Projekat iz predmeta: Računarska Inteligencija

Generisanje epizoda serije na osnovu postojećih

Luka Farkaš SV63/2021
Boris Markov SV73/2021

Sadržaj

Uvod i struktura programa

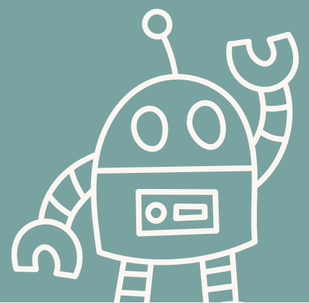
- Ideja ovog projekta je da se generiše skripta epizode na osnovu tekstualnih podataka (sve emitovane epizode).

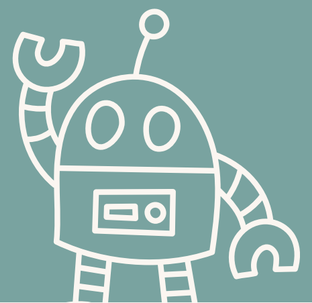
- Koristimo LSTM neuronsku mrežu za realizovanje ovog zadatka.

- Program ima mogućnosti treniranje neuronske mreže na osnovu izabranih podataka, sačuva modele, da ih učita i generiše novu skriptu.

- **Korišćeni moduli:**
  - ‘os’ i ‘glob’ za operacija nad fajlovima
  - ‘torch’ za operacije neuronske mreže
  - ‘sklearn’ za razdvajanje podataka
  - ‘collections’ za kreiranje rečnika
  - CUDA za ubrzanje procesa treniranja korišćenjem GPU-a

- **Klase**:
  - ‘TextDataset’ posebna klasa za baratanje sa sekvencama teksta
  - ‘LSTMModel’ definiše LSTM neuronsku mrežu sa embeddingom, LSTMom i u potpunosti povezanim slojevima



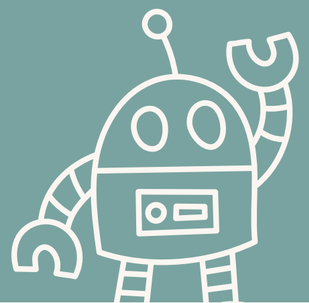


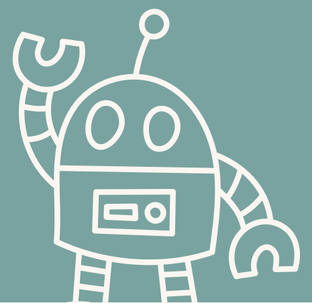
- **Baratanje modelom:**
  - save_model(model) sačuvava model u vidu fajla
  - load_model(vocab_size, embedding_dim, hidden_dim, num_layers) učitava model sa fajl sistema

- **Generisanje teksta:**
  - generate_text(model, start_text, vocab, idx_to_word) generiše tekst na osnovu istreniranog modela
  - clean_generated_text(text) formatira generisan tekst

- **Treniranje i evaluacija:**
  - train(model, dataloader, criterion, optimizer, epoch, vocab_size) vrši treniranje nad modelom za jednu epohu
  - evaluate(model, dataloader, criterion, vocab_size) evaluira model na osnovu validacionih ili test podataka

- **Ostale funkcije:**
  - load_scripts(path) učitava sve tekstualne fajlove iz označenog direktorijuma
  - tokenize(text) tokenizira tekst u reči
  - build_vocab(tokenized_texts) pravi rečnik od tokena teksta
  - text_to_sequences(tokenized_texts, vocab) pretvara tokenizovane tekstove i sekvence na osnovu rečnika
  - create_sequences(tokenized_texts, sequence_length=50) kreira sekvence određene dužine za treniranje
  - collate_fn(batch) dodaje padding za sekvence

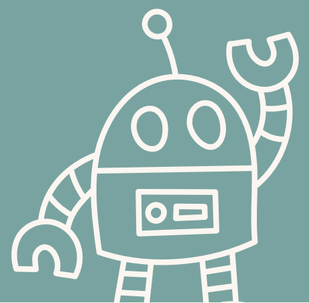


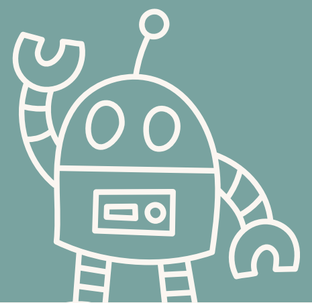


Izvršavanje programa

- Tok programa se odvija na sledeći način:

1. Učitavanje i tokenizovanje tekstualnih podataka
2. Pravljenje rečnika
3. Inicijalizacija uređaja za komputaciju (CPU ili GPU)
4. Izvršavanje jedne od opcija:
    a. Treniranje novog modela i čuvanje u file sistem
    b. Učitavanje istreniranog modela sa file sistema
    c. Generisanje skripte na osnovu učitanog modela uz biranje dužine skripte, i njegovo čuvanje u zaseban .txt fajl
    d. Izlaz iz programa



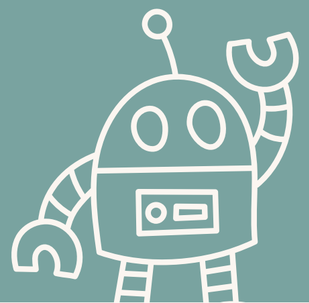


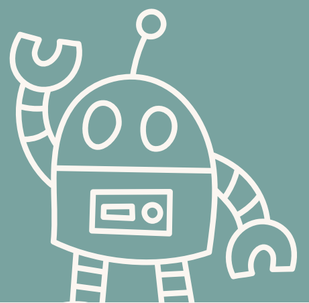
Najbitnije metode: train()

- train(model, dataloader, criterion, optimizer, epoch, vocab_size)
  - model - LSTM model koji će se trenirati
  - dataloader - DataLoader objekat koji sadrži podatke za treniranje
  - criterion - funkcija gubitka
  - optimizer - algoritam optimizacije (u našem slučaju Adam)
  - epoch - trenutna epoha
  - vocab_size - veličina rečnika

- **Koraci:**

1. Postavljanje modela u trening mod
2. Inicijalizacija sakrivenih vrednosti
3. Iteracija kroz segmente podataka:
   a. premeštanje podataka na uređaj
   b. otkačivanje sakrivenih stanja da bi se sprečilo ‘backpropagation’ kroz celu istoriju učenja
   c. pomeranje napred i komputacija podataka
   d. komputacija gubitka
   e. ‘backpropagation’
   f. ažuriranje parametara modela uz pomoć algoritma optimizacije
4. Ispis napredovanja kroz učenje svakih 100 batch-ova
5. Ispis gubitka epohe na kraju



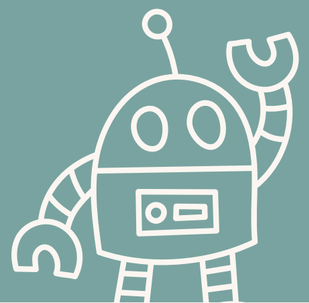


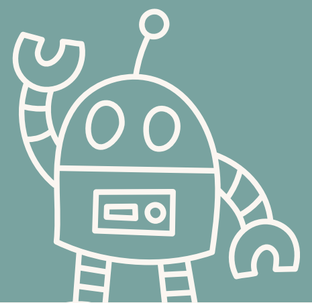
Najbitnije metode: generate_text()

- generate_text(model, start_text, vocab, idx_to_word)
  - model - istreniran LSTM model
  - start_text - tekst koji se uvek ispisuje na početku
  - vocab - rečnik koji mapira reči na indekse
  - idx_to_word - rečnik koji mapira indekse na reči

- **Koraci:**

1. Dobijanje korisničkog unosa za parametar temperature i dužine teksta
2. Postavljanje modela na evaluacioni mod
3. Tokeniziranje početnog teksta i pretvaranje u indekse
4. Inicijalizacija sakrivenih stanja i priprema tensora unosa
5. Iterativno generisanje sledećih reči:
   a. kretanje napred da bi smo dobili šanse output-a
   b. primenjivanje parametra temperature
   c. uzimanje indeksa sledeće reči kroz distribuciju šanse
   d. apendiranje reči na tekst
   e. priprema inputa za sledeću iteraciju
6. Čišćenje generisanog teksta
7. Vraćanje generisanog teksta



