UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA
LABORATORIJA ZA VEŠTAČKU INTELIGENCIJU

DOKUMENTACIJA

RAZVOJ CHATBOT-A KORIŠĆENJEM PYTORCH RAZVOJNOG OKRUŽENJA I NLTK ALATA

Profesor:
Zoran Ševarac

Student:
Dejan Milovanović 3714/20

Beograd, 2021.

Sadržaj

1. Uvod

U ovoj dokumentaciji biće opisane tehnike i alati korišćeni za razvoj chatbot aplikacije sa korisničkim interfejsom u svrhe predmeta Primena veštačke inteligencije u okviru master akademskih studija fakulteta organizacionih nauka, modul Softversko inženjerstvo i računarske nauke. Opšte poznato je da veštačka inteligencija sve više zastupljena u našem svakodnevnom životu. U ovom radu akcenat će biti na *obradi prirodnog jezika (eng. Natural Language Processing - NLP).*

2. Tehnologije

2.1. Python

Predstavlja jedan od najpopularnijih programskih jezika za razvoj softvera, analizu podataka, mašinsko učenje, automatizaciju sistema kao i razvoj web aplikacija. Ono što ga čini toliko popularnim jeste lakoća učenja. Python sintaksa je jednostavna i čitljiva, čineći ga idealnim za početnike. Ono što ga karakteriše jeste i veliki broj dostupnih biblioteka koje olakšavaju proces razvoja. Osnovna primena jeste automatizacija i skriptovanje.

2.2. PyTorch

Predstavlja biblioteku otvorenog koda za mašinsko učenje razvijen od strane Facebook tima za veštačku inteligenciju. Koristi se najviše u istraživačkim i razvojnim procesima na polju dubokog učenja, uglavnom usredsređen na razvoj neuronskih mreža, kroz vrlo jednostavan interfejs.

2.3. NLTK

NLTK (Natural Language Toolkit) prestavlja jednu od najpopularnijih i najmoćnijih biblioteka za obradu prirodnih jezika. Veliki broj modela, alata i biblioteka za obradu teksta je odmah dostupan korisniku.

2.4. Tkinter

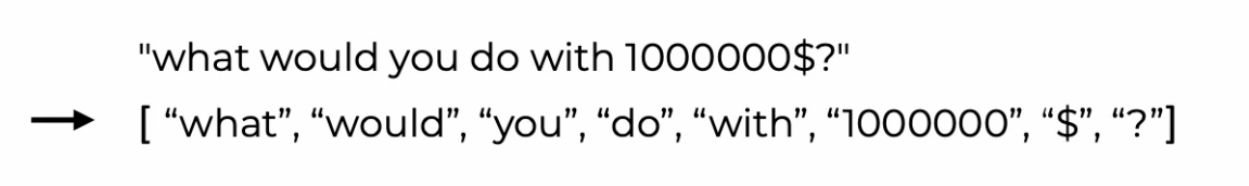
Python biblioteka za kreiranje korisničkog interfejsa.

3. Koncepti obrade prirodnih jezika

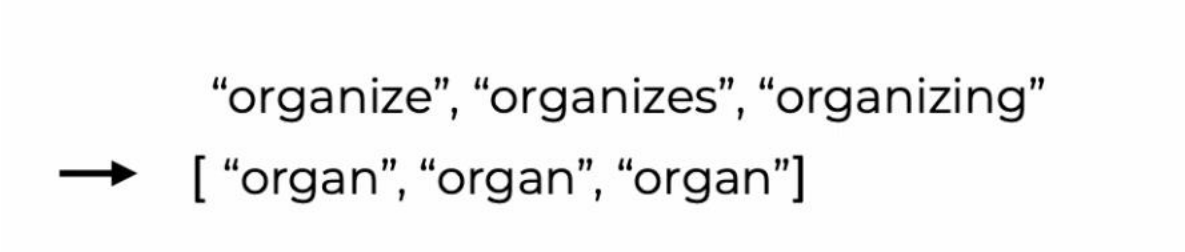
Obrada prirodnih jezika *(eng. Natural Language Processing - NLP)* fokusira se na to kako programirati računare da obrađuju i analiziraju velike količine podataka prirodnih jezika. Pod „prirodni“ odnosi se na jezik kojim ljudi komuniciraju. Glavni problem predstavlja čitanje i razumevanje jezika. Postoji nekoliko tehnika za samu obradu podataka kako bi računar umeo da razume.

