

Piotr Chmiel, 200608

Kamil Machnicki, 200752

Łukasz Matysiak, 200646

Michał Polański, 200852

Maciej Stelmaszuk, 200654

Jakub Zgraja, 200609

# **Deeplearning – tagger**

**(POS, lematyzacja)**

## **DOKUMENTACJA PROJEKTOWA**

Kurs „Zastosowania informatyki w gospodarce”  
Rok akad. 2015/2016, kierunek INF, studia II stopnia

PROWADZĄCY:  
dr inż. Tomasz Walkowiak

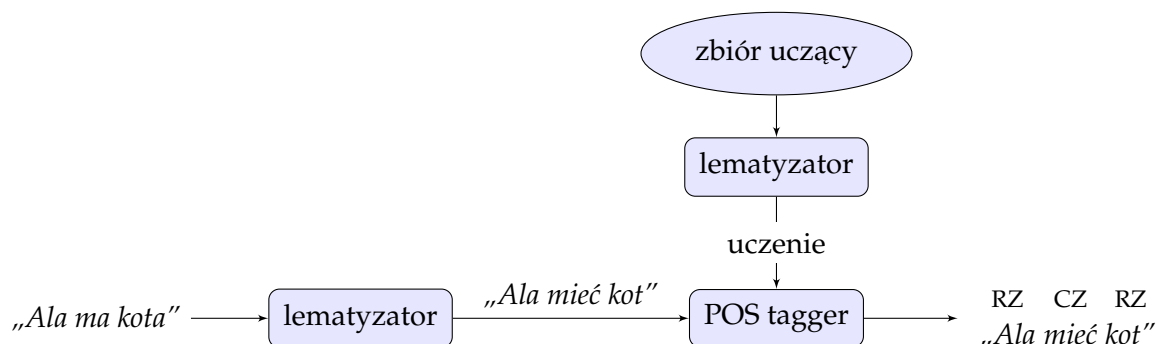
# Spis treści

<b>1</b>	<b>Cel projektu</b>	<b>3</b>
<b>2</b>	<b>Koszty</b>	<b>3</b>
<b>3</b>	<b>Terminy i harmonogram projektu</b>	<b>4</b>
<b>4</b>	<b>Przygotowanie środowiska</b>	<b>4</b>
4.1	Wirtualne środowisko (opcjonalnie)	4
4.2	Instalacja pakietów	5
4.3	Instalacja korpusów	5
4.4	Instalacja NLTK	5
4.5	Instalacja CUDA	5
<b>5</b>	<b>Korzystanie z programu</b>	<b>6</b>
5.1	Konfiguracja środowiska	6
5.2	Ustawienia ścieżek	6
5.3	Konwersja plików xml do csv	6
5.4	Tworzenie bazy końcówek	6
5.5	Tagger - uczenie klasyfikatorów	6
5.6	Testowanie taggera	6
5.7	Uruchomienie benchmarka	7
<b>6</b>	<b>Opis danych</b>	<b>7</b>
6.1	Sposób wybierania cech	7
6.1.1	Zaimplementowana metoda wybierania cech	7
6.1.2	Przykład macierzy cech	7
6.2	Wykorzystane klasyfikatory	10
<b>7</b>	<b>Przetwarzanie danych</b>	<b>10</b>
7.1	Konwersja plików XML do formatu CSV	10
7.2	Mapowanie klas NKJP	11
<b>8</b>	<b>Wyniki pomiarów działania programu</b>	<b>11</b>
8.1	Dokładność modeli w bazie	11
8.2	Pomiary czasów uczenia	14
8.3	Pomiary czasów klasyfikacji	15

# 1 Cel projektu

**Temat projektu:** *Deeplearning – tagger (POS, lematyzacja).*

Zadaniem realizowanego przedsięwzięcia jest stworzenie programu głębokiego uczenia (ang. *deep learning*) z zakresu przetwarzania języka naturalnego. Jego celem jest przypisywanie każdemu wyrazowi w tekście wejściowym odpowiadającej mu części mowy (ang. *part-of-speech, POS*). Wykonanie zadania tagowania zostanie poprzedzone procesem lematyzacji tekstu.



Rysunek 1: Uproszczony schemat działania programu.

Realizacja powinna posiadać formę aplikacji desktopowej lub skryptu. Cel projektu zostanie osiągnięty, jeśli przygotowane oprogramowanie (wyposażone w odpowiedni zbiór uczący) będzie w stanie przetwarzać tekst w języku polskim w czasie i dokładności, które zostaną sprecyzowane przez prowadzącego.

