Antoni Kowalczuk
Jan Jędrzejewski

Las losowy w połączeniu z SVM w problemie klasyfikacji

Dokumentacja końcowa

Temat projektu

Połączenie lasu losowego z SVM w zadaniu klasyfikacji. Postępujemy tak jak przy tworzeniu lasu losowego, tylko co drugi klasyfikator w lesie to SVM. Jeden z klasyfikatorów (SVM lub drzewo ID3) może pochodzić z istniejącej implementacji.

Interpretacja treści zadania

Należy zaimplementować model lasu losowego korzystający z dwóch rodzajów klasyfikatorów: SVM i ID3. Dla tego lasu należy zbadać wrażliwość na parametry takie jak liczba klasyfikatorów, stosunek liczby klasyfikatorów SVM do liczby klasyfikatorów typu ID3, liczba atrybutów dla klasyfikatora. Skuteczność algorytmu należy przetestować w przykładowych zadaniach klasyfikacji, znaleźć słabe i mocne strony tej metody.

Opis algorytmów

Las losowy

Pojęcia

Klasyfikator bazowy

Pojedyncza instancja klasy klasyfikatora typu ID3 lub SVM

Metoda złączeniowa

Polega na wytrenowaniu na tym samym zbiorze danych (dzieląc go na podzbiory) pewnej ilości klasyfikatorów bazowych. Na etapie predykcji każdy z klasyfikatorów bazowych dokonuje predykcji na danych wejściowych, a ostateczny wynik powstaje poprzez agregację wyników klasyfikatorów bazowych. Agregacja może być dokonana poprzez policzenie średniej, wzięcie najpopularniejszego wyniku lub w inny sposób. Na potrzeby problemu klasyfikacji użyta zostanie metoda wyboru najpopularniejszego wyniku.

Algorytm

Las

Las losowy to algorytm używający **metody złączeniowej**, który jako klasyfikatorów bazowych używa drzew decyzyjnych (np. **ID3**). W naszej implementacji będziemy używać również klasyfikatorów **SVM**, dzieląc zbiór klasyfikatorów bazowych lasu po połowie (jeśli to możliwe).

Etap inicjalizacji modelu

Musimy zdefiniować parę hiperparametrów by powstała instancja lasu, są to:

1. Ilość klasyfikatorów w lesie – parametr **B**
2. Hiperparametry klasyfikatorów “bazowych” (np. typ jądra w modelu SVM)
3. Ilość próbek w zbiorze treningowym każdego z klasyfikatorów – parametr **n**

W efekcie tworzony jest zbiór klasyfikatorów bazowych zainicjalizowanych hiperparametrami podanymi do konstruktora

Etap treningu

Etap ten przebiega w sposób opisany w sekcji **Metoda złączeniowa**.

1. Dla każdego klasyfikatora bazowego losujemy bez powtórzeń **n** próbek ze zbioru treningowego (globalnego dla całego lasu), tworzymy w ten sposób zbiór treningowy lokalny dla każdego klasyfikatora
2. Każdy klasyfikator bazowy trenujemy na nowo utworzonym lokalnym zbiorze treningowym zgodnie z metodą treningu specyficzną dla konkretnego typu klasyfikatora

Etap inferencji

Etap ten przebiega w sposób opisany w sekcji **Metoda złączeniowa**. Dla każdej pojedynczej próbki danych na wejściu dokonują się następujące kroki:

1. Każdy z klasyfikator bazowy dokonuje predykcji na próbce
2. Predykcje wszystkich klasyfikatorów są zbierane do zbioru predykcji
3. Na podstawie zbioru predykcji wszystkich klasyfikatorów bazowych dokonujemy agregacji wyników i wybieramy jedną klasę, która jest ostateczną predykcją całego lasu. W przypadku tego algorytmu wybór następuje zgodnie ze strategią “najczęstszy=najlepszy”

