
Rysunek 1: Evil SVM

# Miniprojekt 2: Klasyfikacja binarna ciąg dalszy

Metody Probabilistyczne w Uczeniu Maszynowym

Szymon Szulc

22 maja 2025

# Spis treści

# 1 Wstęp

Niniejszy raport oparty jest na notatnikach `*.ipynb`. Raport ma stanowić zwięzłe i czytelne podsumowanie mojej pracy nad problemem klasyfikacji binarnej korzystając z maszyny wektorów nośnych, głebokiej sieci neuronowej oraz drzewa decyzyjnego.

# 2 Badany problem

## 2.1 Definicja

Dane na jakich pracujemy to jakieś cechy stron internetowych. Naszym zadaniem jest stwierdzić, czy strona internetowa służy do phishingu (1), czy też jest bezpieczna (-1).

## 2.2 Założenia

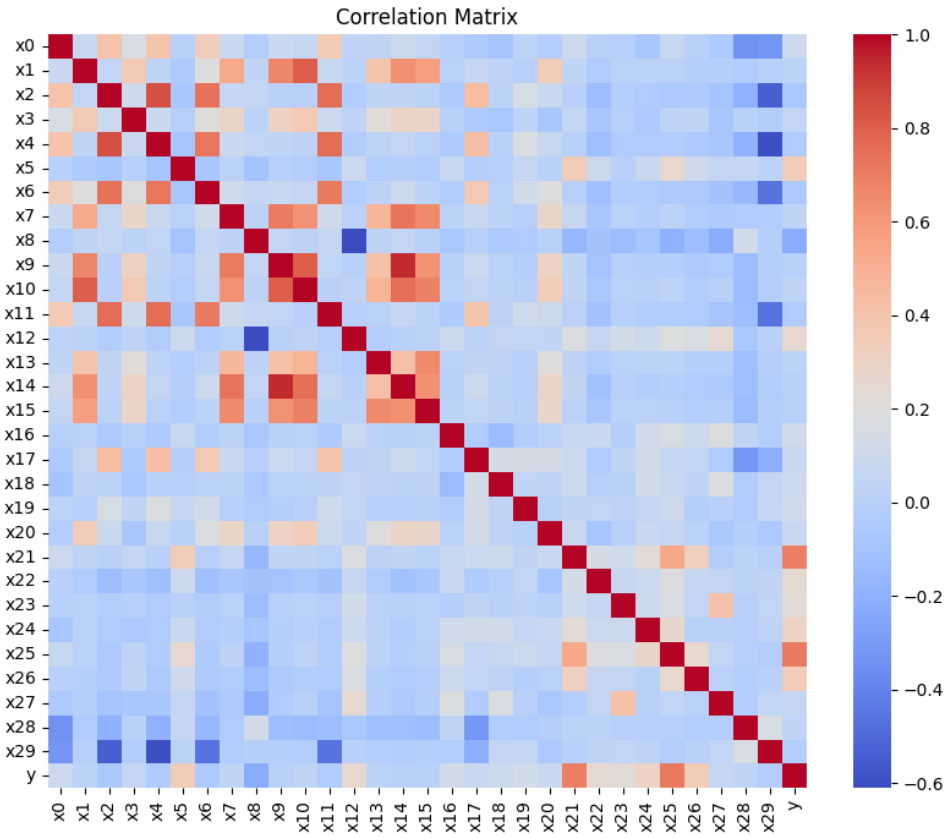
Formalnie:

$$
\begin{aligned}
&y \in \{-1, 1\} \\
&\forall i \in \{0, \ldots, 20\} \;\; x_i \in \{-1, 1\} \\
&\forall j \in \{21, \ldots, 28\} \;\; x_j \in \{-1, 0, 1\} \\
&x_{29} \in \{0, 1\}
\end{aligned}
$$

Mamy do czynienia z cechami dyskretnymi. Nie wiemy nic o ich rozkładach.

# 3 Pierwszy kontakt z danymi

Mamy 30 cech – $x_0, \ldots, x_{29}$ i zmienną binarną $y$. Żeby podtrzymać tradycję, patrzymy na macierz korelacji, którą już bardzo dobrze znamy i przynajmniej empirycznie wiemy, że nie odbiega bardzo od testu chi-kwadrat.