## 2 Koszty

### Szacunkowy czas wykonania

Biorąc pod uwagę zakres tematyczny projektu szacuję się, że do jego wykonania potrzebnych jest 400 godzin roboczych. Około 1/8 czasu poświęcone będzie na zebranie i analizę materiałów dotyczących projektu, ogólne poznanie możliwości i technologii. Największa część czasu zużyta zostanie na implementację projektu (stworzenie lematyzatora oraz taggera). Szacuje się, że będzie to 6/8 czasu. Pozostała część przeznaczona jest na kontakty z prowadzącym oraz testy zaimplementowanego rozwiązania.

### Szacunkowy koszt projektu

Mając na uwadze poziom skomplikowania zadania projektowego stawka za godzinę pracy wynosi 150 zł. Przy szacowaniu kosztów projektu nie uwzględniamy dodatkowych wydatków, które trzeba będzie ponieść w przypadku, gdy np. zajdzie potrzeba wykupienia domeny WWW. Nie są uwzględnione także koszty wdrożenia projektu w środowisku produkcyjnym. W związku z powyższym szacunkowy koszt wykonania projektu jest równy 60 000 zł netto.

Usługi, o których mowa w poprzednich punktach opodatkowane są 23% stawką podatku VAT (podatek od towarów i usług). Szacowana cena brutto wynosi **73 800 zł**.

Tabela 1: Kosztorys.

Nazwa	Jedn.	Ilość	Cena jedn.	Wart. netto	Stawka	Podatek	Wart. brutto
Robocizna	r-g	400	150 zł	60 000 zł	23%	13 800 zł	73 800 zł
<b>Razem:</b>							<b>73 800 zł</b>

### 3 Terminy i harmonogram projektu

Granicznym terminem realizacji projektu jest **7 czerwca 2016 r.** W tym dniu nastąpi prezentacja rezultatów projektu wraz z przekazaniem jego pełnej dokumentacji.

Tabela 2: Harmonogram projektu.

Data	Opis
23.02.2016	Wybór tematu projektu.
1.03.2016	Określenie zakresu tematycznego projektu.
8.03.2016	Przekazanie specyfikacji projektu (wstępna funkcjonalność, określenie kamieni milowych).
9.03.2016 – 06.06.2016	Konsultacja wyników pracy po osiągnięciu kolejnych kamieni milowych projektu.
7.06.2016	Prezentacja rezultatów projektu i przekazanie dokumentacji projektowej.

Tabela 3: Kamienie milowe.

Data	Opis
29.03.2016	Instalacja środowiska
19.04.2016	Stworzenie lematyzatora
10.05.2016	Stworzenie taggera
31.05.2016	Stworzenie instrukcji użytkownika
07.06.2016	Prezentacja projektu

## 4 Przygotowanie środowiska

Wymagany Python 3.5.

### 4.1 Wirtualne środowisko (opcjonalnie)

```
$ python3 -m venv env          # instalacja venv
$ source env/bin/activate      # aktywacja venv
```

... praca w wirtualnym środowisku ...

```
$ deactivate # deaktywacja venv
```

## 4.2 Instalacja pakietów

Podczas instalacji pakietów pod Linuksem, potrzebny jest kompilator `gcc-fortran`.

```
$ pip3 install -r requirements.txt
```

## 4.3 Instalacja korpusów

Należy uruchomić skrypt instalacyjny, który rozpakowuje korpus PWr z pliku `kpwr-1.2.6-disamb.7z` a korpus polski pobiera ze źródła.

```
$ ./installCorpuses.sh
```

## 4.4 Instalacja NLTK

Do tokenizacji słów w podanym tekście użyto NLTK. Dwie opcje instalacji:

1. Interaktywna instalacja w interpreterze:

```
>>> import nltk
>>> nltk.download()
```

2. Instalacja poprzez linię komend:

```
$ sudo python -m nltk.downloader -d /usr/local/share/nltk_data all
```

Dane NLTK zostaną zainstalowane w katalogu `/usr/local/share/nltk_data`.

## 4.5 Instalacja CUDA

1. Pobranie paczki instalującej repozytorium Nvidii ze strony: <https://developer.nvidia.com/cuda-downloads>, testowana wersja: Linux Ubuntu 14.04, architektura x86\_64, paczka deb (local).
2. Instalacja repozytorium w systemie:

```
$ dpkg -i cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb`
```

3. Instalacja sterowników i środowiska CUDA:

```
$ apt-get update && apt-get install -y cuda
```

4. Restart maszyny w celu załadowania sterowników Nvidii zamiast Nouveau.
5. Instalacja Nvidia cuDNN (biblioteki wspomagające sieci neuronowe): należy umieścić zawartość archiwum `cudnn-7.0-linux-x64-v4.0-prod.tgz` w folderze `/usr/local/cuda`. Do ściągnięcia ze strony <https://developer.nvidia.com/rdp/form/cudnn-download-survey>.
6. Instalacja modułu tensorflow dla Pythona.