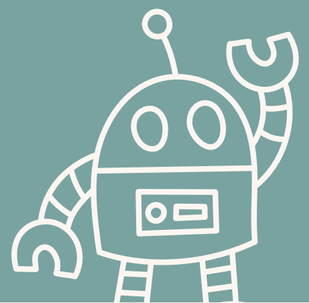


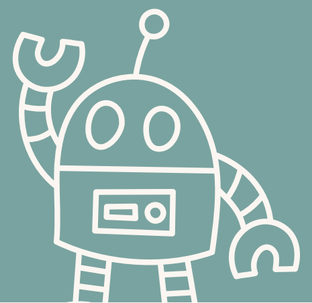
Najbitnije metode: evaluate()

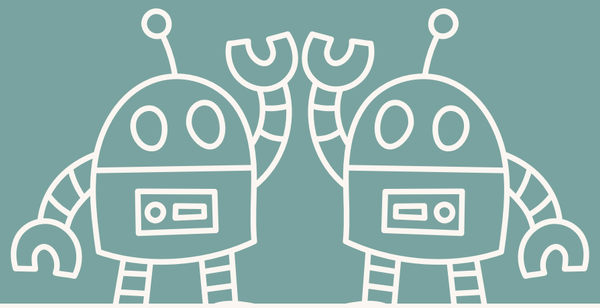
- evaluate(model, dataloader, criterion, vocab_size)
  - model - LSTM model koji će se evaluirati
  - dataloader - DataLoader objekat koji sadrži podatke za treniranje ili validacione
  - criterion - funkcija gubitka
  - vocab_size - veličina rečnika

- **Koraci:**

1. Postavljanje modela na evaluacioni mod
2. Inicijalizacija sakrivenih stanja
3. Poništavanje gradijenta komputacije
4. Iteriranje kroz segmente podataka
   a. pomeranje unosa i targeta na uređaj
   b. kretanje napred i komputiranje izlaza
   c. komputiranje gubitka
   d. akumuliranje gubitka
   e. računanje broja tačnih predikcija
5. Komputiranje preciznosti
6. Vraćanje srednjeg gubitka i preciznosti

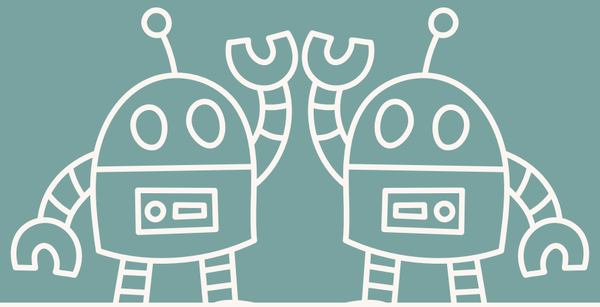






Primer generisanog teksta

Rachel: I don't know... right, he's the pig!
Phoebe: Such a pig!
Rachel: Oh, God, he's such a pig.
Phoebe: Oh yeah, uh, (snapping her fingers at Ross who takes the remote to the OB/GYN...)
Rachel: Oh, we've got a head, we got shoulders, we got arms, we got, oh, look at the little fingers, oh, and a chest, and a stomach. It's a boy, definitely a boy! All right! Ok, legs, knees, and feet. Oh, oh. He's here. He's a person.
Susan: Oh, look at that.
Carol: What does she mean?
Phoebe: I think of it as my uncle Ed, covered in Jell-O.
Carol: Really?
Phoebe: (from the air vent overhead) You guys, he's beautiful!
Ross: Oh, thanks, Pheebs! (They look up towards the vent and wave at Phoebe.)



Evaluacija rezultata

Algoritam evidentno generiše tekst u obliku scenarija koji koriste glumci prilikom snimanja serije.

