Podstawy Sztucznej Inteligencji - laboratorium 1
Klasyfikacja tekstu

Łukasz Jezapkowicz

20 października 2020

1 Notebook z wynikami

Notebook z przeprowadzonymi eksperymentami dostępny jest pod poniższym linkiem.

2 Przygotowanie danych oraz trening modeli

Podstawą do zastosowania biblioteki *simple_transformers* jest odpowiednie przygotowanie danych. Pierwszy ze zbiorów, które są przedmiotem naszego zainteresowania, został już odpowiednio przygotowany. Drugi ze zbiorów został przeze mnie stworzony tak, by spełniać wymogi powyższej biblioteki. Proces tworzenia zbioru widoczny jest poniżej:

```
[104] file_data = ['training_set_clean_only_text.txt', 'training_set_clean_only_tags.txt']

      all_data2 = pd.concat((pd.read_csv(file, sep="\n", header = None) for file in file_data), axis=1)
      all_data2.columns = ['text', 'labels']
      print(all_data2.columns)
      print(all_data2['labels'].value_counts())

      Index(['text', 'labels'], dtype='object')
      0    9190
      1     851
      Name: labels, dtype: int64
```

Otrzymany zbiór następnie podzieliłem na część treningową oraz testową (ze współczynnikiem 0.9). Kolejnym krokiem był oczywiście trening stworzonych modeli. Jako pierwszy przeprowadziłem trening modelu neuronalnego.

2.1 Opis parametrów wynikowych

Poniżej zamieszczam opis poszczególnych parametrów wynikowych:

- parametr accuracy to najprostszy parametr wynikowy będący zwykłym ilorazem liczby dobrze opisanych obserwacji oraz liczby wszystkich obserwacji (TP+TN/TP+FP+FN+TN). Parametr ten jest dobrym wyznacznikiem, gdy liczba przypadków FP oraz FN jest podobna
- parametr precision to iloraz liczby poprawnie opisanych obserwacji positive oraz liczby wszystkich obserwacji opisanych positive (TP/TP+FP). Wysoki współczynnik precision jest pożądany, ponieważ oznacza mało przypadków FP, które są szczególnie groźne
- parametr recall to iloraz liczby poprawnie opisanych obserwacji positive oraz liczby wszystkich obserwacji positive (TP/TP+FN, czyli jaką część zarażonych wirusem poprawnie opisaliśmy)
- parametr f1-score to średnia ważona parametrów precision oraz recall
(2*(Recall * Precision) / (Recall + Precision)). Parametr ten jest szczególnie przydatny, gdy dystrybucja danych klas nie jest podobna (jedna z klas dominuje)
- parametr support nie jest dla nas szczególnie ważny

2.2 Model neuronalny

2.2.1 Pierwszy zbiór danych

Trening pierwszego zbioru danych oraz ewaluacja jego wyników została odpowiednio przygotowana wcześniej. Poniżej zamieszczam więc wyniki wraz z krótkim opisem (eksperyment przeprowadzony dla $num\_train\_epoch = 5$).

```
{'mcc': 0.7337069829549501, 'tp': 121, 'tn': 101, 'fp': 20, 'fn': 14, 'acc': 0.8671875, 'eval_loss': 0.5115653859393205}
{'recall': 0.8962962962962963, 'precision': 0.8581560283687943, 'f1-score': 0.8768115942028987, 'accuracy': 0.8671875}
```

Jak widać wyniki okazały się całkiem zadowalające. Każdy z parametrów wynikowych osiągnął wartość powyżej 0.85 co jest bez wątpienia dobrym wynikiem (w tym zbiorze każdy parametr ma duże znaczenie, ponieważ klasy są podobnej wielkości). Porównanie wyników z modelem bayesowskim w rozdziale 2.4.

2.2.2 Drugi zbiór danych

Trening drugiego zbioru oraz jego ewaluacja zostały przeze mnie stworzone bazując na gotowym modelu dla pierwszego zbioru. Poniżej widoczny kod źródłowy oraz wyniki eksperymentu przeprowadzonego dla $num\_train\_epoch = 5$.

```
[113] ClassificationModel.tokenizer = tokenizer
      cls_model_2_2 = ClassificationModel('roberta', './')
      cls_model_2_2.train_model(train_df2, args={"num_train_epochs": 5})
```

```
result2, model_outputs2, wrong_predictions2 = cls_model_2_2.eval_model(test_df2, acc=sklearn.metrics.accuracy_score)
```

```
{'mcc': 0.0, 'tp': 0, 'tn': 925, 'fp': 0, 'fn': 80, 'acc': 0.9203980099502488, 'eval_loss': 0.2815232598888023}
{'recall': 0.0, 'precision': nan, 'f1-score': nan, 'accuracy': 0.9203980099502488}
```

W tym przypadku wyniki okazały się bardzo słabe. Wszystkie ważne parametry dla tego zbioru (recall, precision oraz f1-score) okazały się albo zerowe albo nieokreślone (nieokreśloność wynika z dzielenia przez 0). Jak widać nasz model nie rozpoznał ani jednego przypadku szczególnie nas interesującego czyli TP (co ciekawe żadnego FP również nie znalazł). Model wymaga więc poprawy - tym zajmiemy się w rozdziale 3.