## 5 Korzystanie z programu

### 5.1 Konfiguracja środowiska

Należy dodać folder zawierający projekt do zmiennej środowiskowej `PYTHONPATH`. Jeśli akurat się w nim znajdujemy (jest on katalogiem bieżącym), można to zrobić np. poprzez:

```
$ export PYTHONPATH="${PYTHONPATH}:${PWD}"
```

Aby wykonać powyższą komendę wraz z włączeniem `venv`, wystarczy pobrać zawartość pliku `prepare` do shella poprzez:

```
$ source prepare
```

### 5.2 Ustawienia ścieżek

Skrypty korzystają ze ścieżek konfigurowalnych za pomocą pliku `src/settings.py`.

### 5.3 Konwersja plików xml do csv

Projekt zawiera skrypt umożliwiający konwersję plików xmlowych do formatu csv:

```
$ python3 src/scripts/csv_creator.py -h
usage: csv_creator.py [-h] (--use_pwr | --use_national) [--extract_base_words]
optional arguments:
-h, --help            show this help message and exit
--use_pwr             Use pwr corpus
--use_national        Use national corpus
--extract_base_words  Save words with their base form
```

### 5.4 Tworzenie bazy końcówek

Do stworzenia bazy końcówek służy plik `src/scripts/suffix_creator.py`.

### 5.5 Tagger - uczenie klasyfikatorów

```
$ python3 src/tagger_trainer.py -h # więcej o ustawianiu liczby rdzeni
$ python3 src/tagger_trainer.py
```

Domyślnie podczas uczenia używane są wszystkie rdzenie procesora - jeden rdzeń na algorytm. Logi związane z uczeniem zapisywane są do pliku `tagger_factory.log`.

### 5.6 Testowanie taggera

```
$ python3 src/tagger_tester.py # pojedyncze słowo
$ python3 src/text_tagger.py   # tekst
```

Skrypt zapyta się o słowo/tekst do klasyfikacji.

## 5.7 Uruchomienie benchmarka

```
$ python3 src/benchmark.py
```

## 6 Opis danych

System operuje na danych wejściowych, którymi są słowa w języku polskim. Istnieje możliwość wprowadzenia zarówno pojedynczego słowa, jak i wielu (np. epitety czy całe zdania).

### 6.1 Sposób wybierania cech

Cechy wybierane są na podstawie budowy słowa oraz kontekstu, w którym zostało użyte. W celu wyekstrahowania cech ze słowa, system bada  $N$  ostatnich liter wyrazu oraz to, jakie części zdania wystąpiły wcześniej.

W zależności od wyboru użytkownika, system potrafi zebrać dane uczące z korpusu PWr lub z korpusu podmilionowego.

#### 6.1.1 Zaimplementowana metoda wybierania cech

Metoda ekstrakcji cech zaimplementowana w systemie opiera się na wyodrębnianiu końcówek wyrazów.

Możliwe jest wybranie od jednej do czterech ostatnich liter, z których następnie jest tworzona macierz cech – wybierane są najczęściej występujące końcówki, wraz z etykietą, którą część mowy stanowią.

Liczba najczęściej występujących końcówek zależy od ich długości:

- 15 najczęstszych końcówek jednoliterowych,
- 35 najczęstszych końcówek mających od dwóch do czterech liter.

Oprócz końcówek wyrazów, dla zdań występujących w korpusach, podawane są etykiety poprzednich dwóch wyrazów.

#### 6.1.2 Przykład macierzy cech

Przykładowy zbiór uczący:

- jabłkami – rzeczownik,
- niepoważny – przymiotnik,
- kierunku – rzeczownik,
- istnieje – czasownik,
- wolność – rzeczownik,

Podane wyrazy nie posiadają danych na temat poprzednich etykiet wyrazów. Można wyróżnić tutaj następujące końcówki:

- i – rzeczownik,
- y – przymiotnik,
- u – rzeczownik,
- e – czasownik,
- ć – rzeczownik,
- mi – rzeczownik,

- ny – przymiotnik,
- ku – rzeczownik,
- je – czasownik,
- ść – rzeczownik,
- ami – rzeczownik,
- żny – przymiotnik,
- nku – rzeczownik,
- eje – czasownik,
- ość – rzeczownik,
- kami – rzeczownik,
- ażny – przymiotnik,
- unku – rzeczownik,
- ieje – czasownik,
- ność – rzeczownik.

