Grupa projektowa: **WT 9:15** Wrocław, 28 maja 2016r.

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

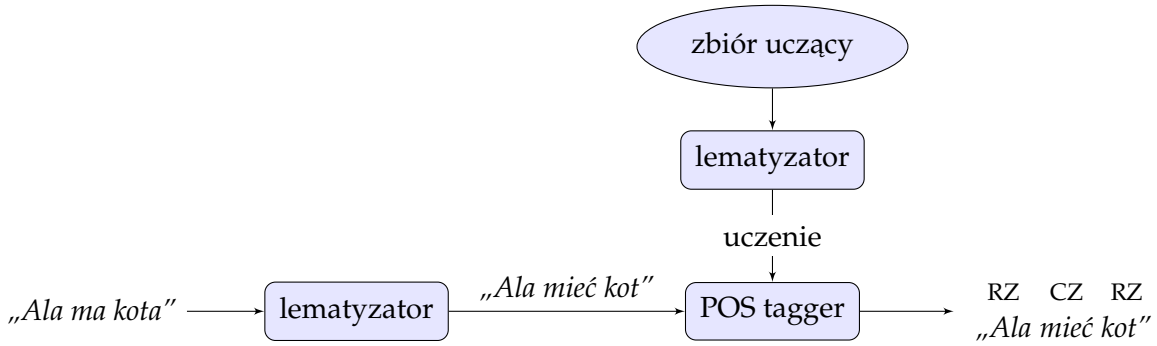
PROWADZĄCY:
dr inż. Tomasz Walkowiak

Spis treści

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
						Razem:	**73 800 zł**

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 użytkowania
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 rozpakowywuje 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 Przetwarzanie danych

6.1 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 wydobycia 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.

6.2 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 4: 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, rozkaźnik, bezosobnik, bezokolicznik, im. przys. współczesny, im. przys. uprzedni, odsłownik, im. przym. czynny, im. przym. biern, winien, predykatyw.
`przyimek`	←	przyimek, wykrzyknik.
`spójnik`	←	spójnik współrzędny, spójnik podrzędny.
`partykuła`	←	kublik.
`forma nierozpoznana`	←	skrót, burkinostka, interpunkcja, ciało obce, forma nierozpoznana.

7 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).

7.1 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.

7.1.1 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.

7.1.2 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 $\rightarrow$ 0
- przymiotnik $\rightarrow$ 1
- czasownik $\rightarrow$ 2
- — (b.d.) $\rightarrow$ 3

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

Tabela 5: 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

7.2 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 6: Wykorzystane klasyfikatory.

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

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).

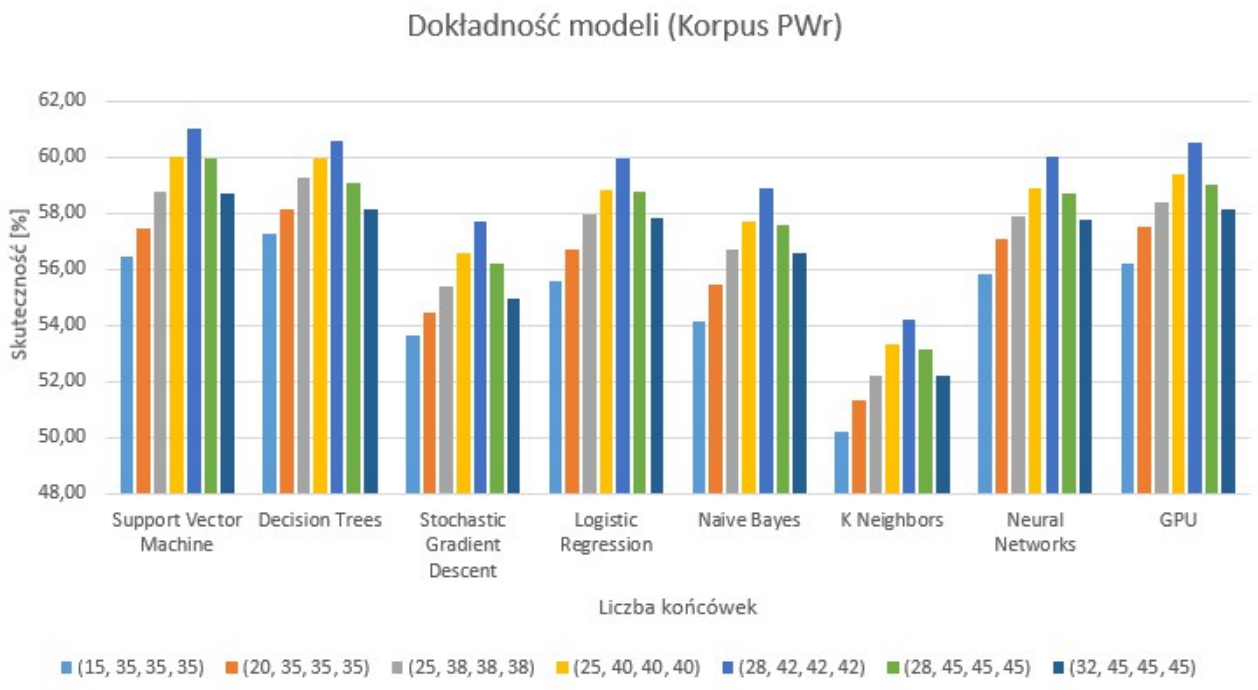
Tabela 7: Skuteczność modeli – korpus PWr.

