Miniprojekt III

Metody probabilistyczne w uczeniu maszynowym

Łukasz Trzos

0 Temat projektu

Plik **phishing.data** zawiera dane o stronach internetowych wraz z informacją, czy dana strona służy do phishingu (1), czy też jest to strona bezpieczna (-1). Każdy wiersz zawiera 31 liczb. Pierwsze 30 z nich to cechy, spośród których:

- pierwsze 21 przyjmuje wartości ze zbioru $\{-1, 1\}$,
- kolejnych 8 cech przyjmuje wartości ze zbioru $\{-1, 0, 1\}$,
- ostatnia cecha przyjmuje wartość ze zbioru $\{0, 1\}$.

W ostatniej kolumnie znajduje się informacja, czy dana strona jest stroną phishingową.

W oparciu o powyższe dane napisz trzy programy uczące, które zwrócą jak najlepsze klasyfikatory stron podejrzanych o phishing. Jeden z programów powinien bazować na **metodzie wektorów nośnych** (SVM), przy czym można korzystać z dowolnych funkcji jądrowych. Pozostałe dwa programy powinny być oparte na dwóch spośród poniższych **klasyfikatorów nieliniowych**:

- metoda k najbliższych sąsiadów (kNN),
- drzewa decyzyjne,
- lasy losowe,
- AdaBoost,
- sieć neuronowa.

Wykorzystaj część zbioru obserwacji jako zbiór treningowy, odpowiednio mniejszą część jako zbiór walidacyjny, a pozostałą część jako zbiór testowy.

W raporcie opisz zastosowane algorytmy uzasadniając wybór parametrów modeli. Dokonaj analizy uzyskanych wyników oraz przedstaw je na odpowiednich wykresach.

W ramach ciekawostki zapoznaj się z dołączonymi artykułami. Czy uzyskane przez Ciebie wyniki są zbliżone do wyników przedstawionych w którymś z artykułów?

1 Klasyfikator SVM

1.1 Opis teoretyczny

Klasyfikator SVM stara się znaleźć hiperpłaszczyznę, która separuje dane należące do dwóch klas. Gdybyśmy otrzymali dane idealnie liniowo separowalne, szukalibyśmy wektora w oraz współczynnika b takich, że:

- $\forall_{i\in[m]} : y^{(i)} = 1 \implies wx^{(i)} + b \geqslant 1$
- $\forall_{i\in[m]} : y^{(i)} = -1 \implies wx^{(i)} + b \leqslant -1$

Można to podsumować jedną formułą przy założeniu, że klasy mają indeksy 1 oraz -1:

$$\forall_{i\in[m]} : y^{(i)}(wx^{(i)} + b) \geqslant 1$$

Marginesem klasyfikatora SVM jest najkrótszy dystans pomiędzy 2 punktami należącymi do przeciwnych klas. Są to wektory nośne, czyli punkty, dla których w powyższej formule zachodzi równość.

Odległością punktu x' od zdefiniowanej hiperpłaszczyzny wyrażamy wzorem $\frac{|wx'+b|}{||w||^2}$. Dla dowolnego x' takiego, że $y'(wx' + b) = 1$ ten dystans wynosi $\frac{1}{||w||^2}$, zatem minimalna odległość 2 punktów z przeciwnych klas od siebie wynosi $\frac{2}{||w||^2}$. Chcemy zmaksymalizować margines, czyli równocześnie zminimalizować normę wektora w.

Zazwyczaj jednak mamy do czynienia z danymi, których nie można idealnie separować. Wprowadzamy więc zmienne ξ_i, które pozwalają na ustępstwa. Redefiniujemy problem następująco:

Minimalizujemy:

$$\frac{1}{2}||w||^2 + \frac{C}{m}\sum_{i=1}^{m}\xi_i$$

Pod warunkami:

- $\forall_{i\in[m]} : y^{(i)}(wx^{(i)} + b) \geqslant 1 - \xi_i$
- $\forall_{i\in[m]} : \xi_i \geqslant 0$

C jest współczynnikiem kosztu służącym karaniu modelu za zbyt duże wartości zmiennych ξ_i. Najelpszą jego wartość możemy wyznaczyć dla konkretnych danych za pomocą zbioru walidacyjnego.