Etykiety są zamieniane na format wykorzystywany przez pakiet `scikit-learn` przy pomocy klasy `DictVectorizer`:

- rzeczownik → 0
- przymiotnik → 1
- czasownik → 2
- — (b.d.) → 3



Dla podanego zbioru uczącego można wyróżnić macierz cech (końcówki liter → czy dany wyraz kończy się na daną końcówkę):

Tabela 4: Przykładowa macierz cech.

	i	y	u	e	ć	mi	ny	ku	je	ść	ami	żny	nku	eje	ość	kami	ażny	unku	ieje	ność	etykieta w. n-1	etykieta w. n-2	etykieta
jabłkami	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	3	3	0
niepoważny	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	3	3	1
kierunku	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	3	3	0
istnieje	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	3	3	2
wolność	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	3	3	0

Legenda:

0 – brak danej cechy w obiekcie

1 – dana cecha występuje dla obiektu

## 6.2 Wykorzystane klasyfikatory

Projekt wykorzystuje wiele klasyfikatorów. Wszystkie klasyfikatory z pakietu `scikit-learn` podczas procesu uczenia używają domyślnych parametrów. Więcej informacji można uzyskać w dokumentacji pakietu, dostępnej pod adresem: <http://scikit-learn.org/stable/modules/classes.html>.

Tabela 5: Wykorzystane klasyfikatory.

Nazwa	Opis
<code>DecisionTreeClassifier</code>	Klasyfikator oparty na drzewie decyzyjnym.
<code>SGDClassifier</code>	Klasyfikator oparty na liniowym modelu z metodą stochastycznego gradientu prostego.
<code>SVC</code>	Klasyfikator reprezentuje model maszyny wektorów nośnych.
<code>LogisticRegression</code>	Klasyfikator oparty na regresji logistycznej.
<code>BernoulliNB</code>	Klasyfikator wykorzystujący model naiwnego klasyfikatora bayesowskiego z rozkładem Bernoulliego.
<code>KNeighborsClassifier</code>	Klasyfikator implementujący mechanizm $k$ -najbliższych sąsiadów.
Sieć neuronowa	Klasyfikator wykorzystujący sieć neuronową opartą na bibliotece <code>sknn</code> . Posiada dwie warstwy: <code>Rectifier</code> (z liczbą 100 neuronów) i <code>Softmax</code> . Współczynnik uczenia wynosi 0,01, a liczba iteracji wynosi 10. Sieć neuronowa wykorzystująca kartę graficzną – klasyfikator, podobnie jak powyższy wykorzystuje sieć neuronową, jednakże do implementacji wykorzystano bibliotekę <code>TensorFlow</code> , ponieważ wspiera wykorzystanie karty graficznej. Podobnie jak w poprzednim klasyfikatorze, sieć ta posiada dwie warstwy – <code>Gated Recurrent Unit</code> z liczbą 50 neuronów oraz warstwę regresji logistycznej, współczynnik uczenia również wynosi 0.01, liczba iteracji wynosi 1000. Klasyfikator wykorzystuje klasę biblioteki <code>TensorFlow</code> – <code>tf.nn.rnn</code> – rekurencyjną sieć neuronową. Ze względu na ograniczoną pamięć kart graficznych, dane uczące dzielone są na części, domyślnie po 10 tysięcy próbek.

## 7 Przetwarzanie danych

### 7.1 Konwersja plików XML do formatu CSV

Ze względu na nieefektywność przetwarzania plików XML oraz na dużą ilość niepotrzebnych danych znajdujących się w korpusach, postanowiono przetworzyć je do formy plików CSV. Proces zamiany plików XML odbywa się dla obydwu korpusów – PWr oraz podkorpusu milionowego. Niestety, ze względu na rozbieżność struktur, nie mogą one być przetworzone w ten sam sposób.

Do pliku CSV zapisywane są dwie informacje – słowo oraz część mowy. Wyrażenia XPath służące do wydobywania tych danych z korpusu PWr wyglądają następująco: `//tok/orth` – dla wyrazu, oraz `//tok/lex[disamb='1']` dla części mowy. Dla podkorpusu milionowego zapytania XPath prezentują się w sposób następujący: `//f[name='orth']/string` dla wyrazu, oraz `//f[name='disamb']/f[name='interpretation']/string` dla części mowy.

## 7.2 Mapowanie klas NKJP

Oba korpusy wykorzystują zestaw znaczników morfosyntaktycznych NKJP, który wyróżnia 36 rodzajów klas. Na potrzeby projektu liczba ta została ograniczona do 10 podstawowych części mowy. Poniżej zostało przedstawione mapowanie z klas NKJP na uproszczony model.

