanje podataka te lematizacija i tokenizacija riječi. Lematizacija je proces konvertovanja riječi u njihovu osnovnu formu i koristi se kako bi se smanjila dimenzionalnost podataka i olakšalo procesiranje. Tokenizacija je proces dijeljenja teksta na manje jedinice, koje se obično nazivaju tokeni. Ovi tokeni su obično riječi, ali mogu biti i pojedinačni znakovi, interpunkcijski znakovi, itd. Tokenizacija se koristi kako bi se razumjelo značenje teksta i izdvojile ključne informacije. Izvršeno je izdvajanje kategorija, riječi i namjena iz skupa podataka. Ovo se radi kako bi se stvorile liste kategorija i riječi koje se koriste u procesu klasifikacije teksta.

Osim toga, u sklopu preprocesiranja izvršen je i stemming, uklanjanje stop riječi, brojeva, interpunkcijskih znakova. Također je izvršeno i pretvaranje svih slova u mala.

```
import json
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('wordnet')
```

```
# load dataset
with open("dataset.json") as file:
    data = json.load(file)
```

```
import re
import json
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, PorterStemmer
from nltk.tokenize import word_tokenize
import string

lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Pretvaranje u mala slova
```

```

text = text.lower()

# Uklanjanje brojeva
text = re.sub(r'\d+', '', text)

# Uklanjanje znakova interpunkcije
text = text.translate(str.maketrans("", "", string.punctuation))

# Tokenizacija teksta
tokens = word_tokenize(text)

# Lematizacija i stemming
tokens = [lemmatizer.lemmatize(stemmer.stem(token)) for token in tokens]

# Uklanjanje stop riječi
tokens = [token for token in tokens if token not in stop_words]

# Spajanje tokena u rekonstruirani tekst
preprocessed_text = ' '.join(tokens)

return preprocessed_text

# Učitavanje dataseta
with open('dataset.json') as file:
    dataset = json.load(file)

pat = []
res = []

# Preprocesiranje podataka
for intent in dataset['intents']:
    patterns = intent['patterns']
    responses = intent['responses']

    # Preprocesiranje uzoraka
    preprocessed_patterns = [preprocess_text(pattern) for pattern in patterns]
    pat.append(preprocessed_patterns)

    # Preprocesiranje odgovora
    preprocessed_responses = [preprocess_text(response) for response in
responses]
    res.append(preprocessed_responses)

# Ispis preprocesiranog dataseta
print(pat)

```

```
print(res)
```

Na Slici 1 je prikazan dio rezultata uspješno izvršenog koda.

```
[[ 'hello', 'hi', 'hey', 'good morn', 'good afternoon', 'good even'], [ 'thank', 'thank', 'thank lot', 'th  
[[ 'hi help today', 'hello assist', 'hi help today', 'hey assist', 'good morningafternooneven may help',
```

Ilustracija 1 - Preprocesiranje podataka

Inženjering karakteristika

U sklopu treće faze našeg projekta urađen je inženjering karakteristika. Skupo podataka koji smo predstavili u prethodnim fazama projekta, podijeljen je na dvije grupe - pitanja i odgovori, a zatim su ti podijeljeni podaci iskorišteni za prikazivanje u različitim formama. Najprije smo koristili vektorski, a nakon toga i embedding prikaz. Što se tiče vektorskog prikaza, implementirali smo jednokratno (one-hot) kodiranje, Bag of words (BoW) i TF-IDF. Od embedding prikaza urađen je Word2Vec i Fast Text.

Kao što je naglašeno, podaci su podijeljeni u skupove questions i answers. Prikaz koda nalazi se u nastavku:

```
from nltk.tokenize import word_tokenize

# Prikupljanje pitanja i odgovora iz podataka
text = data['intents']
questions = []
answers = []

for intent in text:
    for pattern in intent['patterns']:
        questions.append(pattern)
    for pattern in intent['responses']:
        answers.append(pattern)
```

One-hot kodiranje (engl. one-hot encoding) je tehnika koja se koristi za reprezentaciju kategoričkih varijabli u obliku vektora. Ova tehnika se često koristi u mašinskom učenju i obradi prirodnog jezika. Kada se primjenjuje one-hot kodiranje, svaka kategorija varijable predstavlja se kao binarni vektor. Vektor ima dužinu jednaku broju jedinstvenih kategorija u varijabli. Svaka pozicija u vektoru predstavlja jednu kategoriju, a vrijednost na toj poziciji je 1 ako primjer pripada toj kategoriji, odnosno 0 ako ne pripada. Ova tehnika omogućava obradu kategoričkih varijabli koje su učestale u mnogim zadacima mašinskog učenja. Na taj način se kategorička informacija transformiše u format koji je pogodan za daljnju

```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import numpy as np
from numpy import array
from sklearn.preprocessing import OneHotEncoder
import itertools

tokens_q = [doc.split(" ") for doc in questions]

# stvaranje liste tokena i kreiranje riječnika koji mapira riječ u id riječi,
all_tokens = itertools.chain.from_iterable(tokens_q)
word_to_id = {token: idx for idx, token in enumerate(set(all_tokens))}
print(word_to_id)

token_ids = np.zeros((len(questions), len(word_to_id)))
for i, sentence in enumerate(questions):
    for word in sentence.split():
        token_ids[i, word_to_id[word]] = 1

# pretvaranje token-id liste u one-hot reprezentaciju
vec=OneHotEncoder()
X = vec.fit_transform(token_ids).toarray()
print(X)
```

```
{'Hey': 0, 'which': 1, "we're": 2, 'are': 3, 'days': 4, 'have.': 5, 'a': 6,
[[1. 0. 1. ... 0. 1. 0.]
 [1. 0. 1. ... 0. 1. 0.]
 [1. 0. 1. ... 0. 1. 0.]
 ...
 [1. 0. 1. ... 0. 1. 0.]
 [1. 0. 1. ... 0. 1. 0.]
 [1. 0. 1. ... 0. 1. 0.]
 ]]
```

Bag of Words (BoW), ili vreća riječi, je jednostavna tehnika za obradu teksta koja se koristi u prirodnom jeziku i mašinskom- učenju. Ova tehnika predstavlja dokument ili tekstualni korpus kao vreću (skup) riječi, bez uzimanja u obzir gramatičke strukture ili redoslijeda riječi.

Proces bag of words se sastoji od sljedećih koraka:

1. Tokenizacija: Tekst se podijeli na pojedinačne riječi ili tokene. Obično se koristi jednostavna podjela na riječi, ali može se primijeniti i naprednija tokenizacija.
2. Izgradnja rječnika: Sve jedinstvene riječi u tekstu se bilježe kako bi se stvorio rječnik. Svaka riječ predstavlja jedan atribut.
3. Kodiranje dokumenata: Svaki dokument se predstavlja kao vektor koji sadrži broj pojavljivanja svake riječi iz rječnika u tom dokumentu. Broj pojavljivanja može biti apsolutan broj (count) ili binarna vrijednost (0 ili 1) koja označava prisutnost ili odsutnost riječi.

Bag of Words se često koristi kao jednostavan način reprezentacije teksta u mašinskom učenju, ali nema uključene informacije o semantičkom značenju ili kontekstu riječi. Također, ovaj pristup ne uzima u obzir redoslijed riječi i može dovesti do gubitka nekih važnih informacija u tekstu. Ipak, Bag of Words je koristan za mnoge zadatke, kao što su klasifikacija teksta, grupisanje i prepoznavanje uzoraka u velikim skupovima podataka.

Implementacija BoW je vrlo jednostavna zahvaljujući već gotovoj funkciji `CountVectorizer`.

```
#Izgradnja BoW reprezentacije za korpus
count_vect = CountVectorizer()
bow_rep = count_vect.fit_transform(questions)

#Prikaz dobijenog vokabulara
print(" Vokabular je:", count_vect.vocabulary_)

#Ispis kodiranih vektora
for sentence, vector in zip(questions, bow_rep):
    print(f'Rečenica: {sentence}')
    print(f'Bow: {vector.toarray()}\n')
```

Dio izlaza prikazan je na slici ispod.


```
tfidf_matrix = vectorizer.fit_transform(questions)

# Pretvaranje rijetke matrice u gusto oblik
tfidf_matrix = tfidf_matrix.toarray()

# Ispis vektora
for i, document in enumerate(questions):
    print("\nRečenica: ", questions[i])
    print(tfidf_matrix[i])
```

```
Rečenica: I need help with an issue
[0. 0. 0.37279812 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0.42694198 0. 0. 0. 0.
 0.51950152 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0.51950152 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.37279812
 0. 0. 0. ]

Rečenica: I have a question
[0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.63492463
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0.77257408 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0.
 0. 0. 0. ]
```

Ilustracija 4 - Rezultat TF-IDF

Word2Vec je popularna tehnika u obradi prirodnog jezika koja se koristi za prevođenje riječi u vektorski prostor, gdje se riječi predstavljaju kao gusti vektori brojeva. Ova tehnika omogućava modeliranje semantičkih sličnosti među riječima i otkrivanje njihovih latentnih značenja na temelju konteksta u kojem se pojavljuju. Word2Vec modeli se temelje na pretpostavci da su riječi koje se pojavljuju u sličnim kontekstima međusobno slične i da

dijele slična semantička značenja. Trenirani Word2Vec model generiše guste vektore za svaku riječ u vokabularu. Vektori imaju svojstvo da riječi koje su semantički slične imaju slične vektorske reprezentacije. Ova svojstva vektora omogućavaju razne primjene, kao što su pronalaženje sličnih riječi, grupisanje riječi po značenju, prepoznavanje semantičkih veza među riječima i mnoge druge.

