Miniprojekt 3

Na początku stworzyłem moduł **division_and_scaling.py**, w którym w zależności od zapotrzebowania mogę podzielić w losowy sposób dane wejściowe na 2 zbiory - treningowy (2/3 obserwacji) i testowy (1/3 obserwacji) lub na 3 zbiory – treningowy (60%), walidacyjny (20%) i testowy (20%). Ponadto w module jest możliwość skalowania danych metodami: **min-max1, min-max2, standaryzacja.** Parametry skalowania, czyli średnia, odchylenie standardowe, minimum, maximum wyznaczam na podstawie zbioru treningowego, a następnie stosuję w procesie skalowania dla zbioru treningowego, testowego i walidacyjnego.

W następnej kolejności stworzyłem foldery dla rozważanych modeli:

- Metoda wektorów nośnych (SVM)
- AdaBoost
- Metoda k najbliższych sąsiadów (kNN)

W folderze SVM zaimplementowałem algorytm sekwencyjnej minimalnej optymalizacji (SMO) dla maszyny wektorów nośnych z zastosowaniem kilku różnych funkcji jądrowych.
Zgodnie ze specyfikacją modelu maksymalizowałem funkcję
$\sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i\, \alpha_i k(\mathbf{x_i}, \mathbf{x_j}) y_j \alpha_j$ pod warunkiem
$\sum_{i=1}^{m} \alpha_i\, y_i = 0$ oraz $0 \le \alpha_i \le \frac{C}{2m}$ dla każdego i
Dzięki tej maksymalizacji otrzymywałem funkcję decyzyjną postaci
f(z) = $sgn(\mathbf{w^T}\varphi(\mathbf{z}) - b) = sgn([\sum_{i=1}^{m} \alpha_i\, y_i k(\mathbf{x_i}, \mathbf{z})] - b)$ dla b = $[\sum_{j=1}^{m} \alpha_j\, y_j k(\mathbf{x_j}, \mathbf{x_i})] - y_i$

W folderze AdaBoost zaimplementowałem algorytm AdaBoost, który spośród wielu słabych klasyfikatorów - pni konstruuje silny klasyfikator postaci:
h(x) = $sgn\left(\sum_{i=1}^{T} \alpha_i\, h_i(x)\right)$

W folderze kNN zaimplementowałem algorytm k najbliższych sąsiadów, który klasyfikuje obserwacje testowe na podstawie liczniejszej grupy spośród k najbliższych obserwacji ze zbioru treningowe.

W rozważanych modelach błąd jest podany jako szacowane prawdopodobieństwo złej klasyfikacji, czyli $\frac{FN + FP}{TP+TN+FN+FP}$, a frakcje danych to `[0.005, 0.01, 0.02, 0.03, 0.075, 0.125, 0.25, 0.5, 0.625, 0.75, 0.875, 1]`

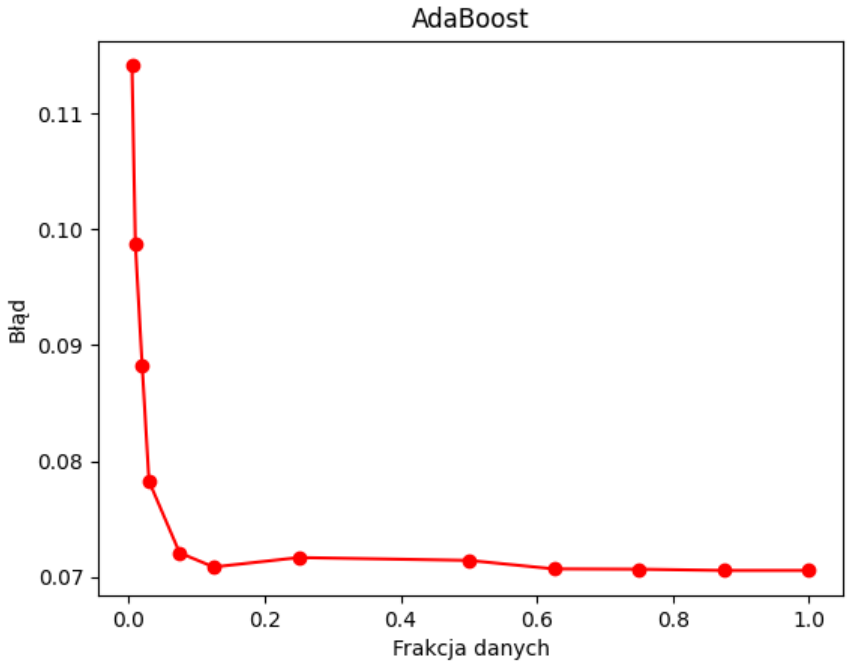
AdaBoost

Wyniki w powyższym modelu uśredniałem na 10 przebiegach algorytmu dla losowo wybranych danych do zbioru treningowego i testowego.