Rysunek 2: Macierz korelacji Pearsona

Możemy zauważyć, że silnie skorelowanych jest tylko kilka cech. Wydaje się, że moglibyśmy się ich pozbyć bez straty na dokładności modelu. Ta hipoteza potwierdziła się dopiero przy drzewach decyzyjnych.

# 4 Podział danych

Dokonałem losowego podziału danych (60% – zbiór treningowy, 20% zbiór walidacyjny, 20% zbiór testowy, zachowując również taki stosunek w obrębie klas $y$) 10 razy, żeby uśrednić wyniki.

# 5 Maszyna wektorów nośnych (SVM)

## 5.1 Wstęp

Będę korzystał z wariantu z regularyzacją.

$$\begin{aligned} &\text{Chcemy zminimalizować} && \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ &\text{pod warunkiem} && y^{(i)}\left(w^T x^{(i)} + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, m, \\ & && \xi_i \geqslant 0, \quad i = 1, \ldots, m. \end{aligned}$$

Od razu przeszedłem na problem dualny z mnożnikami Lagrange'a $\alpha$

$$\begin{aligned} \min_{\alpha} \Psi(\alpha) = \min_{\alpha} \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}K(x^{(i)}, x^{(j)})\alpha_i\alpha_j - \sum_{i=1}^{m}\alpha_i \\ 0 \leqslant \alpha_i \leqslant C, \quad \forall i \in \{1, \ldots, m\} \\ \sum_{i=1}^{m} y^{(i)}\alpha_i = 0. \end{aligned}$$

Margines dla punktu $x$ jesteśmy w stanie policzyć bez liczenia $w$ bezpośrednio ze wzoru

$$u = \sum_{i=1}^{m} y^{(i)}\alpha_i K(x^{(i)}, x) - b.$$

Żeby wytrenować model będę korzystał z algorytmu sekwencyjnej minimalnej optymalizacji. Ale zanim przejdziemy do mojej historii o trenowaniu SVM to jeszcze słowo jak odzyskujemy $b$. Zmienną $b$ aktualizujemy na bieżąco zgodnie z artykułem Johna C. Platta *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. W każdym obiecującym wyborze $\alpha_1, \alpha_2$

$$\begin{aligned} b_1 &= u_1 - y^{(1)} + y^{(1)}(\alpha_1^{\text{new}} - \alpha_1)K(x^{(1)}, x^{(1)}) + y^{(2)}(\alpha_2^{\text{new,clipped}} - \alpha_2)K(x^{(1)}, x^{(2)}) + b \\ b_2 &= u_2 - y^{(2)} + y^{(1)}(\alpha_1^{\text{new}} - \alpha_1)K(x^{(1)}, x^{(2)}) + y^{(2)}(\alpha_2^{\text{new,clipped}} - \alpha_2)K(x^{(2)}, x^{(2)}) + b \\ b^{\text{new}} &= \frac{b_1 + b_2}{2}. \end{aligned}$$

Nie musimy zatem odzyskiwać $b$, ponieważ zawsze je znamy :).
Żeby zwrócić predykcję $y$ wystarczy `sign`$(u)$.

## 5.2 Implementacja

Jak wyglądała moja podróż przez piekło SVMa:

1. Dzielnie przeczytałem artykuł i przepisałem ze zrozumieniem pseudokod, o zgrozo z użyciem pętli `for`. Tego nie dało się wytrenować. Po 2 godzinach mielenia danych

nie było widać końca.

2. Wprowadziłem maksymalną liczbę epok i pozbyłem się z kodu wszystkich warunków, które uważałem za niezrozumiałe – na przykład, jeśli $\eta \leqslant 0$ to nie próbuję tego ratować – zawsze możemy wybrać inne $\alpha_1, \alpha_2$ ;). Udało się wytrenować SVMa, ale teraz predykcja trwała dłużej niż trening.

3. Wektoryzacja jąder. Tutaj spędziłem tylko chwilę z `numpy`. Udało się, mam wynik

   `SVM accuracy:` 0.5109

4. Załamałem się, użyłem wszystkich heurystyk wyboru $i_1, i_2$ z artykułu. Nawet wprowadziłem `error_cache`, żeby móc uczyć przez więcej epok. Wszystko na nic, również dobrze mógłbym rzucać monetą w czasie stałym.

5. Oświecenie. Źle liczyłem samą predykcję, funkcja `np.nonzero` (która miała na celu optymalizację – zwracała indeksy $\alpha_i \neq 0$ – wektory nośne, których było zazwyczaj $\approx 900$ – znacznie mniej niż wielkość zbioru danych) jest bardzo podła i zwraca coś czego byśmy się nie spodziewali. Do podmianie na `np.argwhere` w końcu można zabrać się za dobieranie hiperparametrów.