ID3

Pojęcia

Entropia

Jest to miara średniej ilości informacji per próbka danych z jakiegoś konkretnego zbioru. Im większa jej wartość tym więcej informacji możemy średnio otrzymać z każdej z próbek. Wzór prezentuje się następująco (źródło):

$$\mathrm{H}(X) := -\sum_{x \in \mathcal{X}} p(x) \log p(x) = \mathbb{E}[-\log p(X)],$$

Gdzie **X** – cały zbiór danych, **p(x)** – ilość wystąpień wartości **x** (gdzie **x** – unikalna wartość) w zbiorze **X** podzielona przez rozmiar **X**, **log** – logarytm naturalny.

Dla zbioru podzielonego na podzbiory za pomocą atrybutu

Zbiór danych może mieć wiele atrybutów. Jeśli dzielimy go na podstawie wartości jednego z nich otrzymujemy rozłączne podzbiory. Entropia takiego zbioru to

$$\mathbf{InfH(X, d)} = \sum_{j} \frac{|X_j|}{|X|} \mathbf{H(X_j)}$$

Gdzie ***d*** – atrybut, na podstawie którego dzielimy zbiór danych, X_j – zbiór powstały po podzieleniu atrybutem ***d***, dla **j-tej** wartości atrybutu, **|*X*|** – rozmiar **X**.

Przykłady

Dla **X** = {1, 1, 1, 2, 2, 3, 3, 5} entropia wynosi ⅜ * log(⅜) + ¼ * log(¼) + ¼ * log(¼) + ⅛ * log(⅛), co daje nam około -1.32 jako wynik.
Dla **X** = {1, 2} entropia to log(½), czyli około -0.7
Dla **X** = {1} entropia wynosi 0

Zysk informacyjny

Jest to miara zysku informacyjnego, który uzyskujemy po podzieleniu **X** na podzbiory atrybutem ***d***. Wzór:

$$\mathbf{InfGain(X, d)} = \mathbf{H(X)} - \mathbf{InfH(X, d)}$$

Algorytm

Etap tworzenia drzewa

Wejście: **Y, D, X** – zbiór klas, atrybutów wejściowych, par uczących
ID3(Y, D, X):

1. Jeśli dla każdej próbki $\{x_i, y_i\}$ z **X** prawdą jest, że y_i = **y** (gdzie **y** – jedna z klas z **Y**) → zwróć liść zawierający klasę **y**
2. Jeśli rozmiar **D** jest równy 0 → zwróć liść, który zawiera **y**, gdzie **y** = najczęstsza klasa ze zbioru próbek $\{x_i, y_i\}$ z **X**
3. Niech **d = argmax(InfGain(X, d))**, gdzie **d** – atrybut z **D**
4. Tworzymy *j* zbiorów par uczących z **X**, gdzie dla każdego zbioru prawdziwe jest, że wartość atrybutu **d** dla każdej próbki x_i jest równa *j-tej* wartości tego atrybutu. Oznaczamy te zbiory przez $\mathbf{X_j}$
5. Zwracamy drzewo, które ma korzeń **d**, którego krawędzie prowadzą do drzew powstałych poprzez wywołanie **ID3(Y, D-{d}, $\mathbf{X_j}$)** dla każdej wartości atrybutu *j*.

Algorytm został zaczerpnięty z prezentacji dr inż. Pawła Zawistowskiego na przedmiot WSI w realizacji 2021Z.
W efekcie tworzy się drzewo decyzyjne, w którym:

1. Każdy węzeł ma przypisaną nazwę/identyfikator atrybutu
2. Każdy węzeł ma przypisaną jedną z wartości atrybutu, który jest przypisany jego rodzicowi
3. Każde dziecko węzła ma przypisaną inną wartość atrybutu przypisanego do rodzica
4. Liście drzewa mają przypisaną klasę, która to jest predykcją drzewa dla każdej z zaobserwowanych kombinacji atrybutów użytych do stworzenia liścia