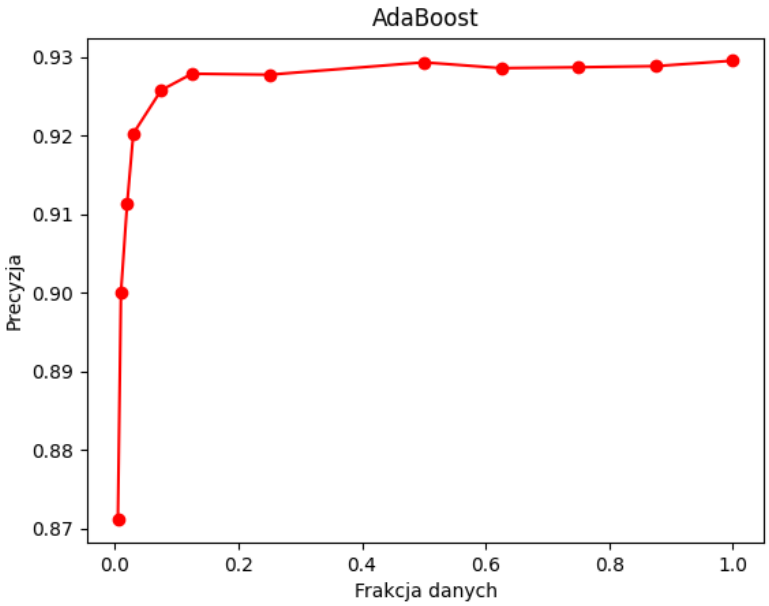
Na samym początku warto zauważyć, że najlepszy spośród słabych klasyfikatorów już w pierwszej iteracji miał średnią skuteczność predykcji w wysokości 89%. Tak wysoką skutecznością charakteryzował się najczęściej klasyfikator dla 26 cechy, ale zdarzało się również, że najlepszy był klasyfikator 22 cechy.

Liczba iteracji wykorzystywana w procesie uczenia wynosiła zwykle 100 – 150, a jej zwiększanie nie powodowało już większego zmniejszania błędu na zbiorze testowym - błąd pozostawał na stabilnym praktycznie stałym poziomie.

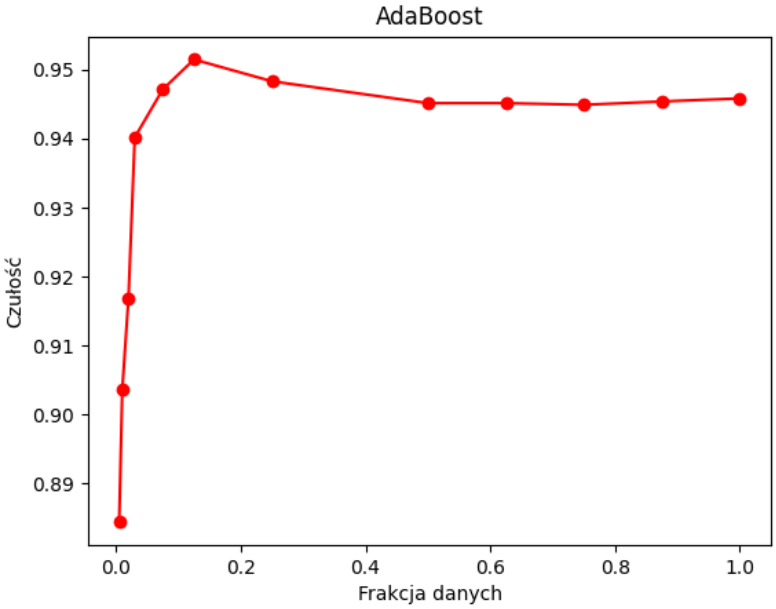
Z uwagi na już dosyć niski błąd słabego klasyfikatora, błąd dla frakcji danych 0.005 wynosił także średnio już niewiele - `0.114.` Największy spadek błędu na wykresie jest widoczny aż do frakcji 0.075 kiedy wynosił `0.072`, a następnie utrzymywał się na względnie stałym poziomie. Najmniejszy błąd otrzymałem dla frakcji 1.0 i wyniósł on `0.07056`



