Politechnika Wrocławska
Wydział Elektroniki
Informatyka

Zastosowania informatyki w medycynie

Analiza stopnia złośliwości komórek nowotworowych wykorzystująca sieci neuronowe.

Autorzy:
Łukasz Matysiak
Kamil Machnicki

Prowadzący:
Dr inż. Bartosz Krawczyk

Termin zajęć:
Czwartek, godz. 11:15

Wrocław
1 czerwca 2016

Spis treści

Rozdział 1

Wstęp

1.1 Cele i założenia projektowe

Celem projektu było zapoznanie się z tematem uczenia maszynowego, a konkretnie z sieciami neuronowymi. Realizacja projektu miała umożliwić zrozumienie zasady działania oraz porównanie wydajności sieci neuronowych używających propagacji wstecznej oraz wariantu o nazwie *Extreme Learning Machines*, na podstawie analizy danych pacjentów.

1.2 Wstęp teoretyczny

Rozwój uczenia maszynowego w ostatnich latach jest bardzo ważny z punktu widzenia wielu dziedzin, jednak szczególnie ważny dla medycyny.

Komputery stają się szybsze i są w stanie przetwarzać większe ilości danych, co przekłada się na możliwość użycia ich do analizowania np. wyników badań. Dzięki rozwojowi techniki, mogą one pomagać lekarzom w diagnozowaniu, co pozwala na szybsze decyzje na temat leczenia pacjenta.

Sieci neuronowe stanowią jedno z możliwych podejść do tematu uczenia maszynowego. Są one modelowane na podobieństwo neuronów w mózgu, a zastosowanie znajdują między innymi w problemach klasyfikacji – np. klasyfikacji stopnia złośliwości raka, tak jak to miało miejsce w projekcie. Wzrost popularności zawdzięczają pojawieniu się szybszych komputerów oraz ogromnej ilości danych dostępnych do uczenia ich, co pozwoliło na zredukowanie wpływu głównych wad sieci neuronowych – wolnego uczenia się oraz wymagania dużych zestawów danych uczących.

Problem badany podczas projektu jest to problem klasyfikacji komórek rakowych. Jest on trudny, ponieważ jest problemem, który **nie jest zbalansowany** – przypadki nowotworu złośliwego stanowią mniejszą część badanych. Wpływa to znacząco na proces uczenia się klasyfikatora i jakość klasyfikacji.

Klasyfikator miał za zadanie nauczyć się rozpoznawać jedną z dwóch klas – G2 oraz G3 – z której każda odpowiada odpowiednio 6 i 7 oraz 8 i 9 punktom w systemie Scarffa-Blooma-Richardsona.

Rozdział 2

Aplikacja

2.1 Wykorzystane technologie

Aplikacja została stworzona przy wykorzystaniu języka `Python` w wersji `3.5`.

Biblioteka `scikit-learn` zawiera wiele narzędzi przydatnych w tematyce uczenia maszynowego, dlatego została wykorzystana w projekcie. Użyto biblioteki w wersji deweloperskiej – `0.18.dev0`[1] – ze względu na dostępność wymaganych narzędzi, m.in. funkcji służących do

- zbudowania klasyfikatora opartego o sieci neuronowe,
- przeprowadzenia walidacji krzyżowej,
- stworzenia rankingu cech,
- stworzenia macierzy pomyłek.

Ponieważ `scikit-learn` nie posiada zaimplementowanych *Extreme Learning Machines,* w projekcie wykorzystano moduł `Python-ELM`[2]. Wyznaczony on był na oficjalnej stronie *ELM*[3] jako jedna z bazowych implementacji. Udostępniony kod nie był przystosowany do działania pod wersją *Pythona 3*, wymagał więc poprawek.

[1] http://scikit-learn.org/dev/documentation.html
[2] https://github.com/dclambert/Python-ELM
[3] http://www.ntu.edu.sg/home/egbhuang/elm_codes.html

Wszystkie wykorzystane moduły i ich przeznaczenie widnieją w tabeli 2.1.

Tabela 2.1: Wykorzystane moduły.