Etap ten jest de facto etapem treningu

Etap inferencji

Dla każdej próbki danych ustalamy ścieżkę w drzewie prowadzącą do liścia na podstawie atrybutów próbki i zwracamy klasę przechowywaną w liściu na końcu ścieżki. Zaczynając od korzenia, który ma przypisany do siebie atrybut, sprawdzamy jego wartość dla próbki danych. Na podstawie tej wartości wybieramy dziecko korzenia, do którego jest przypisana wartość atrybutu zgodna z wartością próbki. Powtarzamy całą procedurę do momentu osiągnięcia liścia, z którego zwracamy wartość klasy, która jest naszą predykcją.

Przykład tworzenia drzewa

Niech **Y** = {0, 1}, **D** = {‘a’, ‘b’}
X jest zadany tabelą poniżej

	a	b	y
0	x	8	1
1	d	5	1
2	x	5	0
3	x	8	0
4	f	8	1
5	x	5	0
6	x	8	0
7	x	8	0
8	f	8	0
9	d	5	1

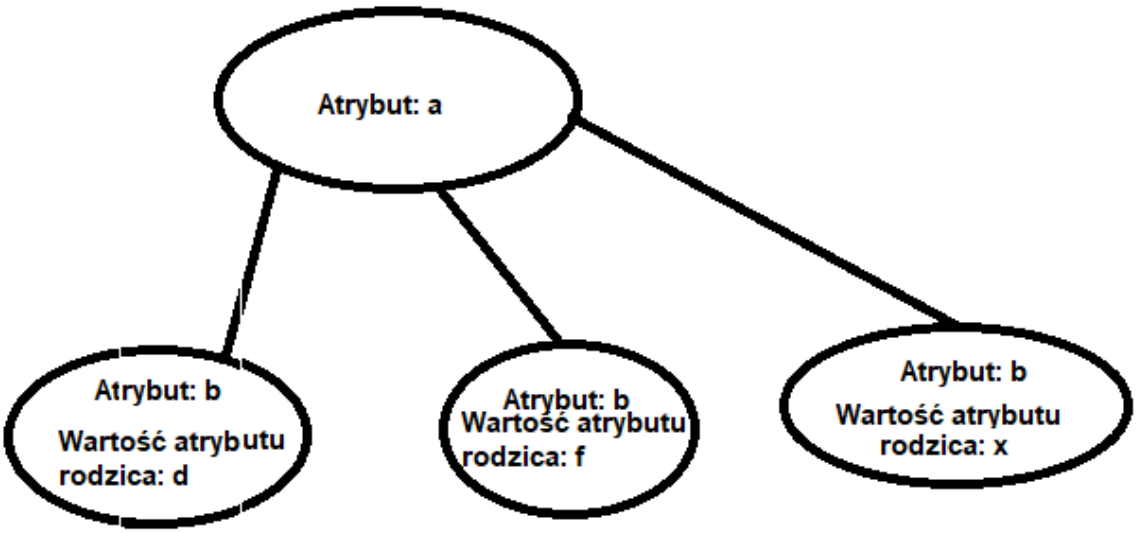
1. Wybieramy atrybut o największym **InfGain**. Jest to **a**

```
[46]  1  infgain(X, 'a'), infgain(X, 'b')
      ✓ 0.4s
...   (0.26404550557748474, 0.013844293808390695)
```

Mamy atrybut korzenia, teraz dzielimy zbiór **X** na podzbiory względem wartości atrybutu **a**

```
[47]  1  np.unique(X.a)
      ✓ 0.3s
...   array(['d', 'f', 'x'], dtype=object)
```

2. Pozostał nam tylko jeden atrybut – **b** w każdym z dzieci korzenia.
Dzielimy teraz zbiory **X** dla każdego z nich na podstawie wartości atrybutu w tych zbiorach



Drzewo po kroku 2.

