

# MPUM MINIPROJEKT III

MICHAŁ HOFFMANN

Abstract. Celem miniprojektu było wykorzystanie kilku nieliniowych algorytmów klasyfikacji do analizy danych z zadanego pliku. W projekcie zaimplementowałem: algorytm SVM, drzewo decyzyjne oraz algorytm K najbliższych sąsiadów. Użyłem też gotowych rozwiązań z bibliotek jako benchmarków.

## 1. Dane

Każdy algorytm był trenowany na tych samych danych. Plik *phishing.data* zawiera dane na temat stron internetowych. Dane to: 11055 rekordów zawierających 30 cech binarnych jak i ternarnych oraz zmienną objaśnianą  $y \in \{-1, 1\}$  - informację czy dana strona podszywała się pod inną.

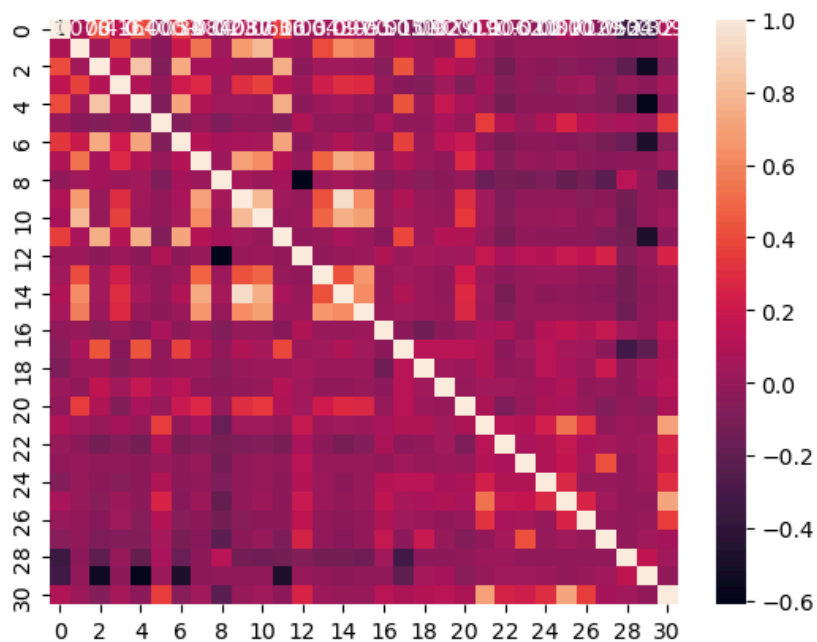
Algorytmy (Poza SVM) testowałem zarówno na początkowym formacie danych, jak i na wersji, gdzie wszystkie cechy ternarne zastąpiłem dwoma cechami binarnymi. Nie spowodowało to żadnych istotnych zmian, poza tym, że drzewa decyzyjne musiały być głębsze dla uzyskania tych samych wyników, a trzy z czterech metryk, które stosowałem w algorytmie KNN stały się równoważne. Czwarta, ( $\ell_\infty$ ) natomiast osiągała dużo gorsze wyniki niż na wyjściowych danych, nic dziwnego, widocznie pewna cecha ternarna najlepiej określała zmienną objaśnianą, co zostało utracone przy przekształceniu na zmienne binarne.

### 1.1. Rozkład danych.

Początkowe dane zawierają 6157 stron „czystych” i 4898 podszywających się. Do treningu będę wykorzystywał pełne dane, trenowanie na podzbiorze, w którym oba zdarzenia są równie prawdopodobne było odrobinę gorsze w wynikach.