1.2 Problem dualny

Za pomocą mnożników Lagrange'a możemy sformułować problem dualny do powyższego. Jego treść wygląda następująco:

Maksymalizujemy:

$$W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}((x^{(j)})^T x^{(i)})$$

Pod warunkami:

- $\sum_{i=1}^{m}\alpha_i y_i = 0$
- $\forall_{i\in[m]} : 0 \leqslant \alpha_i \leqslant C$

Formułujemy ten problem, ponieważ pozwala to na wykorzystanie funkcji jądrowych. Będziemy chcieli korzystać z funkcji odwzorowania cech $\Phi(x)$ bez liczenia ich wartości bezpośrednio. W nowym wzorze, jedyne miejsce występowania wektorów cech $x^{(i)}$ to ich iloczyn, możemy więc funkcję w zastąpić:

$$W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y^{(i)}y^{(j)}\kappa(x^{(i)}, x^{(j)})$$

gdzie κ to stosowana funkcja jądrowa

Żeby znaleźć wektor α maksymalizujący W skorzystamy z biblioteki scikit-learn.

```python
N = len(trainingDataset)

K = np.array([np.array([kernelFunction(X_train[i], X_train[j]) for j in range(N)]) for i in range(N)])

def W(alpha):
    alpha = np.array(alpha)
    return float(-1) * (np.sum(alpha) - 0.5 * alpha @ np.diag(y_train) @ K @ np.diag(y_train) @ alpha)

EqualityConstraint = {
    'type': 'eq',
    'fun': lambda alpha: np.sum(alpha * y_train)
}
Bounds = [(0, C)] * N

initialAlpha = np.zeros(N)

result = minimize(W, initialAlpha, constraints=[EqualityConstraint], bounds=Bounds)
alpha = result.x
```

Rysunek 1: Rozwiązanie wspomnianego problemu korzystające z scikit-learn

Po rozwiązaniu problemu dualnego otrzymujemy następującą funkcję decyzyjną:

$$f(X) = \text{sgn}(\sum_{i=1}^{m} \alpha_i y^{(i)} \kappa(x^{(i)}, X) + \beta)$$

Wektory których odpowiadające współczynnik Lagrange'a α_i są ściśle większe od ustalonego ϵ (u nas $\epsilon = 10^{-5}$) to wektory wspierające, tylko one mają wpływ na rezultat finalnej funkcji decyzyjnej. Leżą one najbliżej hiperpłaszczyzny oddzielającej klasy, dlatego "wspierają ją".

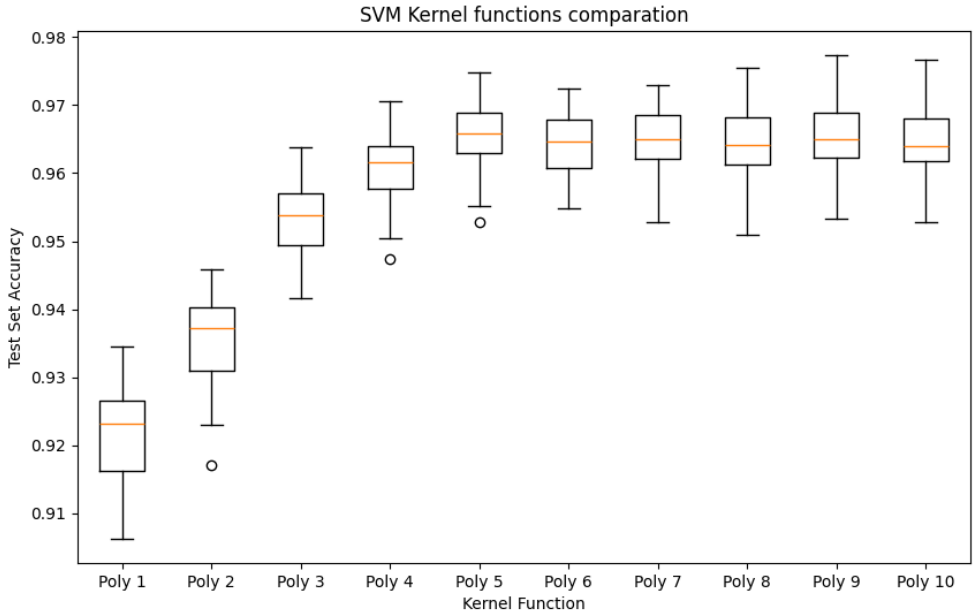
1.3 Eksperymenty

Wypróbujemy jak różne funkcje jądrowe sprawdzają się przy objaśnianiu naszych danych. Wykorzystamy funkcje wielomianowe: $\kappa(x_i, x_j) = (\gamma x_i^T x_j + \beta)^d$ dla różnych wartości parametrów (d, γ, β), oraz funkcję radialną: $\kappa(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$ z parametrem γ. Poniżej przedstawiam wyniki poszczególnych eksperymentów wraz z wyciągniętymi z nich wnioskami.