Tabela 6: Mapowanie klas NKJP.

rzeczownik	←	rzeczownik, rzeczownik deprecjatywny.
liczebnik	←	liczebnik główny, liczebnik zbiorowy.
przymiotnik	←	przymiotnik, przymiotnik przyprzym., przymiotnik poprzyimkowy, przymiotnik predykatywny.
przysłówek	←	przysłówek.
zaimek	←	zaimek nietrzecioosobowy, zaimek trzecioosobowy, zaimek siebie.
czasownik	←	forma nieprzeszła, forma przyszła być, aglutynant być, pseudoimiesłów, rozkaznik, bezosobnik, bezokolicznik, im. przys. współczesny, im. przys. uprzedni, odsłownik, im. przym. czynny, im. przym. bierny, winien, predykatyw.
przyimek	←	przyimek, wykrzyknik.
spójnik	←	spójnik współrzędny, spójnik podrzędny.
partykuła	←	kublik.
forma nierozpoznana	←	skrót, burkinostka, interpunkcja, ciało obce, forma nierozpoznana.

## 8 Wyniki pomiarów działania programu

### 8.1 Dokładność modeli w bazie

Badania przeprowadzono dla różnych liczb n-literowych końcówek wyrazów.

Aby lepiej zobrazować otrzymane wyniki użyto notacji:

(liczba 1 literowych końcówek, liczba 2 literowych końcówek, liczba 3 literowych końcówek, liczba 4 literowych końcówek).

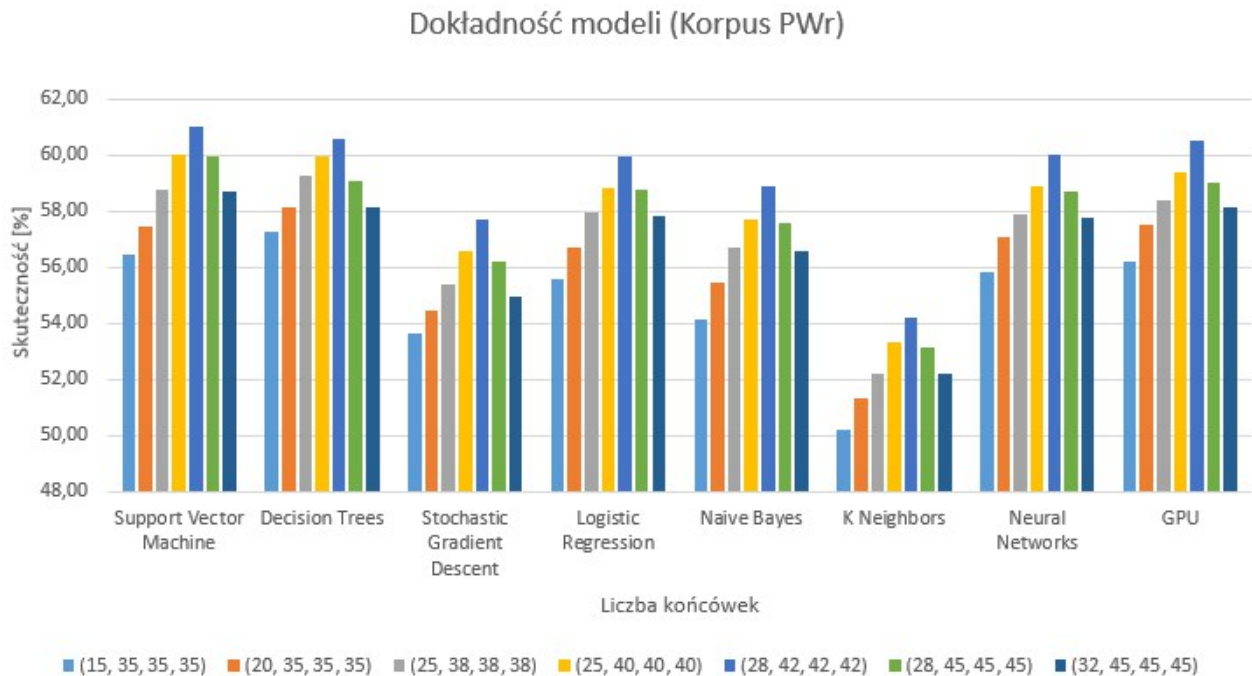
Tabela 7: Skuteczność modeli – Korpus PWt.

Algorytm	(15, 35, 35, 35)	(20, 35, 35, 35)	(25, 38, 38, 38)	(25, 40, 40, 40)	(28, 42, 42, 42)	(28, 45, 45, 45)	(32, 45, 45, 45)
Support Vector Machine	56,46	57,46	58,75	60,03	61,01	59,94	58,68
Decision Trees	57,24	58,12	59,30	59,97	60,58	59,10	58,14
Stochastic Gradient Descent	53,62	54,48	55,37	56,58	57,72	56,22	54,97
Logistic Regression	55,57	56,68	57,93	58,82	59,93	58,79	57,84
Naive Bayes	54,14	55,42	56,69	57,72	58,89	57,58	56,56
K Neighbors	50,17	51,35	52,18	53,30	54,22	53,12	52,20
Neural Networks	55,80	57,08	57,91	58,88	60,04	58,72	57,75
Klasyfikator GPU	56,24	57,46	58,34	59,23	60,51	59,1	58,22