Prilikom razvoja ove aplikacije korišćeno su koncepti: **Tokenization, Stemming, Tagging, Bag of words.**

1. Tokenizaton – Predstavlja deljenje string-a na smislene celine, odnosno listu tokena. Posmatrajte token kao reč u rečenici. Na primer:



2. Stemming – Predstavlja proces skraćivanja reči na njen koren. Koristi se kako bi se poboljšala analiza i prepoznatljivost reči prilikom pretrage. Na primer:



4.Tagging – „Etiketiranje“ reči prema unapred definisanim pravilima. Pristupi etiketiranju su: pristup zasnovan na pravilima (lingvističko znanje) ili stohastički pristup (podaci za obučavanje). U razvoju korišćen je stohastički pristup.

3. Bag of words – Predstavlja tehniku za modelovanje i klasifikaciju teksta. Čini reprezentaciju pojave reči u tekstu. Na osnovu koje se vrše obrade i analize.

4. Postavljanje razvojnog okruženja

Za početak preuzeti i instalirati **Anaconda package manager** - https://www.anaconda.com/products/individual. Neophodan je za kreiranje i upravljanje okruženjima neophodnim za rad. Naravno pre toga neophodno je instalirati **Python** na svom računaru - https://www.python.org/downloads/. Kako je moguće kreirati sopstvena okruženja tako je moguće koristiti i već unapred kreirana okruženja sa paketima već spremim za rad. Jedno takvo okruženje jeste i **PyTorch** koji poseduje sve što nama treba. Posetiti https://pytorch.org/ i u okviru **Get started** sekcije možete izabrati podešavanja koja odgovaraju vašem računaru i ispratiti proces za instalaciju. Proces možete ispratiti na sledećem linku: https://www.youtube.com/watch?v=EMXfZB8FVUA&list=PLqnslRFeH2UrcDBWF5mfPGpqQDSta6VK4

1. Kreiranje okruženja pomoću Anaconda package manager-a

conda create -n pytorch python=VERZIJA (python=3.7)

2. Aktivacija okruženja

conda activate pytorch

3. Instalacija PyTorch-a (Zavisi od operativnog sistema)

conda install pytorch torchvision torchaudio -c pytorch

Nakon ovoga u okviru novog okruženja kojeg smo napravili pomoću Anaconda package manager-a imamo instaliran Pytorch sa svim neophodnim paketima koje on pruža. Ono što nam je ostalo za kraj jeste instalacija **NLTK paketa.** U okviru našeg pytorch okruženja pokrenuti komandu: **conda install nltk** ili korišćenjem alternativnog package manager-a Pip: **pip install nltk.**

5. Razvoj

5.1. Pomoćne funkcije

Za početak neophodno je kreirati pomoćne funkcije iz NLTK paketa kako bi mogli da koristimo neophodne koncepte obrade prirodnih jezika koje smo napomenuli gore.

```
import nltk
import numpy as np
nltk.download('punkt')
from nltk.stem.porter import PorterStemmer

# Kreiramo stemmer
stemmer = PorterStemmer()

def tokenize(sentence):
    return nltk.word_tokenize(sentence)

def stem(word):
    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence, all_words):
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(all_words), dtype=np.float32)
    for idx, w in enumerate(all_words):
```

```
        if w in sentence_words:
            bag[idx] = 1.0

    return bag
```

5.2. Podaci za treniranje

Za potrebe treniranja našeg modela, odnosno chatbot-a, neophodni su nam podaci. Kreirati **.json fajl** sa željenim podacima prema sledećem modelu:

```
{
  "dictionary": [
    {
      "tag": "greeting",
      "patterns": ["Ćao", "Cao", "Zdravo", "Dobar dan"],
      "responses": [
        "Zdravo :-)",
        "Pozdrav, kako vam mogu pomoći",
        "Zdravo, šta mogu da učinim za vas?"
      ]
    }
  ]
}
```

Tag - predstavlja naše „etiketiranje“ rečenica pomoću kojih će model prepoznati date rečenice i nasumično vratiti odgovor.

Patterns - modelu govori koje reči odnosno rečenice da prepoznaje pod ovim tagom.

Responses – Lista mogućih odgovora na prepoznati pattern.