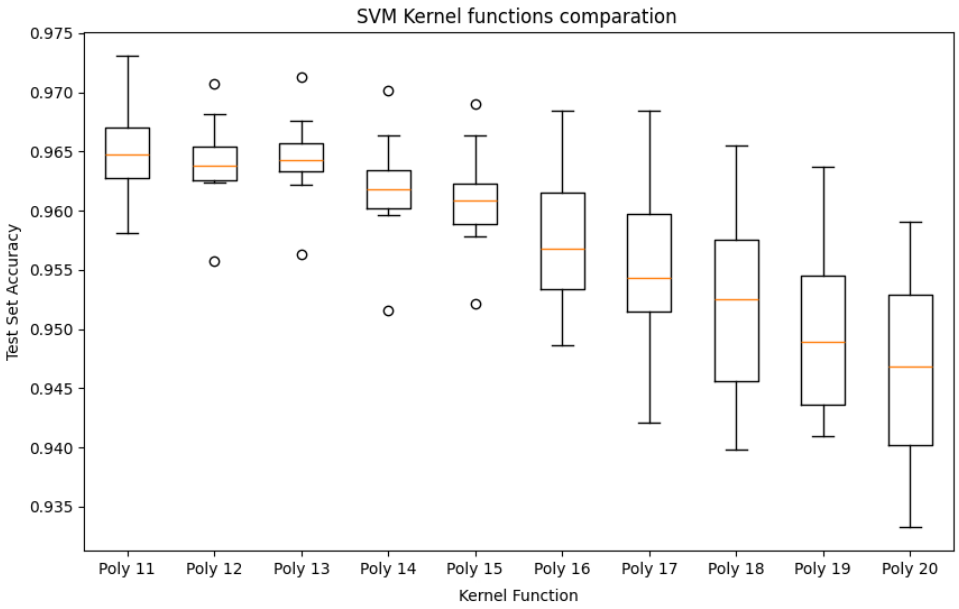
Po porównaniu różnych modeli i wybraniu najlepszego oraz przetestowaniu ich na stosunkowo dużej liczbie niezależnych testów, najlepszy model korzystający z metody wektorów nośnych osiągnął skuteczność 96.6%.

Na początku porównałem jak wzrasta skuteczność modelu wraz ze wzrostem stopnia wielomianu. Jak widać, już model liniowy osiąga skuteczność na poziomie 92%. W dalszych eksperymentach rozważałem model "Poly 5", ponieważ był ostatnim, który w zauważalny sposób poprawił skuteczność modelu, wyniosła ona ok. 96.5%.

Dla wyższych stopni wielomianu następował już wyraźny spadek skuteczności ze względu na dopuszczenie zbyt skomplikowanych granic między klasami i przeuczenie modelu.