Nazwa	**Wersja**	**Opis**
`scikit-learn`	0.18	Sieć neuronowa uczona metodą wstecznej propagacji błędu, selekcja cech, walidacja krzyżowa, macierz błędu.
`Python-ELM`	0.3	*Extreme Learning Machines*.
`numpy`	1.11.0	Obliczenia numeryczne.
`matplotlib`	1.5.1	Tworzenie wykresów.

2.2 Budowa

Na aplikację składa się kilka modułów:

- `main.py` - główny moduł uruchamiający badania.
- `algorithms.py` - uruchamia eksperymenty.
- `dataset.py` - importuje dane z pliku CSV i przechowuje je w wygodnej dla programisty postaci.
- `grapher.py` - tworzy wykresy z wyników badań.
- `helper.py` - zawiera klasy pomocnicze do przechowywania danych.
- `consts.py` - zawiera ustawienia badań i parametry dla algorytmów.

Po uruchomieniu badania przez moduł `main.py`, w module `algorithms.py` odbywa się wykonywanie owych badań. Najpierw, na zbiorze zawierającym dane uczące `X` oraz tablicę wyników `y` wykonywana jest 10-krotna walidacja krzyżowa przy pomocy `StratifiedKFold` z modułu `sklearn.model_selection`, w wyniku której otrzymywane są indeksy elementów podzielonych na dwa podzbiory - zbiór uczący i testowy:

```
StratifiedKFold(n_folds=n_folds).split(X, y):
```

W tym momencie następuje też iteracja przez wszystkie wyniki walidacji, gdzie w każdej iteracji odbywa się selekcja `n`-cech, gdzie `n` to liczba cech, dla której aktualnie wykonywane są badania:

```
SelectKBest(k=k_best_features).fit(X, y).get_support(indices=True)
```

Na tak wyselekcjonowanych cechach odbywa się uruchomienie dwóch algorytmów - uczonego metodą wstecznej propagacji błędu (algorytm *BP*) oraz jego szybka wersja losowa *Extreme Learning Machines* (algorytm *ELM*). Najpierw tworzona jest instancja klasyfikatora algorytmu *BP* (opis parametrów użytych do badania znajduje się w rozdziale o badaniach), następnie odbywa się jego uczenie przekazując do funkcji `fit` zbiór uczący. Na końcu dokonywana jest klasyfikacja algorytmu przekazując do funkcji `score` zbiór testowy.

```
clf = MLPClassifier(...)
clf.fit(X_train, y_train)
score = clf.score(X_test, y_test)
```

Dla algorytmu `ELM` procedura wygląda bardzo podobnie:

```
elmc = ELMClassifier(...)
elmc.fit(X_train, y_train)
score = elmc.score(X_test, y_test)
```

Na końcu dla każdego algorytmu wyliczana jest macierz pomyłek, korzystając z `confusion_matrix` z modułu `sklearn.metrics`.

```
conf_matrix = confusion_matrix(y_test, clf.predict(X_test))
conf_matrix = confusion_matrix(y_test, elmc.predict(X_test))
```

2.3 Wdrożenie

W celu przygotowania aplikacji do działania, należy dodać katalog projektu do zmiennej środowiskowej `PYTHONPATH`. Można to zrobić za pomocą polecenia:

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

Kolejnym krokiem jest uruchomienie skryptu instalacyjnego `install.sh`, który zainstaluje wymagane pakiety z pliku `requirements.txt` oraz pobierze poprawiony moduł `Python-ELM` do katalogu `modules`:

```
./install.sh
```

Teraz można uruchomić aplikację poprzez wywołanie skryptu głównego `main.py`:

```
python3 main.py
```

Działanie aplikacji powinno zakończyć się wygenerowaniem wykresów.

Rozdział 3

Badania

3.1 Przeprowadzone badania

Zgodnie z założeniami, badania przeprowadzono dla 5 różnych liczb neuronów w warstwie ukrytej: **5, 10, 20, 40, 60**. W celu uzyskania możliwie jak najdokładniejszych wyników, każdy eksperyment powtórzono 5 razy, zaś wyniki uśredniono i wraz z wyliczonym odchyleniem standardowym, zaprezentowano na wykresach. Pojedynczy eksperyment składa się z następujących kroków:

1. Wybranie rozmiaru warstwy ukrytej.
2. Wybranie liczby cech.
3. Przeprowadzenie 10-krotnej walidacji krzyżowej, na którą składa się
   (a) Podział zbioru na podzbiory uczący oraz testujący.
   (b) Wybranie najlepszych cech.
   (c) Nauczenie klasyfikatora.
   (d) Ocena jakości klasyfikacji.
4. Uśrednienie wyników.
5. Prezentacja wyników.