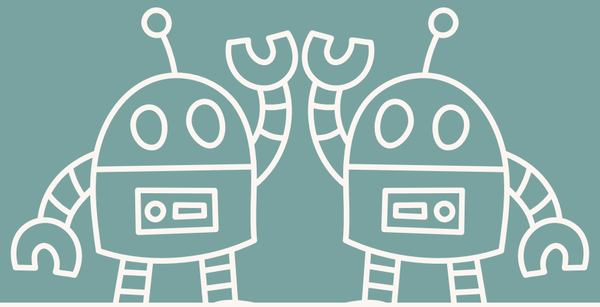
Likovi iz serije su pristuni, takođe se povremeno generišu likovi koji nisu glavni likovi što je dobar pokazatelj da algoritam razume strukturu jedne epizode serije “Friends”.

Model zna da razlikuje radnje (u zagradama) od samog dijaloga.

Gramatički je u glavnom korektan, međutim slabo se vezuje za jedan konkretan dijalog i često pravi digresije što dovodi do toga da se u jednoj sceni dešava mnogo različitih radnji.

Poslednja stavka se može fino korigovati kroz veći opseg treniranja i vrednosti ‘temperature’ parametra, kako bi skripta bila verodostojna pravoj.

Posledica ovakvo generisanog teksta je svakako humoristična nota koja pruža određenu dozu zabave čitaocima, što je i jednim delom bio i cilj.



Evaluacija programa

Bilo je izuzetno izazovno otkloniti greške prilikom razvijanja programa, jer proces treniranja zahteva izvesno vreme (čak i kod korišćenja GPU za treniranje).

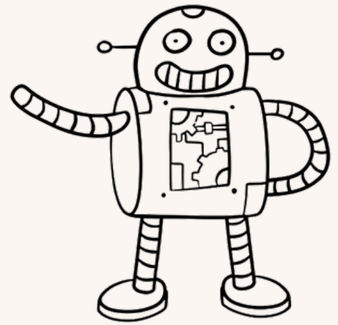
Za treniranje mreže sa 240000 ulaznih podataka (najveći ulaz sa kojim smo testirali), potrebno je oko sat vremena na računaru sa:

- 32GB RAM
- AMD Ryzen 5 1500X Quad-Core Processor 3.50 GHz
- NVIDIA GeForce GTX 1660 SUPER

Kao optimizaciju pri treniranju smo koristili platformu CUDA za ubrzavanje procesa učenja, alokacijom grafičkih procesorskih jedinica za obradu podataka.

Generisan tekst se ispisuje na konzolu i čuva u zaseban fajl “script_{id}.txt”.

Prostor za razvijanje



Kvalitet skripte:

1. Poboljšanja logika interpretacije generisanog teksta
2. Veći opseg treniranja

Funkcionalnosti:

1. Omogućavanje korisniku da podešava sve parametre, uz default vrednosti
2. Generisanje teksta za specifičnog karaktera
3. Mogućnost treniranja na udaljenim serverima

Interfejs

1. Razvijanje u obliku Web aplikacije
2. Više feedback-a korisniku šta se trenutno dešava u programu

Ostalo

1. Dodavanje skripti drugih serija za treniranje
2. Proširivanje funkcionalnosti da korisnik može da čuje generisane skripte prođe uz pomoć ‘text-to-speech’
3. Deljenja koda na module radi boljeg snalaženja

Klasifikacija teksta prema kategoriji - Naive Bayes

Sadržaj

Uvod i struktura programa

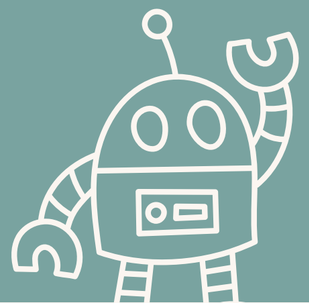
- Prilikom pisanja specifikacije zbog nedovoljne upućenosti nismo pred predvideli da nije moguce ovim algoritmom uraditi predikciju nove epizode, te smo se odlučili za klasifikaciju po ulozi.

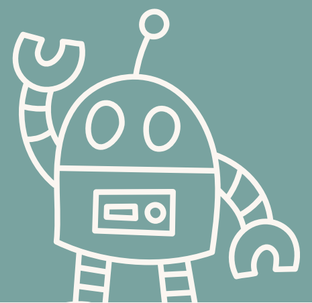
- Medjutim ustanovili smo da su tekstovi koji role imaju nedovoljno duački pa tačnost algoritma ne može da pređe 20% čak ni sa ugrađenim algoritmom iz sklearn biblioteke. Te smo iskoristili neki složeniji dataset

.