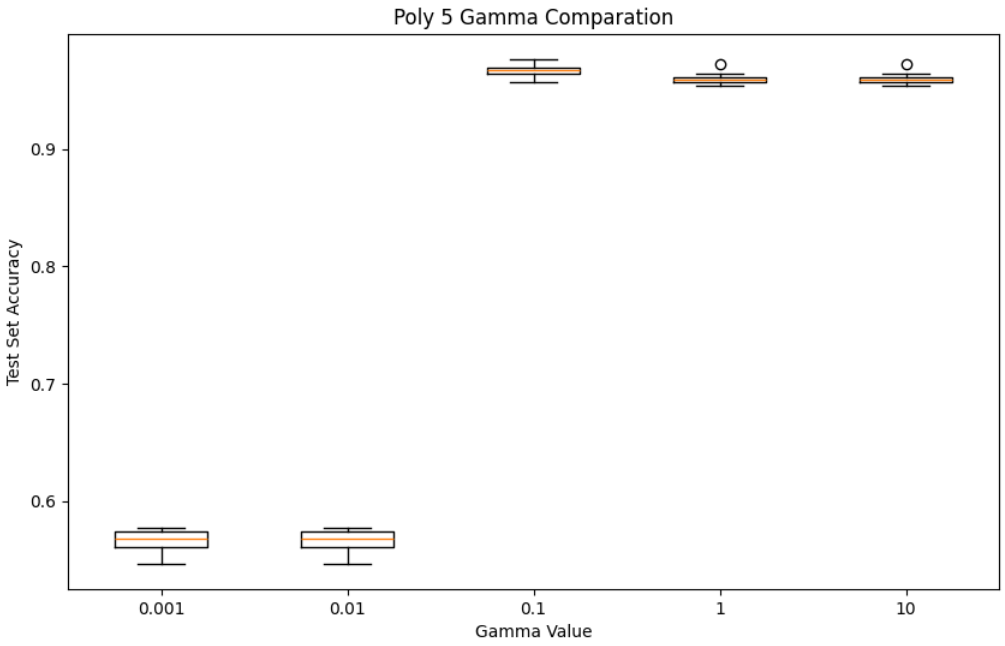


Rysunek 2: Porównanie wyników osiąganych przez różne stopnie wielomianu

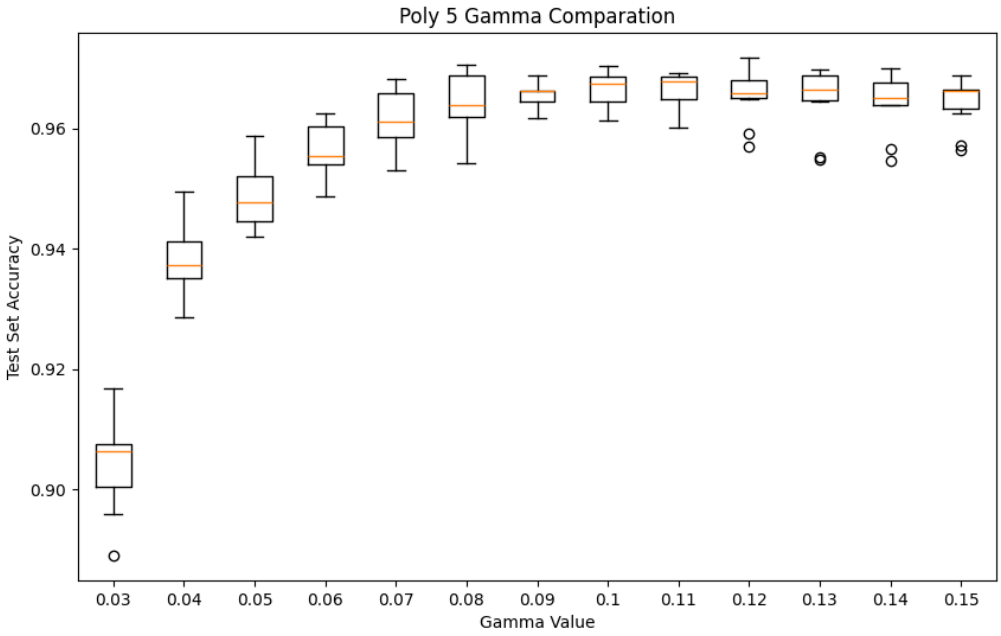


Rysunek 3: Wyższe stopnie wielomianu

Jak widać, zbyt mała wartość parametru gamma doprowadza do modelu, który jest niewiele skuteczniejszy niż model rzucający monetą. Stało się tak, ponieważ w takiej sytuacji wartość funkcji jądrowej κ jest bardzo niewielka, przez co funkcja decyzyjna jest praktycznie stała. Dodatkowo sprawdziłem wyższe wartości parametru, oraz poszukałem dookoła lokalnego ekstremum w $\gamma = 0.1$. Nie poprawiło to jednak znacząco skuteczności modelu.



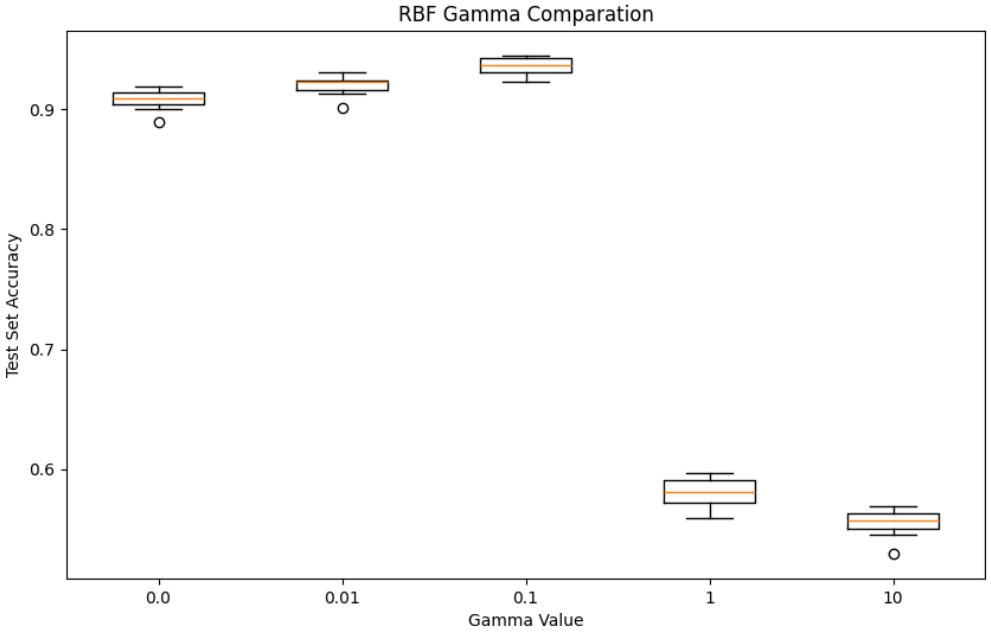
Rysunek 4: Parametr gamma dla funkcji Poly 5