Parametry, których użyto w badaniach znajdują się w tabeli 3.1.

Tabela 3.1: Parametry algorytmów.

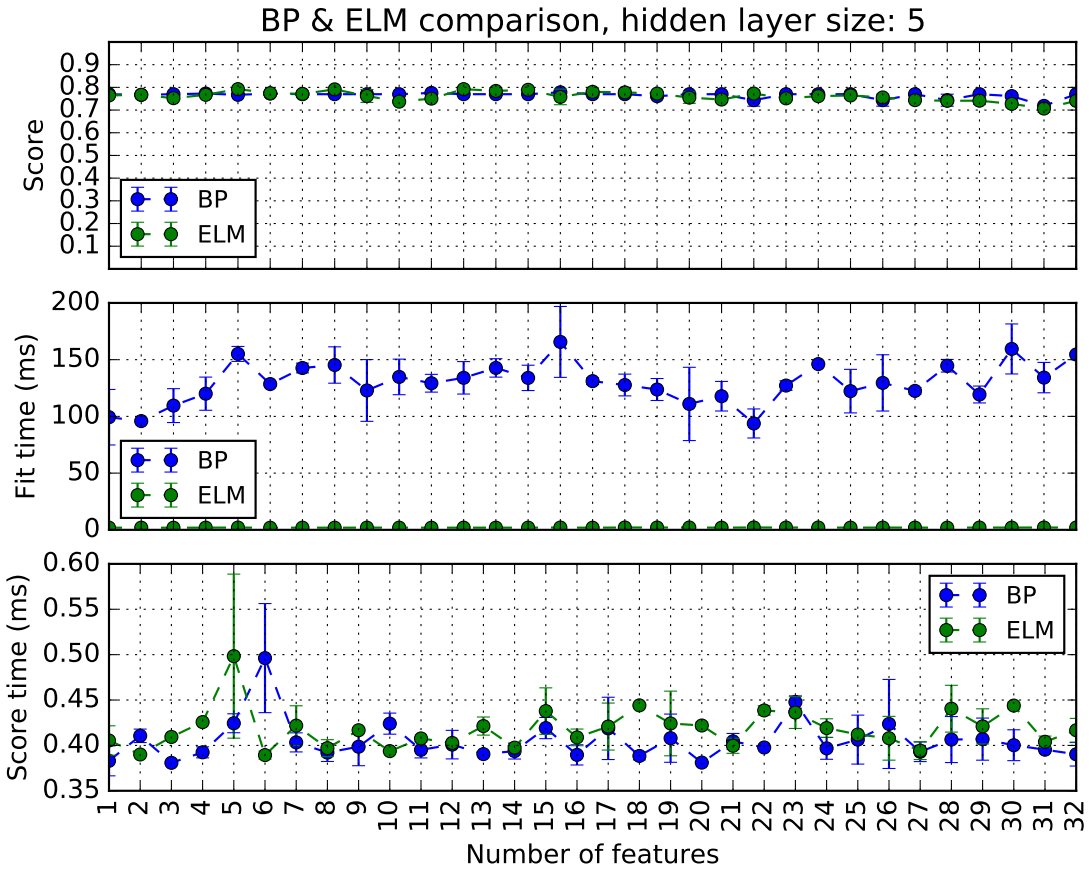
Parametr	**Wartość**	**Opis**
`algorithm`	`sgd`	Algorytm działania (*stochastic gradient descent*).
`max_iter`	`10000`	Maksymalna liczba iteracji.
`alpha`	`1e-6`	Parametr regularyzacji.
`learning_rate`	`constant`	Tempo uczenia.
`activation`	`logistic`	Funkcja aktywacji algorytmu `BP`.
`activation`	`multiquadric`	Funkcja aktywacji algorytmu `ELM`.

3.2 Porównanie algorytmów

Dla 5 neuronów w warstwie ukrytej można zauważyć, że oba algorytmy osiągnęły prawie takie same wyniki oscylujące w okolicy 76%, niezależne od liczby wyselekcjonowanych cech.