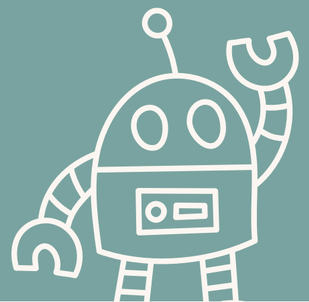
- **Korišćeni moduli:**
  - ‘os’ i ‘glob’ za operacija nad fajlovima
  - napravili smo custom NaiveBayes klasu
  - ‘sklearn’ za ugrađen NaiveBayes
  - numpy biblioteka za množenje verovatnoća
  - stopwords i za uklanjanje znakova interpukcije

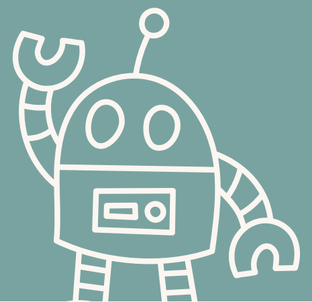
- **Klase**:
  - ‘MultinomialNB’ iz biblioteke sklearn
  - ‘NBTextClassifier’ custom made klasa





- Najpre se podaci učitaju iz csv-a preko pandas biloteke, potom se napravi data frame sa nazivima kolona category i text (kategorija je tip proizvoda dok je tekst opis tog proizvoda). Potom napravimo trening skup od 80% seta podataka i testni od preostalih 20%.
- Potom se definise NBTextClassifier custom made i poziva se metoda fit gde se pripremaju podaci neophodni za predikciju (računaju se verovatnoće prisutnosti u nekoj od kategorija, ukupan broj reči...)
- Potom se u funkciji za predikciju računaju verovatnoće pripadnosti svake reči u određenoj kategoriji i onda se sve te verovatnoće pomnože.
- U istoj funckji se za dobijanje konačnih verovatnoća prethodni rezultat se pomnoži sa prior verovatnoćama koje definišu (ukupan broj reci u nekoj kategoriji) / (ukupan broj reci na nivo svih kategorija)
- Potom se poziva provera tačnosti za test skup sa izračunatim predikcijama i poredimo rezultate.



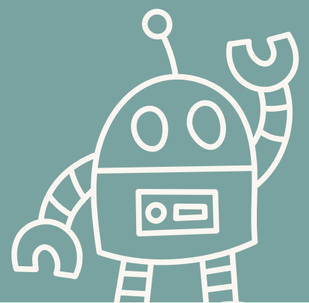


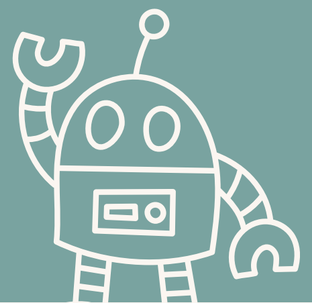
```
Classification Report:
                        precision    recall  f1-score   support

                 Books       0.98      0.52      0.68      2320
Clothing & Accessories       0.95      0.79      0.86      1784
           Electronics       0.94      0.47      0.62      2110
             Household       0.59      0.97      0.74      3758

              accuracy                           0.73      9972
             macro avg       0.86      0.69      0.73      9972
          weighted avg       0.82      0.73      0.72      9972


Accuracy: 0.73
Bayes as python library
Accuracy for NaiveBayes Model on Test Data is: 0.9557761732851986
**********************************************************
```





Pozitivne strane:

- Visoka preciznost za Books, Clothing & Accessories, i Electronics kategorije.
- Visok odziv za Household, što znači da model prepoznaje skoro sve instance ove kategorije.

Negativne strane:

- Nizak odziv za Books i Electronics, što znači da model propušta mnoge stvarne instance ovih kategorija.
- Niska preciznost za Household, što ukazuje na mnogo pogrešnih pozitivnih prognoza.

Balansiranje podataka je potencijalno rešenje za poboljšanje:

- Povećanje broja instanci za manje zastupljene kategorije može poboljšati performanse.

