Ocena nacechowania depresyjnego wypowiedzi przez algorytmy uczenia głębokiego

Urszula Wąsowska Maciej Kapitan

20 stycznia 2024

Szkic prezentacji

- Wprowadenie
- Motywacja
- Zbiór danych
- Literatura
- Metody
- Model
- Eksperyment
- Wyniki

Wprowadzenie

Celem naszej pracy jest badanie możliwości zastosowania algorytmów przetwarzania języka naturalnego w ocenie depresji. Praca opisuje analizę przykład analizy sentymentów w oparciu o zastosowanie rekurencyjnej sieci neuronowej

Motywacja

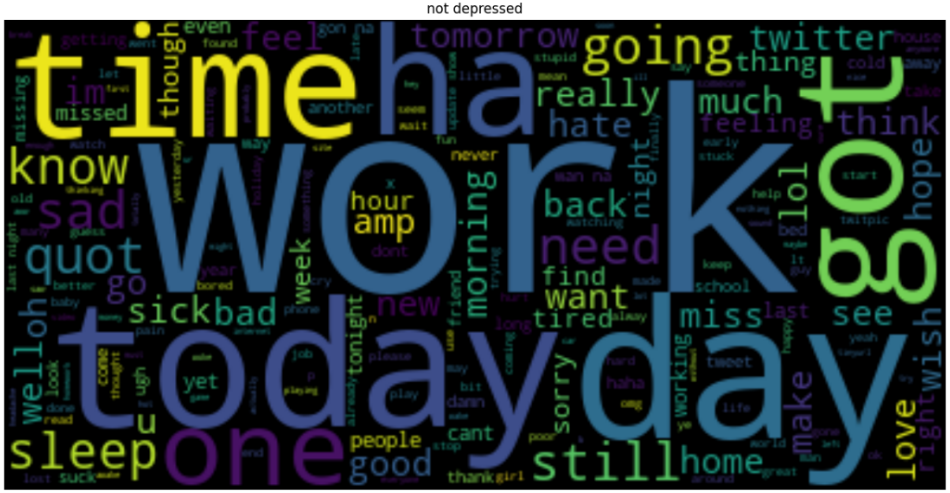
- Wzrost zastosowań NLP i LLM
- Możliwość rozwoju - diagnoza psychologiczna
- Wspomaganie lekarzy w procesie rozpoznawania chorób

Zbiór danych

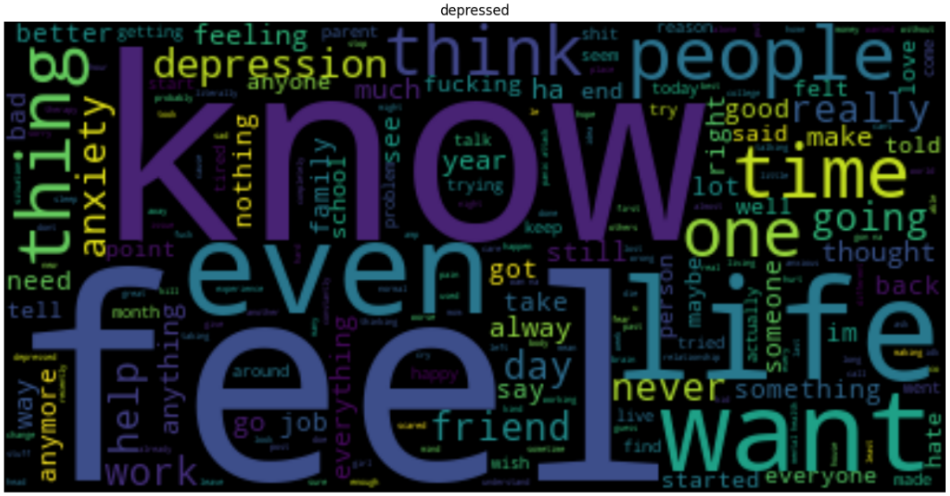
- Dane pochodzą ze strony kaggle.com
- Dane zawierają wypowiedzi zescrapowane z reddita i ocenione pod kątem nacechowania depresyjnego
- Dane zawierają czysty tekst, dzięki czemu może być on łątwiej przetwarzany przez algorytmy
- Przykład wypowiedzi o charakterze depresyjnym: "anyone else instead of sleeping more when depressed stay up all night to avoid the next day from coming sooner may be the social anxiety in me but life is so much more peaceful when everyone else is asleep and not expecting thing of you "
- Przykład wypowiedzi o niedepresyjnym charakterze: "is upset that he can t update his facebook by texting it and might cry a a result school today also blah"