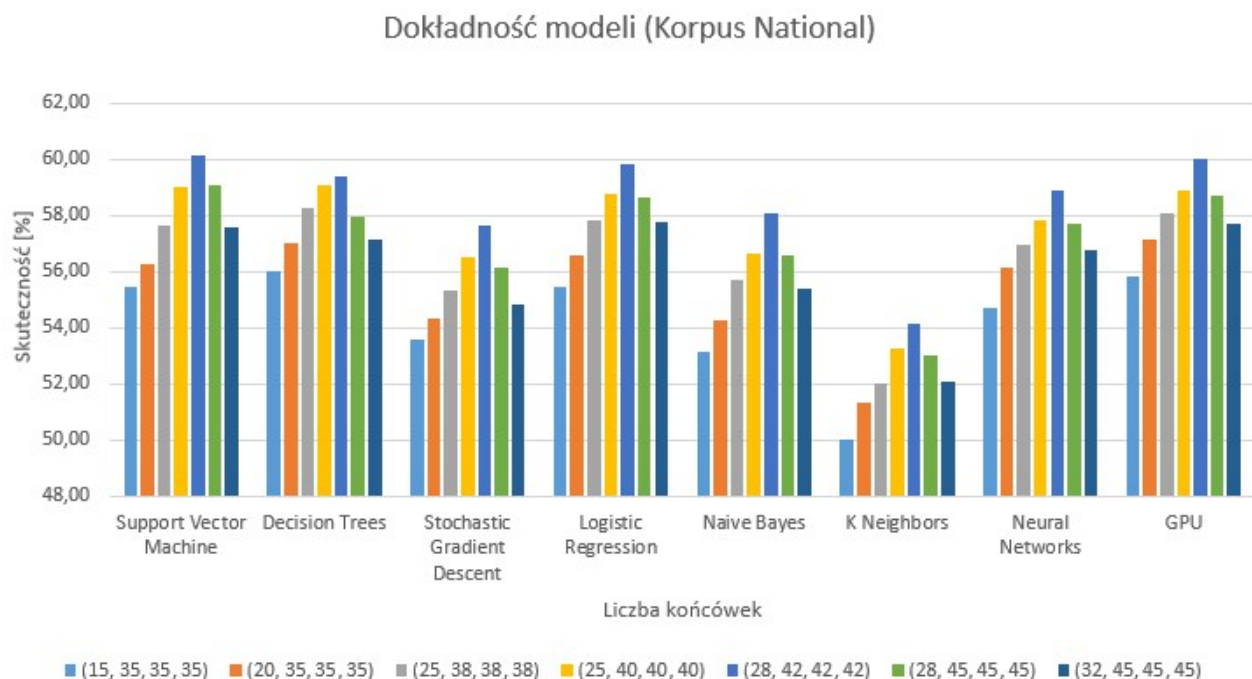
Tabela 8: Skuteczność modeli – korpus National.

Algorytm	(15, 35, 35, 35)	(20, 35, 35, 35)	(25, 38, 38, 38)	(25, 40, 40, 40)	(28, 42, 42, 42)	(28, 45, 45, 45)	(32, 45, 45, 45)
Support Vector Machine	55,43	56,26	57,63	59,05	60,13	59,07	57,55
Decision Trees	56,04	57,03	58,25	59,10	59,38	57,97	57,15
Stochastic Gradient Descent	53,56	54,35	55,30	56,53	57,62	56,16	54,85
Logistic Regression	55,48	56,55	57,82	58,76	59,86	58,64	57,77
Naive Bayes	53,11	54,28	55,70	56,61	58,06	56,59	55,39
K Neighbors	50,02	51,29	52,03	53,25	54,13	53,00	52,08
Neural Networks	54,69	56,17	56,95	57,81	58,87	57,73	56,77
Klasyfikator GPU	56,04	57,3	58,02	58,93	60,22	58,88	57,96

Najlepszym klasyfikatorem okazał się być Support Vector Machine, a najgorszym K Neighbors. Konfiguracje końcówek, w których liczba dwu i więcej literowych końcówek była mniejsza niż 35 nie dawały satysfakcjonujących rezultatów. (co najmniej 50% skuteczności) Dla konfiguracji, w których liczba tych końcówek przekracza 42 zauważono trend ku obniżaniu się skuteczności klasyfikacji dlatego też zaprzestano dalszych badań.



Rysunek 2: Skuteczność modeli – korpus PWr.



Rysunek 3: Skuteczność modeli – korpus National.