Za porebe implementacije ovog embedding prikaza prvo je bilo potrebno instalirati biblioteku gensim. Zatim, opet zahvaljujući gotovoj funkciji Word2Vec iz istoimene biblioteke, formiran je prikaz riječi i njihovih vektora. Radi bolje preglednosti semantičke sličnosti između riječi, napisan je i kod za izgradnju matrice sličnosti.

```
pip install gensim nltk
```

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize

# Tokenizacija teksta
tokenizirani_dokumenti = [word_tokenize(dokument.lower()) for dokument in
questions]
```

```
# Izgradnja modela Word2Vec
model = Word2Vec(tokenizirani_dokumenti, vector_size=100, window=5,
min_count=1, workers=4)
```

```
# Prikaz riječi i njihovih vektora
for riječ in model.wv.key_to_index:
    vektor = model.wv[riječ]
    print("Riječ:", riječ, "Vektor:", vektor)
```

```
# Izgradnja matrice sličnosti
broj_rijeci = len(model.wv.key_to_index)
similarity_matrix = np.zeros((broj_rijeci, broj_rijeci))

for i, rijec1 in enumerate(model.wv.index_to_key):
    for j, rijec2 in enumerate(model.wv.index_to_key):
        similarity_matrix[i, j] = model.wv.similarity(rijec1, rijec2)

rijeci = model.wv.index_to_key
df = pd.DataFrame(similarity_matrix, index=rijeci, columns=rijeci)
styled_df = df.style.set_properties(**{'text-align': 'center', 'border': '1px
solid black'})
styled_df
```

	?	i	a	you	order	an	my	how	can	with	payment	is	what	item	help
?	1.000000	-0.007508	-0.049312	-0.109571	-0.025465	-0.065767	0.018516	0.094848	0.030119	0.214408	0.065156	0.083465	0.094599	-0.039342	-0.036187
i	-0.007508	1.000000	-0.022526	0.068754	0.005715	0.011561	-0.112593	-0.113181	0.034877	-0.095906	-0.132608	0.010141	-0.002865	0.138244	0.161166
a	-0.049312	-0.022526	1.000000	-0.012469	0.171387	0.066378	0.147030	-0.000433	0.200794	-0.032731	-0.100475	0.174746	0.047758	-0.134746	-0.184432
you	-0.109571	0.068754	-0.012469	1.000000	-0.043767	0.133233	0.042468	-0.013293	0.075460	-0.170104	0.013668	0.042499	-0.008516	0.007421	0.178617
order	-0.025465	0.005715	0.171387	-0.043767	1.000000	0.140032	0.035383	-0.025982	-0.066594	-0.172854	-0.257205	-0.004661	0.151146	0.253049	-0.085300
an	-0.065767	0.011561	0.066378	0.133233	0.140032	1.000000	0.020878	-0.056047	0.063495	-0.105404	0.022113	0.169480	-0.143195	0.044845	-0.108105
my	0.018516	-0.112593	0.147030	0.042468	0.035383	0.020878	1.000000	0.006443	0.009636	0.002248	0.051523	-0.051617	0.001941	0.013285	0.008624
how	0.094848	-0.113181	-0.000433	-0.013293	-0.025982	-0.056047	0.006443	1.000000	-0.142406	-0.093469	0.108833	0.113410	-0.039674	-0.024329	0.105812
can	0.030119	0.034877	0.200794	0.075460	-0.066594	0.063495	0.009636	-0.142406	1.000000	0.044915	0.027072	0.040342	-0.038961	-0.121986	-0.076054
with	0.214408	-0.095906	-0.032731	-0.170104	-0.172854	-0.105404	0.002248	-0.093469	0.044915	1.000000	0.014482	-0.074181	-0.046008	-0.106477	-0.045989
payment	0.065156	-0.132608	-0.100475	0.013668	-0.257205	0.022113	0.051523	0.108833	0.027072	0.014482	1.000000	0.111265	0.129186	-0.000417	-0.046191
is	0.083465	0.010141	0.174746	0.042499	-0.004661	0.169480	-0.051617	0.113410	0.040342	-0.074181	0.111265	1.000000	-0.028554	-0.161716	-0.058006
what	0.094599	-0.002865	0.047758	-0.008516	0.151146	-0.143195	0.001941	-0.039674	-0.038961	-0.046008	0.129186	-0.028554	1.000000	-0.075960	0.092448
item	-0.039342	0.138244	-0.134746	0.007421	0.253049	0.044845	0.013285	-0.024329	-0.121986	-0.106477	-0.000417	-0.161716	-0.075960	1.000000	0.027663
help	-0.036187	0.161166	-0.184432	0.178617	-0.085300	-0.108105	0.008624	0.105812	-0.076054	-0.045989	-0.046191	-0.058006	0.092448	0.027663	1.000000
good	0.053270	0.122797	-0.020021	-0.154201	0.015720	-0.026915	-0.146571	0.080349	-0.014861	0.010708	-0.042701	0.012581	-0.004022	-0.144913	-0.098434
do	0.220876	0.024490	-0.015027	-0.260140	0.110190	0.048934	0.164580	0.176089	-0.118897	0.006392	-0.011059	-0.145866	-0.115376	-0.031516	0.049407
have	-0.112414	0.082916	0.002368	0.041844	-0.028817	0.072667	-0.106594	0.115933	0.013314	-0.160430	0.096336	0.062001	-0.066929	0.116771	0.247133
it	-0.012521	0.159437	0.153945	-0.040993	0.033759	-0.139539	0.075134	0.080818	0.017787	0.020639	-0.049677	0.134038	0.035824	0.142925	-0.005402
to	0.007774	-0.158942	-0.034317	-0.001557	-0.276522	-0.126390	0.111200	0.318798	0.096588	-0.038053	0.086196	0.098000	-0.001606	-0.049760	0.162063

Ilustracija 5 - Matrica sličnosti

FastText je biblioteka i tehnika za obradu prirodnog jezika koja se temelji na vektorizaciji riječi, slično kao i Word2Vec. Razvijena je u okviru Facebook AI Research laboratorija i predstavlja proširenje Word2Vec modela. Glavna razlika između FastText-a i Word2Vec-a je u načinu na koji se riječi tretiraju. Dok Word2Vec tretira riječi kao cjeline, FastText riječi tretira kao niz manjih podriječi ili "n-grama". Na taj način, FastText može uhvatiti semantičke veze ne samo na riječnoj razini, već i na razini podsastava riječi. FastText model se trenira na velikim korpusima teksta i generiše vektorsku reprezentaciju za svaku riječ u korpusu. Slično kao i Word2Vec, vektori riječi se koriste za izračunavanje sličnosti između riječi, grupisanje riječi i različite zadatke u obradi prirodnog jezika.

Jedna od ključnih prednosti FastText-a je sposobnost rada s rijetkim ili manje frekventnim riječima koje se često susreću u specifičnim domenima ili jezicima. Budući da FastText razbija riječi na manje podsastave, može izvući značenje iz tih podsastava, čak i ako nisu poznata u trening korpusu.

Kao što je to bio slučaj sa Word2Vec implementacijom, i za ovaj algoritam smo koristili biblioteku gensim i funkciju FastText. Model je istreniran u 100 epoha, a ispod je prikazan kod, kao i analiza sličnosti riječi process sa ostalim pronađenim riječima.

```
from gensim.models import FastText
from gensim.test.utils import common_texts
```

```

# Ručno kreiran dokument
document = questions

# Tokenizacija dokumenta
tokenized_document = [doc.split() for doc in document]

# Inicijalizacija FastText modela
model = FastText(vector_size=100, window=5, min_count=1, workers=4)

# Treniranje modela
model.build_vocab(corpus_iterable=tokenized_document)
model.train(corpus_iterable=tokenized_document,
            total_examples=len(tokenized_document), epochs=100)

word = 'process'
# Pronalaženje najsličnijih riječi
similar_words = model.wv.most_similar(word)
print(f"Najsličnije riječi za '{word}': {similar_words}")

```

Rezultat:

```

Najsličnije riječi za 'process': [('processed?', 0.9199726581573486),
('payment', 0.8016743063926697), ('ordering?', 0.7820302844047546),
('order?', 0.7658452987670898), ('What', 0.7481674551963806), ('order',
0.7403881549835205), ('website?', 0.7266000509262085), ('your',
0.7029959559440613), ('received', 0.6907960176467896), ('receive',
0.6850530505180359)]

```

Implementacija algoritama

Random forest (Nasiha Imamović)

Osnovna ideja iza Random Forest algoritma je kombinovanje više stabala odlučivanja kako bi se postigla bolja prediktivna točnost. Random Forest se može koristiti i za klasifikacijske i za regresijske probleme. Dobro se nosi sa visokom dimenzionalnošću problema i daje dobre rezultate za velike skupove podataka. Najčešće se primjenjuje u zadacima gdje je potrebno izvršiti određenu detekciju uzoraka ili analizu teksta.

Prvi korak pri kreiranju Random Forest modela je podjela podataka na skupove za trening i testiranje. Skup za trening se koristi za treniranje modela, dok se skup za testiranje koristi za evaluaciju performansi modela. Za svako stablo u Random Forestu, generiše se podskup trening podataka bootstrapping metodom. Ovaj korak omogućava svakom stablu da vidi samo dio podataka, stvarajući različite skupove trening podataka. Za svako stablo, iz podskupa trening podataka se nasumično odabire podskup značajki koje će se koristiti za izgradnju stabla. Ovaj korak omogućava raznolikost između stabala i smanjuje korelaciju između njih. Za izgradnju stabla odlučivanja, koriste se odabrani podskup trening podataka i značajke. Stablo se gradi rekurzivno tako da se nasvakom čvoru vrši podjela na temelju odabrane značajke i granice. Cilj je maksimizirati čistoću ili smanjiti gubitak informacije pri svakoj podjeli. Postupak izgradnje stabala ponavlja se za svako stablo u Random Forestu. Kada se sva stabla izgrade, predikcija se vrši tako da se svako stablo primijenina ulazni primjer i dobiju se predikcije. Konačna predikcija se izračunava na temelju ansambla predikcija pojedinačnih stabala, na primjer, pomoću glasanja za klasifikaciju ili prosječne vrijednosti za regresiju. Performanse modela se evaluiraju na temelju skupa za testiranje. To uključuje usporedbu predikcija modela s pravim oznakama i računanje relevantnih metrika, poput točnosti, preciznosti, odziva ili srednje kvadratne pogreške. Koristeći ovaj algoritam, chatbot može napraviti preciznija predviđanja o namjeri i značenju korisničkih inputa, što može pomoći da se poboljša kvalitet njegovih odgovora. Na primjer, Random Forest algoritam bi se mogao koristiti za klasifikaciju korisničkih inputa u različite kategorije, kao što su zahtjevi za informacijama, zahtjevi za pomoć ili zahtjevi za izvođenje određene radnje. Radeći to, chatbot može lakše razumjeti namjere korisnika i dati prikladniji odgovor. Osim toga, može se koristiti i za identifikaciju i ispravljanje grešaka u NLP sistemu chatbota, pomažući da se poboljšaju njegove ukupne performanse i pouzdanost.

Još neke od prednosti koje se mogu izdvojiti su:

- Skalabilnost - Random Forest može rukovati s velikim skupovima podataka bez gubitka performansi. To ga čini pogodnim za chatbotove koji moraju obraditi veliki broj korisničkih upita u stvarnom vremenu.

- Robustnost - Algoritam je otporan na preprilagođavanje zbog upotrebe ansambla stabala odlučivanja. Ansambliranje više stabala pomaže u smanjenju varijanse i poboljšava općenitost modela. Ovo je važno kod chatbota jer se može suočiti s različitim vrstama upita i mora biti sposoban generalizirati odgovore.
- Efikasnost - Random Forest ima visoku brzinu treniranja i predviđanja u usporedbi s nekim drugim kompleksnijim algoritmima mašinskog učenja. To znači da se chatbot može brzo prilagoditi novim podacima i generisati odgovore u stvarnom vremenu.
- Otkrivanje značajki - Random Forest pruža informacije o važnosti značajki. To znači da chatbot može identificirati ključne riječi ili obrasce u korisničkim upitima koje može koristiti za bolje razumijevanje i generisanje odgovora.
- Interpretabilnost - Random Forest omogućava razumijevanje i interpretaciju rezultata. Ukazuje na najvažnije značajke za donošenje odluka i kako svako stablo u ansamblu doprinosi konačnom rezultatu. To pruža mogućnost uvid u unutarnje funkcionisanje chatbota i prilagodbe na temelju povratnih informacija.

U nastavku će biti objašnjen Random Forest algoritam primijenjen na našem skupu podataka.

Učitavanje i preprocesiranje podataka

```
import json
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Učitavanje podataka iz JSON datoteke
with open("dataset.json") as file:
    data = json.load(file)

patterns = []
labels = []
responses = []

# Prikupljanje uzoraka, oznaka i odgovora iz podataka
```

```

for intent in data['intents']:
    for pattern in intent['patterns']:
        patterns.append(pattern.lower())
        labels.append(intent['tag'])
        responses.extend(intent['responses'])

# Preprocesiranje teksta: lematizacija, uklanjanje stop riječi i tokenizacija
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    tokens = [token for token in tokens if token.lower() not in stop_words and
not token.isnumeric()]
    return ' '.join(tokens)

patterns = [preprocess_text(pattern) for pattern in patterns]

```

Prvo je prikazano učitavanje podataka koje se vrši putem “json.load()” funkcije i pohranjuje se u varijablu “data”. Nakon toga se iz podataka prikupljaju uzorci, oznake i odgovori. Petljom se prolazi kroz strukturu podataka iz JSON datoteke i uzorci se dodaju u listu “patterns”, oznake u listu “labels”, a odgovori u listu “responses”. Uzorci se pretvaraju u mala slova pomoću metode “lower()”. Nakon toga se vrši preprocesiranje teksta. Prvo se učitavaju potrebni resursi za obradu teksta pomoću funkcija “nltk.download()” a zatim se definiše funkcija “preprocess_text(text)” koja vrši lematizaciju, uklanjanje stop riječi i tokenizaciju teksta. Tekst se tokenizira pomoću “word_tokenize()” funkcije a zatim se primjenjuje lematizacija pomoću “WordNetLemmatizer”. Na kraju se uklanjaju stop riječi i brojevi. Preprocesirani uzorci se pohranjuju u varijablu “patterns”.

Podjela podataka na trening i testni skup podataka i kreiranje modela

```

# Podjela podataka na trening i testni skup
train_patterns, test_patterns, train_labels, test_labels =
train_test_split(patterns, labels, test_size=0.2, random_state=42)

# Pretvaranje uzoraka u vektore značajki koristeći TF-IDF

```



```
vectorizer = TfidfVectorizer()
train_features = vectorizer.fit_transform(train_patterns)
test_features = vectorizer.transform(test_patterns)

# Izgradnja i treniranje Random Forest modela
rf_model = RandomForestClassifier()
rf_model.fit(train_features, train_labels)
```

Za podjelu uzoraka ("patterns") i oznaka ("labels") na trening i testni skup koristi se "train_test_split()" funkcija. Trening skup čini 80% podataka, dok testni skup čini preostalih 20%. Rezultati podjele se pohranjuju u varijable "train_patterns", "test_patterns", "train_labels" i "test_labels". Za pretvaranje tekstualnih podataka (uzoraka) u vektore značajki koristeći TF-IDF koristi se "TfidfVectorizer". Za trening skup se koristi "fit_transform()" metoda, dok se za testni skup koristi samo "transform()" metoda. Rezultirajući vektori značajki se pohranjuju u varijable "train_features" i "test_features". Za izgradnju modela inicijalizira se "RandomForestClassifier" koji predstavlja Random Forest model. Model se trenira pomoću metode "fit()" na trening skupu, koristeći vektore značajki "train_features" i oznake "train_labels".

Evaluacija modela nad testnim skupom podataka

```
# Evaluacija modela na testnom skupu
predictions = rf_model.predict(test_features)
accuracy = accuracy_score(test_labels, predictions)
precision = precision_score(test_labels, predictions, average='weighted')
recall = recall_score(test_labels, predictions, average='weighted')
f1 = f1_score(test_labels, predictions, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

U dijelu koda prikazanom iznad se vrši predikcija i evaluacija modela. Linija "predictions = rf_model.predict(test_features)" koristi obučeni Random Forest model "rf_model" za predikciju oznaka na testnom skupu značajki "test_features". Metoda predict se koristi za generisanje predikcija na temelju dobivenih značajki.

Što se tiče evaluacijskih metrika, dobiveni su sljedeći rezultati:

- Tačnost = 0.71
- Preciznost = 0.76

- Recall = 0.57
- F1 = 0.61

Linija "accuracy = (predictions == test_labels).mean()" izračunava tačnost modela tako da uspoređuje predikcije sa stvarnim oznakama u testnom skupu. Operator "==" uspoređuje elemente iz predictions i test_labels kako bi se dobila boolean vrijednost za svaku usporedbu. Metoda mean se koristi za izračunavanje prosječne vrijednosti boolean vrijednosti, što predstavlja postotak tačnosti. Nakon uspješno izvršenog koda, postignuta tačnost je 71.43%

Korištenje modela za generisanje odgovora

```
import random

# Korištenje modela za generisanje odgovora
def generate_response(input_text, model, vectorizer):
    input_text = preprocess_text(input_text)
    input_vector = vectorizer.transform([input_text])
    predicted_class = model.predict(input_vector)[0]

    # Pronalaženje odgovarajućeg skupa odgovora na temelju predviđene klase
    response_set = [response for tag, response in zip(labels, responses) if tag
== predicted_class]

    # Nasumičan odabir odgovora iz skupa
    response = random.choice(response_set) if response_set else "I'm sorry, I
don't have an appropriate response."
    return response
```

Funkcija "generate_response" koristi model Random Forest za generiranje odgovora na temelju unesenog teksta i kao ulaz prima tri parametra - "input_text", "model" i "vectorizer" što predstavljaju uneseni tekst za koji se generiše odgovor, obučeni Random Forest model koji se koristi za predikciju i TfidfVectorizer koji se koristi za pretvaranje unesenog teksta u vector značajki, respektivno.

Kako bi se generisao odgovor uneseni tekst se preprocesira pomoću funkcije "preprocess_text", koja vrši lematizaciju, uklanjanje stop riječi i tokenizaciju a zatim se preprocesirani tekst pretvara u vektor značajki pomoću TfidfVectorizera "vectorizer". Koristi se obučeni model Random Forest "model" za predikciju klase odgovora na temelju vektora značajki. Nakon toga se pronalazi skup odgovora koji su povezani s predviđenom klasom i nasumično se odabire jedan odgovor iz skupa ako postoji, inače se koristi zadani

odgovor "I'm sorry, I don't have an appropriate response." I zatim se odabrani odgovor vraća kao rezultat funkcije.

Kreiranje chatbota

```
while True:
    user_input = input("User: ")
    if user_input.lower() == "exit":
        break

    response = generate_response(user_input, rf_model, vectorizer)
    print("ChatBot:", response)
```

U završnom koraku kreiran je i chatbot sa kojim će korisnik razgovarati i kao što je prikazano, chatbot je uspješno odgovorio na sve postavljene upite.

```
User: Hiiii
ChatBot: Hi! How can I help you today?
User: How can I place an order
ChatBot: To place an order, please visit our website and select the items you wish to purchase.
User: Thank you
ChatBot: Anytime!
User: Bye
ChatBot: Bye, take care!
User: exit
```

Ilustracija 6 - Primjena chatbota

BERT i transformeri (Neira Novalić i Nejra Rovčanin)

BERT je jedan od najpoznatijih i najutjecajnijih modela baziranih na transformatorima. On se bazira na ideji preobučavanja jezičnog modela na velikom korpusu podataka iz različitih jezičnih izvora, poput Wikipedije. Tijekom preobučavanja, BERT model uči Masked Language Modeling (MLM) i Next Sentence Prediction (NSP). MLM zahtijeva predviđanje zamagljenih riječi u rečenici, dok NSP predviđa da li su dvije rečenice uzastopne u izvornom tekstu.