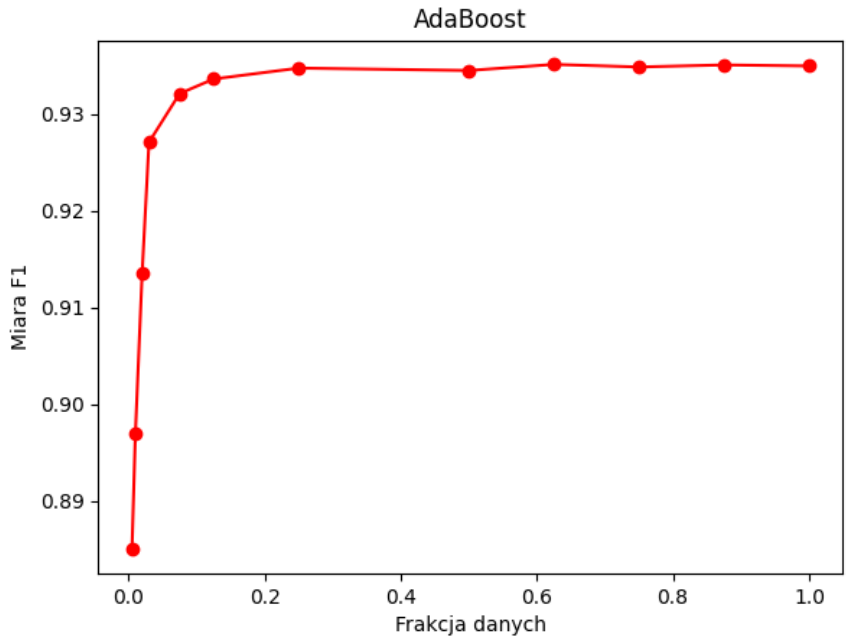
Precyzja dla małych frakcji zbioru treningowego wynosiła np. `0.8711` dla frakcji 0.005, a następnie rosła do `0.9279` dla frakcji 0.125, a następnie była praktycznie stała i osiągała maksymalnie `0.9295` dla 1.0



Natomiast czułość już dla frakcji 0.005 była lekko wyższa i równa `0.884`, a potem rosła aż do `0.9514` dla frakcji 0.125. Co ciekawe najczęściej osiągała tu swoje maksimum, a później lekko spada. Jednakże spadek jest praktycznie niezauważalny - jest mniejszy niż 0.005. Ostatecznie dla frakcji 1.0 czułośc jest równa `0.9459.`



Z uwagi na podobny kształt wykresu dla precyzji i czułości - wykresy dla miary F1 wygląda również w bardzo zbliżony sposób.



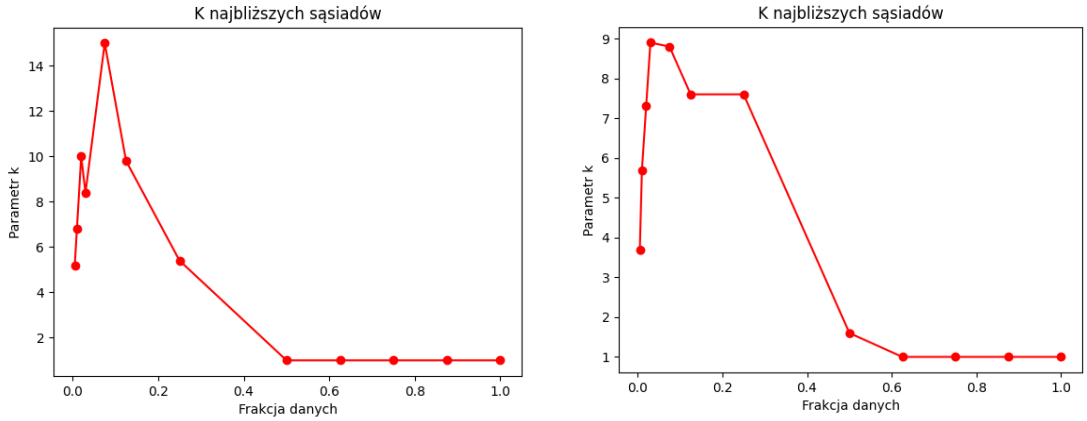
Biorąc pod uwagę niski błąd dla bardzo prostego klasyfikatora spodziewałem się ostatecznego wyniku klasyfikacji stron internetowych na nieco niższym poziomie – tzn. wynoszącym ok. 0.04 - 0.05 dla frakcji 1.0.

K najbliższych sąsiadów

Wyniki w tym modelu również uśredniałem na 10 przebiegach dla losowego podziału danych na zbiory. Jednakże tym razem oprócz zbioru treningowego i testowego wykorzystałem również zbiór walidacyjny. Biorąc pod uwagę fakt, że niektóre cechy miały 3 wartości a inne 2 przeskalowałem dane z użyciem standaryzacji.

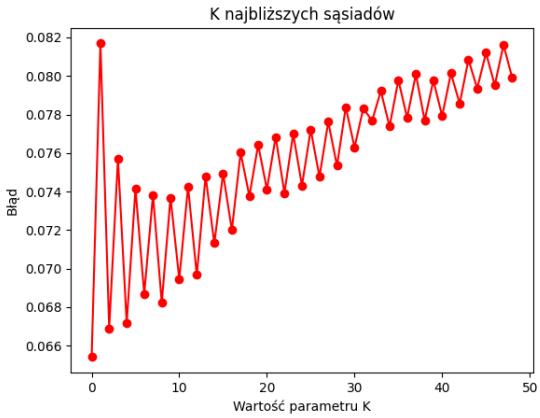
Badając ten model sprawdziłem jaki wpływ na wynik może mieć metryka jaką używamy i dlatego też moje wyniki będą podzielone na te z użyciem metryki l^2, a także na te z użyciem miary Hamminga.