## 8.2 Pomiary czasów uczenia

Tabela 9: Pomiary czasów uczenia.

Algorytm	Korpus uczący PWr (h:m:s)	Korpus uczący National (h:m:s)
Support Vector Machine	0:44:52	16:35:24
Decision Trees	0:01:22	0:05:48
Stochastic Gradient Descent	0:01:20	0:04:56
Logistic Regression	0:01:32	0:07:10
Naive Bayes	0:01:18	0:04:32
K Neighbors	0:01:18	0:05:20
Neural Networks	0:20:28	1:21:04
GPU Classifier	0:14:55	—

Do testów został użyty procesor Intel(R) Core(TM) i7-4930K CPU @ 3.40GHz. Jest to wydajna jednostka z 2013 roku pracująca na 6 rdzeniach w technologii Hyper-Threading (12 wątków). W przypadku klasyfikatora opartego na GPU została wykorzystana karta graficzna NVIDIA GeForce GT 740 (chip GK107) – model z średniej półki z 2014 roku.

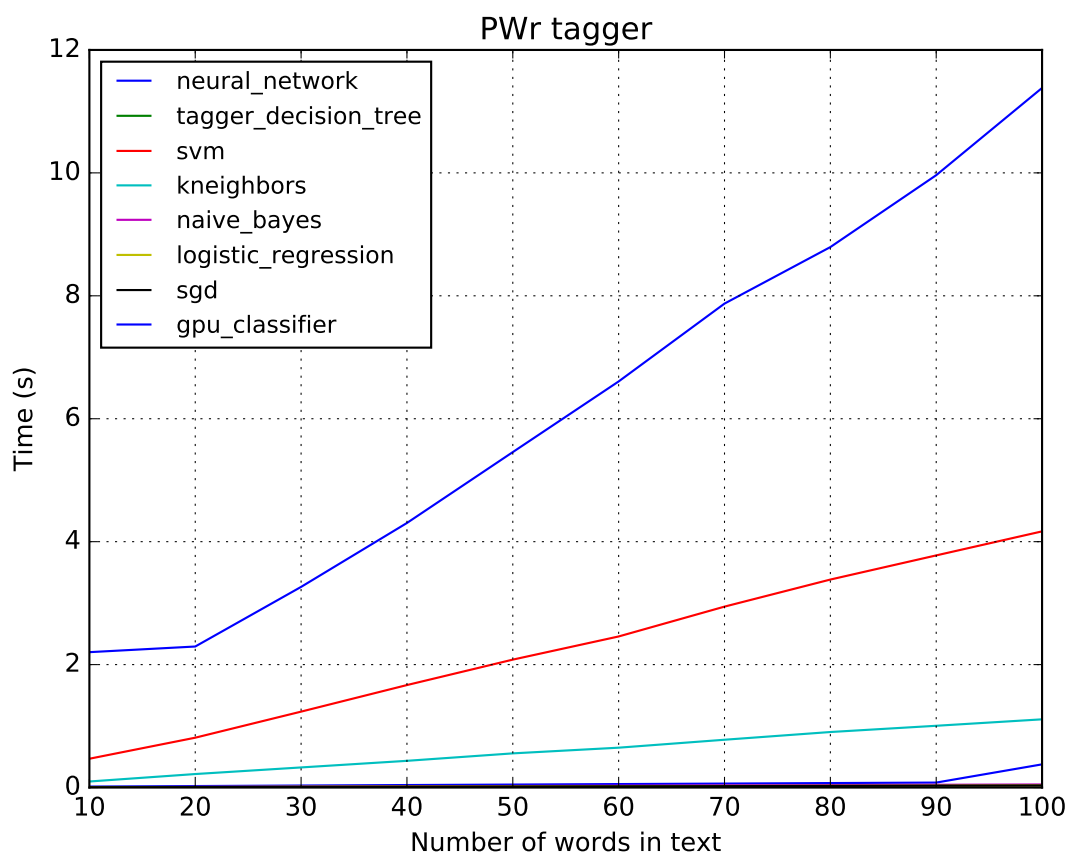
W testach każdy algorytm korzystał z jednego wątku procesora.

Uczenie algorytmów: drzew decyzyjnych, stochastycznego spadku gradientu, regresji logistycznej, naiwnego klasyfikatora bayesowskiego oraz  $k$ -najbliższych sąsiadów zakończyło się na korpusie uczącym PWr w niewiele ponad minutę. Daje to możliwość wielokrotnego zmieniania parametrów tych algorytmów w poszukiwaniu lepszych wyników. Uczenie sieci neuronowych trwało zdecydowanie dłużej od wcześniej wymienionych, natomiast maszyny wektorów nośnych okazały się prawdziwym wyzwaniem dla procesora w postaci wielogodzinnych obliczeń modelu danych. Uczenie klasyfikatorów za pomocą korpusu National, ze względu na wielkość danych uczących, jest kilka razy bardziej czasochłonne w porównaniu do korpusu PWr.