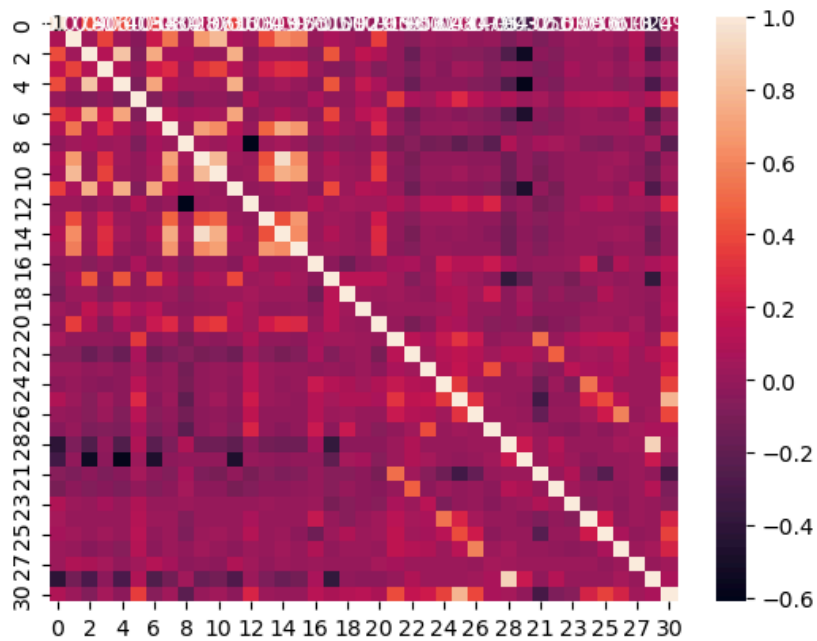
### 1.2. Wykres korelacji.

Użyłem gotowej funkcji z biblioteki *numpy*, aby stworzyć wykres korelacji między cechami jak i zmienną objaśnianą. Można zauważyć istotną korelację między niektórymi cechami, a wartością  $y$  - zdecydowanie najlepiej jest opisywana przez cechy 22 i 25.



Rysunek 1. Wykres korelacji

Podobną analizę przeprowadziłem na zbiorze po przekształceniu cech na binarne. Tym razem osiągnąłem jednego faworyta jeśli chodzi o korelację z  $Y$  - cechą 25, która odpowiada na pytanie „Czy pierwotna cecha 25 ma wartość 1”.



Rysunek 2. Wykres korelacji

## 2. Algorytm SVM

### 2.1. Porażka z SMO.

Dużo czasu spędziłem próbując napisać działającą wersję algorytmu SMO. Niestety, pomimo wspierania się oryginalną pracą, mój klasyfikator uczył się 10 minut i osiągał skuteczność 49%, dlatego w pewnym momencie porzuciłem to podejście pomimo jego istotnych zalet i skupiłem się na algorytmie wykorzystującym podejście gradientowe.

### 2.2. Gradientowy SVM.

Maszyna wektorów nośnych wykorzystująca algorytm gradientowy przyniosła mi naprawdę dobre wyniki przy całkiem krótkim czasie szkolenia i implementacji. Końcowo, był to najgorszy z trzech zaimplementowanych algorytmów, ale zdecydowanie konkurencyjny.

### 2.3. SMO - podejście drugie.

Po wielu niepowodzeniach postanowiłem spytać bardziej doświadczonych kolegów, czy może nie mieli tego samego problemu w implementacji algorytmu SMO. Okazało się to strzałem w dziesiątkę, bo już pierwsza osoba którą spytałem potwierdziła, że rzeczywiście problemem jest stała  $C$ , która musi zawierać się w konkretnym przedziale, żeby zwracać „rozsądne” wyniki, czyli cokolwiek istotnie powyżej 50%. Po wprowadzeniu koniecznych zmian, algorytm uczył się jeszcze dłużej, ale przynajmniej jakkolwiek działał. Z ciekawych wniosków, to uznałem za niepotrzebne wywoływanie pętli algorytmu po osiągnięciu  $\text{changes}=0$ . Wprowadziłem modyfikację, która sprawdzała wyniki dla zbioru walidacyjnego za każdym razem, gdy licznik nie znalazł żadnej zmiany, co pokazało, że w moim przypadku już za pierwszym/drugim razem algorytm na ogół znajdował maksimum swoich możliwości, w celu przetestowania większej liczby stałych  $C$ , usunąłem ten warunek z pętli na czas szukania metaparametrów, jednak potem przywróciłem w celu testowania.