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

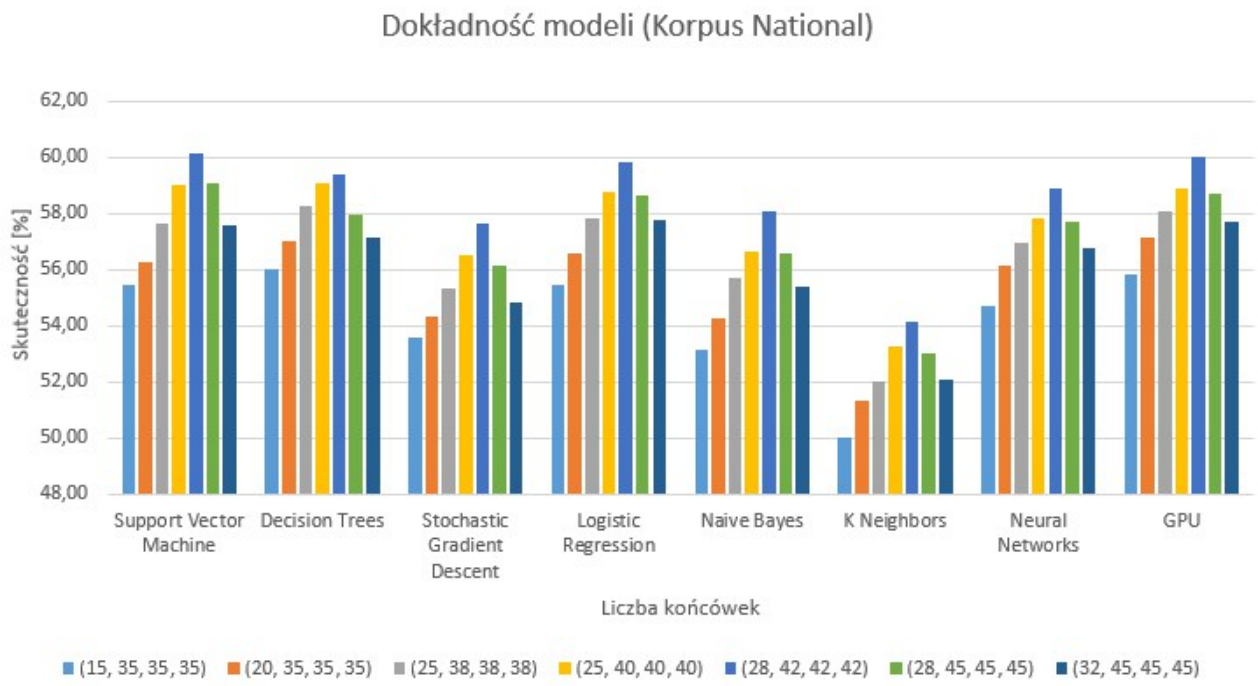
Tabela 8: Skuteczność modeli – korpus National.

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	23:29:53
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 technologi 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 prawdziwnym 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 14 zbiorów tekstów zawierających kolejno od 10 do 100 wyrazów z przeskokiem co 10, oraz większe rozmiary: 250, 500, 1000, 2000. Wyniki rozdzielono na osobne grupy przedstawiające odpowiednio klasyfikatory nauczone na korpusie PWr oraz na korpusie podmilionowym.

Tabela 10: Czasy klasyfikacji w zależności od ilości słów w tekście – Korpus PWr.