Zbiór walidacyjny wykorzystywałem do wyznaczenia optymalnego metaparametru k w naszym algorytmie dla danego podziału na obserwacji na zbiory. Co ciekawe niezależnie od podziału obserwacji, a także wykorzystywanej metryki okazywało się, że optymalne k wynosi 1. Zaciekawiony tą kwestią sprawdziłem jaki byłby wynik dla różnych rozmiarów zbioru walidacyjnego. Okazało się wtedy, że dla bardzo małych frakcji zbioru walidacyjnego optymalny metaparametr k był większy niż 1 i wynosił ok. 5, potem nawet przez chwilę jeszcze rósł i przekraczał nawet 10, a następnie spadał do 1 i utrzymywał się na tym poziomie do końca.

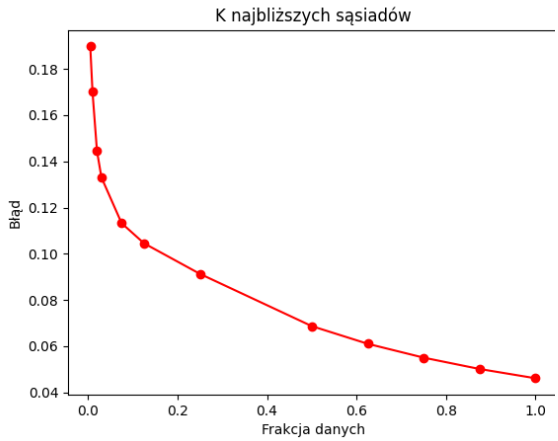


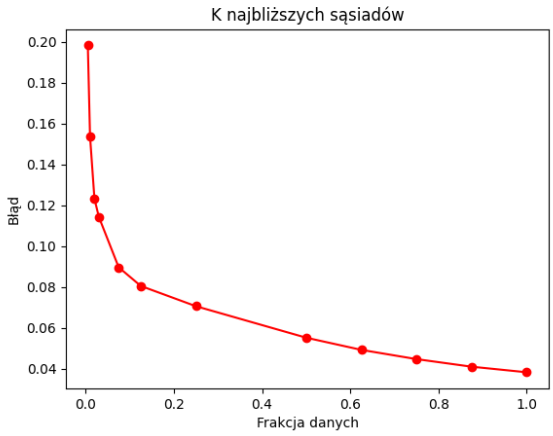
Po lewej wykres dla miary l^2, a po prawej dla Hamminga (będę stosował ten układ za każdym razem w tej sekcji)

Sprawdziłem również jak wygląda wykres błędu na całym zbiorze walidacyjnym w zależności od parametru k. Można na nim zauważyć stopniowy wzrost błędu, przy czym jest on skokowy - dla parzystych k błąd jest wyższy niż dla następujących po nich k nieparzystych. Wynika to pewnie z faktu, że w przypadku remisu 2 zbiorów w algorytmie dla k parzystych wybieram zawsze w klasyfikacji klasę 1. Warto zauważyć, że dla Hamminga wahania są większe.