Prednosti transformera s BERT-om uključuju:

- Bi-direkcionalnost - BERT koristi kontekstualno ugrađivanje riječi koje uključuju informacije iz prethodnih i sljedećih riječi, omogućujući bolje razumijevanje konteksta.
- Duboki slojevi - Transformeri s BERT-om imaju više slojeva, što im omogućuje učenje složenijih značajki i složenih odnosa u tekstu.

- Preobučavanje na velikom korpusu - BERT se preobučava na velikom broju različitih jezičnih izvora, što mu omogućuje razumijevanje širokog spektra jezičnih obrazaca.

Transformeri s BERT-om su dokazali svoju moć u različitim zadacima obrade prirodnog jezika, poput klasifikacije teksta, prepoznavanja entiteta, generisanja teksta i mašinskog prevođenja. Njihova sposobnost razumijevanja konteksta i oblikovanja kvalitetnih reprezentacija riječi čini ih izuzetno korisnim alatom za izgradnju učinkovitih NLP modela, uključujući i chatbote.

Korištenje BERT-a za chatbote ima nekoliko prednosti:

- Kontekstualno razumijevanje - BERT koristi bi-direkcionalno ugrađivanje riječi, što znači da razumije riječi u kontekstu prethodnih i sljedećih riječi. To omogućuje chatbotu da bolje razumije korisničke upite i pruži preciznije odgovore.
- Obrada prirodnog jezika - BERT je prethodno treniran na velikom korpusu podataka, što mu omogućuje razumijevanje jezičnih obrazaca i konteksta. Chatbot koji koristi BERT može bolje obraditi prirodni jezik, uključujući različite stilove izražavanja, idiome i fraze.
- Generisanje relevantnih odgovora - BERT ima sposobnost generiranja reprezentacija riječi koje sadrže semantičke informacije. To omogućuje chatbotu da generira relevantne odgovore na osnovu korisničkog upita, poboljšavajući kvalitetu interakcije s korisnikom.
- Fleksibilnost i prilagodljivost - BERT može biti prilagođen specifičnom domenu ili ciljnoj publici putem dodatnog treniranja na relevantnim podacima. To omogućuje chatbotu da bude prilagođeniji i da bolje odgovara potrebama korisnika.
- Efikasnost i performanse - BERT je optimiziran za brzu obradu podataka, što rezultira bržim i efikasnijim chatbotom. Također, BERT modeli često postižu visoke performanse u zadacima obrade prirodnog jezika, što znači da chatbot može pružiti kvalitetne odgovore korisnicima.

Sve ove prednosti čine BERT idealnim izborom za chatbote.

U nastavku će biti objašnjeni transformer sa BERT-om primijenjeni na našem skupu podataka.

Učitavanje i preprocesiranje podataka

```
with open("dataset.json") as file:
    data = json.load(file)

# Preprocesiranje podataka: lematizacija, uklanjanje stop riječi i brojeva
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Tokenizacija
    tokens = word_tokenize(text)
    # Lematizacija
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    # Uklanjanje stop riječi i brojeva
    tokens = [token for token in tokens if token.lower() not in stop_words and
not token.isnumeric()]
    # Vraćanje preprocesiranog teksta kao spojene riječi
    return ' '.join(tokens)

patterns = []
labels = []

for intent in data['intents']:
    for pattern in intent['patterns']:
        patterns.append(pattern.lower())
        labels.append(intent['tag'])

patterns = [preprocess_text(pattern) for pattern in patterns]
```

Učitavanje i preprocesiranje podatka je urađeno na isti način kao za Random Forest algoritam. Prvo je prikazano učitavanje podataka a nakon toga se iz podataka prikupljaju uzorci, oznake i odgovori. Petljom se prolazi kroz strukturu podataka iz JSON datoteke i uzorci se dodaju u listu "patterns", oznake u listu "labels", a odgovori u listu "responses". Uzorci se pretvaraju u mala slova pomoću metode "lower()". Nakon toga se vrši preprocesiranje teksta. Prvo se učitavaju potrebni resursi za obradu teksta pomoću funkcija "nltk.download()" a zatim se definiše funkcija "preprocess_text(text)" koja vrši lematizaciju, uklanjanje stop riječi i tokenizaciju teksta. Tekst se tokenizira pomoću "word_tokenize()" funkcije a zatim se primjenjuje lematizacija pomoću

“WordNetLemmatizer”. Na kraju se uklanjaju stop riječi i brojevi. Preprocesirani uzorci se pohranjuju u varijablu “patterns”.

Podjela podataka na trening i testni skup i pretvaranje uzoraka u tokene

```
# Podjela na trening i testni skup
train_patterns, test_patterns, train_labels, test_labels =
train_test_split(patterns, labels, test_size=0.2, random_state=42)

# Pretvaranje uzoraka u vektore značajki
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
train_sequences = tokenizer.batch_encode_plus(train_patterns,
padding='longest', truncation=True, return_tensors='tf')
train_input_ids = train_sequences['input_ids']
train_attention_masks = train_sequences['attention_mask']

test_sequences = tokenizer.batch_encode_plus(test_patterns, padding='longest',
truncation=True, return_tensors='tf')
test_input_ids = test_sequences['input_ids']
test_attention_masks = test_sequences['attention_mask']
```

U ovom dijelu koda vrši se podjela uzoraka na trening i testni skup kako bi se omogućila evaluacija performansi modela. Funkcija “train_test_split” iz biblioteke “sklearn.model_selection” koristi se za tu svrhu. Uzorci, oznake, “patterns” i “labels”, koji su prethodno prikupljeni, dijele se na trening i testni skup u omjeru 80 - 20. To znači da će 80% podataka biti korišteno za treniranje modela, dok će preostalih 20% biti korišteno za evaluaciju performansi.

Nakon podjele uzoraka, vrši se pretvaranje uzoraka u vektore značajki. Koristi se “BertTokenizer” iz biblioteke “transformers” kako bi se tokenizirali uzorci, tj. podijelili na tokene a zatim se primjenjuje metoda “batch_encode_plus” na tokenizeru kako bi se izvršila kodiranja svih uzoraka odjednom. Uzorci se oblikuju u odnosu na najdužu duljinu, a zatim se vraćaju kao “input_ids” i “attention_mask”.

Pretvaranje oznaka u kategorizirane varijable

```
# Pretvaranje oznaka u kategorizirane varijable
label_encoder = LabelEncoder()
train_encoded_labels = label_encoder.fit_transform(train_labels)
test_encoded_labels = label_encoder.transform(test_labels)
num_labels = len(label_encoder.classes_)
```

U ovom dijelu koda vrši se pretvaranje oznaka u kategorizirane varijable kako bi se pripremili za upotrebu u modelu. Koristi se "LabelEncoder" iz biblioteke "sklearn.preprocessing" za tu svrhu. Prvo se stvara instanca "LabelEncoder" objekta, nazvana "label_encoder". Zatim se primjenjuje metoda "fit_transform" na trening oznakama ("train_labels"), koja istovremeno obavlja dva koraka treniranje encodera na trening oznakama i transformaciju trening oznaka u kategorizirane varijable. Rezultat pretvaranja se pohranjuje u varijablu "train_encoded_labels".

Nakon toga se primjenjuje metoda "transform" na testnim oznakama ("test_labels"), koja vrši samo transformaciju testnih oznaka u kategorizirane varijable. Rezultat se pohranjuje u varijablu "test_encoded_labels". Također, broj različitih kategorija oznaka se pohranjuje u varijablu "num_labels", koja se računa kao dužina atributa "classes_" u "label_encoder" objektu. Ova vrijednost će biti korisna prilikom definisanja izlaznog sloja modela.

```
# Izgradnja i treniranje BERT modela
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=num_labels)
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-4)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
model.fit([train_input_ids, train_attention_masks], train_encoded_labels,
epochs=13, batch_size=256)

# Evaluacija modela na testnom skupu
test_loss, test_accuracy = model.evaluate([test_input_ids,
test_attention_masks], test_encoded_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

U ovom dijelu koda vrši se izgradnja i treniranje BERT modela za klasifikaciju sekvenci teksta. Prvo se stvara instanca BERT modela za klasifikaciju sekvenci "TFBertForSequenceClassification" iz biblioteke "transformers". Koristi se prethodno trenirani BERT model 'bert-base-uncased' koji je učitao pomoću metode "from_pretrained". Prilikom stvaranja modela, također se specificira broj različitih kategorija oznaka ("num_labels"), što će odrediti izlazni sloj modela. Zatim se definiraju optimizator ("optimizer") i gubitak ("loss") za treniranje modela. Optimizator je Adam optimizator s definiranom stopom učenja ("learning_rate") od 2e-4. Definirani gubitak je "SparseCategoricalCrossentropy" koji se koristi za klasifikaciju s kategoriziranim varijablama. Oba optimizator i gubitak se specificiraju u modelu pomoću metode "compile".

Nakon toga se model trenira na trening podacima koristeći metodu "fit". Ulazni podaci za model su trening input ID-ovi ("train_input_ids") i attention maske ("train_attention_masks"). Oznake za treniranje su kategorizirane varijable ("train_encoded_labels"). Parametri kao što su broj epoha ("epochs") i veličina grupa ("batch_size") također su specificirani.

Nakon završetka treniranja, vrši se evaluacija modela na testnim podacima pomoću metode "evaluate". Ulazni podaci za evaluaciju su testni input ID-ovi ("test_input_ids") i attention maske ("test_attention_masks"). Oznake za evaluaciju su također kategorizirane varijable ("test_encoded_labels"). Rezultati evaluacije, gubitak ("test_loss") i tačnost ("test_accuracy"), se ispisuju na izlazu.