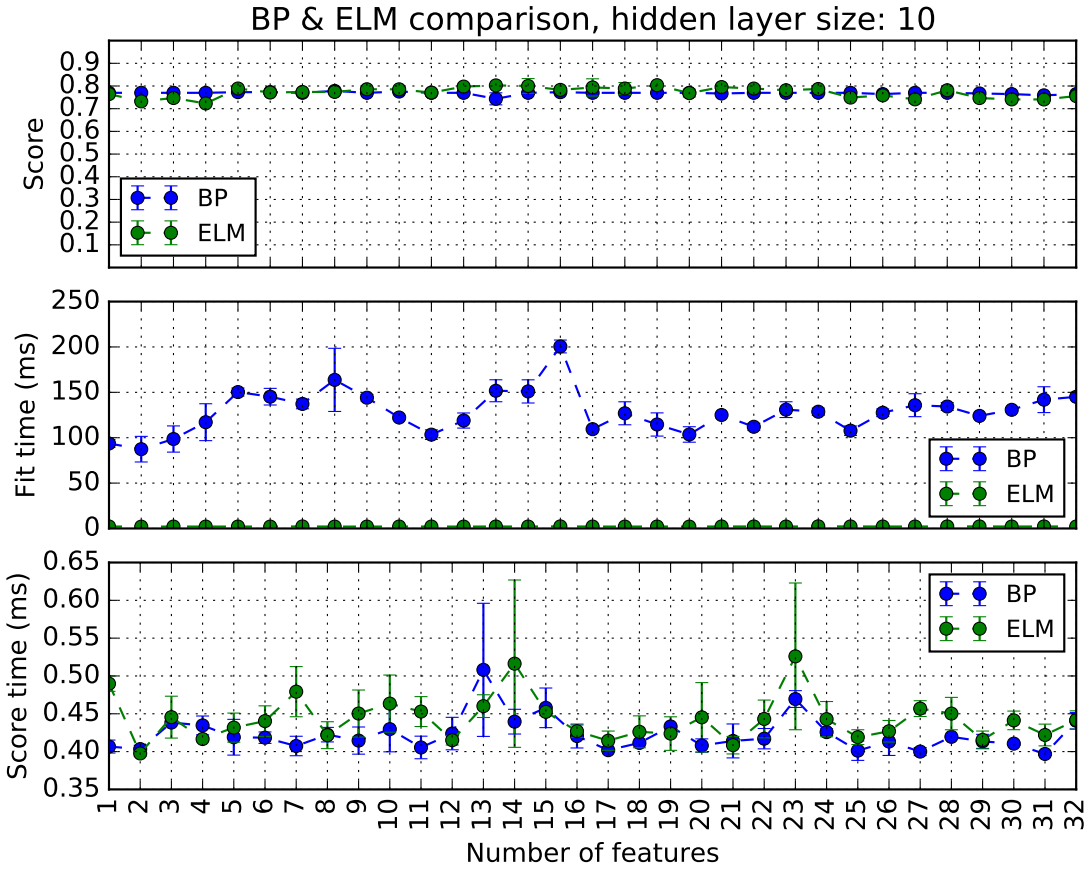
Również czasy wykonywania klasyfikacji okazały się bardzo podobne. Wystąpiły tutaj jednak większe fluktuacje, spowodowane przede wszystkim niedokładnością w mierzeniu bardzo małych czasów (rzędu 0.5 milisekundy) oraz dużym odchyleniem standardowym sygnalizowanym na wykresie przez słupki błędu.

Największą różnicę zauważyć jednak można w średnich czasach uczenia – 2ms dla `ELM` i 130ms dla sieci neuronowej.



