Algorytm SVM

Mikołaj Szawerda 318731

Opis polecenia

Zadanie polega na zaimplementowaniu algorytmu SVM i zbadaniu działania na przykładzie zbioru danych Wine Quality Data Set. Zadaniem algorytmu jest określenie na podstawie podanych właściowości fizykochemicznych wina jakość wina zmapowaną do dwóch wartości - klasyfikacja binarna.
SVM polega na wyznaczeniu linii, która jak najlepiej separuje obie klasy - równoważnie wyznaczenie jak największego obszaru separującego wektory różnych klas. Ponieważ nie każdy zbiór danych jest liniowo separowalny do algorytmu należy zastosować dodatkowe kroki zwiększające jego skuteczność: - dodanie marginesu - linia nie musi separować od siebie dokładnie wszystkich reprezentatnów danych klas - zastosowanie "sztuczki jądrowej" - użycie funkcji jądrowej, która pozwala policzyć iloczyn skalarny dwóch wektorów w przestrzeni wyznaczanej przez funkcję jądrową - praktycznie, pozwala to nadać dowolny kształt linii separującej

Niech:

$\hat{f}\left(x\right) = w^T x + b$ - postać prostej, która najlepiej separuje dwie klasy

$$y_i = \begin{cases} -1 & \hat{f}\left(x_i\right) \leqslant 0 \\ 1 & \hat{f}\left(x_i\right) > 0 \end{cases}$$

funkcja decyzyjna, o przynależności do klasy

$k(u, v)$ - przekształcenie jądrowe

Zadaniem algorytmu jest wyznacznie w i b, a ponieważ algorytm ma mieć możliwość stosowania różnych funkcji jądrowych, program będzie realizował algorytm zapisany w postaci dualnej, gdzie:

$\hat{w} = \sum_i^N \alpha_i x_i y_i$

$\hat{b} = median(\{b_i : x_n \in X_n \land b_i = |y_n - k(\hat{w}, x_n)|\})$, gdzie X_n - zbiór wektorów wspierających.

Ponieważ nie zawsze wektory wspierające muszą znajdować się idealnie na marginesie liczona jest absolutna odległość od marginesu($y_n \in -1, 1$) i z powstałych wartości brana jest wartość środkowa

$a_{1,\dots,N} = arg\,min_\alpha(\frac{1}{2}\sum_i^N\sum_j^N y_i y_j \alpha_i \alpha_j k(x_i, x_j) - \sum_i^N \alpha_i)$

Przy ograniczeniach: $\begin{cases} \sum_i^N y_i \alpha_i = 0, \\ (\forall_i = 1, \dots, N)\ 0 \leqslant \alpha_i \leqslant C \end{cases}$

Hiperparametrami algorytmu są więc:

- C - współczynnik kosztu - mnożnik kary za użycie rozluźnienia
- $k(u, v)$ - przekształcenie jądrowe
- $parametry\ k(u, v)$

Optymalizacja obliczeń numerycznych

Ponieważ trenowanie SVM w postaci dualnej jest bardzo kosztowne obliczeniowo(problem zmienia wymiarowość na ilość danych trenujących), funkcję celu zapiszę w postaci macierzowej celem skorzystania z pakietu numpy

$a_{1,...,N} = arg\,min_{\alpha}\frac{1}{2}\alpha^{T}YKY\alpha - \sum\alpha_{i}$

Gdzie:

$K_{ij} = k(x_i, x_j)$

$Y = diag(y)$

W celu poprawy szybkości zbieżności solvera, wyprowadzam gradient optymalizowanej funkcji postaci:

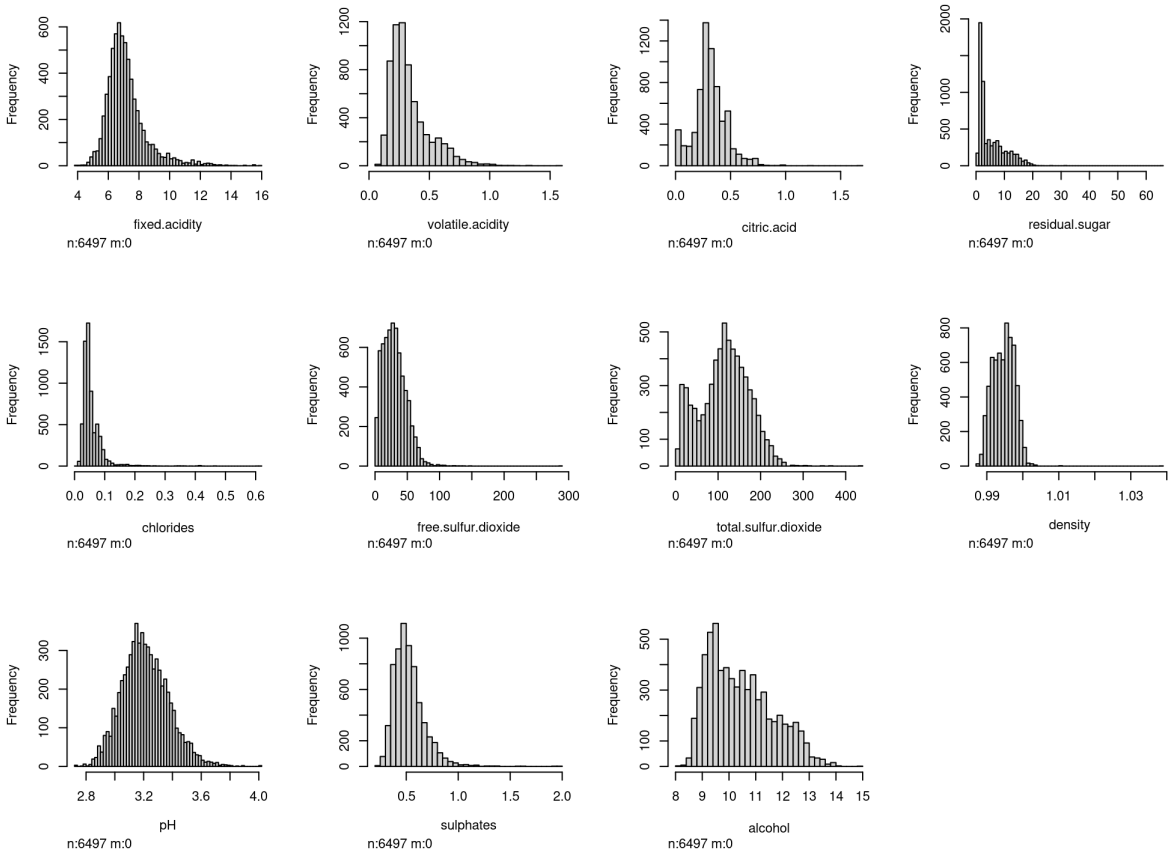
$\nabla\mathbb{D}(\alpha) = \langle K, \alpha\rangle - \varepsilon$

Planowane eksperymenty numeryczne

- podczas uczenia zostanie zastosowane 5-krotna walidacja krzyżowa na zbiorze danych N=500, z mapowaniem $q(q_{old}) = sgn(q_{old} - 5)$, gdzie q to nowa wartość jakości, a dane win czerwonych i białych zostały połączone w jeden dataset i ułożone losowo
- W celu oceny konieczności zastosowania normalizacji/standaryzacji(celem zwiększenia skuteczności i szybkości zbieżność solvera) przedstawię histogramy cech
- Do oceny wpływu hiperparametrów przedstawię macierz pomyłek, oraz wykresy prezentujące TPR i FPR
- algorytm wywołam dla dwóch funkcji jądrowych:
    - $k(u, v) = u^{T}v$
    - $k(u, v) = exp(\frac{-|u-x|^2}{2\sigma^2})$