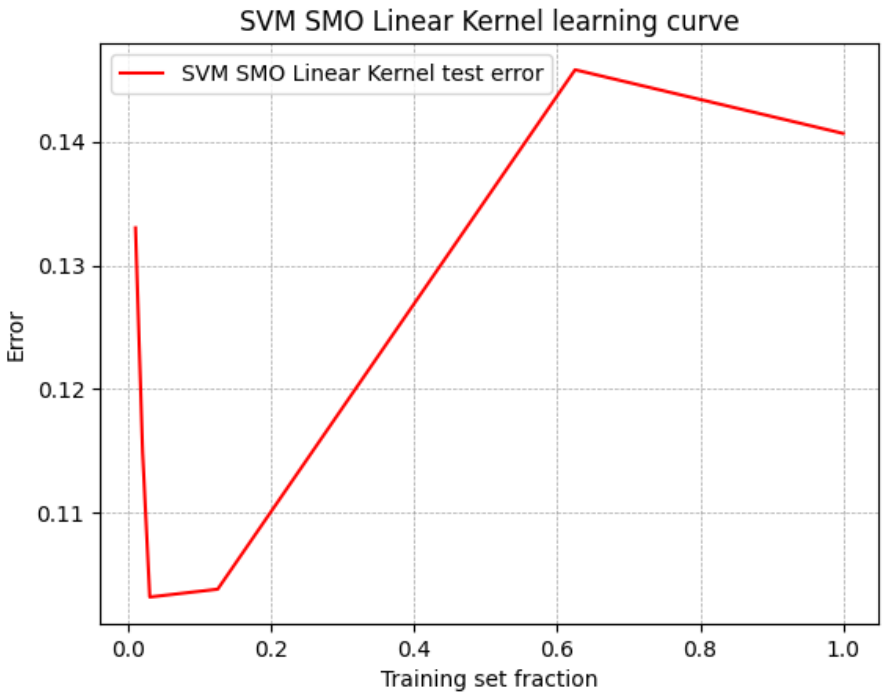
## 5.3 Hiperparametry i jądra

Zaznaczę, że przy SVMach nie stosuję standaryzacji cech.
Zaczynamy od jądra liniowego – zwykłego iloczynu skalarnego. Od tego momentu błąd to znany błąd zero-jedynkowy.

`Average validation error for SVM SMO with C=5.0:` 0.1372
`Average validation error for SVM SMO with C=1.0:` 0.1304
`Average validation error for SVM SMO with C=0.1:` 0.1373
`Average validation error for SVM SMO with C=0.01:` 0.1373

Jak widać $C = 1$ tworzy lokalne minimum. Trenowałem model przez 5000 epok. Zajęło to 4.8s. Dało przyzwoity wynik.

`Average test accuracy for SVM SMO with C=1.0:` 0.8680




Rysunek 3: Krzywa uczenia SVMa z liniowym jądrem

Ten wykres wygląda co najmniej dziwnie. Być może jest to wina doborów $\alpha_1, \alpha_2$ lub zmiany wektorów nośnych.

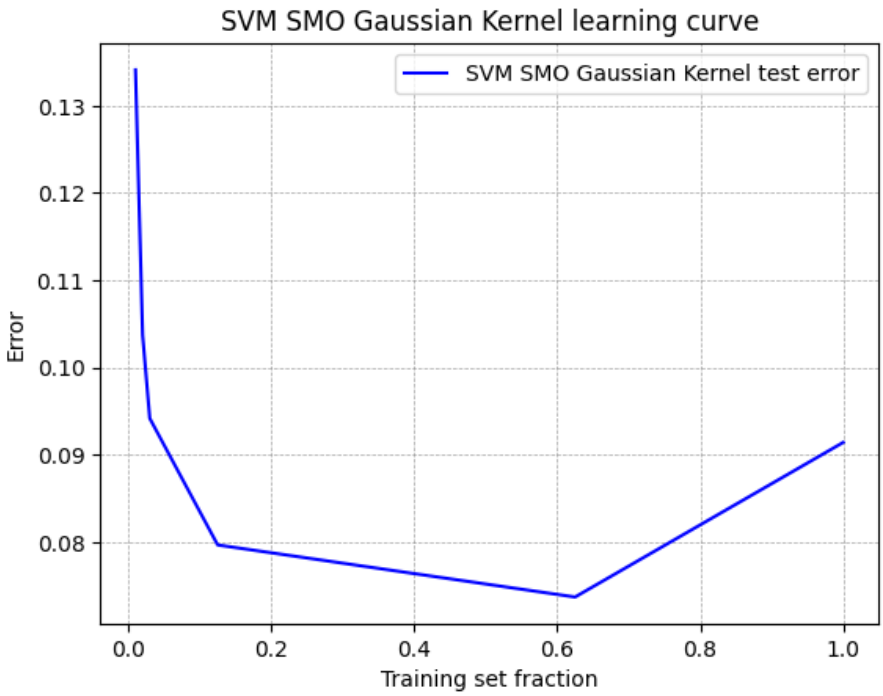
Teraz jądro gaussowskie. Oczekujemy, że wynik się poprawi, a czas nauki wydłuży, ponieważ liczenie `exp` jest kosztowne. Zostawiłem $C = 1$, a walidacja wskazała $\sigma = 7$.