```
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score

# Predikcija na testnom skupu
predictions = model.predict([test_input_ids, test_attention_masks])
predicted_labels = np.argmax(predictions.logits, axis=1)

# Izračunavanje preciznosti, odziva i F1 mjere
precision = precision_score(test_encoded_labels, predicted_labels,
                             average='weighted')
recall = recall_score(test_encoded_labels, predicted_labels,
                       average='weighted')
f1 = f1_score(test_encoded_labels, predicted_labels, average='weighted')

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Što se tiče evaluacijskih metrika, dobiveni su sljedeći rezultati:

- Preciznost = 0.76
- Recall = 0.71
- F1 = 0.7

Postignuta tačnost je 85.71% te je to ujedno i najbolja tačnost postignuta od strane svih implementiranih i evaluiranih algoritama u ovom radu.

Korištenje modela za generisanje odgovora

```
import random

# Korištenje modela za generisanje odgovora
def generate_response(input_text, model, tokenizer):
    input_sequence = tokenizer.encode_plus(input_text.lower(),
padding='longest', truncation=True,
max_length=128, return_tensors='tf')
    logits = model(input_sequence.input_ids)[0]
    predicted_class = tf.argmax(logits, axis=-1).numpy()[0]
    tag = label_encoder.inverse_transform([predicted_class])[0]

    # Pronalaženje odgovarajućeg skupa na temelju taga
    for intent in data['intents']:
        if intent['tag'] == tag:
            responses = intent['responses']
            break

    # Odabir slučajnog odgovora iz skupa
    response = random.choice(responses)
    return response
```

Ovaj dio koda se odnosi na korištenje modela za generisanje odgovora na temelju ulaznog teksta.

Definisana je funkcija “generate_response” koja prima ulazni tekst, model i tokenizer kao argumente. Koristi se metoda “encode_plus” koja vrši kodiranje teksta dodjeljujući mu ID-ove i generiše tenzore za ulaz u model. Postavljene su odgovarajuće opcije kao što su dodavanje paddinga ograničavanje maksimalne dužine teksta i trunciranje teksta. Nakon kodiranja teksta, ulazni ID-ovi se prosljeđuju modelu, a dobiveni izlazi se pohranjuju u varijablu “logits”. Logiti predstavljaju rezultate predikcije modela za svaku klasu. Pomoću funkcije “argmax” iz TensorFlow biblioteke, dobiva se indeks klase s najvišom vrijednošću logita. Taj indeks se zatim koristi za dobivanje odgovarajuće oznake koristeći inverznu transformaciju label encoder-a. Oznaka se koristi za pronalaženje odgovarajućeg skupa odgovora u strukturi podataka “data”. Nakon pronalaska skupa odgovora, funkcija koristi biblioteku “random” za odabir slučajnog odgovora iz tog skupa. Odabrani odgovor se vraća kao rezultat iz funkcije.

Kreiranje chatbota

```
while True:
```

```
user_input = input("User: ")
if user_input.lower() == "exit":
    break

response = generate_response(user_input,model,tokenizer)
print("ChatBot:", response)
```

I chatbot kreiran korištenjem ovog algoritma je uspješno odgovorio na sve upite postavljene od strane korisnika.

```
User: Hi
ChatBot: Hey there, how can I assist you?
User: How can I place an order
ChatBot: If you need assistance with placing an order, we're happy to help. You can reach out to us through
User: How can I return a product
ChatBot: Our return policy allows you to return items within 30 days. To initiate a return, please fill out
User: And what are methods of payment
ChatBot: We accept credit cards, debit cards, PayPal, and Apple Pay. Our payment process is secure and all
User: Thank you
ChatBot: Glad to help!
User: Bye
ChatBot: Goodbye, have a great day!
User: exit
```

Ilustracija 7 - Primjena chatbota

Rekurentna neuronska mreža sa LSTM slojem (Rovčanin Nejra i Novalić Neira)

RNN (Recurrent Neural Network) je tip neuronske mreže koji je posebno dizajniran za obradu sekvencijalnih podataka, poput niza riječi u rečenici ili vremenskih serija. RNN ima sposobnost da zadržava unutrašnje stanje (memoriju) koje se osvježava sa svakim novim unosom podataka. To omogućava RNN-u da uzme u obzir kontekstualne informacije iz prethodnih koraka i koristi ih u obradi trenutnog koraka.

LSTM (Long Short-Term Memory) je specifična vrsta RNN-a koja je razvijena kako bi riješila problem dugoročne ovisnosti u RNN-ovima. Klasični RNN-ovi mogu imati problema s održavanjem informacija na duži rok zbog problema zvanog "problema nestajućeg gradijenta". LSTM arhitektura rješava ovaj problem kroz korištenje ćelija memorije.

LSTM ćelija sastoji se od nekoliko ključnih elemenata:

- Input gate - kontroliše koliko novih informacija treba biti primljeno i dodano u stanje memorije

- Forget gate - kontroliše koliko informacija iz prethodnog stanja memorije treba biti zaboravljeno
- Output gate - kontroliše koliko informacija iz stanja memorije treba biti izloženo kao izlaz

LSTM koristi ove vrste gateova kako bi odlučio koje informacije treba zadržati u stanju memorije, a koje treba zaboraviti. Ova sposobnost omogućava LSTM-u da nauči dugoročne ovisnosti u podacima, što je posebno korisno u zadacima obrade prirodnog jezika, kao što su mašinsko prevođenje, generisanje teksta i analiza sentimenta. Ovo je zapravo specifična vrsta RNN-a koja koristi ćelije memorije i gateove kako bi naučila i zadržala dugoročne ovisnosti u sekvencijalnim podacima. Ova arhitektura je postala vrlo popularna u području obrade prirodnog jezika i ima široku primjenu u raznim zadacima analize teksta.

LSTM je dobar za implementaciju chatbota iz nekoliko razloga. Chatboti obično rade s sekvencijalnim podacima, poput niza riječi u korisničkom unosu ili generisanju odgovora. LSTM, kao vrsta RNN-a, može obraditi ove sekvence korak po korak, uzimajući u obzir kontekstualne informacije iz prethodnih koraka. To omogućava chatbotu da uzme u obzir kontekst korisničkog unosa i generiše odgovor koji je koherentan i prilagođen situaciji. LSTM je posebno dizajniran za rješavanje problema dugoročnih ovisnosti. Chatboti često moraju razumjeti kontekstualne informacije iz prethodnih koraka kako bi dali adekvatan odgovor. LSTM arhitektura omogućava chatbotu da nauči i pamti ove dugoročne ovisnosti, što rezultuje kvalitetnijim i koherentnijim odgovorima. Chatboti se najčešće koriste za obradu tekstualnih podataka. LSTM je posebno dobar u obradi teksta jer može naučiti i predstaviti složene obrasce i značenja u tekstu. To omogućava chatbotu da razumije korisničke upite, prepoznaje ključne riječi i generira odgovore koji su relevantni i informativni. LSTM može biti korišten za generisanje prirodnog jezika, što je ključna funkcionalnost chatbota. Chatboti mogu koristiti LSTM za generisanje koherentnih rečenica koje su slične ljudskom govoru. Također može naučiti pravilnu sintaksu, gramatiku i stil jezika iz trening podataka i koristiti tu informaciju za odgovore.

U nastavku će biti objašnjena implementacija RNNsa LSTM slojem, a kasnije će biti urađena i evaluacija modela kao i primjer upotrebe chatbota.

Učitavanje podataka i izdvajanje uzoraka

Ispod je prikazano učitavanje podataka iz JSON datoteke i izdvajanje uzoraka i oznaka iz tih podataka. Prvo, postavljamo putanju do JSON datoteke u varijablu `dataset_path`. Nakon toga, otvaramo datoteku koristeći `open` funkciju i koristimo `json.load` funkciju da

bismo učitali podatke iz datoteke u varijablu *dataset*. Zatim, inicijaliziramo prazne liste *patterns* i *labels*. Koristimo dvije petlje da bismo iterirali kroz svaki intent u datasetu. Intent predstavlja određeni skup podataka koji ima određene uzorke i oznake.

Unutar petlje, prolazimo kroz svaki uzorak unutar trenutnog intentu i dodajemo pretvoreni uzorak u listu *patterns*. Također, dodajemo oznaku ("tag") trenutnog "intenta" u listu *labels*. Na kraju izvršavanja ovog koda, varijable *patterns* i *labels* će sadržavati sve uzorke i oznake iz učitane JSON datoteke, pripremajući ih za daljnju obradu ili analizu.

```
# Učitavanje podataka iz JSON datoteke
dataset_path = 'dataset.json'
with open(dataset_path) as file:
    dataset = json.load(file)

# Izdvajanje uzoraka i oznaka iz podataka
patterns = []
labels = []
for intent in dataset['intents']:
    for pattern in intent['patterns']:
        patterns.append(pattern.lower()) # Pretvorba u mala slova
        labels.append(intent['tag'])
```

Preprocesiranje podataka

Naredni kod se odnosi na preprocesiranje podataka koji su prethodno učitani iz JSON datoteke. Preprocesiranje uključuje lematizaciju, uklanjanje stop riječi i brojeva.

Prvo, koristimo biblioteku *nltk* (Natural Language Toolkit) za obradu prirodnog jezika. Uvezeni su sljedeći moduli:

- *nltk.corpus.stopwords* - Koristimo stopwords modul za dobivanje skupa stop riječi na engleskom jeziku. Stop riječi su česte riječi poput "the", "is", "and", koje obično ne nose puno značenja i mogu se ukloniti iz teksta tokom obrade.
- *nltk.stem.WordNetLemmatizer* - Koristimo WordNetLemmatizer za lematizaciju riječi. Lematizacija je proces smanjivanja riječi na njihovu osnovnu formu (lemu), poput pretvaranja "running" u "run".
- *nltk.tokenize.word_tokenize* - Koristimo word_tokenize za tokenizaciju teksta, što znači razdvajanje teksta na pojedinačne riječi ili tokene.