Wyniki

Charakterystyki liczbowe datasetu



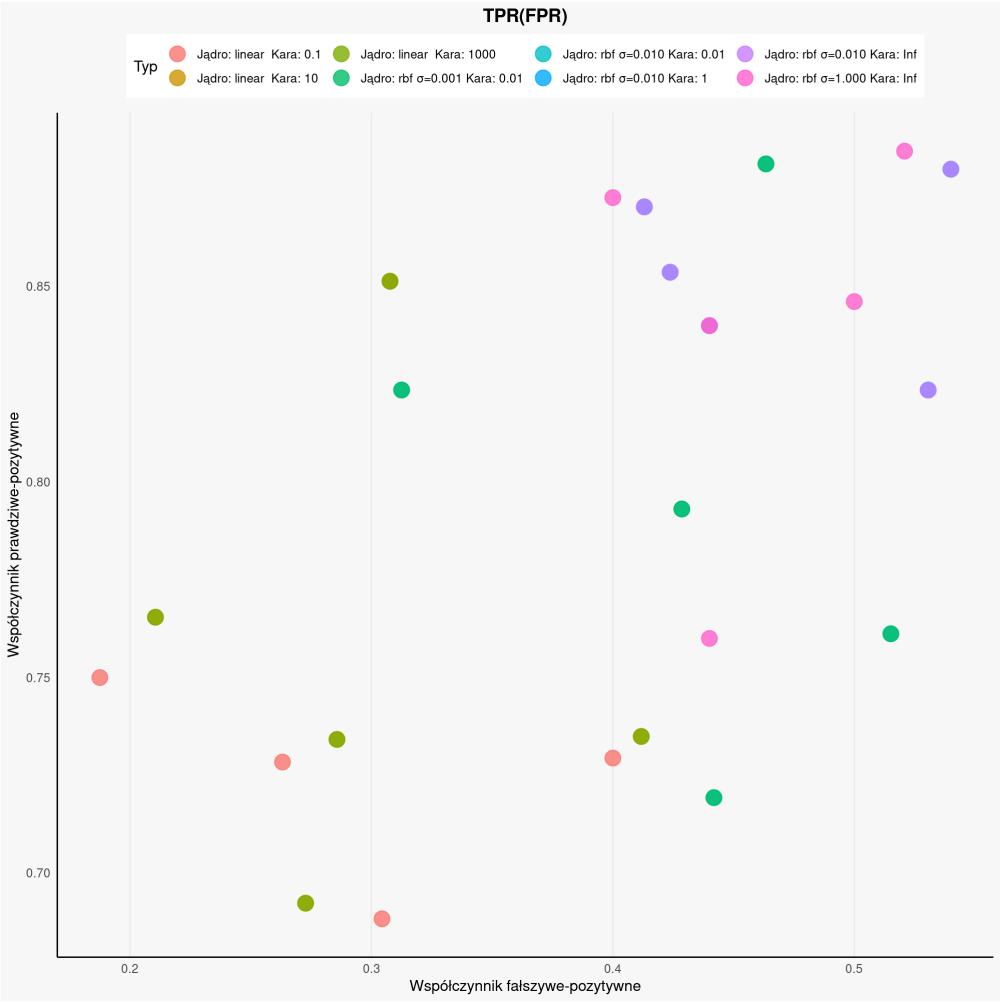
Charakterystyka cech

name	Min.	Mean	Max.
fixed.acidity	3.80000	7.21530706	15.90000
volatile.acidity	0.08000	0.33966600	1.58000
citric.acid	0.00000	0.31863322	1.66000
residual.sugar	0.60000	5.44323534	65.80000
chlorides	0.00900	0.05603386	0.61100
free.sulfur.dioxide	1.00000	30.52531938	289.00000
total.sulfur.dioxide	6.00000	115.74457442	440.00000
density	0.98711	0.99469663	1.03898
pH	2.72000	3.21850085	4.01000
sulphates	0.22000	0.53126828	2.00000
alcohol	8.00000	10.49180083	14.90000
quality	-1.00000	0.26612283	1.00000

Na podstawie histogramów oraz min/max wartości cech, można stwierdzić iż konieczna będzie standaryzacja wartości - dane mają różne skale, a wszystkie cechy mają rozkład normalny.

Na podstawie tabeli można również stwierdzić, iż rekordów o $y = 1$ jest więcej.