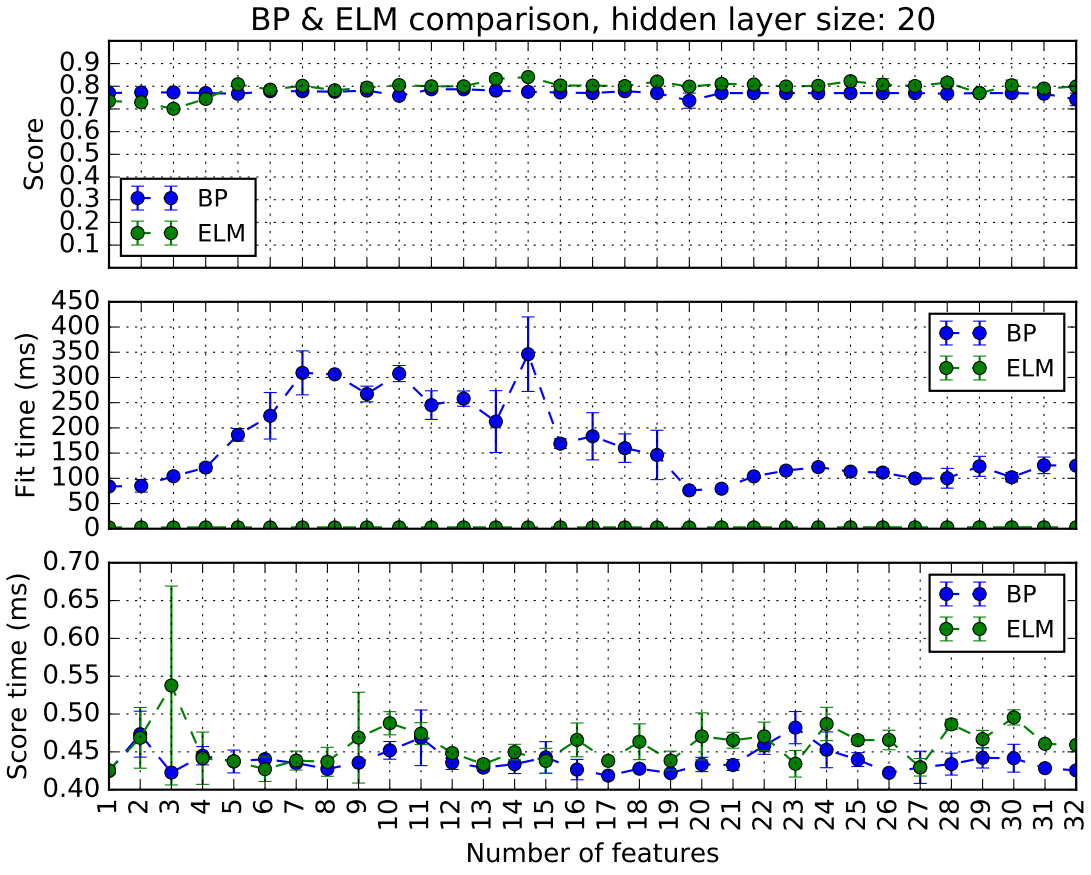
Rysunek 3.1: 5 neuronów w warstwie ukrytej - porównanie obu algorytmów

Dla 10 neuronów w warstwie ukrytej zachodzą te same zależności. Widoczny jest jednak nieznaczny wzrost czasu uczenia dla algorytmu `BP` do około 250 milisekund w okolicy 10-14 wyselekcjonowanych cech. Algorytm `ELM` pozostaje bez zmian.



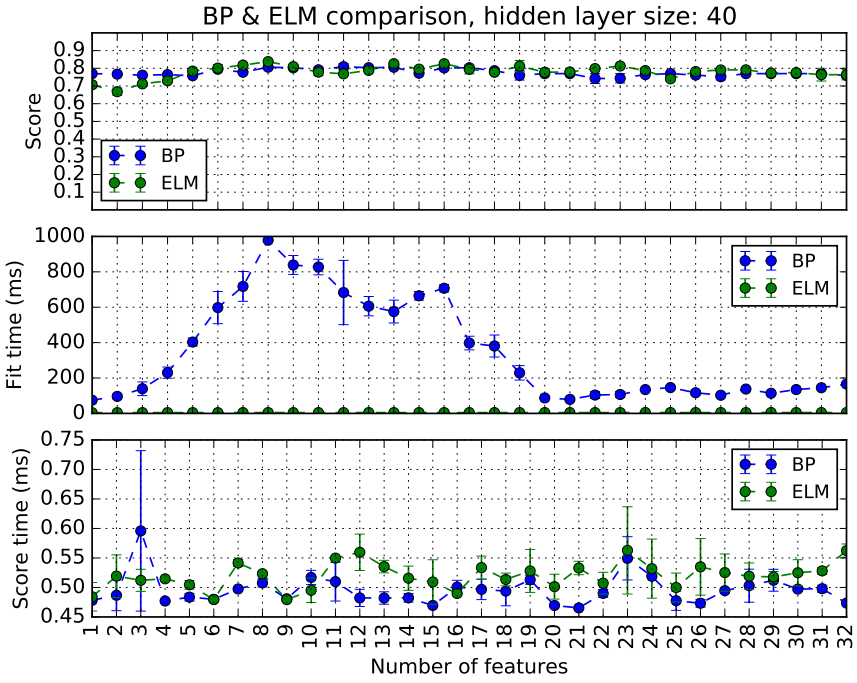
Rysunek 3.2: 10 neuronów w warstwie ukrytej - porównanie obu algorytmów