`Average validation error for SVM SMO with Gaussian kernel sigma=0.1:` 0.4370
`Average validation error for SVM SMO with Gaussian kernel sigma=1:` 0.3433
`Average validation error for SVM SMO with Gaussian kernel sigma=7:` 0.1297
`Average validation error for SVM SMO with Gaussian kernel sigma=14:` 0.1990

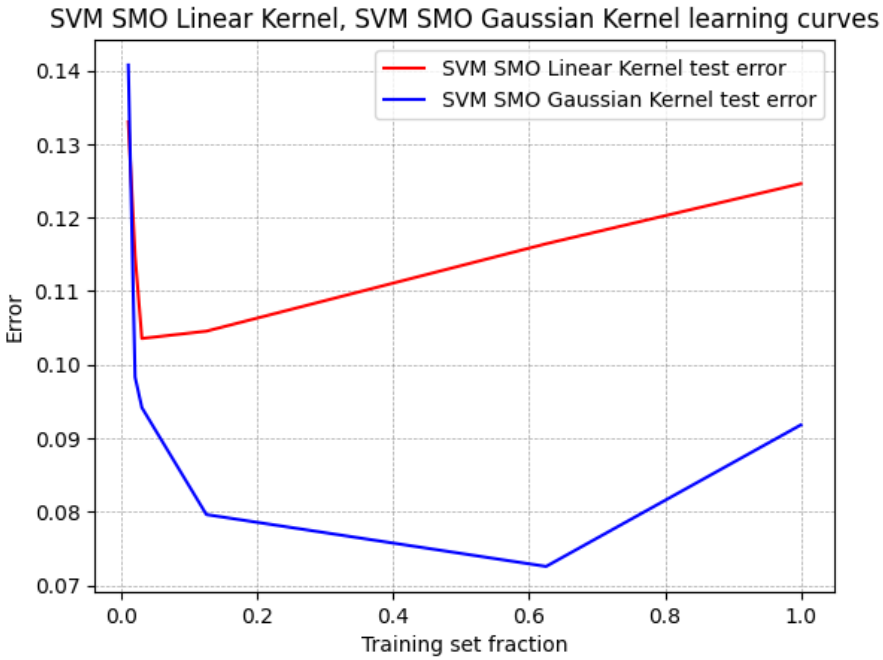
Czas nauki dla 5000 epok wyniósł 18.2s – 4 razy dłużej. Wyniki nie są dużo lepsze, być może dane były prawie liniowo separowalne.

`Average test accuracy for SVM SMO with Gaussian kernel sigma=7:` 0.9091

Natomiast mam wrażenie, że samo uczenie przebiega bardziej gładko co widać na wykresie.




Rysunek 4: Krzywa uczenia SVMa z gaussowskim jądrem




Rysunek 5: Krzywe uczenia SVMów

# 6 Drzewo decyzyjne a miał być las

## 6.1 Wstęp

Drzewa decyzyjne są bardzo intuicyjne i powinny dawać bardzo dobre rezultaty dla zbioru treningowego. One po prostu dzielą przestrzeń na podstawie najlepszych cech i podprzestrzeniom nadają etykiety zgodnie z głosem większości próbek w danej podprzestrzeni.