2.3 Model bayesowski

2.3.1 Pierwszy zbiór danych

Model bayesowski dla pierwszego zbioru danych został odpowiednio przygotowany wcześniej. Poniżej zamieszczam więc wyniki.

	precision	recall	f1-score	support
0	0.94	0.92	0.93	126
1	0.92	0.95	0.94	130
accuracy			0.93	256
macro avg	0.93	0.93	0.93	256
weighted avg	0.93	0.93	0.93	256

Jak widać wszystkie parametry osiągnęły wartości większe od 0.9! Są to więc wyniki lepsze od wyników dla modelu neuronalnego. Dla tego zbioru danych model bayesowski okazał się dobrą alternatywą.

2.3.2 Drugi zbiór danych

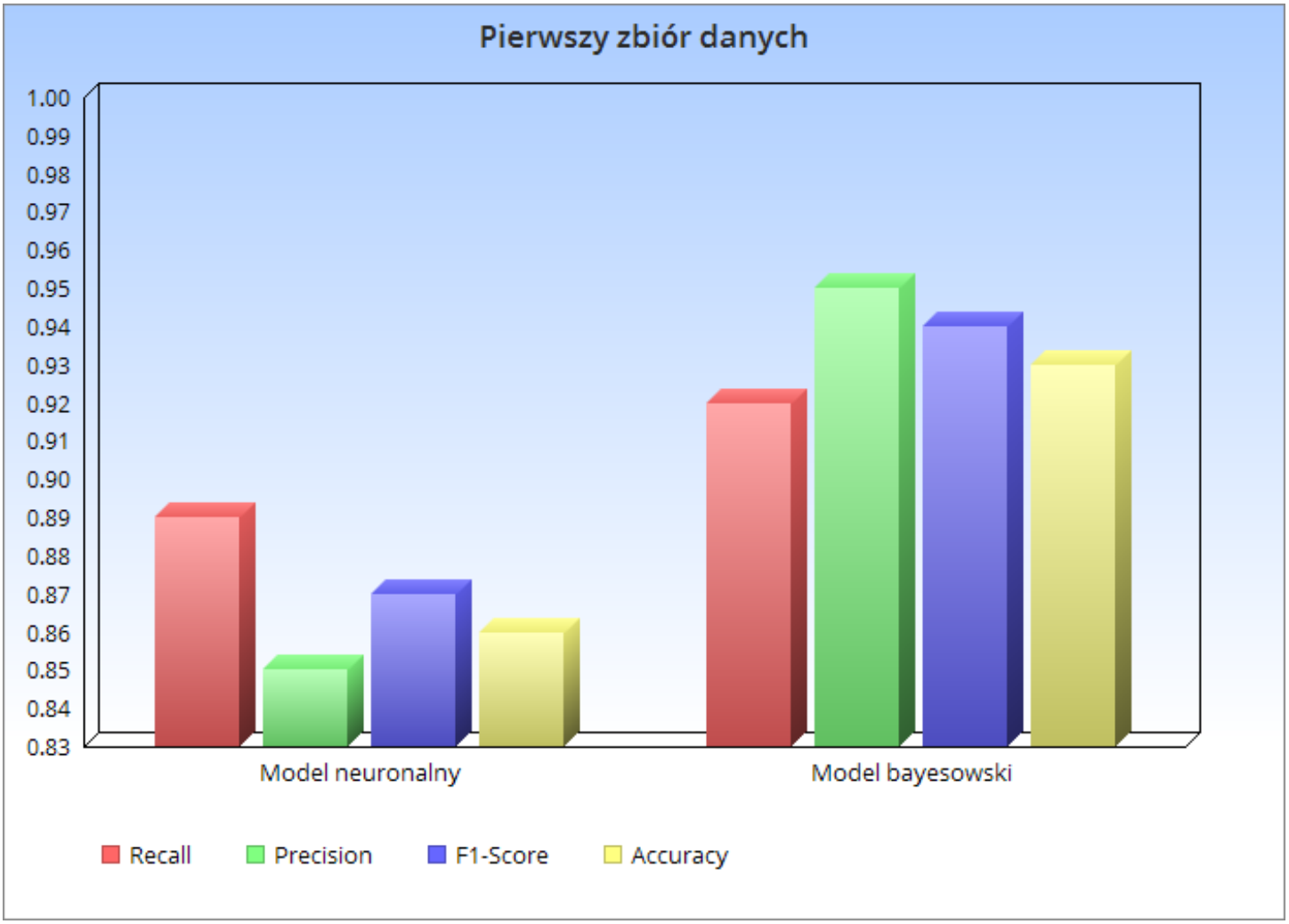
Model bayesowski dla drugiego zbioru danych został przeze mnie stworzony bazując na gotowym modelu dla pierwszego zbioru. Poniżej widoczny kod źródłowy oraz wyniki.