3. Sprawdzamy, czy dla któregoś węzła możemy zwrócić liść z klasą

```
[55]  1  np.unique(X.loc[X.a=='d', 'y']), np.unique(X.loc[X.a=='f', 'y']), np.unique(X.loc[X.a=='x', 'y'])
      ✓ 0.2s

...   (array([1]), array([0, 1]), array([0, 1]))
```

Tak, możemy dla liścia z wartością **d**

4. Dla pozostałych tworzymy nowe węzły

```
[56]  1  np.unique(X.loc[X.a=='f', 'b']), np.unique(X.loc[X.a=='x', 'b'])
      ✓ 0.3s

...   (array([8]), array([5, 8]))
```

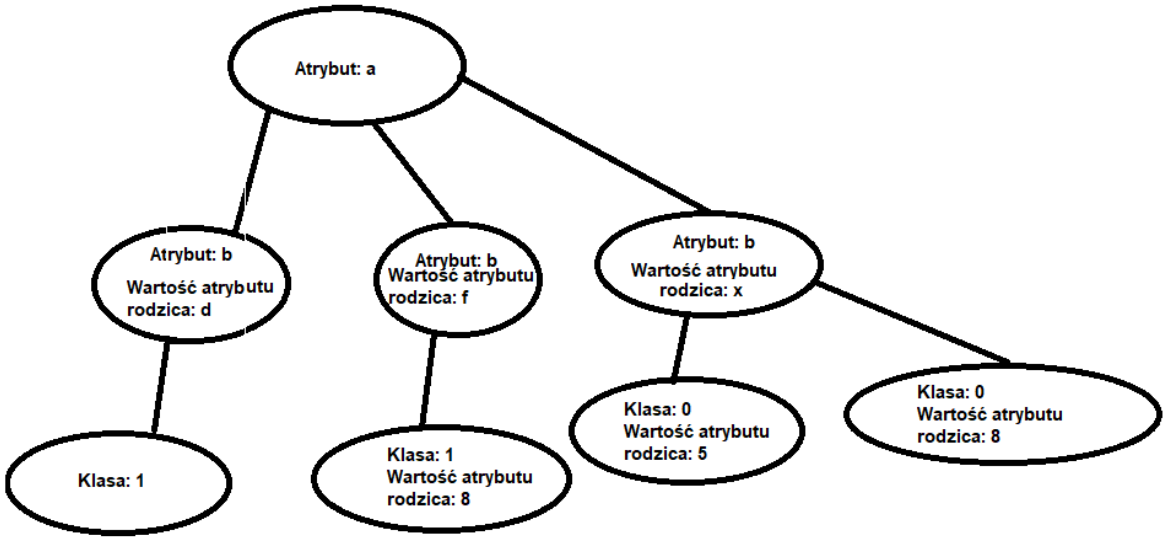
5. Powtarzamy punkt 3.

```
      1  np.unique(X.loc[(X.a=='f')&(X.b==8), 'y']), np.unique(X.loc[(X.a=='x')&(X.b==5), 'y']), np.unique(X.loc[(X.a=='x')&(X.b==8), 'y'])
      ✓ 0.3s

(array([0, 1]), array([0]), array([0, 1]))
```

Dla wartości **b** = 5 w węźle odpowiadającym wartości **x** atrybutu **a** możemy od razu zwrócić klasę 0

6. Nie mamy więcej atrybutów, więc dla pozostałych węzłów sprawdzamy najczęściej występujące wartości klasy **y**. Dla pierwszego węzła wybieramy losowo, bo obie wartości występują równą ilość razy (array([1, 1]) odpowiada liczności każdej z wartości)

```
[62]  1  np.unique(X.loc[(X.a=='f')&(X.b==8), 'y'], return_counts=True), np.unique(X.loc[(X.a=='x')&(X.b==8), 'y'], return_counts=True)
      ✓ 0.5s

...   ((array([0, 1]), array([1, 1], dtype=int64)),
       (array([0, 1]), array([3, 1], dtype=int64)))
```