5.3. Treniranje modela

Kreirati fajl **train.py** koji će biti korišćen zajedno sa našim Neural Network modelom za treniranje chatbot-a**:**

```
import numpy as np
import json

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from nltk_utils import bag_of_words, tokenize, stem
from model import NeuralNet

# Učitavamo naš json fajl
with open("dictionary.json", "r") as f:
    dictionary = json.load(f)

all_words = []
tags = []
xy = []

# Prolazimo kroz svaku rečenicu za svaki kreirani šablon
for item in dictionary["dictionary"]:
    # Dodajemo sve tagove u listu
    tag = item["tag"]
```

```
    tags.append(tag)

    for pattern in item["patterns"]:
        # Tokenizujemo svaku reč rečenice
        w = tokenize(pattern)
        # Dodajemo u našu listu svih reči
        all_words.extend(w)
        # kreiramo niz parova [pattern-tag] koji nam je neophodan za trening chatbot-a
        xy.append((w, tag))

# Stemming i uklanjamo interpukcijske znake
ignore_words = ["?", ".", "!"]
all_words = [stem(w) for w in all_words if w not in ignore_words]
# Uklanjamo duplikate kreiranjem Set-a
all_words = sorted(set(all_words))
tags = sorted(set(tags))

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)

# Kreiranje podataka za treniranje chatbot-a
X_train = []
y_train = []

for (pattern_sentence, tag) in xy:
```

```
    # X: bag of words za svaku tokenizovanu rečenicu
    bag = bag_of_words(pattern_sentence, all_words)
    # Kreiramo naš X vektor za treniranje chatbot-a
    X_train.append(bag)

    # Uzimamo odgovarajući index našeg tag-a
    label = tags.index(tag)
    y_train.append(label)  # Class labels

# Numpy array - Grid of values
X_train = np.array(X_train)
y_train = np.array(y_train)

# Hyper-parameters
num_epochs = 1000
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)


class ChatDataset(Dataset):
    # učitavanje dataseta
    def __init__(self):
        self.n_samples = len(X_train)  # Broj uzoraka
```

```
        self.x_data = X_train
        self.y_data = y_train

    # pristupanje odredjenom indexu dataseta
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # duzina dataseta
    def __len__(self):
        return self.n_samples


dataset = ChatDataset()
train_loader = DataLoader(
    dataset=dataset, batch_size=batch_size, shuffle=True, num_workers=0
)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = NeuralNet(input_size, hidden_size, output_size).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Treniranje modela
for epoch in range(num_epochs):
```

```
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        # Forward pass
        outputs = model(words)
        loss = criterion(outputs, labels)

        # Backward pass and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 100 == 0:
        print(
            "Epoch [{0}/{1}], Loss: {2:.4f}".format(epoch + 1, num_epochs, loss.item())
        )


print("Final loss: {:.4f}".format(loss.item()))

data = {
    "model_state": model.state_dict(),
    "input_size": input_size,
    "hidden_size": hidden_size,
    "output_size": output_size,
    "all_words": all_words,
```

```
    "tags": tags,
}

FILE = "data.pth"
torch.save(data, FILE)

print("training complete. file saved to {}".format(FILE))
```

5.4. Model neuronske mreže

U ovom slučaju korišćen je model neuronske mreže sa prostiranjem signala unapred (*eng. feedforward*) koji ima dva skrivena sloja. Koritićemo Neural net paket u okviru PyTorch-a.

```
import torch
import torch.nn as nn

# Feed Forward Neural Net
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)

        return out
```

Da bi pokrenuli treniranje našeg modela neophodno je da u našem projektu pokrenemo komandu **python train.py.** Komanda će pokrenuti našu Python skriptu **train.py** i u projektu kreirati novi fajl sa našim trening podacima. (U konkretnom projektu fajl će se zvati **data.pth**)

5.5. Chatbot

```
import random
import json

import torch

from model import NeuralNet
from nltk_utils import bag_of_words, tokenize

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

with open("dictionary.json", "r") as json_data:
    dictionary = json.load(json_data)

FILE = "data.pth"
data = torch.load(FILE)

input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data["all_words"]
tags = data["tags"]
model_state = data["model_state"]

model = NeuralNet(input_size, hidden_size, output_size).to(device)
```

```
model.load_state_dict(model_state)
model.eval()
#Naziv chatbot-a
bot_name = "FON Sluzba"

def get_response(msg):
    sentence = tokenize(msg)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    print(prob.item())
    if prob.item() > 0.5:
        for item in dictionary["dictionary"]:
            if tag == item["tag"]:
                return random.choice(item["responses"])

    return "Nisam Vas razumeo..."
```