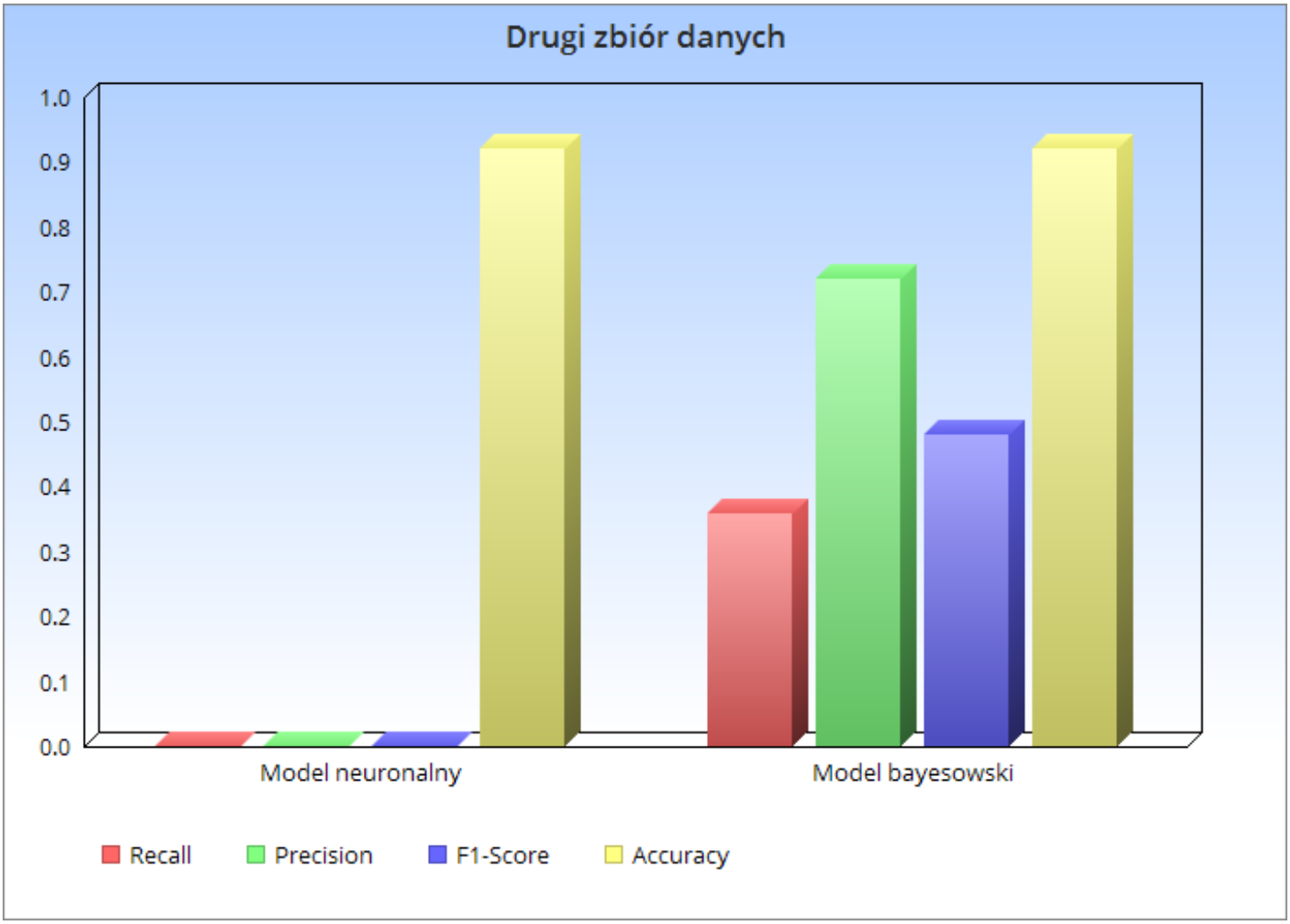
```
pipeline2 = Pipeline([
                ('tfidf', TfidfVectorizer(stop_words=[])),
                ('clf', OneVsRestClassifier(MultinomialNB(
                    fit_prior=True, class_prior=None))),
            ])
parameters2 = {
    'tfidf__max_df': (0.25, 0.5, 0.75),
    'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'clf__estimator__alpha': (1e-2, 1e-3)
}

train_x2 = [x.strip() for x in train_df2['text'].tolist()]
test_x2 = [x.strip() for x in test_df2['text'].tolist()]
train_y2 = [str(x) for x in train_df2['labels'].tolist()]
test_y2 = [str(x) for x in test_df2['labels'].tolist()]
print(len(train_x2), len(test_x2), len(train_y2), len(test_y2))
grid_search(train_x2, train_y2, test_x2, test_y2, ['0', '1'], parameters2, pipeline2)
```

	precision	recall	f1-score	support
0	0.93	0.98	0.96	906
1	0.72	0.36	0.48	99
accuracy			0.92	1005
macro avg	0.83	0.67	0.72	1005
weighted avg	0.91	0.92	0.91	1005

Dla drugiego zbioru wyniki nie są tak okazałe jak dla pierwszego. Są jednak lepsze niż te dla modelu neuronalnego (recall jest tutaj 0.36, podczas gdy tam wyniósł 0, a wartości precision i f1-score istnieją i mają całkiem nienajgorsze wartości!). Dla tego zbioru danych model bayesowski również okazał się nienajgorszą alternatywą.