Wykres zależności TPR(FPR) dla różnych parametrów



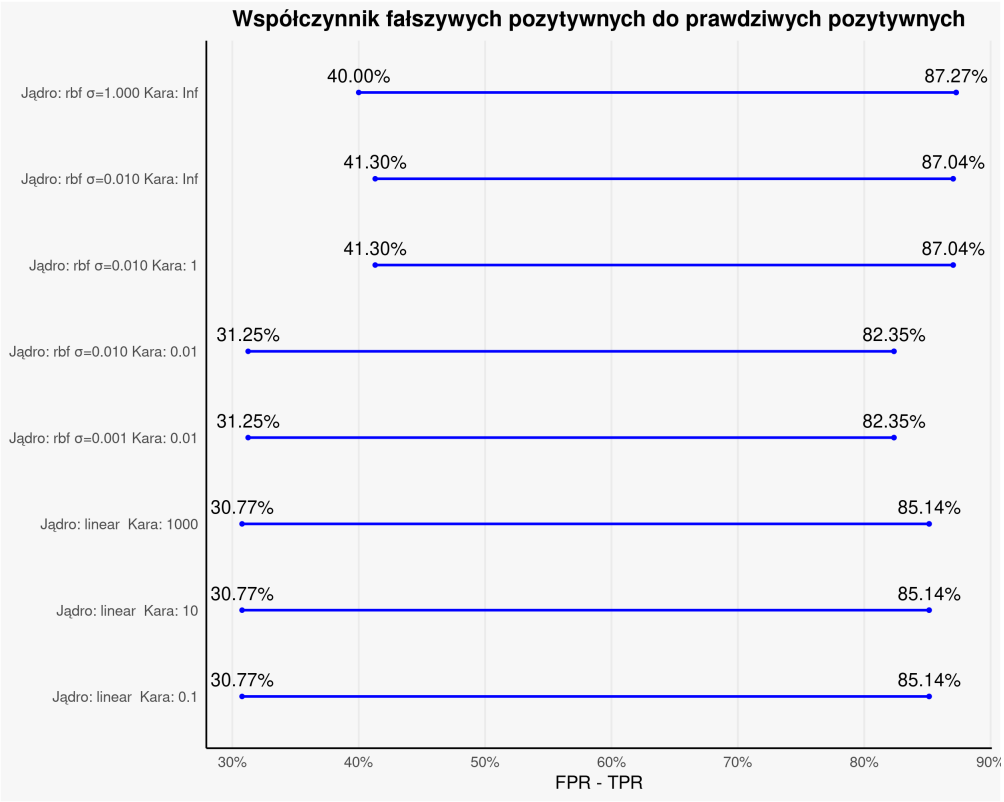
Wykres przedstawia jaki jest stosunek przykładów zaklasyfikowanych poprawnie do błędnie zakfalifikowanych dla klasy o $y = 1$. Można zauważyć trend, gdzie wraz z wzrostem FPR rośnie TPR - ze względu na przewagę liczbną klasy 1 algorytm częściej jako odpowiedź zwraca tą klasę.

Na przykładzie wykresu widać również zależność wyników dla jądra rbf od sigmy i kary - wraz z wzrostem tych parametrów maleje lokalność algorytmu, program globalnie stwierdza, że ta klasa się częściej pojawia - rośnie więc TPR, ale także FPR.

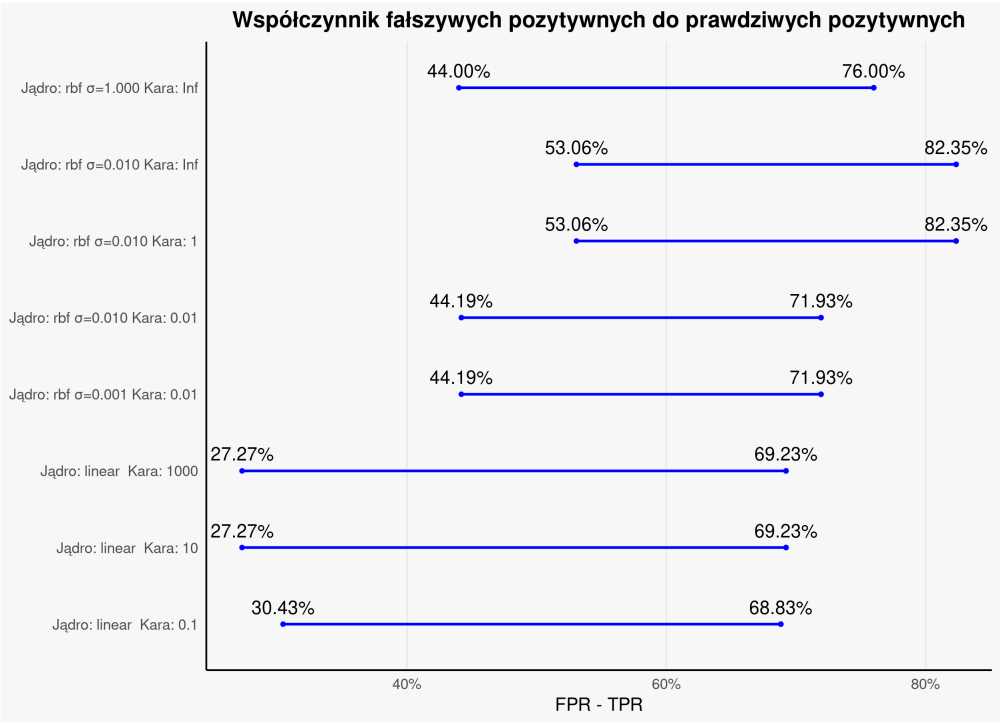
Według tego wykresu, najlepszymi modelami są algorytmy, których punkty są położene jak najbliżej lewego-górnego rogu. Według tej klasyfikacji najlepszym algorytmem jest algorytm z jądrem liniowym i karą 1000 lub 0.1 oraz model rbf z sigmą 1e-3 i karą 1e-2.