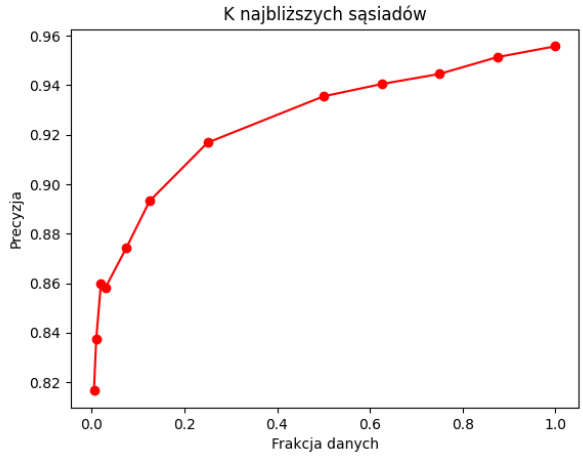


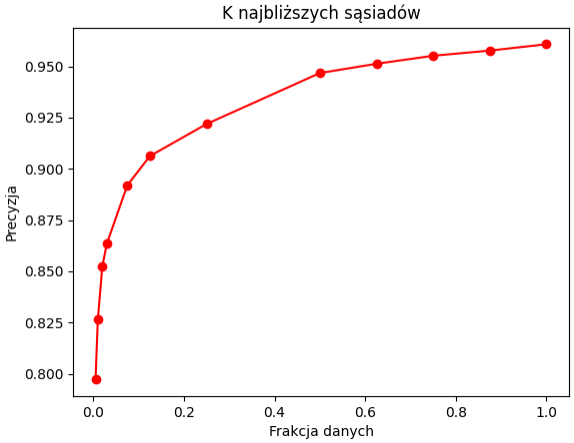
Jak już można zauważyć nawet na powyższych wykresach miara Hamminga daje trochę lepsze wyniki niż miara l^2. Spadek wykresu błędu w porównaniu do wykresu AdaBoost niezależnie od zastosowanej miary jest bardziej wypłaszczony - największy spadek ma cały czas miejsce dla przedziału frakcji danych [0 – 0.1], ale tym razem spadek na przedziale [0.1 - 1.0] jest cały czas zauważalny i wynosi co najmniej 0.04. Obie metryki dla frakcji 0.005 dają bardzo podobny wynik na poziomie `0.1906,` a potem już zaczyna widać wyższy spadek dla miary Hamminga - np. dla frakcji 0.075 dla miary l^2 to `0.113,` a dla Hamminga to `0.092.` Najlepszy wynik w obu przypadkach jest dla frakcji 1.0 i wynosi dla l^2 `0.0460,` a dla Hamminga `0.0380.`



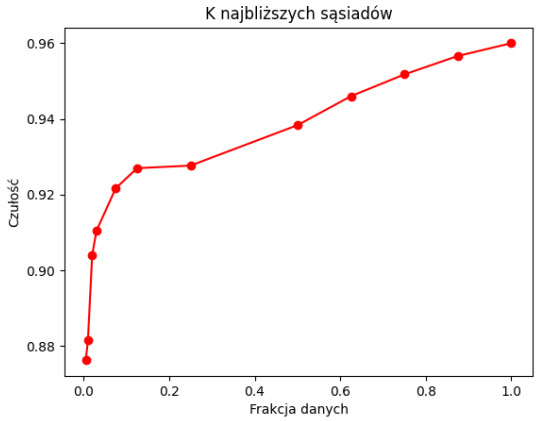


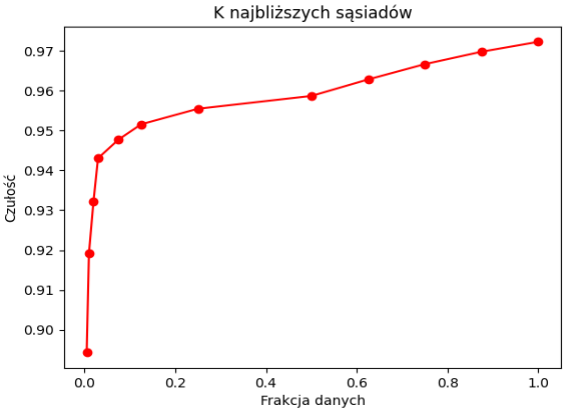
Wykresy precyzji dla obu metryk wyglądają praktycznie tak samo – dla frakcji 0.005 wynosi ok. `0.80,` rosną do wartości `0.906` dla frakcji 0.125, żeby osiągnąć maksimum dla frakcji 1.0 i wartości `0.9608.`