## 6.2 Implementacja

Ja do sprawdzania, jak dobry jest podział, użyłem klasycznej miary – `Gini Impurity`. Implementacja obyła się bez jakichkolwiek problemów. Przeszukujemy dany podzbiór (tak, podzbiór bo w zamyśle chciałem mieć gotową implementację pod las, a dla drzew przekazywać cały zbiór cech zawsze) cech, wybieramy najlepszą i rekursja. Podkradłem ujmującą sztuczkę z kanału na YouTube StatQuest – możemy posortować unikalne wartości danej cechy i sprawdzać podziały ze względu na średnią 2 sąsiednich elementów. To nam załatwia cechy binarne i resztę bez ifowania. Jako hiperparametr wprowadziłem `max_depth`, żeby ograniczyć przeuczenie.

## 6.3 Wielkie odkrycie

Wytrenowałem drzewo z `max_depth` = 8, żeby zobaczyć czy wszystko działa.

`Validation error fot DT:` 0.0644

Ja nie dowierzałem. Oczekiwałem, że będzie źle. Wytrenowałem drzewo o głebokości 1 – to jest dokładnie 1 podział.

`Validation error fot DT:` 0.1111

Co więcej to była cecha $x_{25}$ – przywołując macierz korelacji – cecha o najwyższym współczynniku Pearsona. Dopiero teraz zdałem sobie sprawę jak bardzo mogłem przyspieszyć SVMa. Wystarczyło usunąć nieistotne cechy. Taki wynik drzewa decyzyjnego potwierdza nam również, że dane są prawie liniowo separowalne – przynajmniej tak mi się wydaje.

## 6.4 Hiperparametry

`Average validation error for DT with max_depth=1:` 0.1111

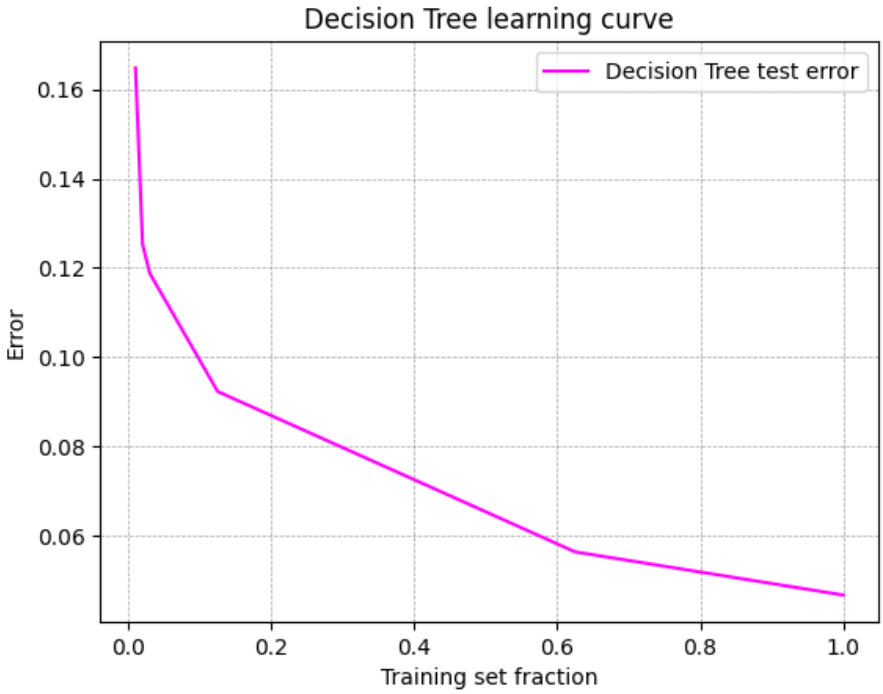
`Average validation error for DT with max_depth=8:` 0.0644

`Average validation error for DT with max_depth=16:` 0.0438

Pewnie moglibyśmy zejść z błędem jeszcze niżej, ale trenowanie trwało już zbyt długo, pętle `for` są bardzo niewydajne. Ostatecznie drzewo decyzyjne z maksymalną głębokością 16 trenowało się 18.5s. Osiągnęło świetny wynik.

`Average test accuracy for DT with max_depth=16:` 0.9533

Podzieliło przestrzeń na 363 podprzestrzenie – być może bym to jakoś skomentował, jeśli obejrzałbym wykład 10., który zdaje się być o wymiarze Vapnika. Trenowało się bardzo gładko.