Zbiór danych

not depressed



depressed



Literatura

- Pierwsze metody przetwarzania języka naturalnego sięgają lat 50 XX wieku - głównie analiza semantyczna tekstu
- Język ludzi jest skomplikowany składniowo i często zależy od kontekstu, nie ma określonych reguł
- Przełom: Zastosowanie rekurencyjnych sieci neuronowych zdolnych przetwarzać sekwencje
- Częstym zastosowaniem algorytmów NLP jest analiza sentymentów

Literatura

- Pierwsze metody przetwarzania języka naturalnego sięgają lat 50 XX wieku - głównie analiza semantyczna tekstu
- Język ludzi jest skomplikowany składniowo i często zależy od kontekstu, nie ma określonych reguł
- Przełom: Zastosowanie rekurencyjnych sieci neuronowych zdolnych przetwarzać sekwencje
- Częstym zastosowaniem algorytmów NLP jest analiza sentymentów

Literatura

- "Opinion Mining and Sentiment Analysis" autorstwa Bo Pang i Lillian Lee:
  Artykuł ten prezentuje ogólny przegląd analizy sentymentów, w tym techniki, wyzwania i kierunki badań.
- "Sentiment Analysis: A Survey" autorstwa Duyu Tang i Bing Qin:
  Ten artykuł jest przeglądem literatury, który prezentuje różne metody i techniki analizy sentymentów, zarówno te oparte na regułach, jak i te oparte na uczeniu maszynowym.
- "Deep Learning for Sentiment Analysis: A Survey" autorstwa Liang Wu i in.:
  Artykuł ten skupia się na zastosowaniu głębokich sieci neuronowych w analizie sentymentów, prezentując różne modele i podejścia.

Metody - tokenizacja

- Celem tokenizacji jest przekształcenie ciągu znaków na liczby, aby algorytmy uczenia maszynowego mogły je przetwarzać
- Tworzenie słownika najczęstszych słów z przypisanymi tokenami - liczbami naturalnymi
- Zamiana słowa na wektor - One hot encoding
- Zamiana słowa na wektor - Words embedding

Metody - sieć neuronowa

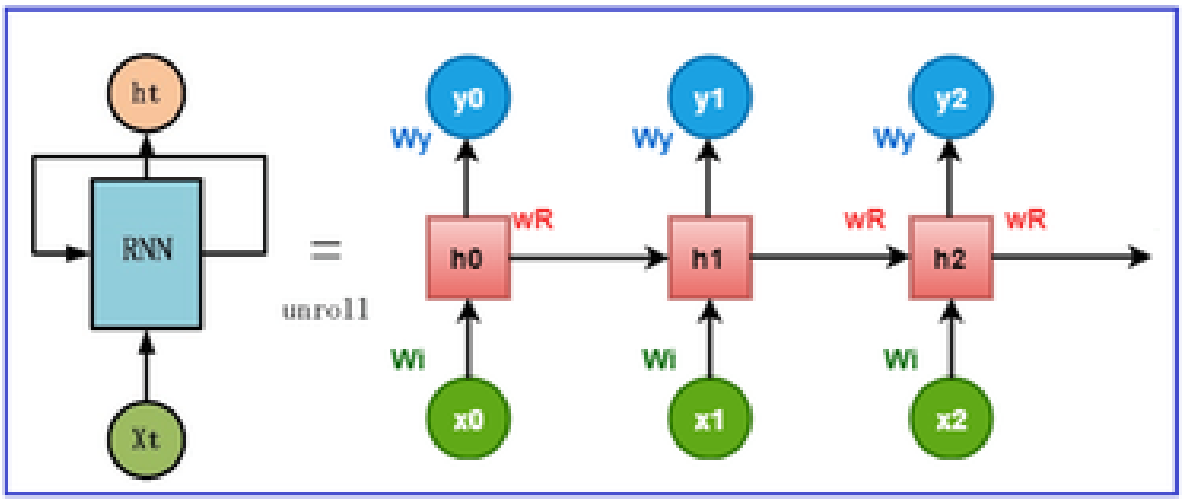
- Warstwa embedding - zamienia tokeny słów na wektory, które są następnie dopasowywane w procesie uczenia maszynowego
- Warstwa rekurencyjna - każdy neuron otrzymuje na wejściu output nie tylko z poprzedniej warstwy lecz także z poprzedniego neuronu w tej samej warstwie
- LSTM - ulepszenie zwykłych warst rekurencyjnych eliminujące problem znikającego gradientu