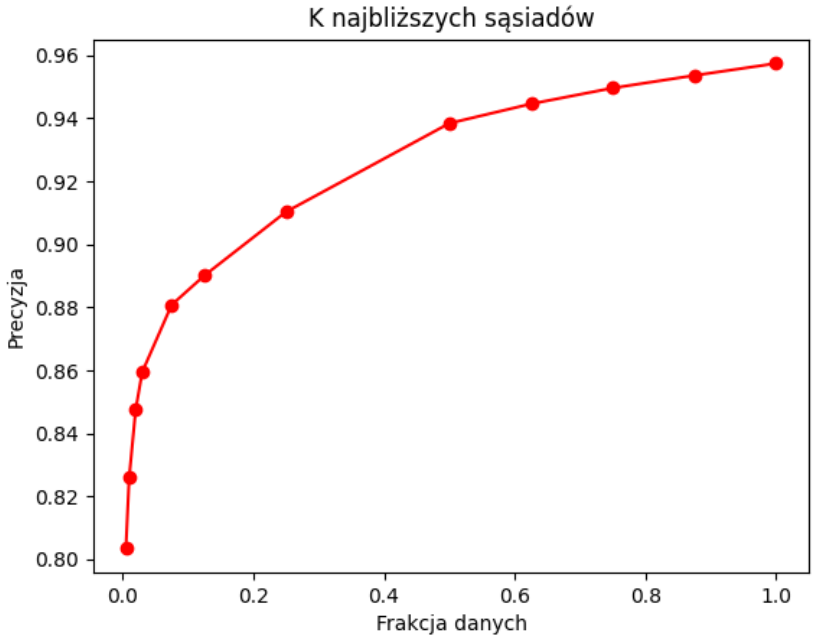


Na wykresie czułości widać już natomiast wyższe wyniki dla miary Hamminga, choć same wykresy są dosyć zbliżone. W zasadzie dla każdej frakcji danych miara Hamminga ma wyższą wartość czułości. Na samym początku dla frakcji 0.005 miara l^2 osiąga `0.876,` a Hamminga `0.891,` po przekroczeniu frakcji 0.075 w l^2 otrzymałem `0.921,` a w Hammingu `0.9488.` Najwyższa czułość została osiągnięta dla frakcji 1.0 zarówno dla l^2 - `0.9599` jak i Hamminga - `0.9722.` Wartość ta już jest naprawdę wysoka i świadczy o wysokim poziomie detekcji pozytywnych przypadków - jest niewiele przypadków FalseNegative, czyli złej klasyfikacji strony internetowej

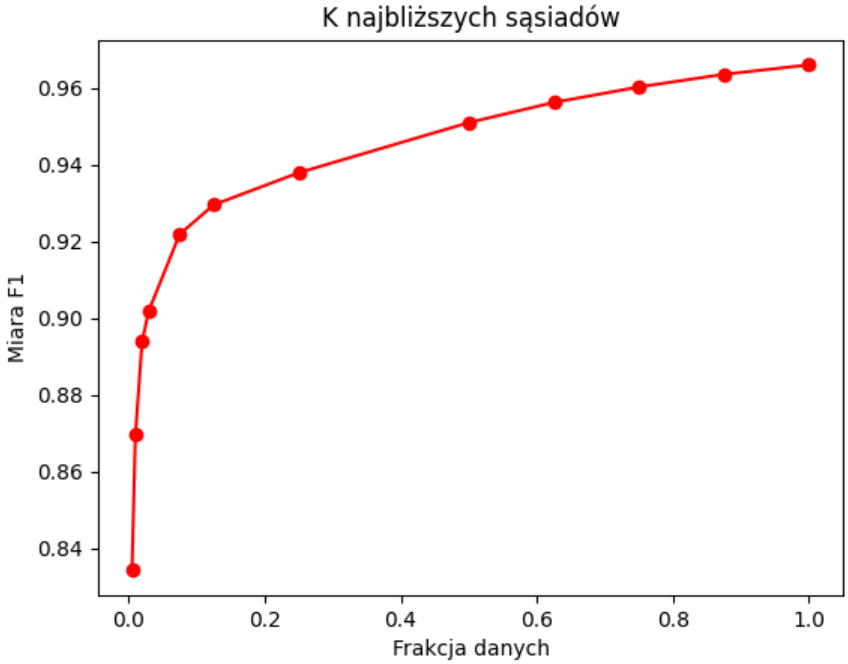




Wykresy miary F1 wyglądały za to w następujący sposób



Miara l^2



Miara Hamminga

Maszyna wektorów nośnych

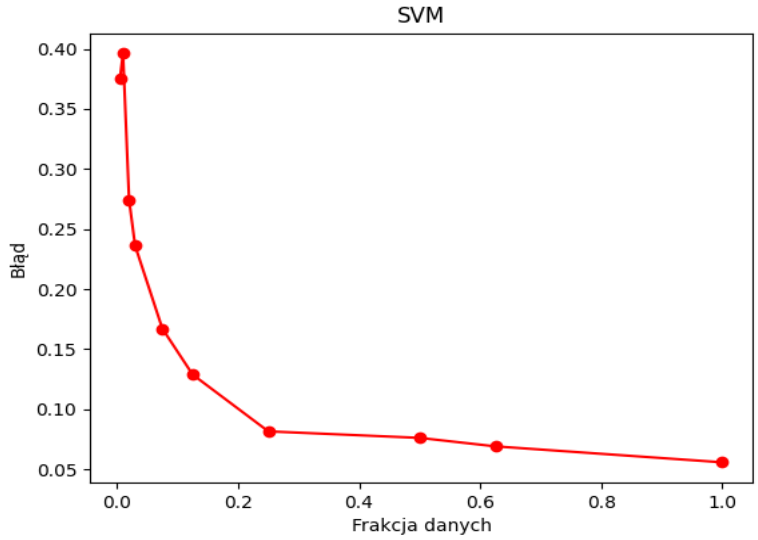
Dla tego modelu wyniki zostały uśrednione na 5 przebiegach algorytmu dla losowego podziału obserwacji na zbiór treningowy, testowy i walidacyjny.
W tym modelu badałem wpływ funkcji jądrowych na ostateczny wynik. Rozważyłem:

- Jądro gaussowskie – $\exp\left(-\frac{\|x-z\|^2}{2\cdot\sigma^2}\right)$
- Jądro liniowe (brak jądra) - $x^T z$

Do wyznaczania metaparametru jądra - σ, a także współczynnika C wykorzystałem zbiór walidacyjny.