- *re* - Koristimo *re* modul (regularni izrazi) za naprednu obradu teksta, poput provjere je li token numerički.

Potom definiramo funkciju *preprocess_text* koja prima tekst kao ulaz i vrši preprocesiranje na njemu. Unutar funkcije, prvo koristimo *word_tokenize* za tokenizaciju teksta i dobivanje liste tokena. Zatim koristimo lematizator *WordNetLemmatizer* kako bismo lematizirali svaki token u listi. To se postiže prolaskom kroz svaki token i primjenom lematizacije na njega. Nakon lematizacije, provjeravamo svaki token u listi i provjeravamo je li token (pretvoren u mala slova) u skupu stop riječi ili je numerički. Ako je token stop riječ ili numerički, izbacujemo ga iz liste tokena. Na kraju, koristimo *join* kako bismo spojili preprocesirane tokene u jedinstveni tekst i vraćamo taj preprocesirani tekst.

Nakon definiranja funkcije *preprocess_text*, primjenjujemo tu funkciju na svaki uzorak u listi *patterns*. Tako će svaki uzorak biti preprocesiran pomoću funkcije *preprocess_text* i rezultat će biti spremljeni u istoimenu listu.

```
# Preprocesiranje podataka: lematizacija, uklanjanje stop riječi i brojeva
import nltk

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re

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

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    # Tokenizacija
    tokens = word_tokenize(text)
    # Lematizacija
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    # Uklanjanje stop riječi i brojeva
    tokens = [token for token in tokens if token.lower() not in stop_words and
not token.isnumeric()]
    # Vraćanje preprocesiranog teksta kao spojene riječi
    return ' '.join(tokens)

patterns = [preprocess_text(pattern) for pattern in patterns]
```

Pretvaranje uzoraka u vektore značajki

Kod u nastavku se odnosi na pretvorbu uzoraka u vektore značajki kako bi se pripremili za obradu. Koristimo *Tokenizer* za pretvaranje tekstualnih podataka u brojne vektore značajki. Pozivamo metodu *fit_on_texts* Tokenizera i proslijeđujemo mu listu uzoraka (*patterns*). Ova metoda prilagođava *Tokenizer* na temelju tekstualnih podataka, stvarajući interni rječnik riječi iz tih podataka. Svaka riječ dobiva jedinstvenu brojčanu oznaku. Nakon toga koristimo metodu *texts_to_sequences* Tokenizera kako bismo pretvorili svaki uzorak iz liste *patterns* u sekvencu brojeva koja predstavlja riječi u tom uzorku. Dobivene sekvence spremljene su u varijablu *sequences*, a zatim računamo maksimalnu dužinu sekvenci koristeći *max* funkciju na listi dužina svih sekvenci. Koristimo funkciju *pad_sequences* kako bismo skratili ili proširili sve sekvence na zajedničku dužinu *max_sequence_length*. Dobivene skraćene sekvence spremljene su u varijablu *padded_sequences*. Potom izračunavamo broj značajki tako da uzmemo dužinu rječnika riječi Tokenizera (*tokenizer.word_index*) i dodamo 1 jer indeksiranje rječnika počinje od 1. Dobiveni broj spremljen je u varijablu *num_features*.

```
# Pretvorba uzoraka u vektore značajki
tokenizer = Tokenizer()
tokenizer.fit_on_texts(patterns)
sequences = tokenizer.texts_to_sequences(patterns)
max_sequence_length = max([len(seq) for seq in sequences])
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length)
num_features = len(tokenizer.word_index) + 1
```

Pretvaranje oznaka u kategorizirane varijable

Ovaj kod se odnosi na pretvorbu oznaka u kategorizirane varijable kako bi se pripremile za obradu. Nakon izvršavanja ovog koda, oznake su pretvorene u numeričke vrijednosti pomoću *LabelEncodera*, što olakšava obradu u modelima mašinskog učenja koji očekuju numeričke ulazne podatke. Broj jedinstvenih oznaka također je određen, što može biti korisno za konfiguraciju izlaznog sloja u modelu.

```
# Pretvorba oznaka u kategorizirane varijable
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(labels)
num_labels = len(label_encoder.classes_)
```

Nastavak se odnosi na podjelu skupa podataka na trening i test skup, izgradnju modela rekurentne neuronske mreže (RNN), kompajliranje modela i treniranje modela. U nastavku je objašnjen svaki korak:

1. Podjela skupa na trening i test

- Koristi se funkcija *train_test_split* iz *scikit-learn* biblioteke kako bi se podaci podijelili na trening i test skup.
- Ulazni podaci *padded_sequences* i ciljne oznake *encoded_labels* se dijele u omjeru 75 - 25 za trening i test skup, koristeći *test_size=0.25*.
- Varijable *X_train*, *X_test*, *y_train* i *y_test* sadrže podatke podijeljene na odgovarajuće skupove.

2. Izgradnja modela RNN

- Kreira se instanca sekvencijalnog modela - *model = Sequential()*.
- Dodaje se embedding layer koji pretvara brojčane vektore značajki u gusto raspoređene vektore fiksne dužine. Parametri ovog sloja su broj jedinstvenih značajki (*num_features*), dimenzionalnost ugrađivanja (128) i dužina ulaznih sekvenci (*max_sequence_length*).
- Dodaje se sloj LSTM koji je vrsta rekurentnog sloja koji se koristi za modeliranje sekvencijalnih podataka. Parametar ovog sloja je broj skrivenih jedinica (128).
- Dodaje se potpuno povezani sloj (dense layer) s brojem izlaznih kategorija (*num_labels*) i aktivacijskom funkcijom softmax koja generiše vjerovatnoće za svaku kategoriju.

3. Kompajliranje modela

- Koristi se metoda *compile* na modelu kako bi se postavile postavke za treniranje. Postavljamo gubitak (loss) na *sparse_categorical_crossentropy* koji je prikladan za višeklasnu klasifikaciju.
- Postavljamo *optimizer* na *adam* koji je popularni optimizacijski algoritam.
- Definišemo metriku *accuracy* za evaluaciju modela.

4. Treniranje modela

- Metoda *fit* se koristi za treniranje modela.
- Ulazni podaci za trening su *X_train* i *y_train*.
- Postavljamo broj epoha na 100, *batch_size* na 32 i dodajemo validacijske podatke (*X_test* i *y_test*) kako bismo pratili performanse modela tokom treninga.

Nakon izvršavanja ovog koda, model RNN je izgrađen, kompajliran i istreniran.

```
# Podjela skupa na trening i test
X_train, X_test, y_train, y_test = train_test_split(padded_sequences,
                                                    encoded_labels, test_size=0.25, random_state=42)
```

```

# Izgradnja modela RNN
model = Sequential()
model.add(Embedding(num_features, 128, input_length=max_sequence_length))
model.add(LSTM(128))
model.add(Dense(num_labels, activation='softmax'))

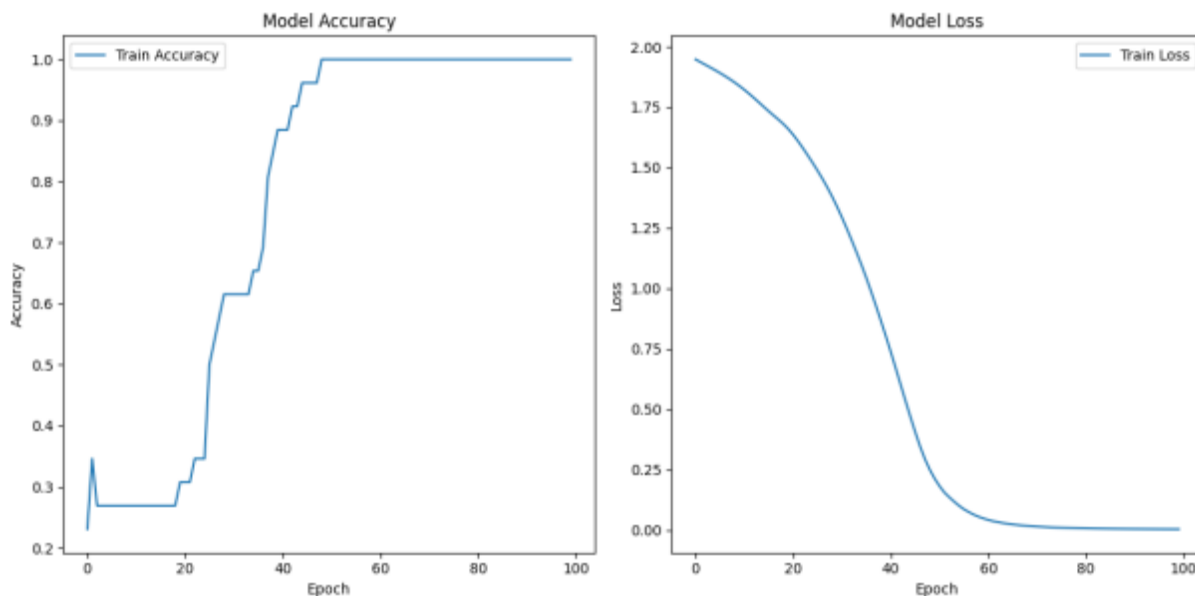
# Kompilacija modela
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Treniranje modela
history = model.fit(X_train, y_train, epochs=100, batch_size=32,
validation_data=(X_test, y_test), verbose=0)

# Evaluacija modela
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)

```

Na slici ispod možemo vidjeti kako je tekao proces treniranja te da se tačnost stalno povećavala, a gubitak smanjivao.