Rysunek 6: Krzywa uczenia drzewa decyzyjnego

# 7 Głęboka sieć neuronowa

## 7.1 Wstęp

Plany były bardzo ambitne. Chciałem pobawić się z różnymi metodami schodzenia gradientem: SGD, Nesterov, Adam. Niestety piekielny SVM pokrzyżował moje plany. Skończyło się na następującej architekturze:

Warstwa 1: input_dim = 30, output_dim = 32
Warstwa 2: input_dim = 32, output_dim = 16
Warstwa 3: input_dim = 16, output_dim = 1
Na końcu: funkcja aktywacji `sigmoid`

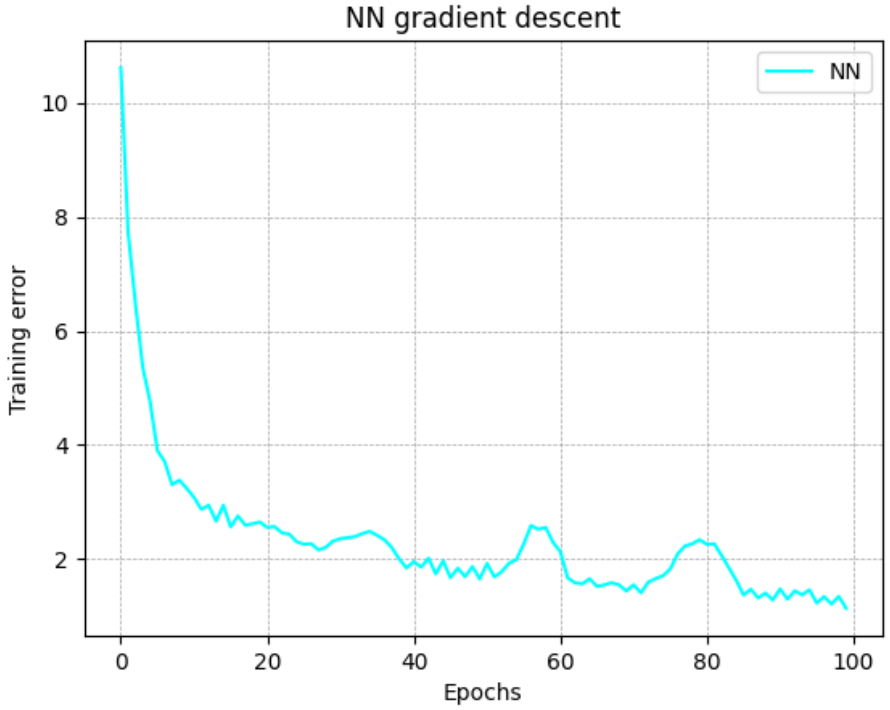
Ta architektura nie ma większego uzasadnienia. Bierzemy liczbę cech i tak stopniowo zmniejszamy aż do 1. Ponadto dla każdej warstwy są wyrazy wolne, wagi inicjalizuję metodą `He` – mam już doświadczenie, że bez tego gradient eksploduje. Gradient to zwykły `SGD` z `batch_size` i stałym `learning_rate`. Jako funkcję aktywacji używam `ReLU`, a funkcja straty to znana binarna entropia krzyżowa.

## 7.2 Implementacja

Jedyną trudnością jest tutaj propagacja wsteczna. Trzeba chwilę pogłówkować jak użyć tam `numpy`.