### 2.4. Wyniki.

#### 2.4.1. SVM.

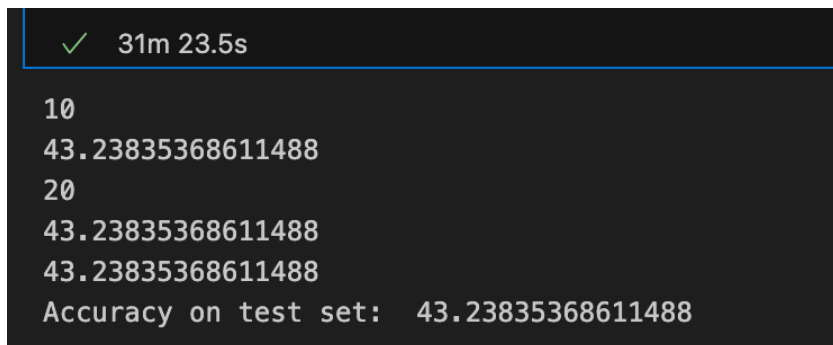
SVM za każdym przebiegiem znajdował bardzo podobny wektor, którego średnia dokładność wynosiła około 93.1%.

#### 2.4.2. SMO.

Początkowa wersja, ze stałą  $C$  w przedziale  $[0, 1]$  osiągała wyniki poniżej 50%, co nie do końca było dla mnie akceptowalne.

Końcowo, w zależności od kernela i parametru  $C$ , algorytm osiągał od 55 do 70% skuteczności, gdzie najlepsze wyniki dawał kernel wielomianowy, dla stałej  $C = 75$ . Czasem, w trakcie szkolenia, algorytm osiągał trochę wyższe dokładności, ale w kolejnych iteracjach wracał on do gorszych wyników.

Zdarzały się też takie wyniki:



Rysunek 3. Bywa i tak.

### 3. Drzewo decyzyjne

Drzewo decyzyjne tworzyłem zarówno dla danych binarnych, jak i początkowego zbioru. Osiągało dokładnie takie same wyniki, ale binarne dane potrzebowały trochę głębszego drzewa, by mieć taką samą skuteczność - ma to intuicyjnie sens.

#### 3.1. Najważniejsza cecha.

Ze względu na strukturę algorytmu tworzenia drzewa, w korzeniu powinna znaleźć się ta cecha, która najlepiej objaśnia  $Y$ . W każdym przypadku była to cecha 25, która była najlepszym kandydatem już na etapie tworzenia macierzy korelacji. Drzewa decyzyjne potwierdziły tę hipotezę.

#### 3.2. Kwestia głębokości.

Na wykresach poniżej bardzo dobrze będzie widać, że już dla niewielkich głębokości, drzewo osiąga bardzo dobre wyniki.

Tworzyłem drzewa o głębokości od 2 do 128, bo wyczytałem gdzieś, że najlepiej gdy  $\max d \leq \sqrt{n}$ , jednak nawet mniejsze wartości zupełnie wystarczały, głębokość 16 (32 dla binarnych klas) wysyłała możliwości drzewa decyzyjnego.