Dla 20 neuronów w warstwie ukrytej zdecydowanie zauważyć już można tendencję w czasie uczenia algorytmu BP, gdzie do około 7 wyselekcjonowanych cech czas znacząco wzrasta, później do około 16 cech pozostaje względnie stały, następnie maleje i od 22 cech utrzymuje się już na stałym, niskim poziomie rzędu 100 milisekund. Wytłumaczyć tę zależność można zjawiskiem przeuczenia, gdzie algorytm osiągnął lokalne minima, a następnie je wykorzystywał przy kolejnych uczeniach.

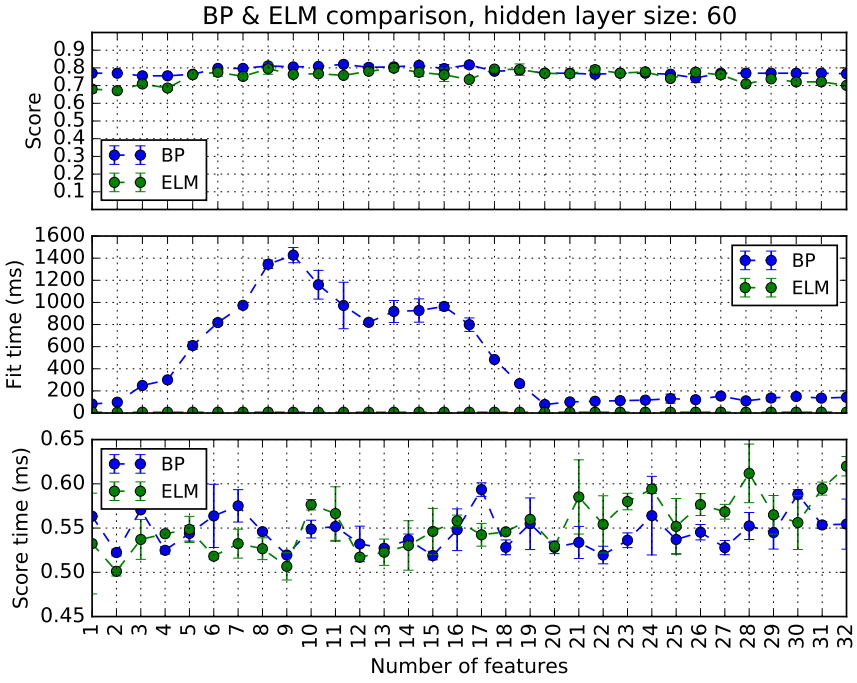


Rysunek 3.3: 20 neuronów w warstwie ukrytej - porównanie obu algorytmów

Dla 40 oraz 60 neuronów w warstwie ukrytej widać już bardzo wyraźnie zauważone wcześniej tendencje dla algorytmu BP. Reszta zależności pozostaje bez zmian.