Metody - sieć neuronowa

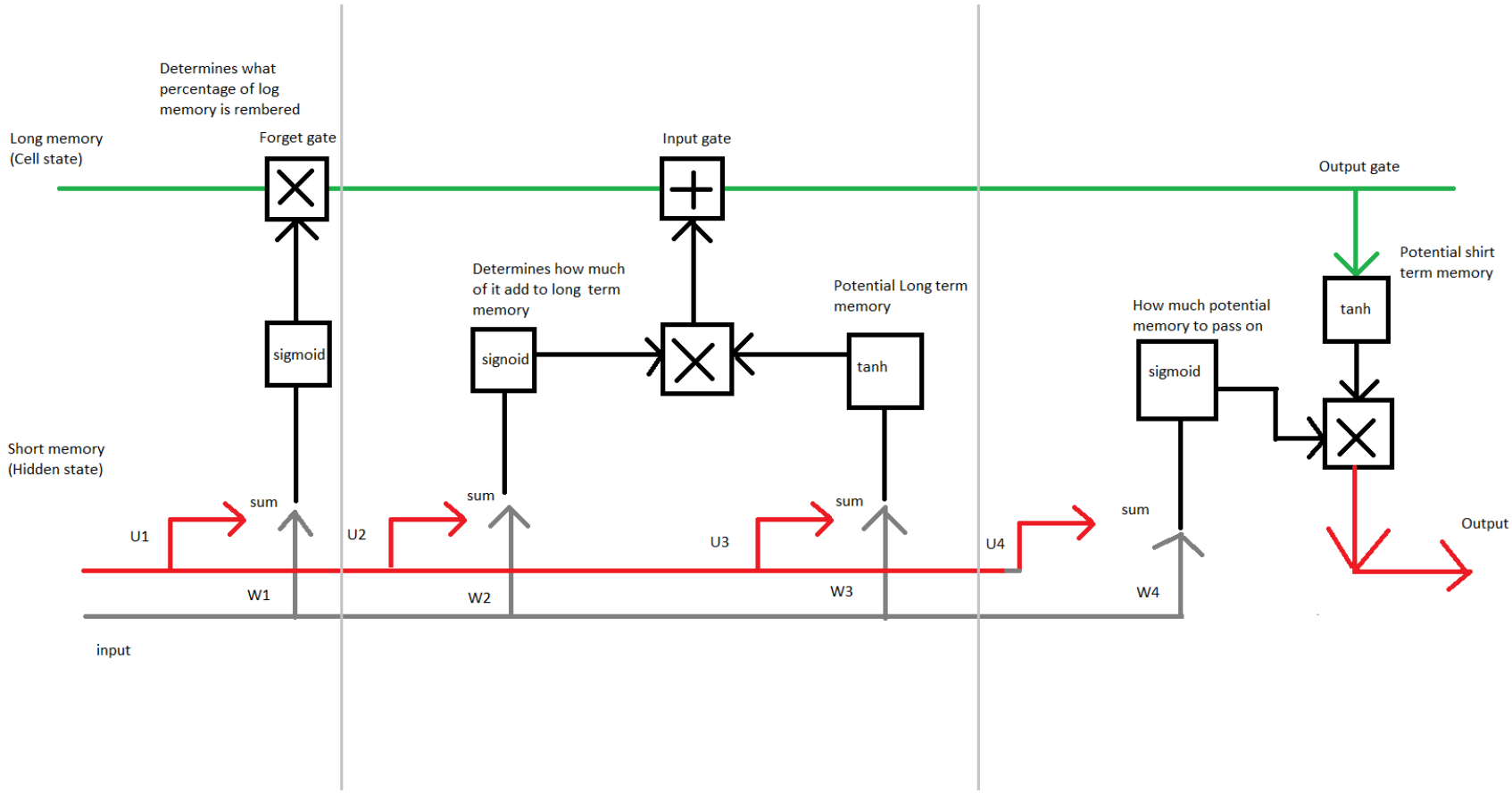
- Warstwa Flatten - wypłaszczenie macierzy reprezentującej zdanie do pojedynczego wektora
- Warstwa Dense z aktywacją relu oraz sigmoid
- Warstwa Drouot - losowe wyłączanie niektóych wag, zapobiega przetrrenowaniu modelu
- Adam optimizer - optymalizator który dobiera odpowiedni współczynnik uczenia do każdego buforu danych dzięki czemu jest ono efektywniejsze