#### 3.3. Funkcje nieczystości.

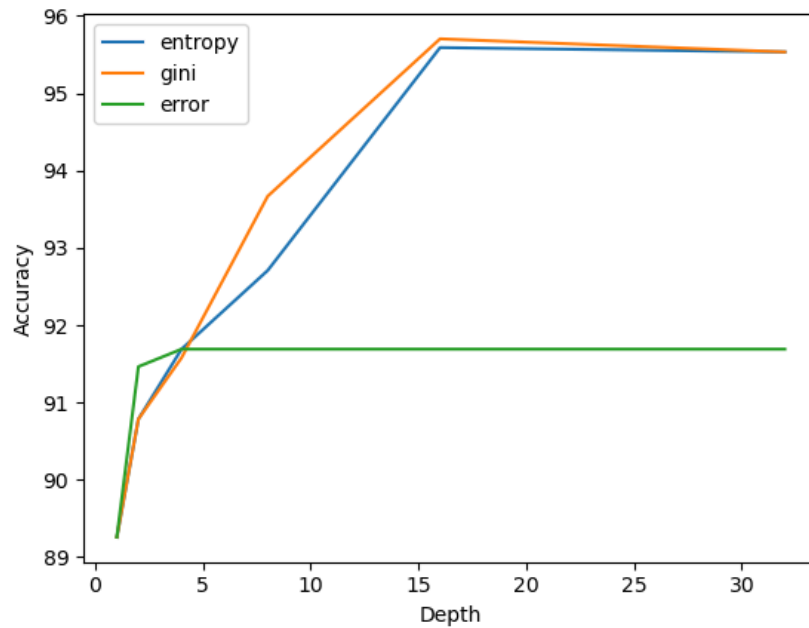
Użyłem trzech prostych funkcji nieczystości.

- funkcji giniego
- entropii
- funkcji, która po prostu liczy prawdopodobieństwo błędnej predykcji

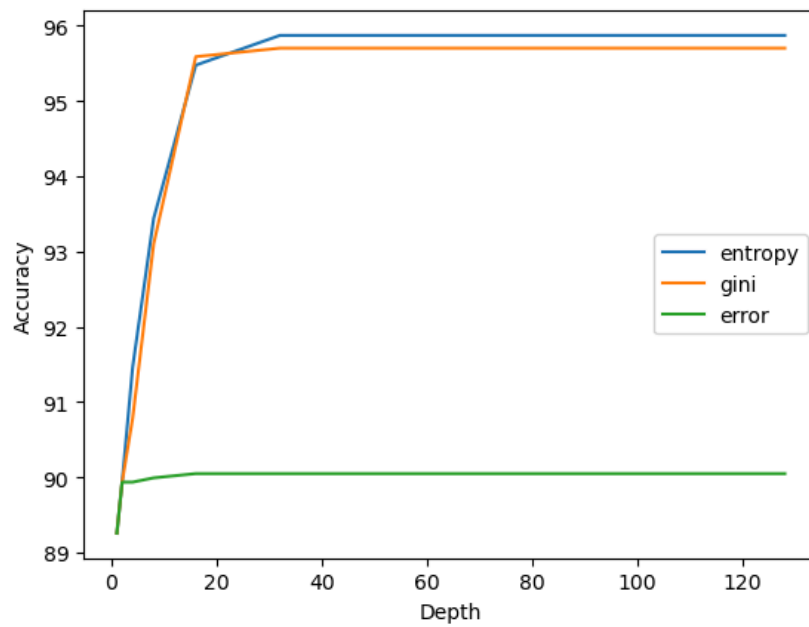
Dla danych z cechami ternarnymi, najlepiej sprawowała się funkcja giniego, a w przypadku binarnym - entropia. Funkcja błędu była w obu przypadkach najgorsza, a po konwersji na dane binarne, traciła jeszcze ok. dwóch punktów procentowych.

#### 3.4. Końcowe wyniki.

Poniższe wykresy przedstawiają dokładność drzew w zależności od maksymalnej głębokości i funkcji nieczystości. Jeden wykres przedstawia dane binarne, drugi zbiór z ternarnymi (różnica w skali wynikła z tego, że dane binarne chciałem przetestować na znacznie głębszych drzewach, okazało się to zupełnie niepotrzebne).