Ilustracija 8 - Rastuća tačnost i opadajući gubitak

Testiranje modela

U narednom koraku je izvršeno testiranje na način da je modelu zadano nekoliko patterna za koje je bilo potrebno utvrditi labelu i dati adekvatan odgovor.

```
# Testiranje modela
test_patterns = [
    "Hello",
    "Thank you",
    "How can I place an order?",
    "What payment methods do you accept?",
    "I want to return an item, how to do it?"
]

# Pretvorba testnih uzoraka u vektore značajki
test_sequences = tokenizer.texts_to_sequences(test_patterns)
padded_test_sequences = pad_sequences(test_sequences,
maxlen=max_sequence_length)

# Predviđanje oznaka za testne uzorke
predictions = model.predict(padded_test_sequences, verbose=0)

# Dekodiranje predviđenih oznaka
decoded_labels = label_encoder.inverse_transform(np.argmax(predictions,
axis=1))

# Ispisivanje rezultata predviđanja
for pattern, label in zip(test_patterns, decoded_labels):
    print(f"Input: {pattern}")
    print(f"Predicted label: {label}")
    print("Response:", end=" ")
    for intent in dataset['intents']:
        if intent['tag'] == label:
            print(np.random.choice(intent['responses']))
            break
    print()
```

```
Input: Hello
Predicted label: greeting
Response: Good [morning/afternoon/evening]! How may I help you?

Input: Thank you
Predicted label: thanks
Response: Glad to help!

Input: How can I place an order?
Predicted label: order
Response: To place an order, please visit our website and select the items you wish to purchase. The stand

Input: What payment methods do you accept?
Predicted label: payment
Response: We accept credit cards, debit cards, PayPal, and Apple Pay. Our payment process is secure and al

Input: I want to return an item, how to do it?
Predicted label: returns
Response: For returns, please initiate the process within 14 days and follow these instructions: pack the
```

Ilustracija 9 - Korištenja chatbota

Evaluacija algoritma

Nakon toga, izvršena je evaluacija algoritma. Pokrenuta je predikcija modela, a nakon dekodiranja predviđenih oznaka, izračunate su mjere F1, recall i preciznost.

```
from sklearn.metrics import classification_report

# Predviđanje vjerojatnosti za testni skup
y_pred_prob = model.predict(X_test)

# Pretvaranje vjerojatnosti u predviđene oznake
y_pred = np.argmax(y_pred_prob, axis=1)

# Dekodiranje predviđenih oznaka
predicted_labels = label_encoder.inverse_transform(y_pred)

# Izračunavanje F1, recall i preciznosti
report = classification_report(label_encoder.inverse_transform(y_test),
                               predicted_labels)

# Ispis rezultata
print("Classification Report:\n", report)
```

Classification Report:				
	precision	recall	f1-score	support
goodbye	1.00	0.50	0.67	2
greeting	0.00	0.00	0.00	0
order	0.50	1.00	0.67	2
payment	0.00	0.00	0.00	2
returns	1.00	1.00	1.00	2
thanks	1.00	1.00	1.00	1
accuracy			0.67	9
macro avg	0.58	0.58	0.56	9
weighted avg	0.67	0.67	0.63	9

Ilustracija 10 - Rezultat evaluacijskih metrika

Iz izvještaja klasifikacije možemo vidjeti sljedeće rezultate:

- Za klasu "goodbye" imamo preciznost od 1.00, recall od 0.50 i F1-score od 0.67. To znači da je model dobro prepoznao pozitivne primjere, ali je propustio neke negativne primjere.
- Za klasu "greeting" nemamo podatke o preciznosti, recallu i F1-scoreu jer ne postoje oznake za tu klasu u testnom skupu.
- Za klasu "order" imamo preciznost od 0.50, recall od 1.00 i F1-score od 0.67. Model je prepoznao sve pozitivne primjere, ali je imao nešto lošiju preciznost.
- Za klasu "payment" nemamo podatke o preciznosti, recallu i F1-scoreu jer ne postoje oznake za tu klasu u testnom skupu.
- Za klasu "returns" imamo preciznost, recall i F1-score od 1.00. Model je savršeno prepoznao sve primjere te klase • Za klasu "thanks" imamo preciznost, recall i F1-score od 1.00. Model je savršeno prepoznao primjer te klase.

Ukupna tačnost modela na testnom skupu iznosi 67%.

Važno je napomenuti da preciznost, recall i F1-score zavise od broja primjera u testnom skupu. U ovom slučaju, neke klase imaju vrlo mali broj primjera ili nemaju oznake u testnom skupu, što utječe na izračunate vrijednosti.

Kreiranje chatbota

Konačno, implementiran je i kod koji pokreće chatbot sa kojim korisnik može razgovarati.

```
# Funkcija za predviđanje odgovora na temelju unesenog teksta
```

```

def get_response(text):
    preprocessed_text = preprocess_text(text)
    sequence = tokenizer.texts_to_sequences([preprocessed_text])
    padded_sequence = pad_sequences(sequence, maxlen=max_sequence_length)
    predicted_label = np.argmax(model.predict(padded_sequence), axis=-1)
    tag = label_encoder.inverse_transform(predicted_label)[0]

    for intent in dataset['intents']:
        if intent['tag'] == tag:
            responses = intent['responses']
            response = np.random.choice(responses)
            return response

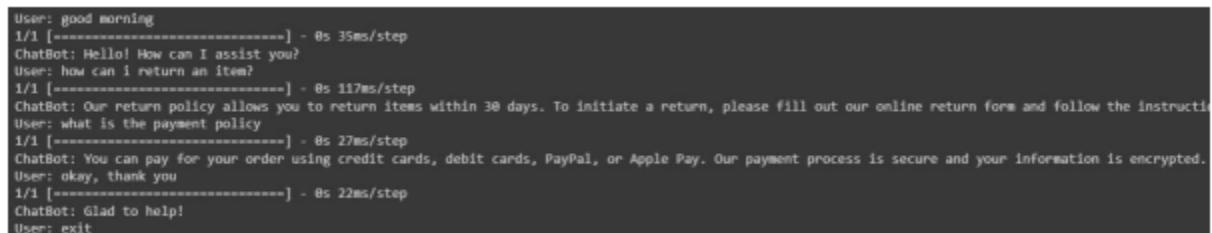
    return "I'm sorry, I didn't understand that."

# Primjer korištenja chatbota
while True:
    user_input = input("User: ")
    if user_input.lower() == "exit":
        break

    response = get_response(user_input)
    print("ChatBot:", response)

```

Na slici ispod možemo vidjeti da je chatbot dao prilično razumne odgovore.



```

User: good morning
1/1 [=====] - @s 35ms/step
ChatBot: Hello! How can I assist you?
User: how can i return an item?
1/1 [=====] - @s 117ms/step
ChatBot: Our return policy allows you to return items within 30 days. To initiate a return, please fill out our online return form and follow the instructions.
User: what is the payment policy
1/1 [=====] - @s 27ms/step
ChatBot: You can pay for your order using credit cards, debit cards, PayPal, or Apple Pay. Our payment process is secure and your information is encrypted.
User: okay, thank you
1/1 [=====] - @s 22ms/step
ChatBot: Glad to help!
User: exit

```

Ilustracija 11 - Upotreba chatbota

Konvolucijska neuronska mreža (CNN) (Rudalija Ema)

Konvolucijska neuronska mreža (CNN) je vrsta neuronske mreže koja je posebno efikasna u obradi i analizi struktuiranih podataka poput slika. Dok su tradicionalne neuronske mreže pune povezanosti prikladnije za obradu sekvencijalnih podataka, poput teksta, CNN-ovi su posebno dizajnirani za ekstrakciju značajki iz prostorno raspoređenih podataka.

Glavna ideja iza CNN-a je korištenje konvolucija za detekciju lokalnih značajki u ulaznim podacima. Konvolucija je matematička operacija koja ima ulaznu matricu (npr. slika) i filter (poznat i kao jezgra ili kernel) te generiše izlaznu matricu koja predstavlja detektovane značajke. Ova svojstva konvolucija čine CNN-ove prikladnim za obradu vizualnih podataka poput slika.

Kada se koristi za implementaciju chatbota, CNN se može koristiti za obradu teksta, posebno u fazi razumijevanja prirodnog jezika. Umjesto izravne primjene konvolucija na slike, primijenjujemo konvolucije na tekstualne ulazne podatke, poput uzoraka korisničkog unosa. Konvolucije u ovom slučaju prepoznaju lokalne sekvencijalne obrasce u tekstu.

Korisnost CNN-a za implementaciju chatbota leži u njegovoj sposobnosti da nauči relevantne značajke iz teksta, poput ključnih riječi ili fraza, što omogućava chatbotu da razumije korisničke upite ili poruke. CNN može izdvojiti bitne informacije iz teksta i prepoznati obrasce koji odgovaraju različitim klasama ili oznakama u skupu podataka. Osim toga, CNN se često kombinira s drugim slojevima neuronske mreže, poput pooling slojeva i potpuno povezanih slojeva, kako bi se postigla veća efikasnost i bolji rezultati.

Učitavanje i preprocesiranje podataka

U nastavku je data implementacija CNN-a koja koristi naš dataset. Nakon importovanja potrebnih biblioteka, izvršeno je učitavanje podataka.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D,
Dense, Dropout
import nltk
from tensorflow.keras.optimizers import Adam
import json
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import numpy as np

# Učitavanje podataka
with open("dataset.json") as file:
    data = json.load(file)

# Preprocesiranje podataka: lematizacija, uklanjanje stop riječi i brojeva
nltk.download('punkt')
```

```

nltk.download('wordnet')
nltk.download('stopwords')

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

```

Definisana je funkcija *preprocess_text* koja obavlja preprocesiranje teksta - tokenizaciju, lematizaciju, uklanjanje stop riječi i brojeva te spajanje preprocesiranog teksta. Ova funkcija se primjenjuje na rečenice koje se nalaze u varijabli *patterns*, dok varijabla *labels* samo čuva podatke o tome iz kojeg domena je pitanje postavljeno.