Rysunek 3.4: 40 neuronów w warstwie ukrytej - porównanie obu algorytmów



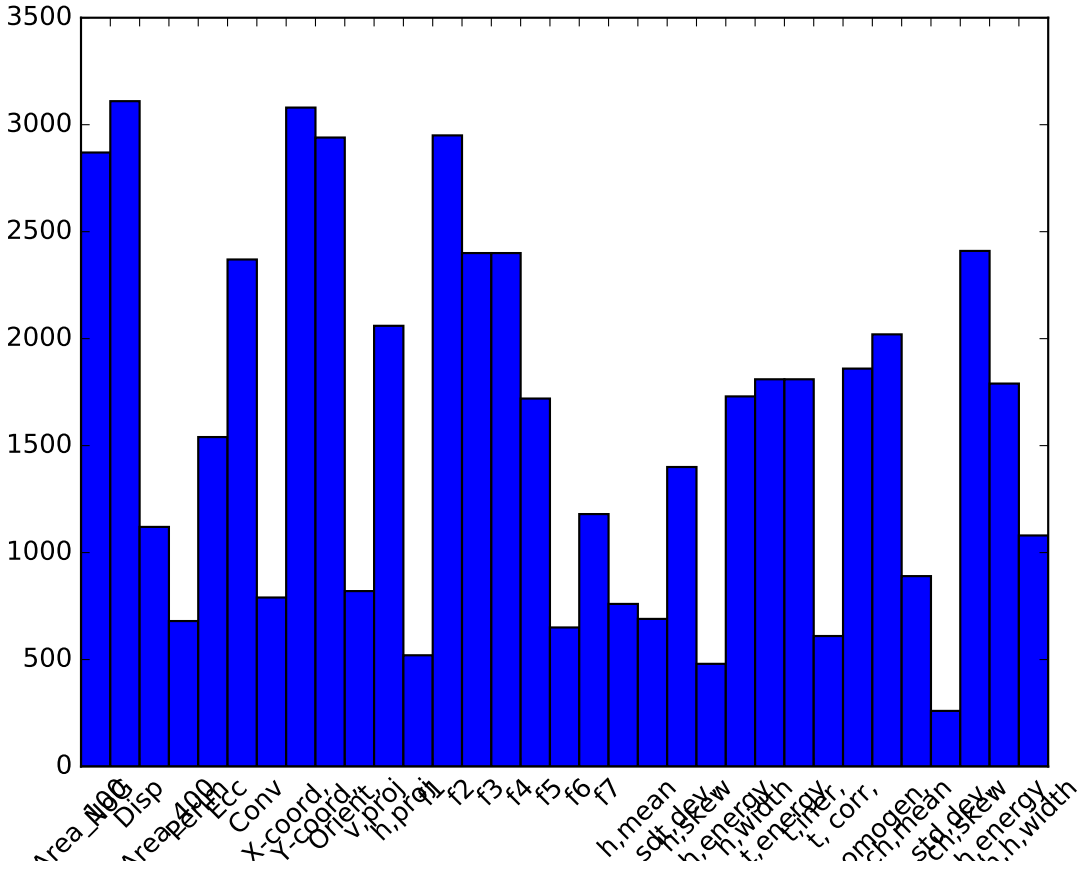
Rysunek 3.5: 60 neuronów w warstwie ukrytej - porównanie obu algorytmów

3.3 Selekcja cech

Selekcja cech polega na wybieraniu podzbioru cech w celu ograniczenia czasu uczenia, uproszczenia modelu oraz minimalizacji zjawiska przeuczenia.

W projekcie skorzystano z klasy `SelectKBest`, z modułu `sklearn.feature_selection`.

Wykres poniżej przedstawia sumę częstości wybierania cech na przestrzeni wszystkich eksperymentów. Widać wyraźnie, że niektóre cechy wybierane były znacznie częściej niż inne, co oznacza, że były o wiele bardziej przydatne w procesie uczenia algorytmów.