Rysunek 4. Dokładność drzewa decyzyjnego dla początkowych danych



Rysunek 5. Dokładność drzewa decyzyjnego dla danych binarnych

#### 4. Algorytm KNN

W zasadzie algorytm nie wymaga żadnego preprocessingu, o ile nie używa się bardziej zaawansowanych struktur do trzymania danych. Pozwala to na więcej

bawienia się metaparametrami. W moim przypadku były to metryki, jak i początkowe zmienianie danych wykorzystując współczynnik korelacji.

#### 4.1. Metryki odległości.

Poza podstawową metryką euklidesową, postanowiłem przetestować kilka innych rozwiązań. Koniec końców użyłem następujących metryk:

- Odległość euklidesowa  $\ell_2$
- Metryka manhattańska  $\ell_1$
- Metryka maximum  $\ell_\infty$  (z góry skazana na niską skuteczność, ze względu na małą liczbę klas cech)
- Metryka *Adam*\* (nazwa pochodzi od pomysłodawcy), w której odległość to liczba różnych cech. Dowód nierówności trójkąta dla tej metryki pozostawiam jako proste ćwiczenie.

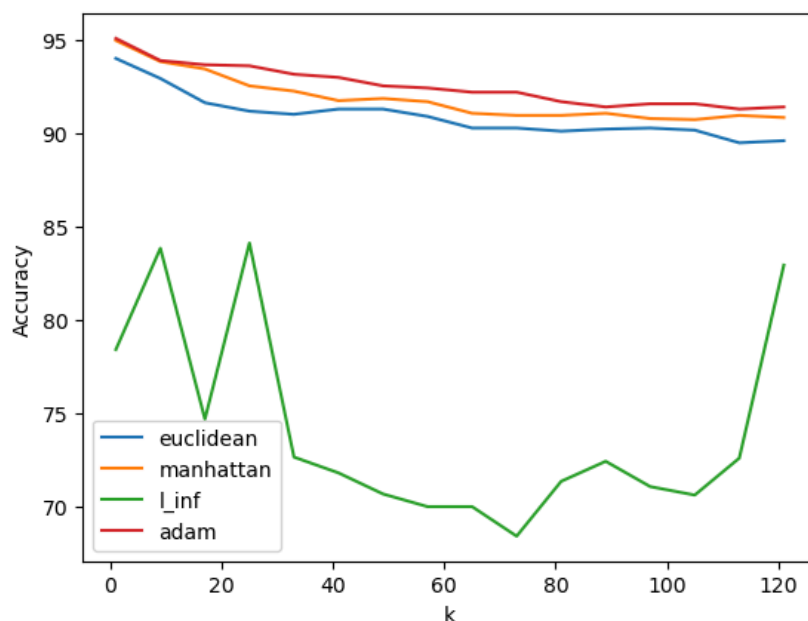
Jako dodatek, spróbowałem przeskalować dane o stałe (a bardziej ich dopełnienia do wartości większej od liczby cech dla odwrócenia porządku) pochodzące z analizy korelacji z pierwszego akapitu, co miało znaczenie w przypadku metryk innych niż maksimum i *Adam*. Próbowałem też użyć odwrotności, ale sprawowały się dużo gorzej. Co to znaczy w praktyce? Chciałem przeskalować przestrzeń cech w taki sposób, żeby odległości cech bardziej znaczących zmieniały więcej w końcowym wyniku. Delikatnie poprawiło to wynik dla tych dwóch metryk, ale jedynie do poziomu wyników z metryki *Adam*.

#### 4.2. Jak znaleźć K?.

Zamiast skupiać się na jednej wartości K, postanowiłem sprawdzić ich wiele i porównać wyniki. Testowałem  $k \in \{1, 9, \dots, 129\}$ , a dla konkretnych metryk także i większe (gdyż wykresy sugerowały tendencję, że może być dla takich wartości lepszy wynik) - bezskutecznie, gdyż najlepszym k okazało się  $k = 1$ .