Przyjmując za próg akceptacji $FPR < 0.5$ około 75% uruchomień było prawidowych.

Porównanie najlepszych i najgorszych uruchomień modeli danego typu

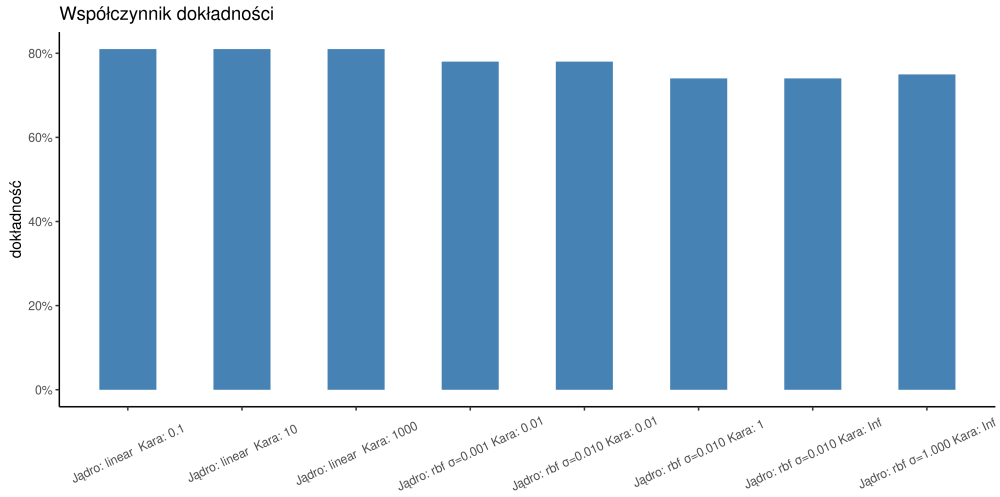


Dla najlepszych uruchomień(wykres jest maksymalnie rozciągnięty - małe FPR, duże TPR) algorytmy tych samych kerneli osiągnęły praktycznie takie same wyniki - każdy model znalazł to samo optimum. Algorytmy z liniowym jądrem okazały się lepsze.