2.3.3 Porównanie modeli





Z wykresów płyną następujące wnioski:
- dla prostych problemów model bayesowski nie ustępuje modelowi neuronalnemu
- dla zbioru pierwszego, gdzie klasy są o podobnej wielkości, model okazał się za słaby, żeby pokonać model bayesowski - oznacza to, że potrzebne są lepsze parametry uczenia modelu
- dla zbioru drugiego, gdzie jedna klasa jest znacząco większa od drugiej, model okazał się bardzo słaby (nie ma realnego zastosowania). Potrzeba o wiele lepszych parametrów (zwłaszcza odpowiedniego wagowania), żeby model mógł podjąć jakąkolwiek walkę z modelem bayesowskim

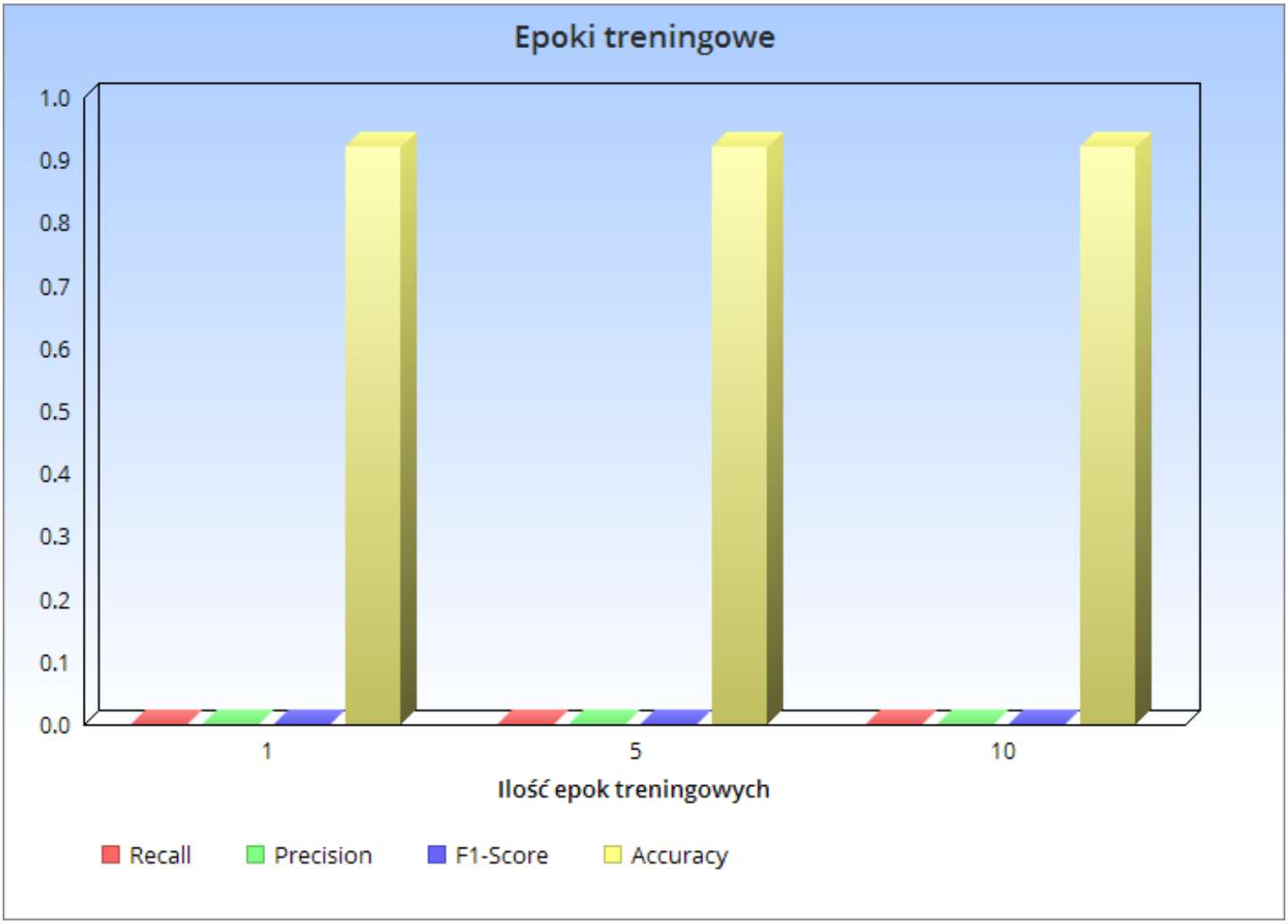
Teraz przejdziemy do szukania odpowiednich parametrów sieci neuronalnej.