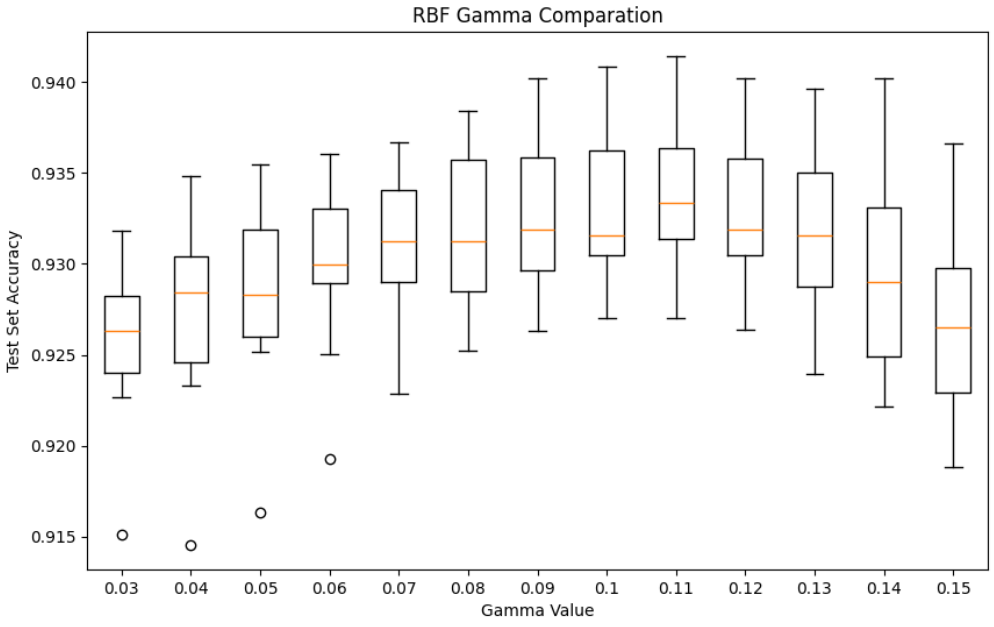
Rysunek 5: Parametr gamma dla funkcji Poly 5

Przetestowałem również funkcję RBF definiowaną wzorem $\kappa(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$. Tutaj co ciekawe, w przeciwieństwie do modelu wielomianowego, bardzo słabe modele uzyskałem już przy większych wartościach parametru γ z zestawu początkowo testowanych wartości. Ponownie, lokalne ekstremum znajdowało się wokół wartości 0.1.

Najlepszy model korzystający z RBF jaki udało mi się wytrenować osiągał skuteczności na poziomie 93.2%. Wyraźnie lepsze niż wcześniej omawiany model liniowy, de facto nie korzystający z dobrodziejstw funkcji jądrowych, ale znacznie gorsze niż modele wielomianowe z odpowiednio dobranymi parametrami.



Rysunek 6: Parametr gamma dla funkcji RBF



Rysunek 7: Parametr gamma dla funkcji RBF

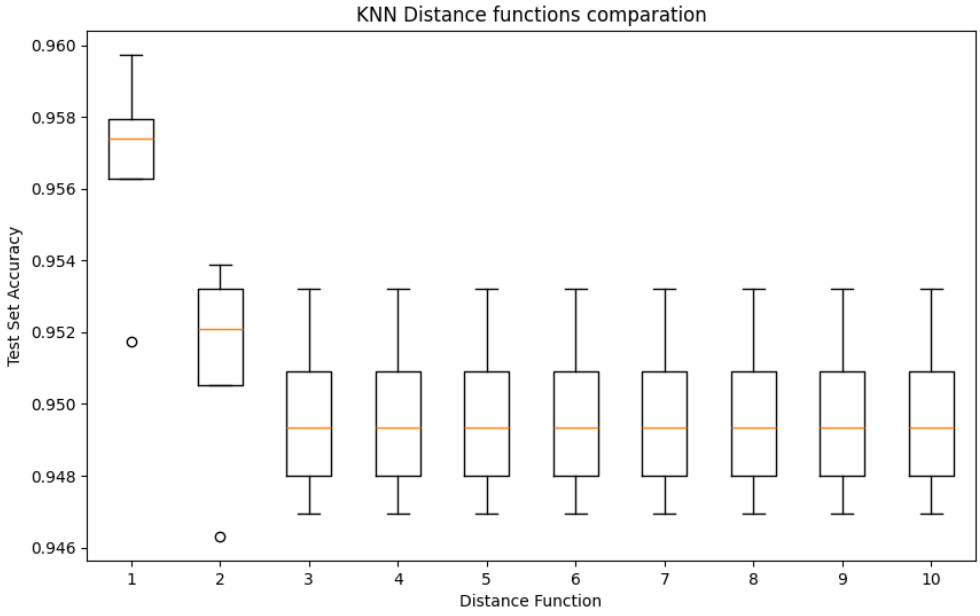
2 Klasyfikator kNN

Działanie tego klasyfikatora jest bardzo proste. Dla nowego wektora cech wybieramy k rekordów ze zbioru treningowego, które znajdują się najbliżej. Następnie wybieramy dla niego tę klasę, która w zbiorze najbliższych sąsiadów występuje częściej.