Algorytm	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**250**	**500**	**1000**	**2000**
Logistic Regression	0.0076	0.0081	0.0114	0.0154	0.0185	0.0215	0.0242	0.0280	0.0305	0.0340	0.0790	0.1580	0.3160	0.4736
Stochastic Gradient Descent	0.0030	0.0064	0.0097	0.0140	0.0171	0.0205	0.0232	0.0268	0.0301	0.0336	0.0803	0.1611	0.3206	0.4812
Decision Trees	0.0039	0.0078	0.0123	0.0167	0.0203	0.0244	0.0277	0.0319	0.0359	0.0399	0.0953	0.1909	0.3814	0.5734
Naive Bayes	0.0046	0.0097	0.0148	0.0210	0.0259	0.0310	0.0355	0.0408	0.0458	0.0509	0.1225	0.2460	0.4902	0.7358
Neural Networks	0.3552	0.0157	0.0244	0.0329	0.0408	0.0484	0.0564	0.0647	0.0729	0.0815	0.4051	0.3911	0.7834	1.1736
K Neighbors	0.0980	0.2192	0.3260	0.4340	0.5545	0.6481	0.7756	0.9020	1.0012	1.1103	2.8379	5.6259	11.2033	16.8673
Support Vector Machine	0.4625	0.7471	1.1418	1.5333	1.9311	2.2751	2.7042	3.1116	3.4938	3.8904	4.1399	8.2825	16.5498	24.7921
Klasyfikator GPU	2.2735	2.1504	3.3241	4.5460	5.4218	6.3891	7.6431	8.7503	10.2526	10.9616	24.4249	52.6103	104.2419	209.9426

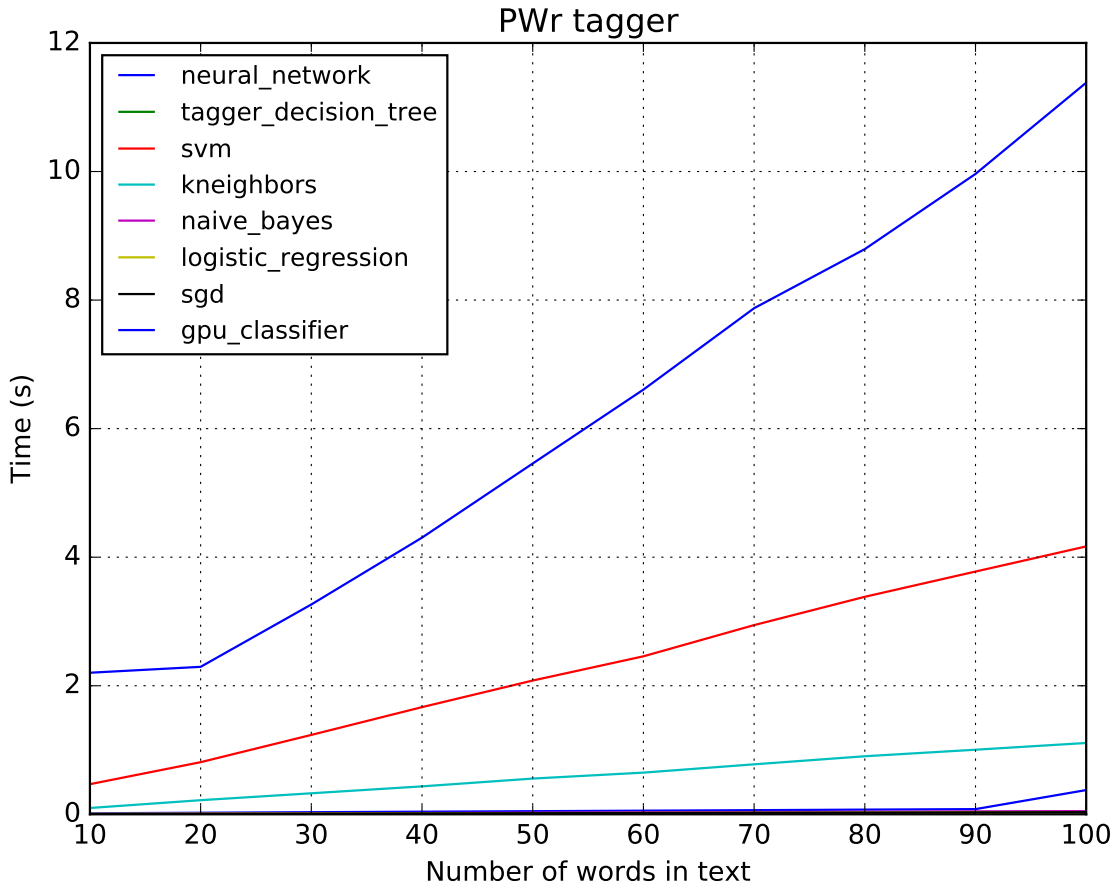
Tabela 11: Czasy klasyfikacji w zależności od ilości słów w tekście – korpus National.

