1 Wstęp

1.1 Autorzy

- Jan Liberacki
- Jakub Ziarko

1.2 Wprowadzenie

Celem naszej pracy inżynierskiej jest zbadanie liczności populacji żubra w Polsce.

2 Cel i wizja produktu

2.1 Definicja problemu

Zadanie weryfikacji autorstwa tekstu to ocena czy autor danego zbioru tekstów jest również autorem kwestionowanego tekstu. Weryfikacja autorstwa ma się odbywać dla języków: angielski, hiszpański, holenderski, grecki. Każdy z tekstów zawiera od kilkuset do kilku tysięcy liter.

2.2 Cel pracy

Celem projektu jest stworzenie biblioteki zdolnej do tworzenia i trenowania sieci dla zadania weryfikacji autorstwa tekstu pisanego. Praca ta ma rozwijać pracę inżynierską z poprzednich lat, która nie wyczerpała tego tematu. Osiągnięto pozytywne wyniki dla języka angielskiego, natomiast dla pozostałych z wyżej wymienionych języków są one nadal niezadowalające.

2.3 Motywacja do pracy

Motywacja do projektu nie zmieniła się znacznie względem pracy na której bazujemy. Mimo pojawienia się wielu zaawansowanych narzędzia do łatwego tworzenia nawet bardzo skomplikowanych sieci, zapewniających przy tym szybką naukę oraz kontrolę nad jej przebiegiem wciąż brakuje dopracowanej biblioteki, która pozwoliłaby na tworzenie i trenowanie sieci dla zadania weryfikacji autorstwa tekstu. Projekt, którego dalszego rozwoju się podejmujemy nie w wyczerpał tematu, stąd pojawia się miejsce dla tej pracy. Największa różnicą względem pracy, na której już bazujemy jest fakt, że naszym celem jest zwiększenie dokładności tego rozwiązania.

2.4 Wizja

Wizją tej pracy jest stworzenie dopracowanej biblioteki pozwalającej na konfigurowalne tworzenie głębokich sieci typu RNN, trening, preprocessing danych wejściowych. Oprócz tego powinna ona zapewnić persystencję wytrenowanych modeli oraz ich dalsze testowanie. System powinien wspierać naukę w czterech językach: angielskim, greckim, hiszpańskim oraz holenderskim. Grecki, hiszpański oraz holenderski wymagają największego dopracowania. Wyżej wspomniana konfigurowalność powinna pozwalać na tworzenie dowolnych sieci parametryzowanych hiperparametrami: głębokość sieci, wymiar wektorów wejściowych, liczba neuronów w warstwach ukrytych, wielkość mini-batchów, itd. Ważna jest też możliwość wizualizacji modelu, np. poprzez generację grafu obrazującego strukturę sieci. Biblioteka powstanie w języku Python przy wykorzystaniu biblioteki PyTorch.

2.5 Analiza ryzyka

2.5.1 Czas ewaluacji i eksperymentów

Stworzenie podobnej biblioteki wiąże się z nieustannym treningiem oraz testowaniem jakości generowanych sieci, stąd jedną z przeciwności będzie czas, który będzie potrzebny na wszystkie obliczenia. Będą one wymagać dużej ilości czasu procesora. W tym wypadku wszystkie obliczenia będą prowadzone z wykorzystaniem Prometeusza.

2.5.2 Wiedza na temat sieci neuronowych

Sieci neuronowe są bardzo złożoną kwestią, która wymaga sprawnego poruszaniem się w wyższej matematyce. W szczególności w kontekście tego projektu potrzebne będzie bardzo dokładne zgłębienie tego tematu. Jest to dla nas największa przeszkoda. Wymaga czasu oraz dużej cierpliwości, szczególnie że jest to dla nas zupełnie nowe zagadnienie.

2.5.3 Znajomość języków programowania oraz bibliotek

Obojga autorów tej pracy cechuje wysoka znajomość języka Python. Natomiast przeszkodą będzie opanowanie biblioteki PyTorch, która jest dla nas zupełną nowością.

2.5.4 Wykonalność ekonomiczna

Będziemy korzystać z darmowych bibliotek oraz środowiska, na które mamy licencje jako studenci. Ewentualne koszty będą wynikać z dostępu do wiedzy, np. kursy internetowe, książki.

3 Preprocessing

Preprocessing to moduł naszej biblioteki odpowiedzialny za przetwarzanie tekstów na formay nadający się do szkolenia sieci.

3.1 Teoria

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
                VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)

    return M
```

3.2 Jak korzystać?

test2

3.3 Budowa

test3