3 Eksperymenty z hiper-parametrami

Wyniki trenowania modelu neuronalnego nie są zadowalające, chcielibyśmy stworzyć o wiele lepszy model. W tym celu będziemy badali zachowanie naszego modelu, gdy będziemy modyfikowali trzy argumenty związane z jego trenowaniem: liczbę epok treningowych, rozmiar batcha oraz wagi poszczególnych klas.

3.1 Liczba epok treningowych

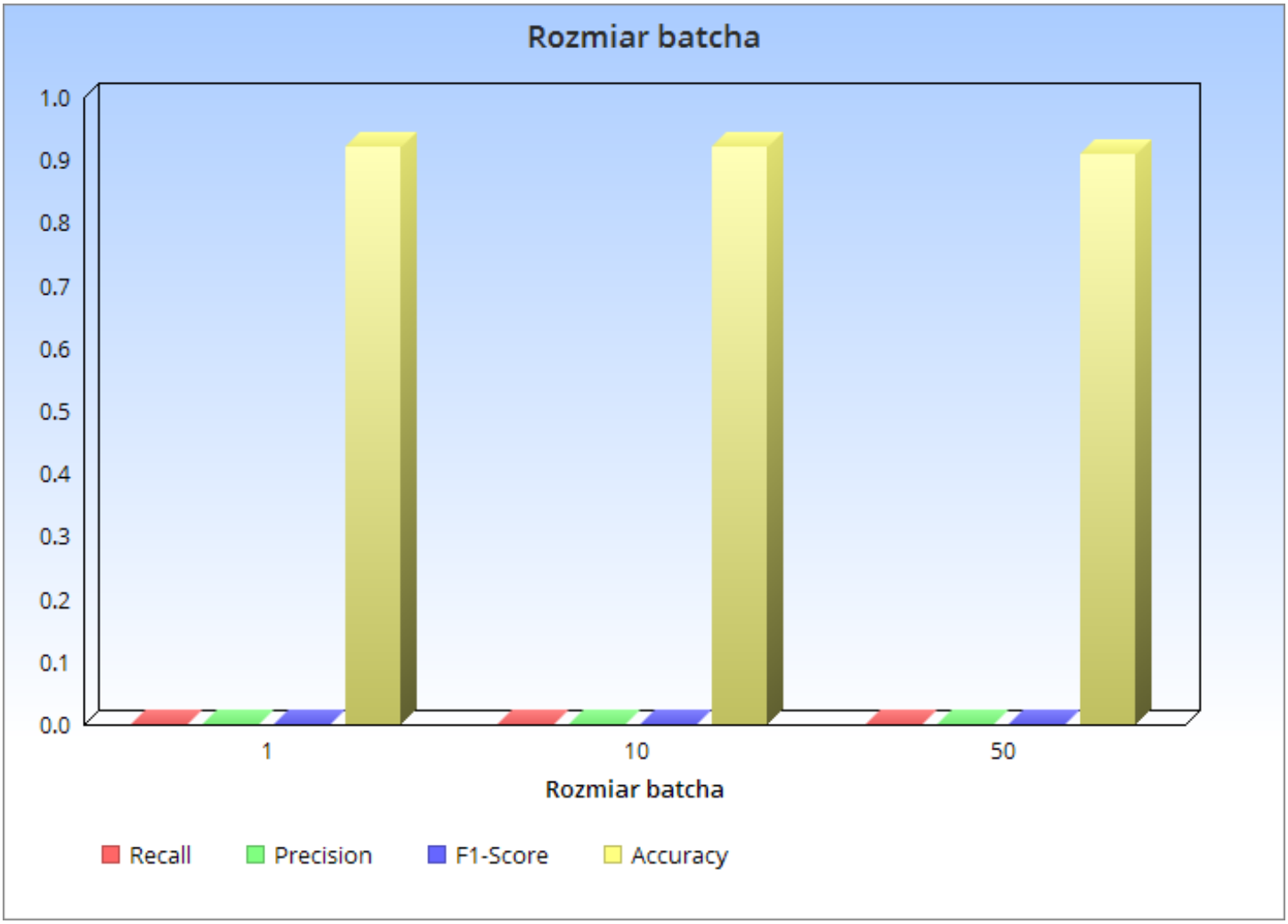
Liczba epok treningowych to nic innego jak liczba całkowitych przejść po naszym zbiorze treningowym. Podejrzewamy, że im większa ich liczba tym lepsze wyniki. Przeprowadzimy teraz eksperyment sprawdzający czy faktycznie tak jest. Eksperyment przeprowadzimy dla wartości: 1, 5, 10. Wyniki widoczne poniżej.



Wyniki okazały się identyczne (wszystkie parametry są identyczne oprócz eval_loss co świadczy o różnicy pomiędzy modelami). Co więcej, każdy z modeli jest tak samo słaby jak wcześniej (nie odgaduje NIC). Eksperyment nie doprowadził nas zatem do polepszenia naszego modelu.
Widać więc, że dla małych, prostych modeli zmiany (tylko) ilości epok treningowych nie polepszyły naszego modelu.