Metody - sieć rekurencyjna

- Sieć rekurencyjna - output z poprzedniego neuronu w danej warstwie jest też inputem do następnego neuronu
- Problem znikającego lub eksplodującego gradientu w długich sieciach rekurencyjnych



Metody - LSTM


Determines what
percentage of log
memory is rembered
Long memory
(Cell state)
Forget gate
Input gate
Output gate
Potential shirt
term memory
Determines how much
of it add to long term
memory
Potential Long term
memory
How much potential
memory to pass on
tanh
sigmoid
signoid
tanh
sigmoid
Short memory
(Hidden state)
sum
sum
sum
sum
U1
U2
U3
U4
Output
W1
W2
W3
W4
input

Eksperyment

- Pozbycie się z tekstu niepotrzebnych informacji - stop words, znaki interpunkcyjne itd
- Przekształcenie słów na tokeny
- Uczenie z wykorzystaniem sieci neuronowej
- Uczenie z wykorzystaniem metod uczenia płytkiego
- Porównanie efektywności dla różnych parametrów

Eksperyment

Pozbycie się stopwords

```
from nltk.corpus import stopwords

stop_words = ["wa", "t", "m", "s", "don", "ve"] + stopwords.words("english")

def remove_stop_words(stop_words):
    def wrapper(text):
        seq = []
        for word in text.split():
            if word not in stop_words:
                seq.append(word)
        text = ' '.join(seq)
        return text
    return wrapper

nltk.download('stopwords')
df['clean_text'] = df['clean_text'].apply(lambda x: x.lower())
df['clean_text'] = df['clean_text'].apply(remove_stop_words(stop_words))
```

Eksperyment

Tokenizacja

```
from sklearn.model_selection import train_test_split
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer

maxlen=50
maxwords=1000

def prepare_data(df, maxlen=maxlen, maxwords=maxwords, one_hot=False):
    X, y = df['clean_text'], df['is_depression']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)

    tokenizer = Tokenizer(num_words=maxwords)
    tokenizer.fit_on_texts(X)

    if not one_hot:
        X_train = tokenizer.texts_to_sequences(X_train)
        X_test = tokenizer.texts_to_sequences(X_test)

        X_train = pad_sequences(X_train, maxlen=maxlen)
        X_test = pad_sequences(X_test, maxlen=maxlen)

    if one_hot:
        X_train = tokenizer.texts_to_matrix(X_train, mode='binary')
        X_test = tokenizer.texts_to_matrix(X_test, mode='binary')

    return X_train, X_test, y_train, y_test
```

Eksperyment

Model

```
from keras.layers import Embedding, Bidirectional, LSTM, Dropout, Dense, Flatten
from keras.models import Sequential


model = Sequential()
model.add(Embedding(maxwords, 8, input_length=maxlen))
model.add(Bidirectional(LSTM(64, dropout=0.5, recurrent_dropout=0.5)))
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))


model.summary()
```