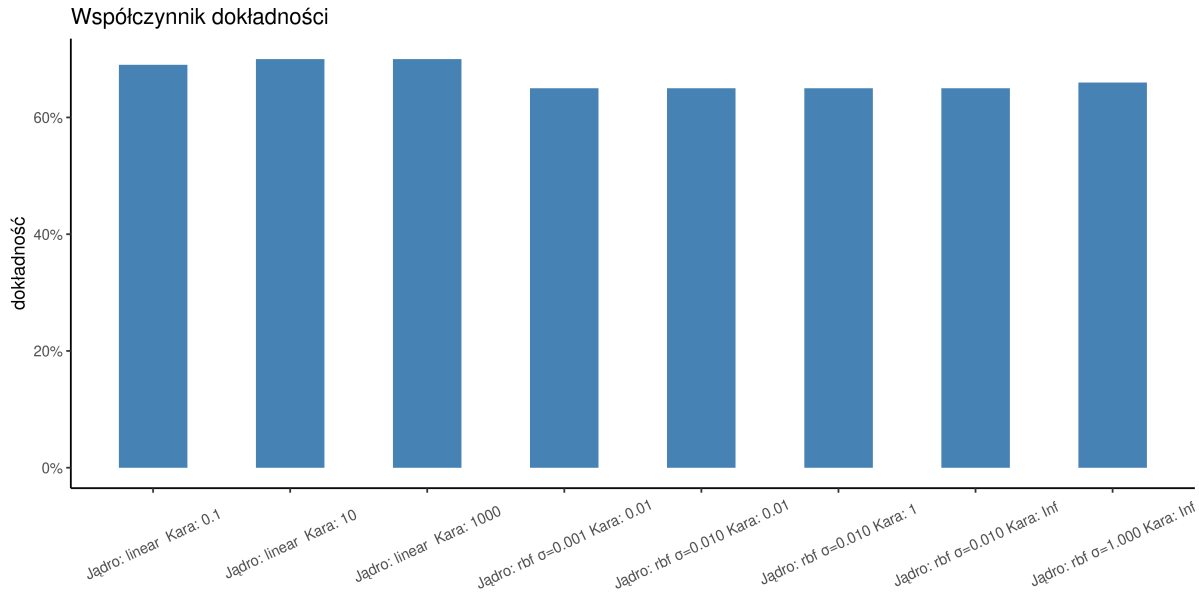


W przypadku najgorszych uruchomień modele z liniowym jądrem osiągnęły mniejszą ilość niepoprawnych klasyfikacji, lecz gorszą ilość poprawnych. Ponownie można zauważyć iż jądro rbf częściej wskazuje klasę 1 jako bardziej prawdopodobną

Najlepsze uruchomienia



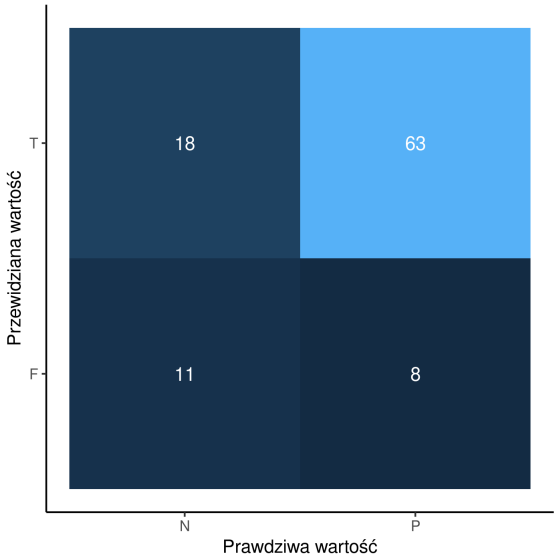
Najgorsze uruchomienia



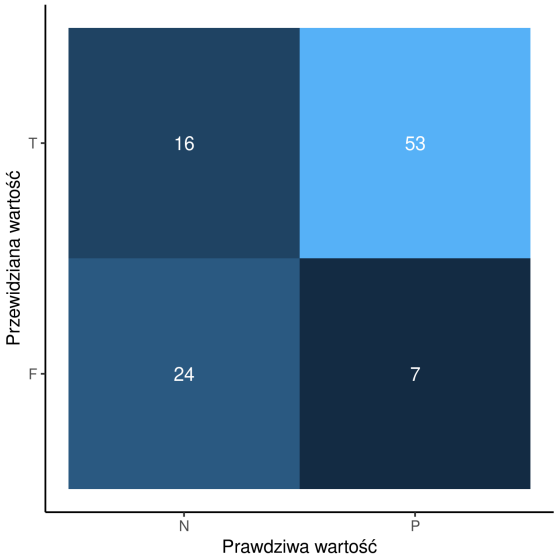
Dokładność wskazuje w ilu przypadkach, bez względu na wartość klasy, model wskazał poprawną odpowiedź. SVM z liniowym jądrem zarówna dla najlepszego jak i najgorszego uruchomienia uzyskał najlepsze wyniki, jednakże można zauważyć iż nie są one znacząco odmienne.