5.6. Korisnički interfejs – Tkinter

```
from tkinter import *
from chat import get_response, bot_name

BG_GRAY = "#ABB2B9"
BG_COLOR = "#0e4775"
TEXT_COLOR = "#EAECEE"

FONT = "Helvetica 14"
FONT_BOLD = "Helvetica 13 bold"


class ChatApplication:
    def __init__(self):
        self.window = Tk() # Top level widget
        self._setup_main_window() # Create layout

    def run(self): # Pokretanje aplikacije
        self.window.mainloop()

    def _setup_main_window(self): # Helper
        self.window.title("FON Chatbot")
        self.window.resizable(width=False, height=False) # NOT RESIZEABLE
        self.window.configure(width=470, height=550, bg=BG_COLOR) # Configuration
```

```
    # Defisanje zaglavlja
    head_label = Label(
        self.window,
        bg=BG_COLOR,
        fg=TEXT_COLOR,
        text="Dobrodošli",
        font=FONT_BOLD,
        pady=10,
    )
    head_label.place(relwidth=1)

    # tiny divider
    line = Label(self.window, width=450, bg=BG_GRAY)
    line.place(relwidth=1, rely=0.07, relheight=0.012)

    # Prostor za ispisavanje poruka
    self.text_widget = Text(
        self.window,
        width=20,
        height=2,
        bg=BG_COLOR,
        fg=TEXT_COLOR,
        font=FONT,
        padx=5,
        pady=5,
    )
```

```
        self.text_widget.place(relheight=0.745, relwidth=1, rely=0.08)
        self.text_widget.configure(cursor="arrow", state=DISABLED) # Readonly

        # Scroll bar
        scrollbar = Scrollbar(self.text_widget) # Postavljamo scrollbar na text_widget
        scrollbar.place(relheight=1, relx=0.974) # Cela visina text_widgeta
        scrollbar.configure(command=self.text_widget.yview) # Osposobljavanje scroll-a

        # Footer
        bottom_label = Label(self.window, bg=BG_GRAY, height=80)
        bottom_label.place(relwidth=1, rely=0.825)

        # Unos poruka
        self.msg_entry = Entry(bottom_label, bg="#0e4775", fg=TEXT_COLOR,
font=FONT) # bottom_label je parent
        self.msg_entry.place(relwidth=0.74, relheight=0.06, rely=0.008, relx=0.011)
        self.msg_entry.focus() # Inicijalni fokus
        self.msg_entry.bind("<Return>", self._on_enter_pressed) # Slanje klikom na
ENTER

        # Dugme za slanje poruke
        send_button = Button(
            bottom_label,
            text="Send",
            font=FONT_BOLD,
            width=20,
```

```
        bg=BG_GRAY,
        command=lambda: self._on_enter_pressed(None),
    )
    send_button.place(relx=0.77, rely=0.008, relheight=0.06, relwidth=0.22)

def _on_enter_pressed(self, event):
    msg = self.msg_entry.get()
    self._insert_message(msg, "You")

def _insert_message(self, msg, sender):
    if not msg:
        return

    self.msg_entry.delete(0, END)
    msg1 = f"{sender}: {msg}\n\n"
    self.text_widget.configure(state=NORMAL) # Kako bi mogli da editujemo
    self.text_widget.insert(END, msg1)
    self.text_widget.configure(state=DISABLED) # Vracamo u readonly

    msg2 = f"{bot_name}: {get_response(msg)}\n\n"
    self.text_widget.configure(state=NORMAL)
    self.text_widget.insert(END, msg2)
    self.text_widget.configure(state=DISABLED)

    self.text_widget.see(END)
```

```
if __name__ == "__main__":
    app = ChatApplication()
    app.run()
```

6. Finalni izgled aplikacije

Kada je sve postavljeno, kada imamo podatke za trening, aplikaciju pokrećemo sa komandom **python app.py.** Interfejs naše aplikacije izgleda ovako. Unosom pitanja, chatbot će analiziranjem poslate rečenice vratiti odgovarajući odgovor. Naravno, što bolji podaci za trening to je i naš chatbot bolji.