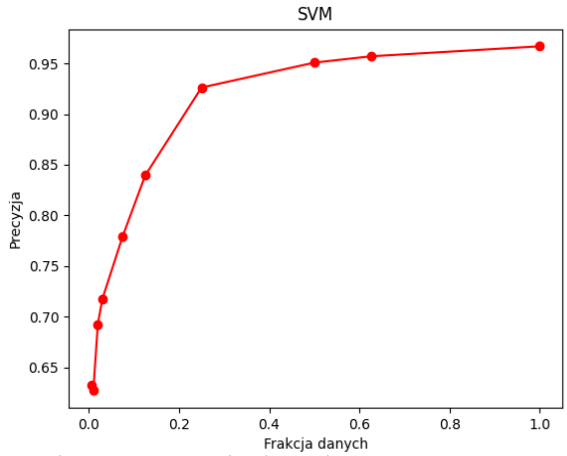
Jądro gaussowskie

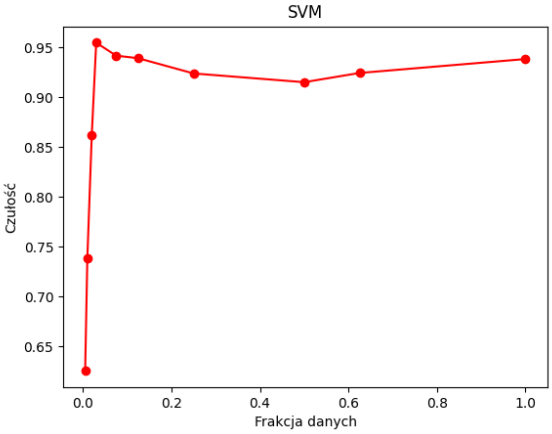
Spośród sprawdzanych wartości σ najmniejszy błąd ostatecznie osiągnąłem dla σ=1. Dla małych frakcji zbioru treningowego dostawałem dosyć duży błąd w porównaniu do poprzednich metod - przykładowo dla 0.005 dostałem błąd `0.37.` Jednakże wraz ze wzrostem frakcji danych i osiągnięciu 0.25 błąd spadł już poniżej 0.1 i wyniósł `0.081.` Ostatecznie dla frakcji 1.0 wyniósł on najmniej - 0.0557, co jest już na pewno dobrym wynikiem.



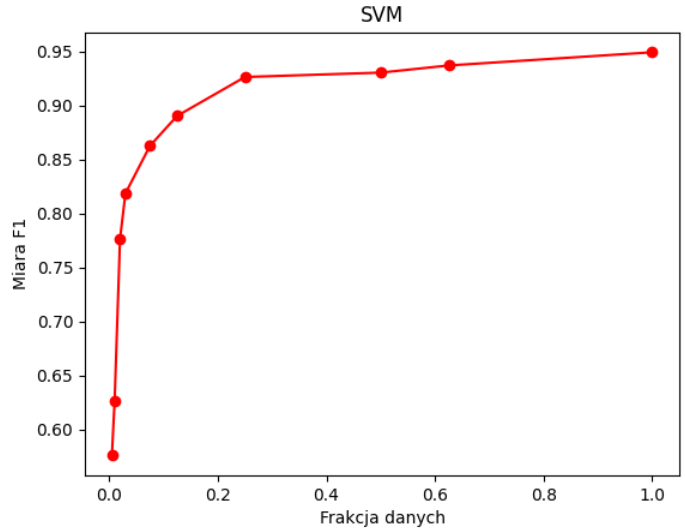
Kształt wykresu precyzji wygląda bardzo podobnie jak dla poprzednich modeli, jednak biorąc pod uwagę znacznie wyższy błąd niż wcześniej początkowa precyzja jest też niższa i wynosi dla 0.005 - `0.63,` a następnie po przekroczeniu frakcji 0.25 wynosi już `0.926.` A swoje maksimum osiąga w 1.0 dla `0.967.` Dla porównania czułość tym razem jest w swoim maksimum

nieco niższa, a także jej wykres jest odbiegający od wcześniejszych, ponieważ swoje maksimum osiąga dla frakcji 0.03 - `0.954`, a potem lekko spada aż do wartości `0.938` dla frakcji 1.0. Taki odstający wykres może być spowodowany mniejszą średnią wyników branych pod uwagę.