Macierze błędów

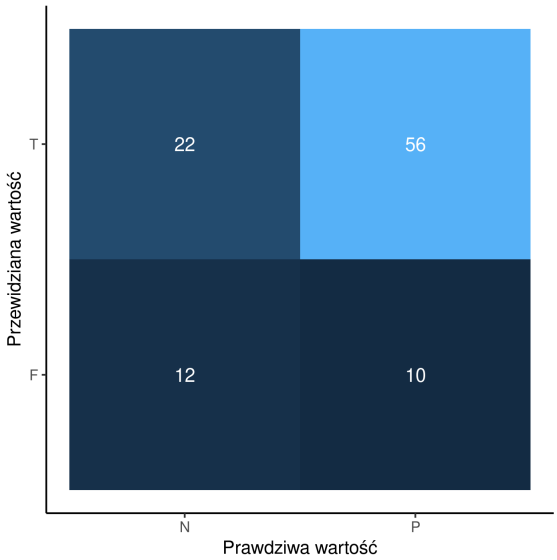
Jądro: linear Kara: 0.1 Dokładność: 81%



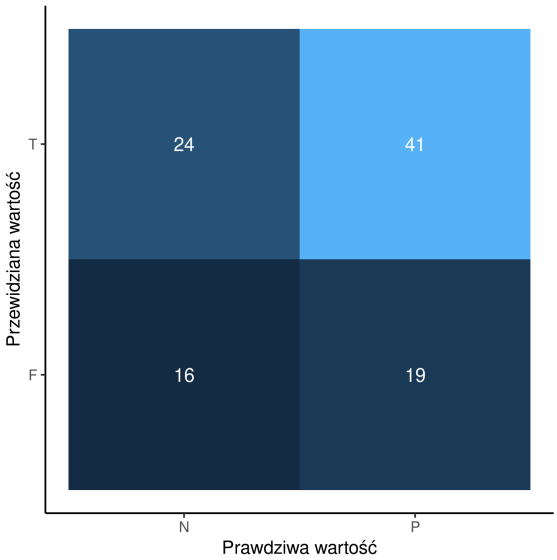
Jądro: linear Kara: 0.1 Dokładność: 69%



Jądro: rbf σ=0.001 Kara: 0.01 Dokładność: 78%



Jądro: rbf σ=0.001 Kara: 0.01 Dokładność: 65%



Podsumowanie wyników						
Typ	TPR	TPR σ	FPR	FPR σ	Czas wykonania[s]	Dokładność
Jądro: linear Kara: 0.1	0.75	0.06	0.29	0.08	41.84	74%
Jądro: linear Kara: 10	0.76	0.06	0.30	0.07	147.25	74%
Jądro: linear Kara: 1000	0.76	0.06	0.30	0.07	707.40	74%
Jądro: rbf σ=0.001 Kara: 0.01	0.80	0.06	0.43	0.07	8.39	71%
Jądro: rbf σ=0.010 Kara: 0.01	0.80	0.06	0.43	0.07	8.33	71%
Jądro: rbf σ=1.000 Kara: Inf	0.84	0.05	0.46	0.05	59.19	70%
Jądro: rbf σ=0.010 Kara: 1	0.85	0.02	0.47	0.06	8.39	69%
Jądro: rbf σ=0.010 Kara: Inf	0.85	0.02	0.47	0.06	8.72	69%

Według tabeli agregującej średnie osiągi każdego z modeli algorytmy z liniowym jądrem osiągały najlepszą dokładność, jednakże rzadziej poprawnie klasyfikowały. Można również zauważyć praktyczny brak wpływy parametru kary na algorytm liniowy, po za czasem wykonania, który pomiędzy $C = 0.1$ a $C = 1000$ jest aż 18 krotnie większy. Widać również, iż dla jądra rbf algorytm szybciej optymalizował funkcję celu - przez wcześniejsze policzenie macierzy jądra został wykluczony element ciężkości obliczeń iloczynu skalarnego w innych przestrzeniach. SVM z RBF był również szybko optymalizowalny, ponieważ funkcja ma charakter bardziej lokalny, przez co solver może szybko natrafić na odpowiednie miejsce do eksploatacji.

Wnioski

Dzięki wielu uruchomieniom tych samych modeli na różnych podzbiorach danych uczących i cross walidacji, można stwierdzić, iż algorytm SVM pozwala na dobre dostosowanie do danych, dzięki możliwości zmiany jądra - pozwala klasyfikować możliwe nieliniowe zależności. Jednakże algorytm w postaci dualnej jest bardzo kosztowny obliczeniowo - każda ewaluacja funkcji celu to wymnożenie macierzy o rozmiarach zbioru uczącego przez kombinację mnożników Lagrange'a. Zauważając, lepsze osiągi algorytmu z jądrem liniowym, oraz praktyczny brak wpływu hiperparametru kary, można stwierdzić, iż zbiór był bliższy separacji liniowej. W przypadku algorytmy z jądrem RBF można zauważyć wpływ hiperparametrów na osiągi oraz czas działania. Mniejsze wartości sigmy i kary zwiększają lokalność rozwiązań, a przez to optymalizator szybciej znajduje optimium lokalne, jednakże może to prowadzić do niesłusznego faworyzowania jednej z klas, zwiększając ilośc niepoprawnie przewidzianych klas z zbioru testowego.