Ważne są wybory 2 parametrów. Po pierwsze, jakiej funkcji odległości danych od siebie używać. Oczywiście przed trenowaniem modelu przeskalowałem wszystkie dane do przedziałów $[0, 1]$, żeby każda miała równe znaczenie przy ustalaniu odległości. Model KNN przetestowałem dla różnych norm L_p:

$$L_p(x, y) = \sqrt[p]{\sum_{i=1}^{d} |x_i - y_i|^p}$$

Po drugie, jaką zastosować wartość parametru k. Oczywiście najlepiej, żeby k było nieparzyste, w ten sposób unikamy remisów. Nie testowałem poszczególnych wartości osobno, model wykorzystywał zbiór walidacyjny żeby ustalić najlepsze k. Tutaj niepotrzebnie dawałem mu dużo możliwych wartości tego parametru, zazwyczaj wybierał spośród kilkudziesięciu kandydatów i i tak decydował się na bardzo niską wartość $k \in \{1, 3, 5\}$, znacząco spowolniło to czas uczenia się.



Rysunek 8: Porównanie norm L_p

Porównałem jak zachowują się poszczególne normy L_p. Jak widać, wyniki dla $p \in [3, 10]$ są identyczne. Wraz ze wzrostem p norma zbliża się coraz bardziej do normy $L_\infty = max_{i \in [d]}(|x_i - y_i|)$. W naszym przypadku mamy 30 niezależnych cech o rozkładzie dyskretnym przyjmującym $2 - 3$ wartości, co oznacza, że dla każdej pary 2 różnych punktów ta norma wyniesie 0.5 lub 1. Z tego powodu nie zdecydowałem się jej testować, ale eksperyment z innymi normami w zasadzie pokazał, jaką skuteczność prawdopodobnie by ona osiągnęła.

Zdecydowanie najlepiej z opisywaniem danych poradziła sobie norma manhattańska, wyraźnie gorsze, choć nadal satysfakcjonujące, wyniki uzyskaliśmy przy pomocy normy euklidesowej.

3 Klasyfikator AdaBoost

3.1 Opis teoretyczny

Algorytm AdaBoost (Adaptive Boosting) opiera się na połączeniu wielu słabych klasyfikatorów w jeden silny klasyfikator. Słaby klasyfikator to niezwykle prosty model, który można bardzo szybko wytrenować, oraz którego skuteczność jest niewiele lepsza od losowej.

Rekordom ze zbioru treningowego przypisujemy wagi, początkowo $\forall_{i\in[m]} : w_1^{(i)} = \frac{1}{m}$. Iteracja algorytmu wygląda następująco:

1. Spośród dostępnych słabych klasyfikatorów wybierz ten, który na danych treningowych osiąga najmniejszy błąd ważony. Błąd ważony klasyfikatora h_t to:

$$\epsilon_t = \sum_{i=1}^{m} w_t^{(i)} \mathbf{1}[h_t(x^{(i)}) \neq y^{(i)}]$$

2. Wybranemu klasyfikatorowi przypisujemy wagę:

$$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$$

3. Aktualizujemy wagi rekordów ze zbioru treningowego według wzoru:

$$w_{t+1}^{(i)} = \frac{w_t^{(i)} exp(-\alpha_t y^{(i)} h_t(x^{(i)})}{Z_t}$$

gdzie Z_t jest współczynnikiem regularyzacji dbającym o to, żeby wagi sumowały się do 1.

Po wykonaniu T iteracji, silna funkcja decyzyjna jest tworzona przez ważoną sumę predykcji wybranych w czasie działania algorytmu klasyfikatorów słabych:

$$f(X) = sgn(\sum_{t=1}^{T} \alpha_t h_t(X))$$

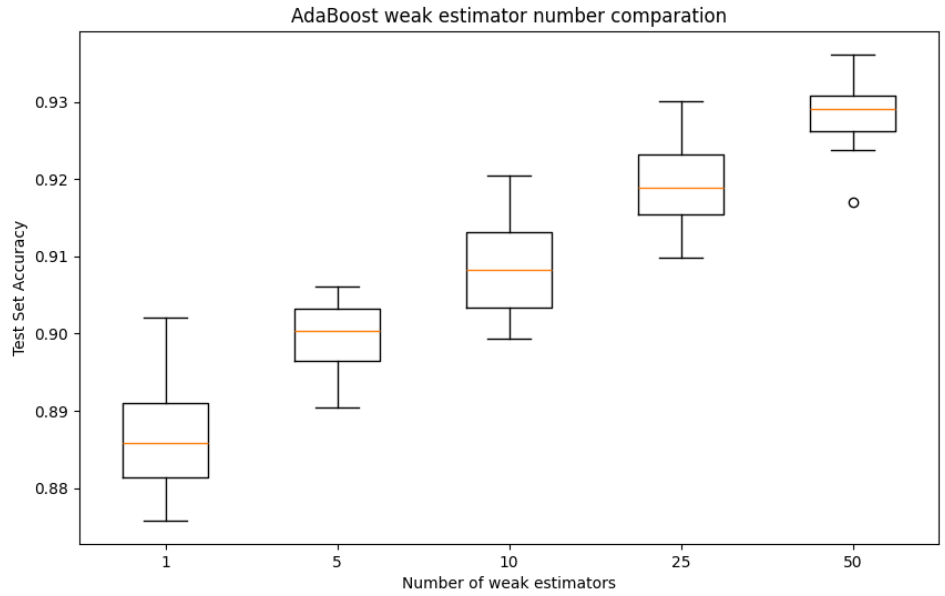
3.2 Wybór słabych klasyfikatorów

Bardzo ważny jest oczywiście wybór przestrzeni hipotez, z której będziemy wybierać najlepszy klasyfikator słaby podczas kolejnych iteracji algorytmu. Oprócz tego, musimy ustalić parametr T, czyli liczbę słabych klasyfikatorów wchodzących w skład silnego. Poniżej opisuję wyniki uzyskane dla klasyfikatorów szkolonych na różnych przestrzeniach hipotez i wartościach parametrów:

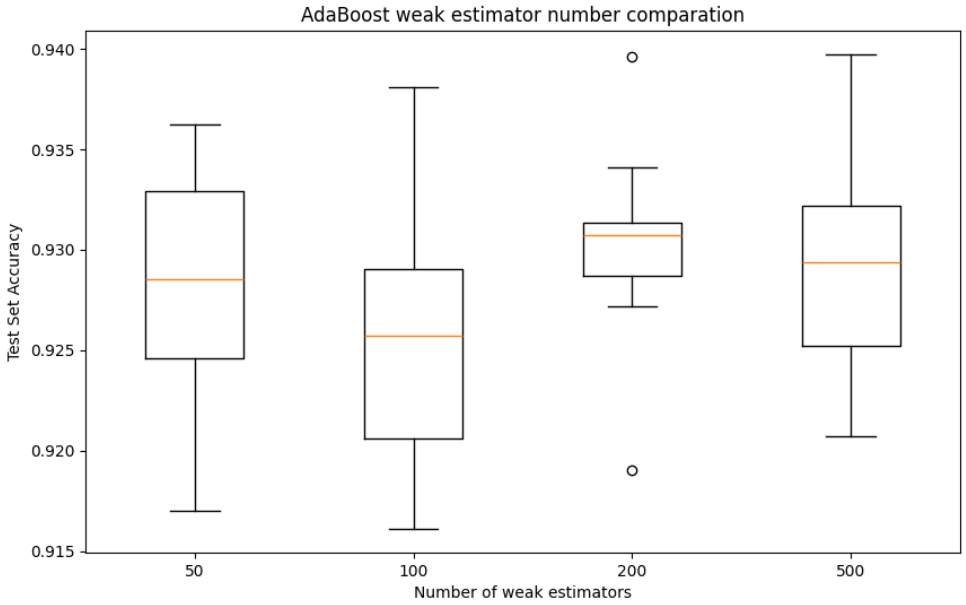
Na początku wykorzystałem zbiór klasyfikatorów skupiających się na pojedynczej cesze rekordu. Wyglądają one następująco: $f_{i,j}(X) = \mathbf{1}[X_i = j] - \mathbf{1}[X_i \neq j]$. Przestrzeń hipotez to lista opisanych klasyfikatorów utworzonych dla każdej z 30 cech i każdej z 3 wartości. Na poniższych wykresach przedstawione są średnie wyniki uzyskiwane przez silne klasyfikatory trenowane na tym zbiorze w zależności od parametru T.

Ciekawe jest to, że dla $T = 1$, czyli de facto używając tylko jednej cechy, uzyskaliśmy zadziwiająco dobrą skuteczność na poziomie 88.5%. Oznacza to, że istniała przynajmniej jedna pojedyncza cecha, która była bardzo silnie skorelowana z przypisywaną klasą.

Gorsze wyniki w przypadku dużych wartości współczynnika wynikają z mojego niedopatrzenia. Przy wykonywaniu tych testów, generowałem osobne podziały na dane treningowe i testowe dla obydwu algorytmów, przez co duży wpływ na wyniki miały niefortunne losowania. Nie ma to jednak większego znaczenia dla eksperymentu, ponieważ później uzyskaliśmy lepsze wyniki dla większej liczby iteracji.