Gotowe drzewo

Klasyfikator SVM

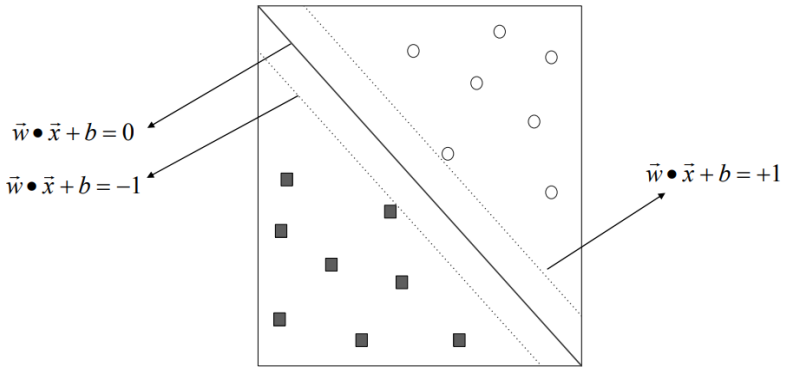
Pojęcia

Funkcja jądra (kernel)

Funkcja matematyczna używana do wygenerowania nowego wymiaru w przestrzeni danych. W nowo powstałym wymiarze wyszukujemy liniowej separowalności danych. Dla prostych przypadków można użyć liniowej funkcji jądra. W trudniejszych zdarzeniach należy wybrać bardziej zdolną funkcję na przykład wielomianową, sigmoidalną, radialną.

Przykład funkcji liniowej

Poszukujemy liniowej funkcji separującej dwie klasy: w · x + b = 0, gdzie w i b to parametry.



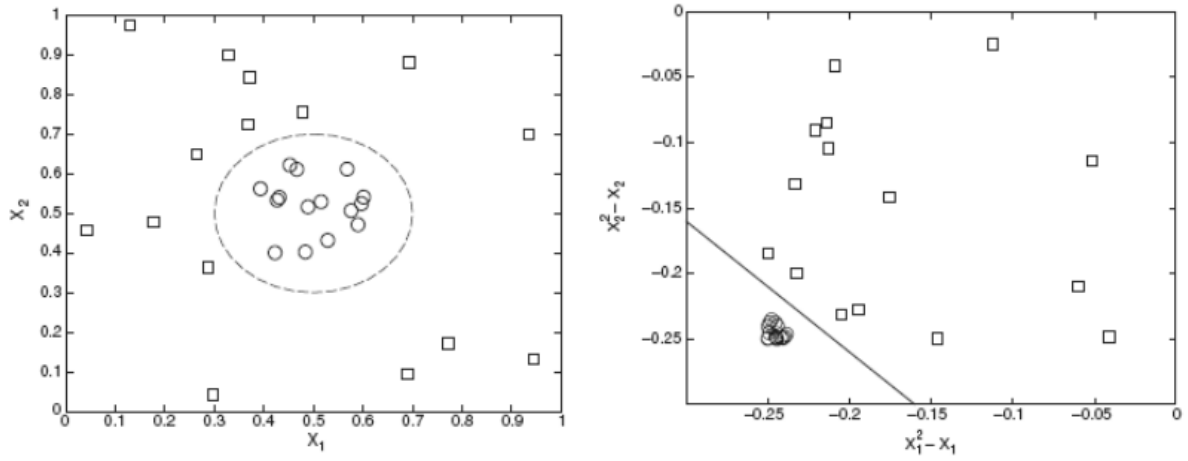
Granice wyznaczymy poprzez linie równoległe przechodzące przez najbliższe punkty każdej z klas tak jak na rysunku obok. (źródło)
Wówczas podział na klasy przedstawimy funkcją:

$$y = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} + \mathbf{b} > 0 \\ -1 & \mathbf{w} \cdot \mathbf{x} + \mathbf{b} < 0 \end{cases}$$

Zadaniem optymalizacji jest maksymalizacja odległości funkcji decyzyjnej od najbliższych punktów.