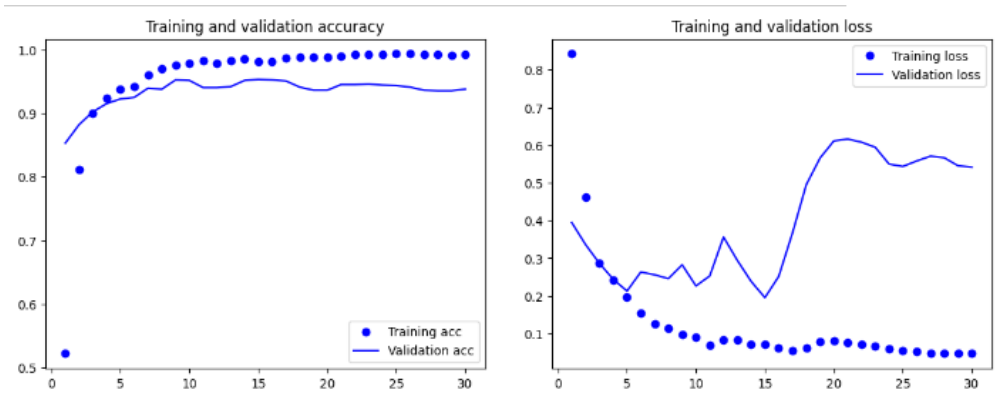
Eksperyment

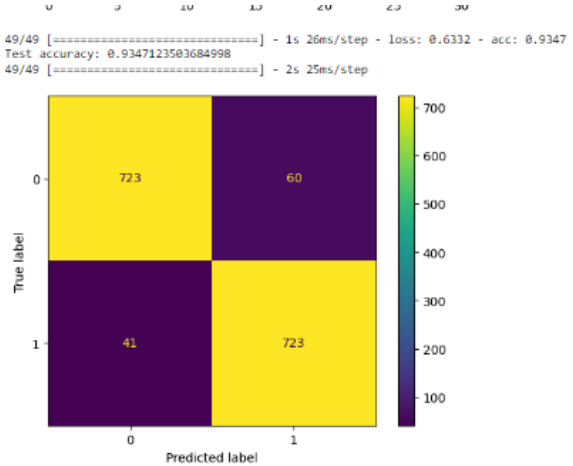
Ocena skuteczności

```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

def evaluate(model, X_test, y_test, epochs=10, history=None):
    if history:
        acc = history.history['acc']
        val_acc = history.history['val_acc']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        epochs = range(1, len(acc) + 1)
        plt.plot(epochs, acc, 'bo', label='Training acc')
        plt.plot(epochs, val_acc, 'b', label='Validation acc')
        plt.title('Training and validation accuracy')
        plt.legend()
        plt.figure()

        plt.plot(epochs, loss, 'bo', label='Training loss')
        plt.plot(epochs, val_loss, 'b', label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()
        plt.show()

    if isinstance(model, Sequential):
        print(f'Test accuracy: {model.evaluate(X_test, y_test)[1]}')
    else:
        print(f'Test accuracy: {model.score(X_test, y_test)}')
    predictions = model.predict(X_test)
    predictions = np.where(predictions > 0.5, 1, 0)

    cm = confusion_matrix(y_test, predictions, labels=[0,1])
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                                  display_labels=[0,1])
    disp.plot()
    plt.show()
```

Wyniki

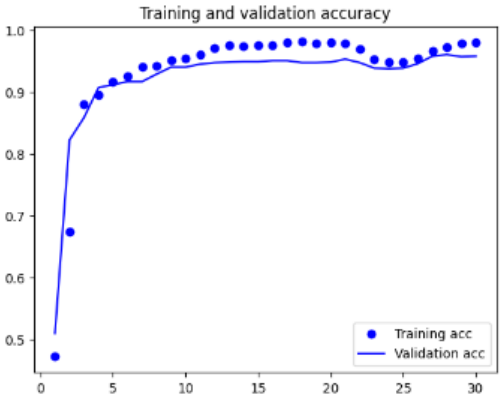
Model bez regularyzacji:

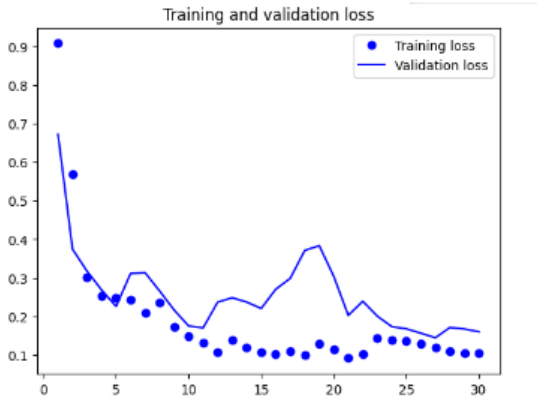


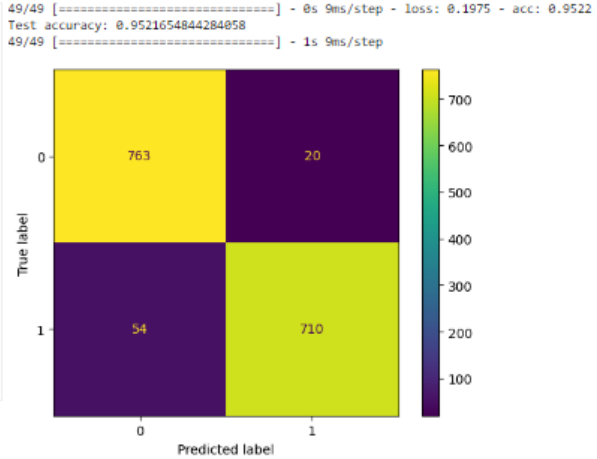


Wyniki

Model z regularyzacją:

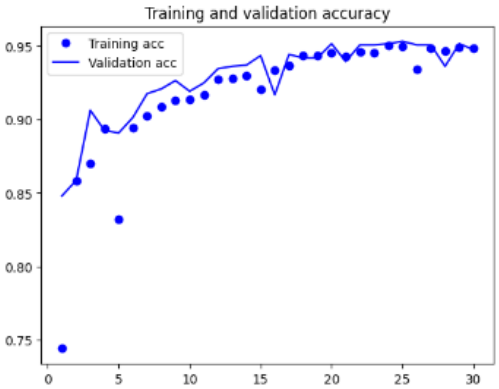


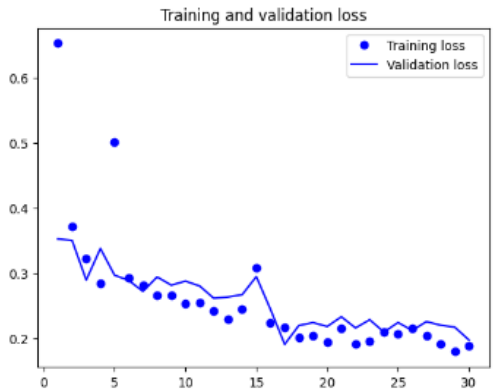


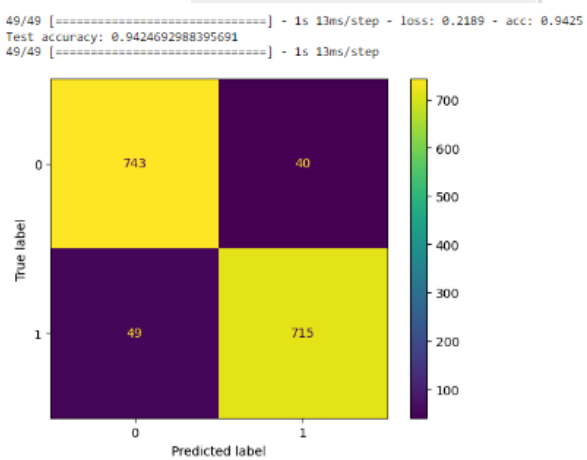


Wyniki

Model pretrenowany GloVe:



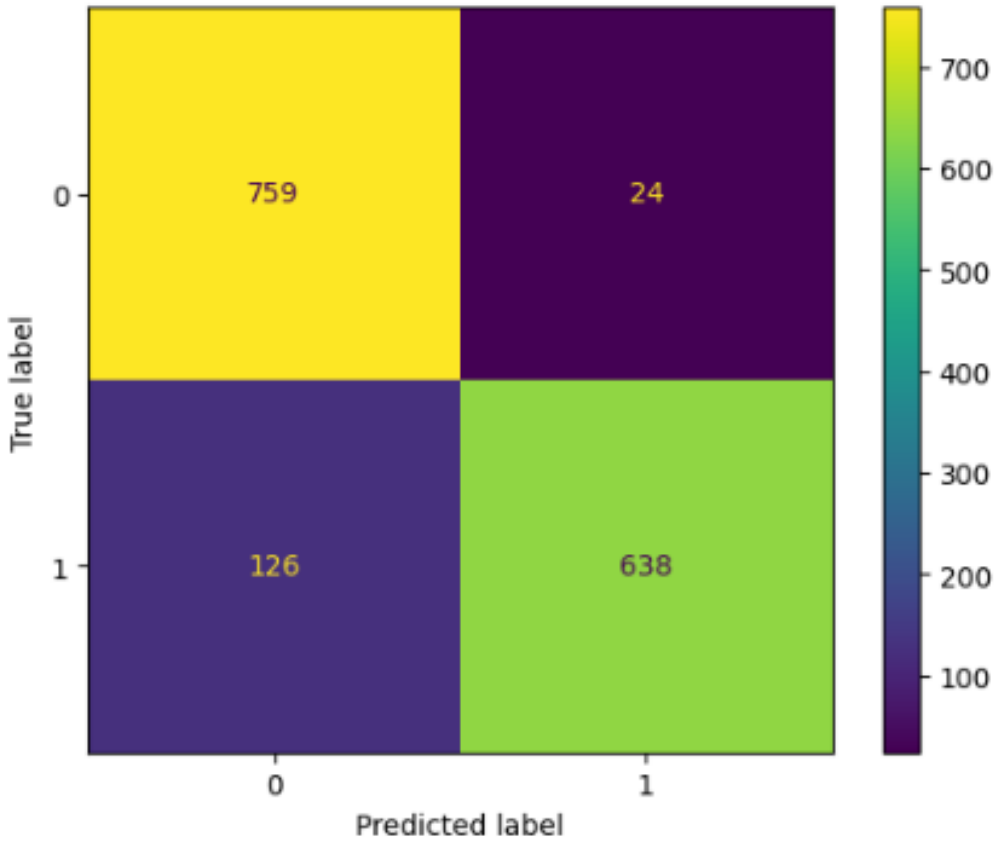




Wyniki

XGBoost:

Test accuracy: 0.903038138332256



Wyniki

Porównanie dokładności algorytmów za względu ma ilość słów i długość wiadomości:

Ilość słów/Długość wiadomości	5	20	100
10	0.784	0.782	0.747
100	0.896	0.935	0.936
500	0.873	0.953	0.948

Analizując wyniki z tabelki widać że skuteczność algorytmu jest wprost proporcjonalna do ilości słów i długości wiadomości. Kolejną cenną informacją jest fakt że nie ma dużej różnicy w skuteczności dla wiadomości zawierających 20 lub 100 słów - oznacza to że można równie dobrze testować algorytm dla krótszych wiadomości co pozwala zaoszczędzić czas i zasoby komputera.

Wnioski

- Sieci neuronowe dobrze sprawdzają się w analizie sentymentów
- Dobranie odpowiedniego modelu wymaga dużej ilości czasu przeznaczonej na porównywanie różnych modeli
- Przed treningiem sieci neuronowej warto sprawdzić jak algorytm uczenia płytkiego radzi sobie z zadanym problemem - być może prostsze i szybsze metody dadzą zadowalające wyniki
- Algorytmy przetwarzania języka naturalnego mogą być pomocne w ocenie psychologicznej, jednak wymagają one wielu danych, opisanych odpowiedni przez psychologów i psychiatrów. Istotne jest również testowanie modelu na bardzo różnych danych w celu zapobiegnięcia przeuczeniu.
- Skuteczność algorytmu jest tym większa, im więcej słów zawrzemy w słowniku z tokenami