### 8.3 Pomiary czasów klasyfikacji

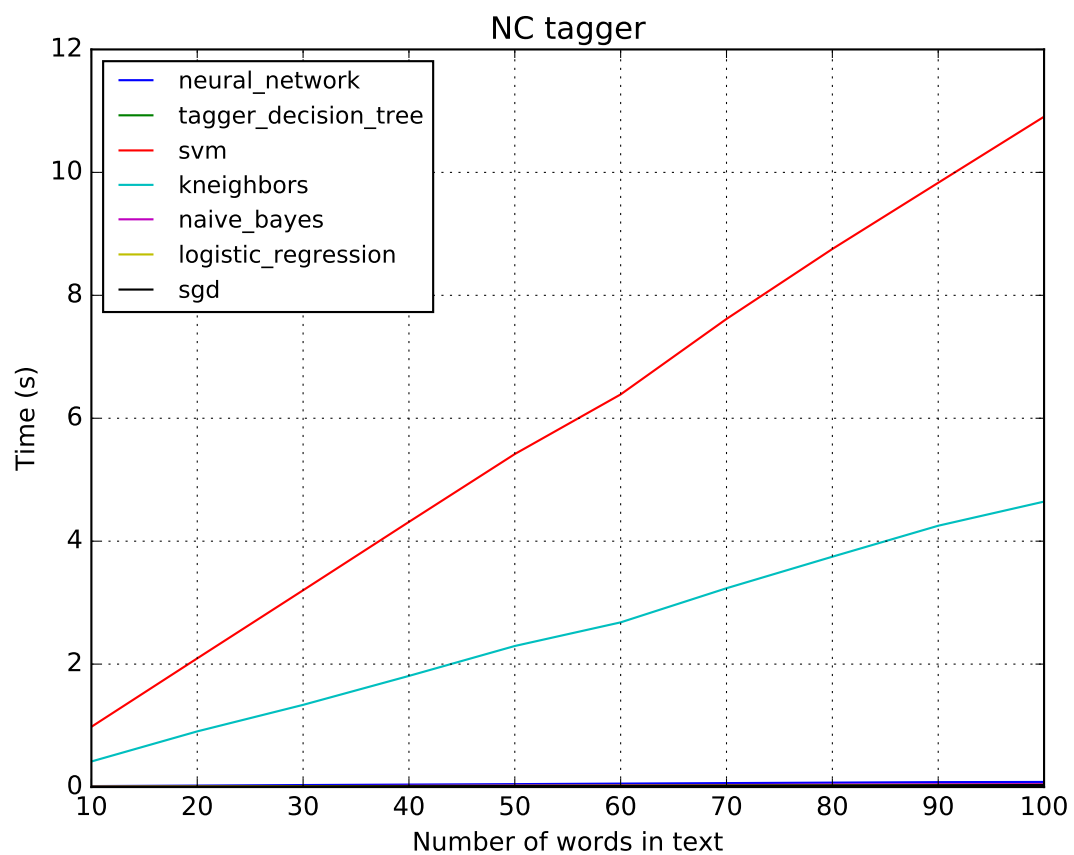
Pomiary czasów klasyfikacji przeprowadzono na próbie 10 zbiorów tekstów zawierających kolejno od 10 do 100 wyrazów. Wyniki rozdzielono na osobne wykresy przedstawiające odpowiednio klasyfikatory nauczone na korpusie PWr oraz na korpusie podmilionowym.

Zauważyć można, iż na pierwszym wykresie przedstawiającym korpus PWr, najdłuższym czasem klasyfikacji wyróżnił się ten wykorzystujący kartę graficzną, osiągając przy 100 wyrazach czas 11,381 s. Dwukrotnie lepszy czas osiągnął algorytm SVM, osiągając 4,167 s. Trzecim, dziesięciokrotnie szybszym od pierwszego był algorytm K Neighbors – 1,109 s. Reszta algorytmów może się pochwalić bardzo niskimi czasami, rzędu setnych sekundy, dlatego też na wykresie są prawie niewidoczne.



Rysunek 4: Czasy klasyfikacji w zależności od ilości słów w tekście – korpus PWr.

W przypadku korpusu podmilionowego sytuacja jest bardzo podobna z tą różnicą, iż pominięty został algorytm bazujący na karcie graficznej albowiem nie można go było uruchomić na udostępnionej maszynie testowej z powodu niewystarczającej ilości pamięci operacyjnej.



Rysunek 5: Czasy klasyfikacji w zależności od ilości słów w tekście – korpus National.