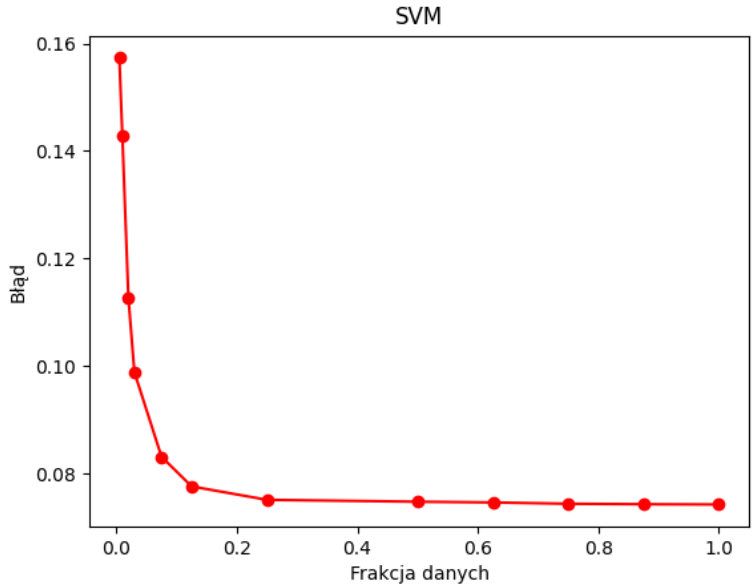


A miara F1wygląda tak:

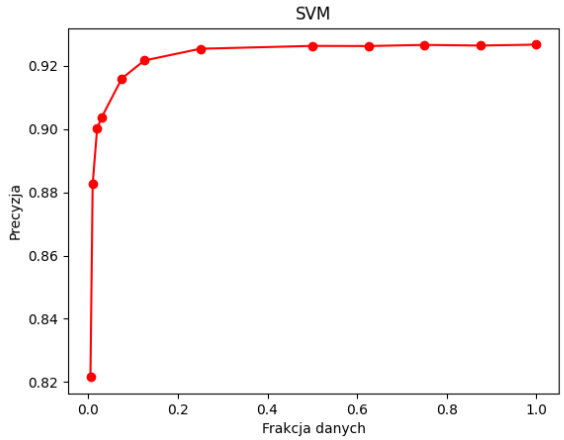


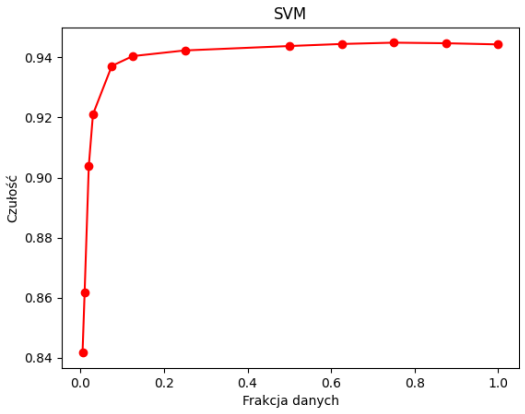
Jądro liniowe

Kształt wykresu i wartości błędu w tym przypadku wyszły mi zbliżone do tych otrzymanych w algorytmie AdaBoost. Dla frakcji 0.005 wynik wyniósł `0.157,` większość spadku miała miejsce do frakcji 0.075 gdzie wartość wynosiła `0.081,` a minimum było dla frakcji 1.0 i osiągnęło `0.0725.`

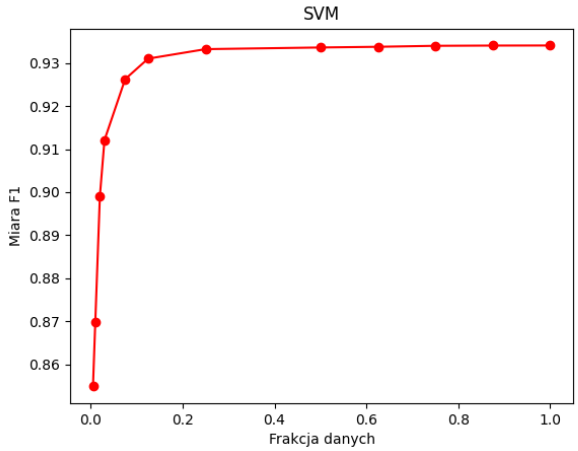


Wykresy precyzji i czułości są bardzo zbliżone do siebie, a natomiast ich wartości są również podobne do tych otrzymanych przy AdaBoost. Maksymalna precyzja wynosi `0.9267` i jest nieco gorsza od precyzji, która osiąga `0.9442` również dla frakcji 1.0.





Miara F1 wygląda tym razem w taki sposób:



Podsumowując, najniższy błąd uzyskałem dla modelu k najbliższych sąsiadów z wykorzystaniem miary Hamminga. Błąd wyniósł wtedy `0.0380.` Dla tego samego modelu miałem także najwyższą czułość. Warto zauważyć, że model k najbliższych sąsiadów z wykorzystaniem miary l^2 był także dobry. Niewiele odstawał od niego SVM z jądrem gaussowskim i możliwe, że przy jeszcze dokładniejszym badaniu metaparemetrów błąd mógłby być niższy. Spośród rozważanych modeli najwyższy błąd dostałem dla AdaBoost i SVM z jądrem liniowym, dla których wyniósł on ok. `0.07.`