3.2 Rozmiar batcha

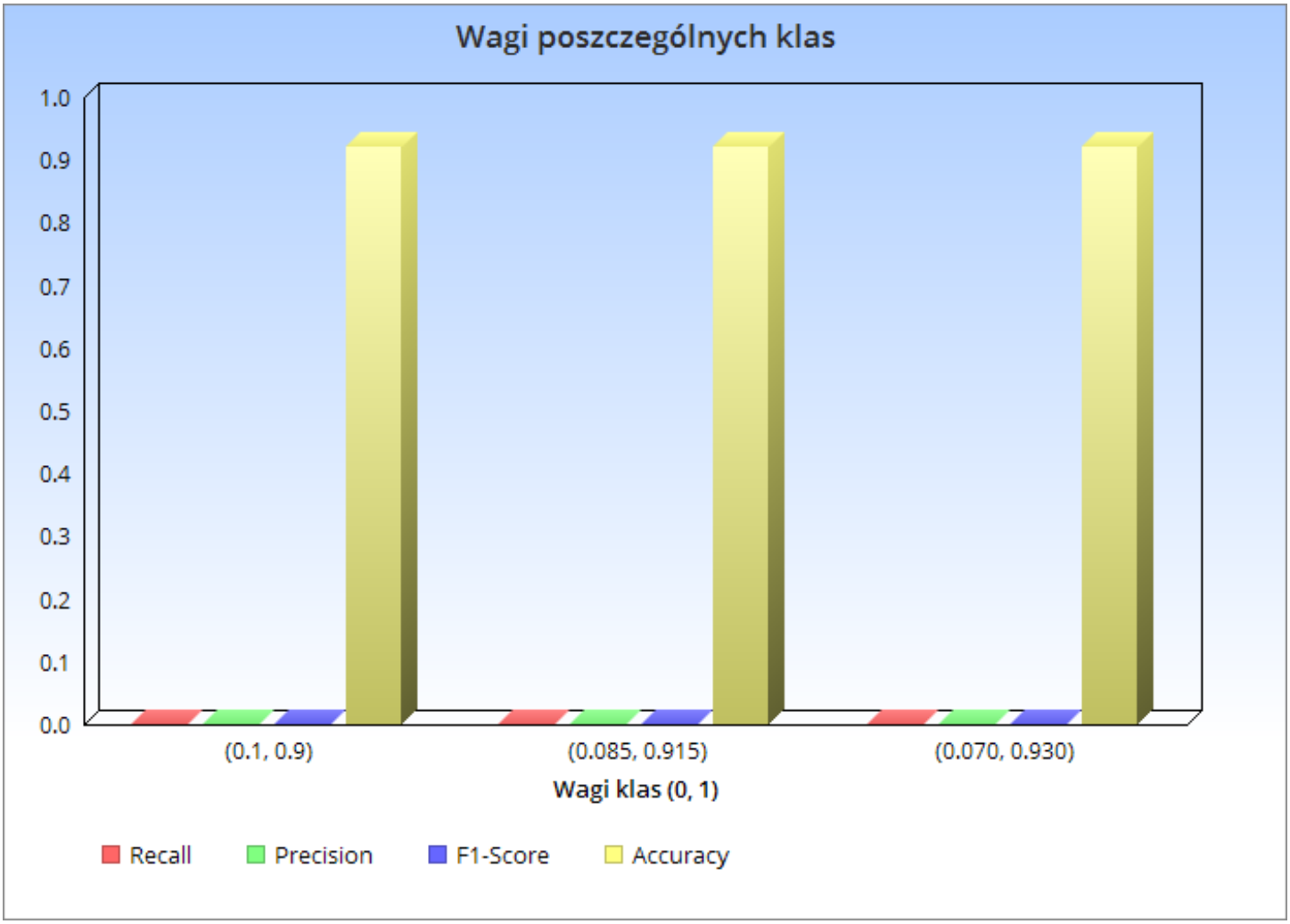
Rozmiar batcha to nic innego jak liczba obserwacji, po których aktualizujemy parametry naszego modelu. Nie zakładamy żadnych specjalnych wyników. Eksperyment przeprowadzimy dla wartości: 1, 10, 50. Wyniki widoczne poniżej.



Wyniki okazały się identyczne (wszystkie parametry są identyczne oprócz eval_loss co świadczy o różnicy pomiędzy modelami). Co więcej, każdy z modeli jest tak samo słaby jak wcześniej (nie odgaduje NIC). Eksperyment nie doprowadził nas zatem do polepszenia naszego modelu.
Widać więc, że dla małych, prostych modeli zmiany (tylko) rozmiaru batcha nie polepszyły naszego modelu.