Algorytm	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**250**	**500**	**1000**	**2000**
Logistic Regression	0.0062	0.0095	0.0129	0.0165	0.0194	0.0223	0.0255	0.0292	0.0322	0.0353	0.0789	0.1582	0.3156	0.4743
Stochastic Gradient Descent	0.0031	0.0063	0.0096	0.0136	0.0164	0.0194	0.0225	0.0264	0.0293	0.0328	0.0803	0.1609	0.3202	0.4816
Decision Trees	0.0039	0.0075	0.0115	0.0161	0.0195	0.0231	0.0270	0.0314	0.0350	0.0389	0.0958	0.1920	0.3817	0.5743
Naive Bayes	0.0047	0.0096	0.0146	0.0203	0.0249	0.0294	0.0345	0.0401	0.0447	0.0498	0.1224	0.2456	0.4895	0.7369
Neural Networks	0.0835	0.0156	0.0236	0.0322	0.0399	0.0471	0.0556	0.0640	0.0718	0.0797	0.2682	0.3896	0.7777	1.1690
K Neighbors	0.4112	0.8940	1.3225	1.7856	2.2718	2.6559	3.1993	3.7022	4.2142	4.6087	12.9492	25.2313	50.9364	77.4012
Support Vector Machine	0.9388	2.0124	3.0720	4.1224	5.1977	6.1370	7.3113	8.3830	9.4158	10.4554	13.7190	27.3826	54.6263	81.9051

Zauważyć można, iż najdłuższym czasem klasyfikacji wyróżnił się ten wykorzystujący kartę graficzną, osiągając przy liczbie 2000 wyrazów bardzo długi czas 209,9426 s. Lepszy czas osiągnął algorytm `SVM`, osiągając dla tej samej liczby wyrazów 24,7921 s. Trzecim, stosunkowo podobnym do poprzedniego okazał się algorytm `K Neighbors` – 16,8673 s. Reszta algorytmów może się pochwalić bardzo niskimi i zbliżonymi do siebie czasami.

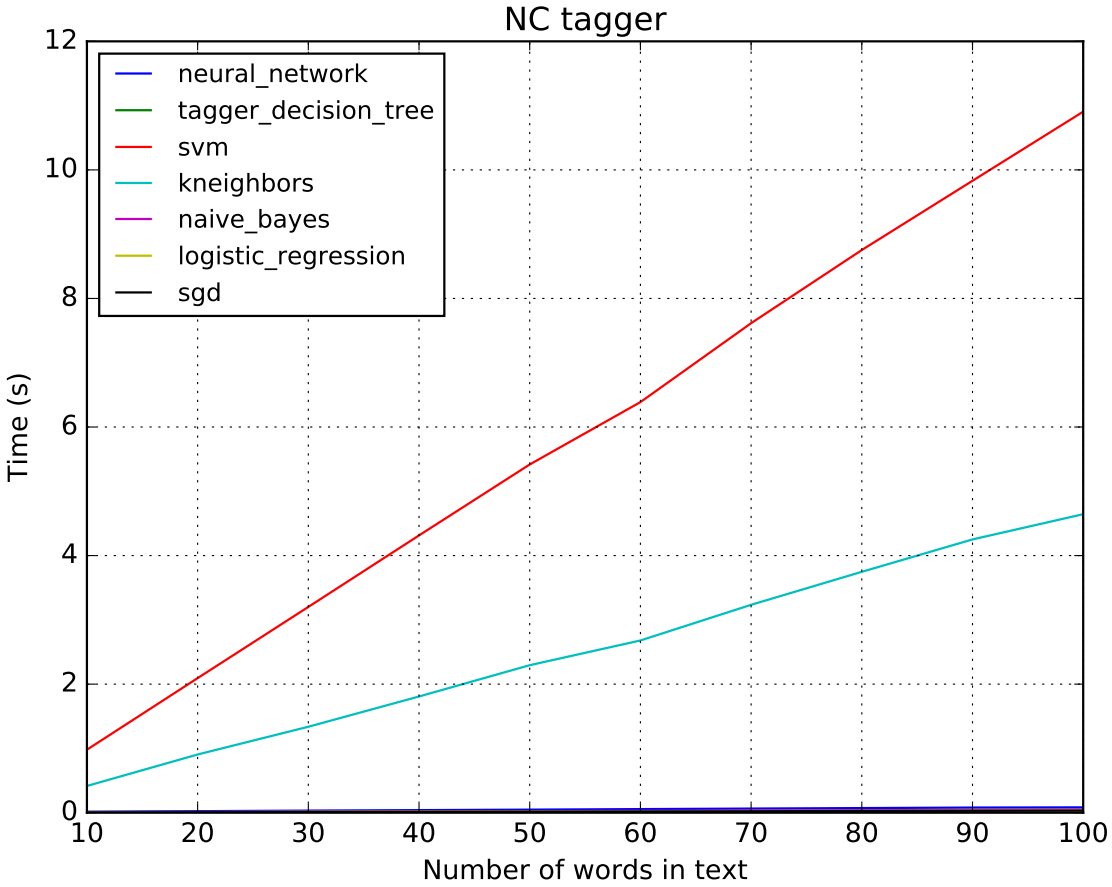
Faktem również jest, że dla pierwszych pięciu algorytmów, które osiągnęły najlepsze czasy (`Logistic Regression`, `SGD`, `Decision Trees`, `Naive Bayes`, `Neural Networks`) czasy klasyfikacji dla korpusu PWr są bardzo zbliżone do czasów klasyfikacji na korpusie podmilionowym. Dla najgorszych algorytmów, czasy klasyfikacji na korpusie podmilionowym są kilkukrotnie wyższe.