Rysunek 3.6: Częstość wybierania cech

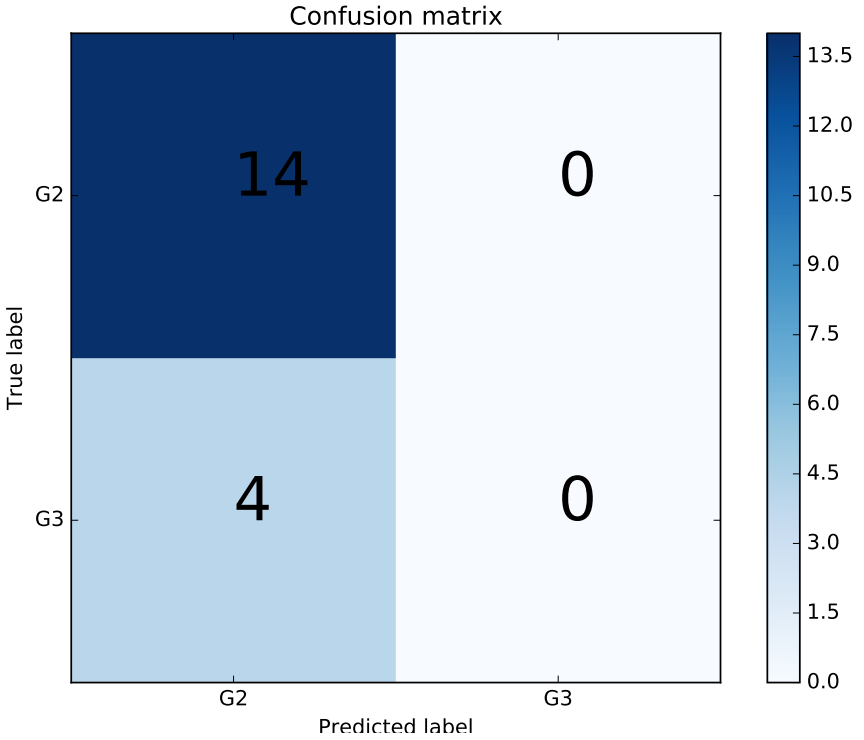
3.4 Macierz pomyłek

Macierz pomyłek umożliwia zobrazowanie jak dobrze klasyfikator radzi sobie ze swoim zadaniem.

Dla problemu z dwiema klasami – tak jak w aktualnie rozpatrywanym problemie – macierz dzieli się na cztery części, w których zliczane są obiekty sklasyfikowane poprawnie – umieszczone na przekątnej macierzy – oraz błędnie – umieszczone poza przekątną.

W projekcie skorzystano z funkcji `confusion_matrix`, z modułu `sklearn.metrics`.

Na przedstawionym rysunku widać wyraźnie, że 14 z badanych obiektów zaklasyfikowanych zostało poprawnie do klasy `G2`. 4 z nich otrzymało błędną klasyfikację jako `G2`. Oznacza to, że w tym przypadku algorytm nauczył się klasyfikować wszystkie obiekty do klasy `G2`, co spowodowane było opisanym wyżej problemem niezbalansowanych danych.



Rysunek 3.7: Przykładowa macierz pomyłek

Rozdział 4

Wnioski

- Sieci neuronowe mogą stanowić potężne narzędzie w walce z chorobą.
- Parametry sieci należy dobrać odpowiednio do problemu, aby unikać zjawiska przeuczenia.
- Dane, z którymi przyszło nam pracować nie były zbalansowane – 76% przypadków znajdowało się w klasie G2 – co powodowało problemy przy uczeniu klasyfikatora.
- Sieci neuronowe potrzebują sporej ilości danych uczących, aby poprawnie generalizować problemy.
- ELM daje bardzo zbliżone jakościowo wyniki do sieci neuronowych ze wsteczną propagacją, jednocześnie proces uczenia jest o wiele szybszy.
- Wbrew początkowej intuicji, kilka najlepszych cech zebranych razem wcale nie musi dawać najlepszych wyników.
- Procesy selekcji cech oraz walidacji krzyżowej nie mogą być przeprowadzane osobno.

Bibliografia

[1] Neural networks and deep learning. http://neuralnetworksanddeeplearning.com/. Aktualne dnia: 2016-06-01.

[2] Scikit-learn documentation. http://scikit-learn.org/dev/documentation.html. Aktualne dnia: 2016-06-01.

[3] G.-B. Huang. *What are Extreme Learning Machines?* http://www.ntu.edu.sg/home/egbhuang/pdf/ELM-Rosenblatt-Neumann.pdf, 2015.

[4] B. Krawczyk, M. Galar, Łukasz Jelen, F. Herrera. *Evolutionary undersampling boosting for imbalanced classification of breast cancer malignancy.* Elsevier, 2015.

[5] A. Krot. Introduction to machine learning with python and scikit-learn. http://kukuruku.co/hub/python/introduction-to-machine-learning-with-python-andscikit-learn. Aktualne dnia: 2016-06-01.

[6] R. Rojas. *Neural Networks - A Systematic Introduction.* https://page.mi.fu-berlin.de/rojas/neural/, 1996.