3.3 Wagi poszczególnych klas

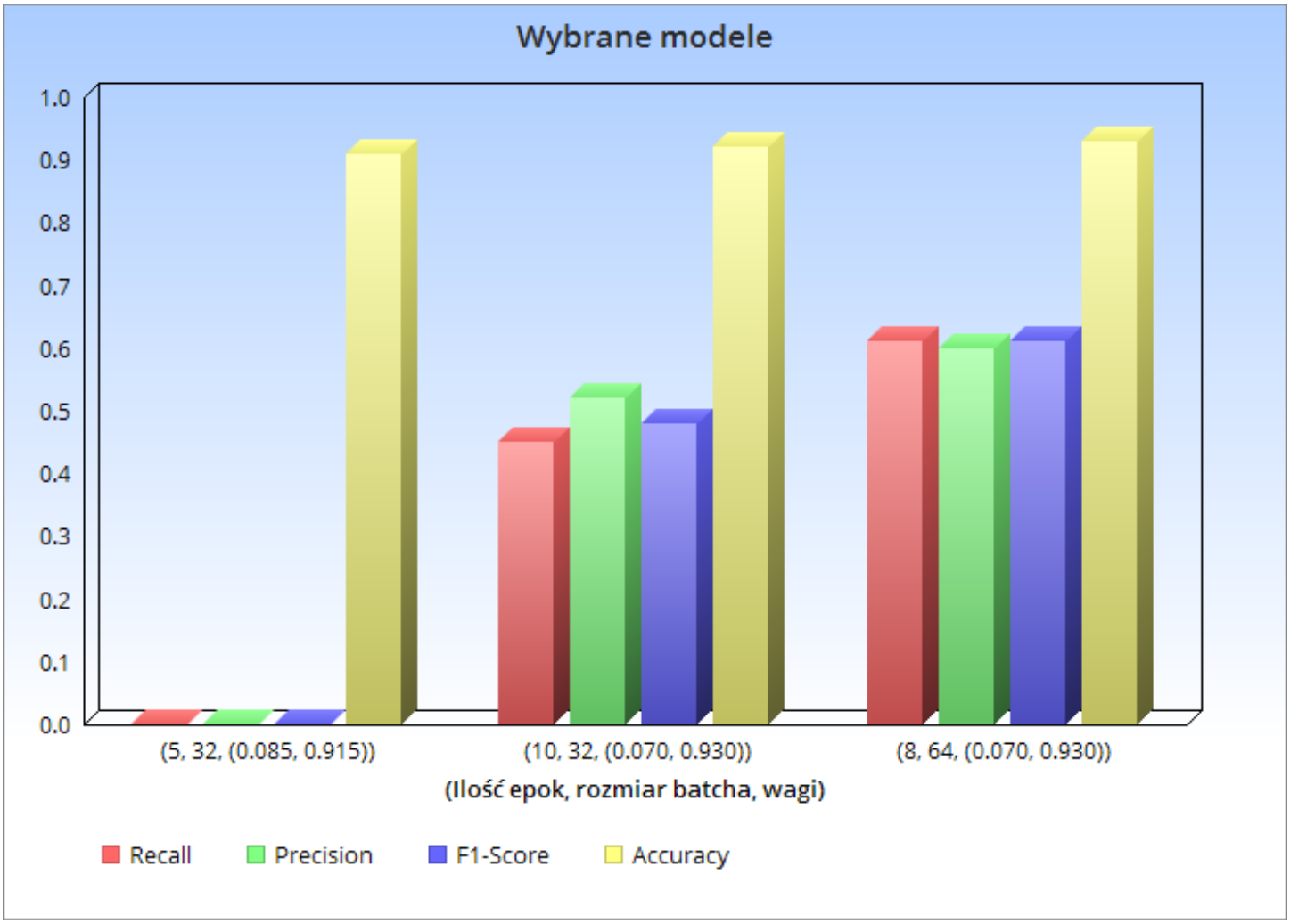
Wagi klas warto modyfikować w przypadku, gdy jedna z klas dominuje nad drugą. W takich przypadkach możemy "wyczulić" model na mniej występująca klasę ustawiając większą wagę tej klasy. Nie zakładamy żadnych szczególnych wyników, sprawdzimy co różne wartości wag zmienią w naszym modelu. Eksperyment przeprowadzimy dla wartości wag: $(0.1, 0.9)$, $(0.085, 0.915)$, $(0.070, 0.930)$. Wyniki widoczne poniżej.