Poniższy wykres przestawia wyniki z korpusu PWr. W celu lepszego zobrazowania zależności zostały na nim przedstawione wyniki od 10 do 100 wyrazów. Najszybsze algorytmy osiągnęły czasu 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.

9 Podsumowanie

W ramach projektu powstał kompletny system, w którym można wytrenować klasyfikatory, w tym jeden oparty o kartę graficzną. Dzięki systemowi można zbudować tagger części mowy języka polskiego. Lematyzator nie powstał z powodu wielu trudności, do których można zaliczyć: kodowanie danych, określanie klas, punkty, w których zamieniane byłyby końcówki i specyfikacja języka polskiego, która zawiera dużą liczbę wyjątków w odmianach słów.

Przeprowadzone badania czasów klasyfikacji wykazały, iż najefektywniejszymi algorytmami są: `Logistic Regression`, `SGD`, `Decision Trees`, `Naive Bayes` oraz `Neural Networks`. Uzyskały one bardzo podobne czasy, gdzie dla liczby 2000 wyrazów czasy ich klasyfikacji wyniosły niecałą sekundę. Można je więc uznać za efektywne. Algorytmy `K Neighbors` oraz `SVM` odznaczyły się stanowczo dłuższymi czasami klasyfikacji, więc nie powinno się ich stosowac dla tekstów mających więcej niż kilka tysięcy wyrazów. Algorytmu uczącego na GPU nie udało się zastosować dla korpusu podmilionowego, zaś dla korpusu PWr osiągnął tak wysokie czasy, że można go uznać za algorytm całkowicie nieefektywny i nie powinien być stosowany przy tego typu zagadnieniach.

Uzyskanie wysokich skuteczności klasyfikatorów wymaga dobrania odpowiednich parametrów – liczb n-literowych końcówek. Badania wykazały, że najwyższą skuteczność klasyfikacji otrzymujemy gdy liczba 2,3,4 literowych końcówek znajuje się w przedziale [35,45], a liczba 1 literowych koncówek w przedziale [15,32]. W przypadku, gdy liczba końcówek jest zbyt duża lub zbyt mała skuteczność klasyfikacji gwałtownie spada. Oprócz tego na skuteczność klasyfikacji źle wpływają duże różnice między liczbami 2,3,4 literowych końcówek. Według przeprowadzonych różnica dwoma liczbami końcówek powinna wynosić nie więcej niż 10. Najlepszym klasyfikatorem okazał się SVM z najlepszym wynikiem 61,08% dla korpusu PWr oraz 60,13% dla korpusu National. Dokładność klasyfikacji można poprawić przez szerszą analizę wejściowych danych (przeprowadzoną przez specjalistów ds. języka polskiego) lub lepszy dobór parametrów.