#### 4.3. Wyniki.

Poniższy wykres przedstawia dokładność, jaką osiągnął algorytm kNN dla różnych wartości k i czterech różnych metryk. Wymiana danych na binarne odrobinię podniosła wyniki, sprawiając dodatkowo, że trzy z metryk stały się izomorficzne, a dodanie ważonych współczynników, polepszyło wyniki metryk euklidesowej i manhattańskiej, przy czym ta druga nawet osiągnęła lepsze wyniki niż *Adam*. Pomimo próbowania wielu możliwych K,  $K = 1$  wciąż pozostawało najlepszym współczynnikiem.



Rysunek 6. wyniki algorytmu kNN

### 5. Czasy szkolenia

Z racji na konieczność szkolenia bardziej czasochłonnych algorytmów, do szkolenia używałem równoległe mojego komputera i platformy Google Colab.

Algorytm SVM okazał się naprawdę wymagającym czasowo. Jego wersja gradientowa potrzebowała średnio 4-5 minut, a SMO, które miało być optymalizacją, potrzebowało 6, do nawet 30 minut ( w zależności od parametrów) na wyszkolenie lokalnie, a na Colabie ponad godzinę.

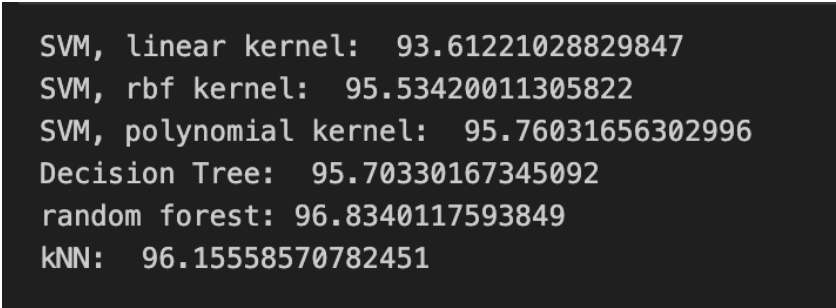
Drzewa decyzyjne tworzyły się niemalże natychmiastowo. Nawet w przypadku największej używanej przeze mnie głębokości - 128, czas powstawania drzewa był niemalże znikomy, szczególnie w porównaniu do pozostałych algorytmów.

W przypadku algorytmu kNN zdecydowałem się nie stosować żadnego preprocessingu, choć po rozmowie z moim współlokatorem, trochę żałuję, gdyż te algorytmy wydają się naprawdę ciekawe. Czas tworzenia predykcji dla zbioru walidacyjnego to średnio półtorej do dwóch minut, a dla zbioru testowego - 20% całości, to równe dwie minuty).

Podsumowując, najbardziej optymalnym czasowo algorytmem były drzewa decyzyjne, a najmniej SVM. Co ciekawe, uzyskana precyzja także jest w tej kolejności, więc mam niekwestionowanego zwycięzcę mojego projektu.

### 6. Benchmarki

Wyniki algorytmów z biblioteki sklearn:



```
SVM, linear kernel: 93.61221028829847
SVM, rbf kernel: 95.53420011305822
SVM, polynomial kernel: 95.76031656302996
Decision Tree: 95.70330167345092
random forest: 96.8340117593849
kNN: 96.15558570782451
```

Rysunek 7. wyniki modeli benchmarkowych

Oznacza to, że moje algorytmy (poza SMO) nie poradziły sobie dużo gorzej niż gotowe rozwiązania biblioteczne, jednak w szczególności SVM i KNN, było tutaj dużo szybsze.

## 7. Adnotacje

\* Jestem świadom, że owa metryka ma też nazwę z innym nazwiskiem, jednak w ramach wdzięczności osobie, która mi ją podsunęła, zostawię ją w taki sposób w raporcie.

## References

Theoretical Computer Science, Jagiellonian University, Kraków  
*Email address:* `michal.hoffmann@student.uj.edu.pl`