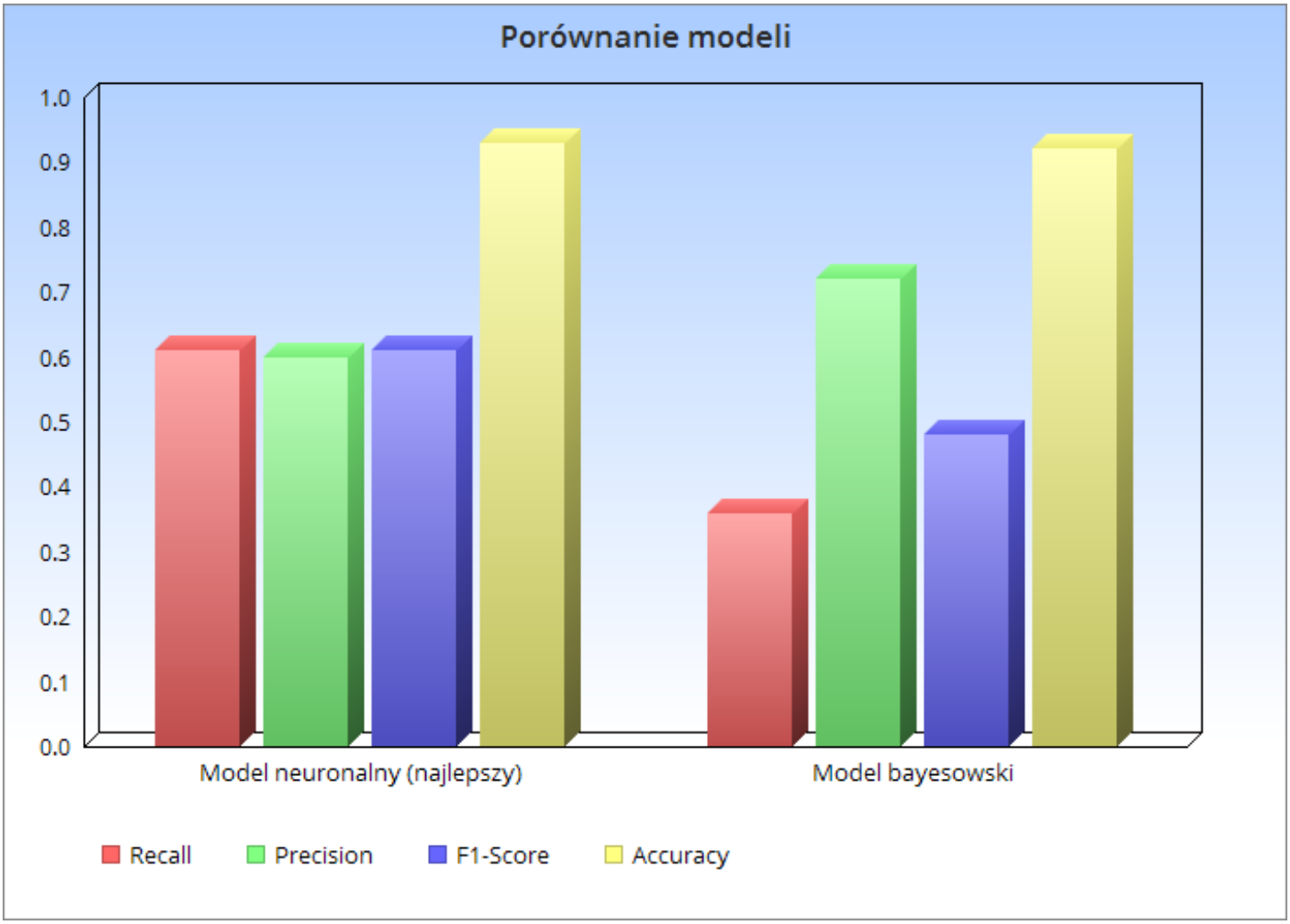
Wyniki okazały się identyczne (wszystkie parametry są identyczne oprócz eval_loss co świadczy o różnicy pomiędzy modelami). Co więcej, każdy z modeli jest tak samo słaby jak wcześniej (nie odgaduje NIC). Eksperyment nie doprowadził nas zatem do polepszenia naszego modelu.
Widać więc, że dla małych, prostych modeli zmiany (tylko) wag poszczególnych klas nie polepszyły naszego modelu.

3.4 Modyfikacja wszystkich parametrów

Wykonane eksperymenty nie polepszyły naszego modelu. Spróbujemy więc zmodyfikować wszystkie parametry naraz. Spróbujemy następujących konfiguracji:
(Ilość epok treningowych, rozmiar batcha, wagi klas)
Konfiguracja 1 = (5, 32, (0.085, 0.915))
Konfiguracja 2 = (10, 32, (0.070, 0.930))
Konfiguracja 3 = (8, 64, (0.070, 0.930))



Jak widać eksperyment okazał się bardzo udany. Konfiguracja 1 nie polepszyła naszego modelu natomiast konfiguracja 2 oraz 3 znacząco podniosły poprawność naszej sieci. Z tego płyną wnioski:
- należy znaleźć odpowiednie wagi klas, takie, które wyczulą nasz model na klasę rzadziej występującą
- ilość epok treningowych musi być wystarczająco duża, żeby nasz model zaczął poprawnie działać
- wzrost rozmiaru batcha znacząco polepsza wyniki naszego modelu co widać na wykresie powyżej
Poniżej widać porównanie wyników dla modelu bayesowskiego oraz najlepszego modelu neuronalnego:



Widać więc, że odpowiednio wytrenowany model neuronalny potrafi być lepszy od prostego modelu bayesowskiego.
Model neuronalny okazał się zatem lepszy niż model bayesowski.

4 Podsumowanie

Wykonane ćwiczenie poznało dobrze zapoznać się z tematem klasyfikacji tekstu oraz pojęciami z nim związanymi. Pozwoliło również praktycznie przekonać się o znaczeniu poszczególnych parametrów modelu neuronalnego oraz budowaniu prostych modeli bayesowskich, które dla prostych przypadków służą równie dobrze co modele neuronalne.