## 7.3 Hiperparametry

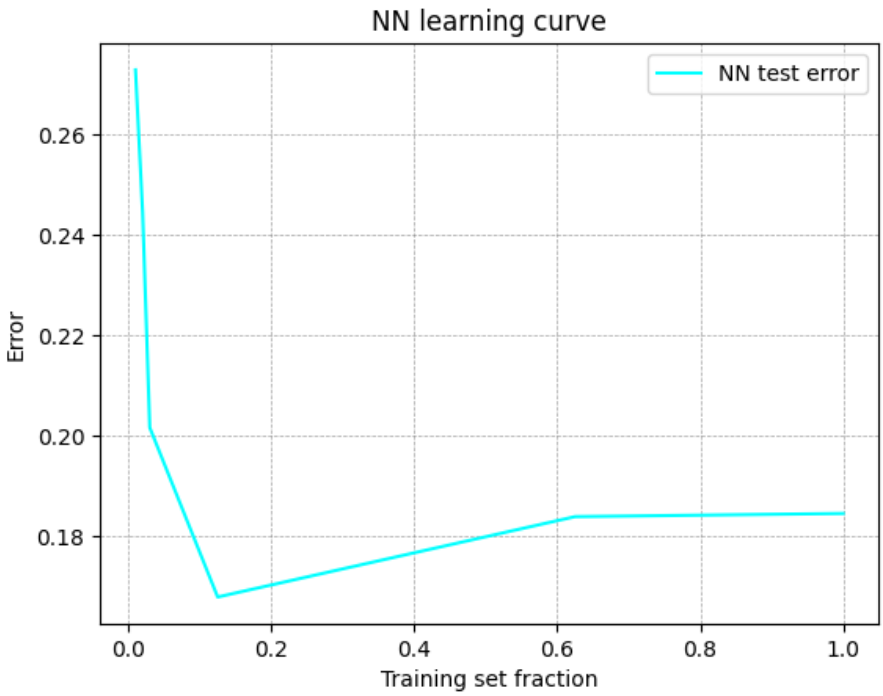
Ostatecznie skończyło się na `batch_size` = 512, `epochs` = 100, `learning_rate` = 0.01. Bardzo lubię oglądać wykresy training error vs epoch, więc taki też przygotowałem. To pokazuje jakie problemy ma optymalizator, żeby wbić się w minimum. Lepsze optymalizatory byłyby gładsze.




Rysunek 7: Spadek gradientowy

Jak można było się spodziewać, taka prosta sieć średnio sobie radzi, ma trudności z zejściem do minimum, być może schodzi do złego minimum.

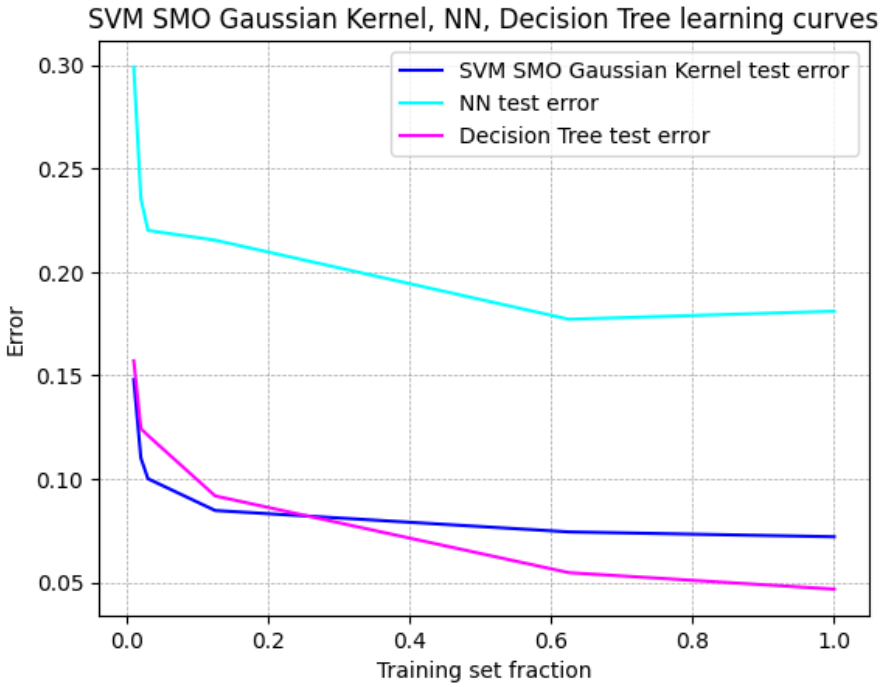
`Average accuracy for NN:` 0.8063




Rysunek 8: Krzywe uczenia NN

# 8 Porównanie

Absolutnie nie spodziewałem się takiego wyniku. Szczerze liczyłem, że piękna matematyka, która stoi za SVM zwycięży, jednak okazała się bardzo trudna do implementacji. Najprostsze drzewo decyzyjne okazało się najlepsze w wykrywaniu czy strona służy do phishingu.




Rysunek 9: Wszystkie krzywe uczenia

# 9 Podsumowanie

Ten miniprojekt nauczył mnie wiele. Po pierwsze nie ufać nazwom funkcji. Do drugie pętla `for` jest bardzo wolna w `Pythonie`. Po trzecie i najważniejsze warto sprawdzić kilka modeli dla 1 problemu, możemy na przykład odkryć, że dane są prawie liniowo separowalne przez 1 cechę.