```

def preprocess_text(text):
    # Tokenizacija
    tokens = word_tokenize(text)
    # Lematizacija
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    # Uklanjanje stop riječi i brojeva
    tokens = [token for token in tokens if token.lower() not in stop_words and
not token.isnumeric()]
    # Vraćanje preprocesiranog teksta kao spojene riječi
    return ' '.join(tokens)

patterns = []
labels = []

for intent in data['intents']:
    for pattern in intent['patterns']:
        patterns.append(pattern.lower())
        labels.append(intent['tag'])

patterns = [preprocess_text(pattern) for pattern in patterns]

```

Podaci su podijeljeni na trening i testniskup tako da je 20% podataka ostavljeno za testiranje i potrebe evaluacije modela. Potom je izvršena tokenizacija i podešavanje dužine sekvenci, kao i pretvaranje oznaka u kategorizirane varijable.

```

# Podjela na trening i testni skup
train_patterns, test_patterns, train_labels, test_labels =
train_test_split(patterns, labels, test_size=0.2)

# Tokenizacija i pretvorba uzoraka u sekvence
tokenizer = tf.keras.preprocessing.text.Tokenizer()

```

```

tokenizer.fit_on_texts(train_patterns)
train_sequences = tokenizer.texts_to_sequences(train_patterns)
test_sequences = tokenizer.texts_to_sequences(test_patterns)

# Podešavanje maksimalne duljine sekvence
max_length = max(len(seq) for seq in train_sequences)
vocab_size = len(tokenizer.word_index) + 1

# Povećavanje duljine sekvence
train_sequences = pad_sequences(train_sequences, maxlen=max_length,
padding='post')
test_sequences = pad_sequences(test_sequences, maxlen=max_length,
padding='post')

# Pretvorba oznaka u kategorizirane varijable
label_encoder = LabelEncoder()
train_encoded_labels = label_encoder.fit_transform(train_labels)
test_encoded_labels = label_encoder.transform(test_labels)
num_labels = len(label_encoder.classes_)

```

Izgradnja CNN modela

Naredni kod prikazuje izgradnju i treniranje konvolucijske neuronske mreže (CNN). Ispod su navedeni koraci treniranja:

- *model - Sequential()* - Stvara se instanca sekvencijalnog modela, koji je osnovna struktura za slojeve u TensorFlow Kerasu.
- *model.add(Embedding(vocab_size, 100, input_length=max_length))* - Dodaje se Embedding layer koji pretvara riječi u vektore značajki. *vocab_size* je ukupan broj jedinstvenih riječi u korpusu, 100 je dimenzionalnost ugrađivanja, a *input_length* je dužina ulaznih sekvenci.
- *model.add(Conv1D(128, 5, activation='relu'))* - Dodaje se sloj konvolucije (Conv1D) s 128 filtera veličine 5 i aktivacijom ReLU. Ovaj sloj obavlja konvoluciju na ulaznim sekvencama kako bi prepoznao lokalne obrasce.
- *model.add(GlobalMaxPooling1D())* - Dodaje se pooling sloj koji izdvaja najveću vrijednost iz svake konvolucijske mape. Ovaj sloj smanjuje dimenzionalnost izlaza i izvlači ključne značajke iz konvolucija.

- `model.add(Dense(64, activation='relu'))` - Dodaje se potpuno povezani sloj s 64 neurona i aktivacijom ReLU. Ovaj sloj služi za učenje složenijih značajki iz izlaza grupisanja.
- `model.add(Dropout(0.5))` - Dodaje se sloj isključivanja (dropout) s stopom isključivanja od 0.5. Ovaj sloj služi za sprečavanje overfittinga modela tako da nasumično isključuje neurone tokom treninga.
- `model.add(Dense(num_labels, activation='softmax'))` - Dodaje se potpuno povezani sloj s brojem neurona jednakim broju klasa u problemu klasifikacije. Aktivacija softmax se koristi kako bi se dobio vektor vjerovatnoće za svaku klasu.

Kompajliranje i treniranje modela

Koraci koji opisuju ovaj dio su:

- `learning_rate` - $2e-4$ - Postavlja se stopa učenja (learning rate) na vrijednost $2e-4$.
- `optimizer` - `Adam(learning_rate=learning_rate)` - Kreira se Adam optimizator s ranije definisanom stopom učenja.
- `model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])` - Model se kompajlira definisanjem gubitka kao sparse kategoričke unakrsne entropije, optimizatora i metrike mjerenja tačnosti.
- `model.fit(train_sequences, train_encoded_labels, epochs=300, batch_size=128)` - Model se trenira na trening skupu podataka (`train_sequences` i `train_encoded_labels`) tokom 300 epoha s veličinom batcha 128.

```
# Izgradnja CNN modela
model = Sequential()
model.add(Embedding(vocab_size, 100, input_length=max_length))
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_labels, activation='softmax'))

# Kompilacija i treniranje modela s prilagođenom stopom učenja
learning_rate = 2e-4
optimizer = Adam(learning_rate=learning_rate)
```



```

model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])

model.fit(train_sequences, train_encoded_labels, epochs=300, batch_size=128)

# Evaluacija modela na testnom skupu
test_loss, test_accuracy = model.evaluate(test_sequences, test_encoded_labels)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

```

Ovaj model je imao tačnost od 57%.

Što se tiče ostalih evaluacijskih metrika i one su nešto lošije u poređenju sa prethodnim algoritmima:

- Preciznost = 0.48
- Recall = 0.57
- F1 = 0.5

Pokretanje chatbota

Na kraju, implementiran je kod koji pokreće chatbot i omogućava analizu dobijenih odgovora.

```

import random

# Korišćenje modela za generisanje odgovora
def generate_response(input_text, model, tokenizer):
    input_sequence = tokenizer.texts_to_sequences([input_text.lower()])
    input_sequence = pad_sequences(input_sequence, maxlen=max_length,
padding='post')
    logits = model.predict(input_sequence)[0]
    predicted_class = np.argmax(logits)
    tag = label_encoder.inverse_transform([predicted_class])[0]

    # Pronalaženje odgovarajućeg skupa na temelju taga
    for intent in data['intents']:
        if intent['tag'] == tag:
            responses = intent['responses']
            break

    # Odabir slučajnog odgovora iz skupa
    response = random.choice(responses)

```

```
return response
```

```
while True:
    user_input = input("User: ")
    if user_input.lower() == "exit":
        break

    response = generate_response(user_input,model,tokenizer)
    print("ChatBot:", response)
```

```
User: Hello
1/1 [-----] - 0s 33ms/step
ChatBot: Hello, how may I assist you today?
User: Help me
1/1 [-----] - 0s 23ms/step
ChatBot: If you have any questions or need help with anything, our customer support team is here for you.
User: How can I place an order
1/1 [-----] - 0s 28ms/step
ChatBot: If you need assistance with placing an order, we're happy to help. You can reach out to us through
User: Thank you
1/1 [-----] - 0s 26ms/step
ChatBot: No problem!
User: Bye
1/1 [-----] - 0s 45ms/step
ChatBot: Goodbye, have a great day!
User: exit
```

Ilustracija 12 - Upotreba chatbota

Zaključak

Na osnovu izvršenih eksperimenata na četiri različita algoritma za rješavanje problema chatbota, dobili smo rezultate koji ukazuju na različitu uspješnost algoritama. Najbolje rezultate pokazao je BERT Transformer model, zatim Random Forest, slijedi RNN, dok je CNN imao najmanju tačnost.

BERT Transformer model je bio najuspješniji algoritam za chatbot, što se može objasniti njegovom sposobnošću da uhvati složene i kontekstualne značajke u tekstu. BERT koristi Transformer arhitekturu koja je pokazala veliku moć u obradi prirodnog jezika. Transformer arhitektura omogućava BERT-u da modelira dugoročne zavisnosti u tekstu, hvata kontekst i razumije suptilnosti značenja riječi. Random Forest, iako tradicionalniji algoritam, pokazao se kao drugi najbolji izbor za chatbot.

Random Forest je ensemble algoritam koji kombinuje više stabala odlučivanja. Ova kombinacija donosi raznovrsnost u modelu i omogućava mu da nauči kompleksne obrasce u podacima. Random Forest se dobro nosi s tekstualnim podacima i može uhvatiti važne karakteristike koje pomažu u klasifikaciji odgovarajućih oznaka.

RNN (Recurrent Neural Network) je također pokazao solidne rezultate, ali nešto slabije od BERT i Random Forest. RNN je posebno dobar za obradu sekvencijalnih podataka poput teksta, jer ima sposobnost pamćenja prethodnih informacija. Međutim, RNN ima izazove u modeliranju dugoročnih zavisnosti i može imati problema sa "nestajućim gradijentom" tokom treniranja.

Napokon, CNN (Convolutional Neural Network) je pokazao najmanju tačnost među četiri algoritma. Iako CNN ima snažne sposobnosti u obradi slika, za obradu teksta se ne pokazuje tako efikasnim. CNN se fokusira na lokalne značajke u podacima i može propustiti šire kontekstualne informacije koje su bitne za chatbot.

Na osnovu ovih rezultata, zaključujemo da BERT Transformer model ima prednost u rješavanju problema chatbota zbog svoje sposobnosti modeliranja konteksta i razumijevanja suptilnosti teksta. Random Forest također pruža dobre rezultate zbog svoje sposobnosti učenja kompleksnih obrazaca. RNN je solidan izbor, ali može imati izazove s dugoročnim zavisnostima. Konačno, CNN se pokazao manje uspješnim za ovaj problem, vjerojatno zbog svoje fokusiranosti na lokalne značajke umjesto na kontekstualno razumijevanje teksta.