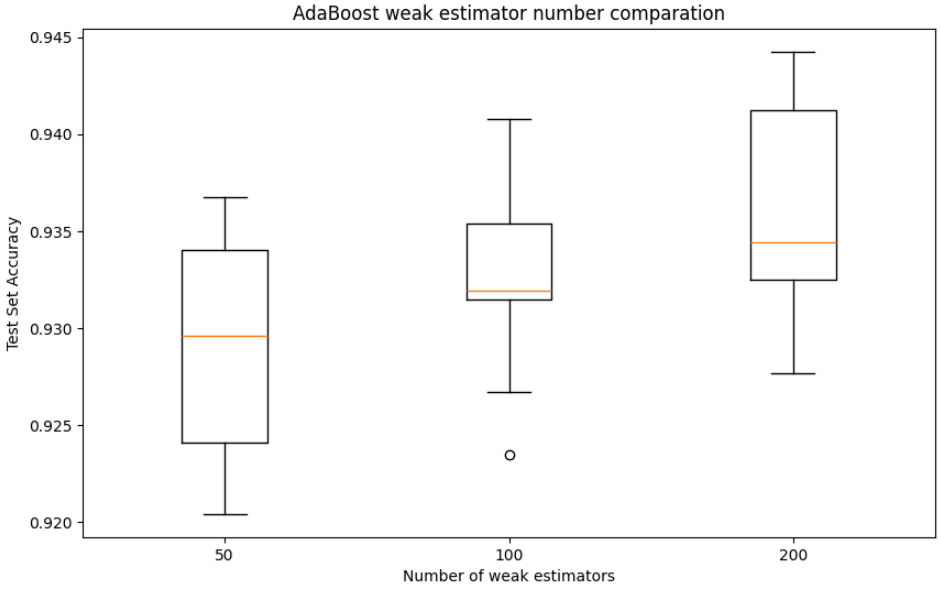


Rysunek 9: Mała liczba słabych klasyfikatorów



Rysunek 10: Większa liczba słabych klasyfikatorów

Dodałem również klasyfikatory oparte o ten sam schemat, ale tym razem biorące pod uwagę nierówności, licząc na niewielką poprawę. Nierówności w zasadzie dają jakąkolwiek różnicę tylko w przypadku cech mogących przyjmować 3 wartości, których nie było dużo, ale i tak udało się zanotować widoczną wyników.



Rysunek 11: Większa liczba słabych klasyfikatorów

Wypróbowałem również słabe klasyfikatory oparte o 2 lub więcej cech. Dało to jednak znikomą poprawę wyników, a znacząco zwiększyło czas potrzebny do wytrenowania algorytmu, dlatego zrezygnowałem z uwzględniania tych wyników w raporcie.

4 Wnioski

Poniższe tabele przedstawiają wyniki uśrednione dla już większej liczby losowych przebiegów algorytmów uzyskane przy pomocy najlepszych wytrenowanych klasyfikatorów z każdej grupy:

SVM:

Miara	**Wartość**
Minimum	0.9558
Pierwszy kwartyl	0.9626
Średnia	0.966
Trzeci kwartyl	0.968
Maksimum	0.977

kNN:

Miara	**Wartość**
Minimum	0.9502
Pierwszy kwartyl	0.9575
Średnia	0.9583
Trzeci kwartyl	0.9633
Maksimum	0.966

AdaBoost:

Miara	**Wartość**
Minimum	0.928
Pierwszy kwartyl	0.9331
Średnia	0.9348
Trzeci kwartyl	0.9419
Maksimum	0.9443

AdaBoost uzyskał ostatecznie dość wyraźnie gorsze wyniki od pozostałych dwóch klasyfikatorów. Wydaje mi się, że ten wynik może być związany z dużą obecnością szumu w zbiorze danych. Klasyfikatory SVM oraz kNN, który wybiera niską wartość k, są dość odporne na odstające dane, natomiast AdaBoost przypisuje im coraz większą wagę w kolejnych iteracjach, starając się do nich dopasować, przez co traci na skuteczności w przypadkach generalnych. AdaBoost uzyskał jednak zadziwiająco dobre wyniki dla najprostszych możliwych klasyfikatorów opartych o jedną cechę, co świadczy o silnej korelacji co najmniej kilku cech ze zmienną objaśnianą.

Fakt, że algorytm kNN wybierał niskie wartości k pokazuje, że dane są dosyć mocno zależne od siebie lokalnie, i do oceny nowej strony możemy użyć jedynie najbliższego jej sąsiada by uzyskać podobne wyniki. Fałszywe strony są często produkowane według podobnych schematów, dlatego jeśli mamy wsród danych bardzo podobny przypadek, który okazał się być oszustwem, bezpiecznie jest również sklasyfikować nową stronę jako podejrzaną.

Najlepiej poradził sobie SVM. Ta metoda generalnie dobrze sobie radzi dla danych o bogatym zestawie cech, gdzie zależności mogą być wielowymiarowe. Dzięki wykorzystaniu funkcji jądrowych mogliśmy szukać granicy decyzyjnej na wielu wymiarach. Załączony artykuły również podkreślały wysoką skuteczność tej metody. Jakby nie patrzeć jest też ona najbardziej skomplikowana przy implementacji, więc dobrze, że ten czas wynagradza skutecznością.

W artykułach podkreślona była również istotność wybrania odpowiedniego algorytmu na podstawie konkretnego zbioru danych i wiedzy o nim. Akurat w przypadku otrzymanych w eksperymencie danych, których poszczególne cechy wyłapywaliśmy podczas omawiania algorytmów, najlepiej sprawdziła się klasyfikacja oprata o wektory nośne.