Klasfikacija teksta prema kategoriji - KNN

Sadržaj

- Uvod i struktura programa
- Izvršavanje programa
- Najbitnije metode:
  - *get_n_grams_frequencies()*
  - *pretprocessing()*
  - *calculate_distance()*
  - *predict*
- *Primer ugrađenog algoritma iz sklearn biblioteke*
- *Primer napisanog algoritma od nule*
- *Pregled classification_report iz sklearn.matrix biblioteke*
- *Analiza izveštaja*

Uvod i struktura programa

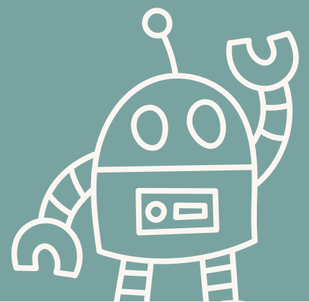
- Prilikom pisanja specifikacije zbog nedovoljne upućenosti nismo pred predvideli da nije moguce ovim algoritmom uraditi predikciju nove epizode, te smo se odlučili za klasifikaciju po ulozi.

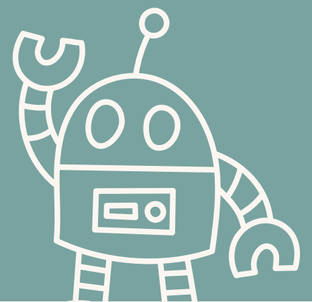
- Medjutim ustanovili smo da su tekstovi koji role imaju nedovoljno duački pa tačnost algoritma ne može da pređe 20% čak ni sa ugrađenim algoritmom iz sklearn biblioteke. Te smo iskoristili neki složeniji dataset

.

- **Korišćeni moduli:**
  - ‘os’ i ‘glob’ za operacija nad fajlovima
  - napravili smo custom KNN klasu
  - ‘sklearn’ za ugrađen KNN
  - vektorizacija texta za ugradjeni KNN
  - stopwords i za uklanjanje znakova interpukcije

- **Klase**:
  - ‘KNeighborsClassifie’ iz biblioteke sklearn
  - ‘KNNClassifier’ custom made klasa



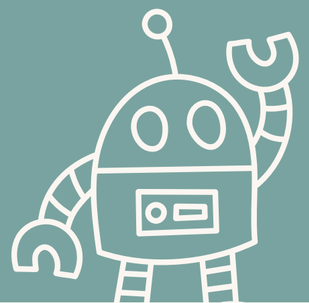


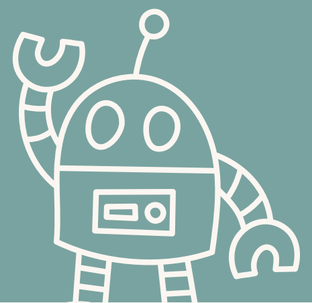
- Kao kao i kod prethodnog algoritma ustanovljen je isti problem, te se koritsi drugi set podataka. Takodje se učitaju podaci preko pandas biblioteke i formira se data frame sa istim kolonama.
- Potom se razdeli podaci na skupove za trening i testiranje. Potom se podaci za trening preprocesiraju koristeći metodu sa n_gram rečima.
- n_gram za zadato n iz teksta uzima broj karaktera od početka do kraja teksta za n koraka pomerajući se za jedan karakter od početka. (ako je npr n jednako sa 3 i imamo reč petar, n_grams će biti redom: pet, eta, tar...)
- potom se računaju frekvencije u odnosu na čitav broj reči unutar svih kategorija i posebno pojavljivanje tih reči u konkretnoj kategoriji, na sličan način se preprocesiraju i test podaci

- Na kraju se poziva predikcija gde se računaju distance tih frekvencija i uzima se prvih K koje mi zadamo (najbolje je da bude neparan broj) i proveravamo kom komšiji je najbliži te će njegovoj grupi pripasti.

Classification Report:

	precision	recall	f1-score	support
Books	0.89	0.61	0.72	2320
Clothing & Accessories	0.56	0.04	0.08	1784
Electronics	0.58	0.07	0.13	2110
Household	0.46	0.97	0.62	3758
accuracy			0.53	9972
macro avg	0.62	0.42	0.39	9972
weighted avg	0.60	0.53	0.44	9972





Hvala Vam na pažnji!