Przykład funkcji wielomianowej

Oryginalna funkcja celu to klasa 1 dla punktów w okręgu o promieniu 0,2 i środku w punkcie (0,5; 0,5) (rysunek po lewej stronie). Za pomocą funkcji wielomianowej przekształcimy dwuwymiarową przestrzeń do trzywymiarowej, aby uzyskać liniową separowalność w trzecim wymiarze (rysunek po prawej stronie).



Transformacja przekształcenia: $\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$

Zadaniem optymalizacji jest znalezienie parametrów funkcji wielomianowej przekształcenia.

Algorytm

Metoda SMV polega na przekształcaniu przestrzeni danych w sposób umożliwiający kategoryzację klas w sposób liniowy, nawet w przypadkach gdy w oryginalnej przestrzeni danych nie dało się ich liniowo oddzielić. Algorytm sprowadza się do znalezienia parametrów funkcji jądra. Schemat Uczenia algorytmu wygląda następująco:

Etap inicjalizacji modelu

1. Wybieramy funkcję jądra (kernela), przykładowe funkcje to:
    a. Liniowa - dla nieskomplikowanej separacji
    b. Wielomianowa
    c. Sigmoidalna
    d. Radialna
    e. Gaussowska
2. Dla wielomianowej funkcji jądra wybieramy stopień wielomianu

Etap treningu

1. Za pomocą funkcji jądra przekształcamy przestrzeń danych dodając dodatkowy wymiar.
2. Poszukujemy parametrów funkcji jądra dla których dane będą liniowo separowalne w nowo powstałym wymiarze.
3. Powstała hiperpłaszczyzna dzieląca próbki jest funkcją decyzyjną odpowiedzialną za rozróżnianie klas.

Etap wdrożenia

1. Algorytm dla podanych danych oblicza wartość funkcji jądra.
2. W zależności od położenia wartości względem wymodelowanej hiperpłaszczyzny, w postaci pewnej funkcji decyzyjnej, określana jest klasa punktu

Plan eksperymentów

Stworzone modele należy przetestować pod względem hiperparametrów zadeklarowanych w inicjalizacji procesu uczenia. W powyższych opisach algorytmów wyszczególniono możliwe wartości parametrów oraz ich wpływ na sposób działania algorytmu. Jednocześnie w ramach projektu przeprowadzimy szereg testów i pomiarów skuteczności oraz funkcjonalności wytworzonych modeli.

Ilość klasyfikatorów w drzewie - parametr B

Podstawowym rodzajem eksperymentów będzie dopasowanie ilości klasyfikatorów do zadanego problemu klasyfikacji. W zależności od liczby klasyfikatorów bazowych model może dawać odmienne rezultaty. Dla niewielkiej wartości B (1-3) model będzie zbyt ogólny, przy zbyt dużej ilości klasyfikatorów w stosunku do ilości próbek w zbiorze treningowym model będzie zbyt szczegółowy.

Jądro SVM - kernel

Dla klasyfikatora SVM należy podjąć decyzję, której z funkcji jądra użyć. Dla prostych przykładów funkcja liniowa może być wystarczająca, dla bardziej skomplikowanych nie da satysfakcjonujących rezultatów. W zależności od problemu i wielowymiarowego rozmieszczenia próbek funkcje jądra będą lepiej lub gorzej pasowały do zbioru danych.

Miary jakości

Do testowania i oceny modelu wykorzystamy znane i powszechnie używane metody do walidacji modeli sztucznej inteligencji.

Obciążenie (ang. Bias)

Błąd klasyfikacji na zbiorze trenującym. Predykcja klas dla próbek ze zbioru trenującego. Bias to stosunek błędnie sklasyfikowanych przykładów do wszystkich.

Skuteczność na zbiorze testowym

To wydzielony fragment ze zbioru danych, nie jest używany do trenowania modelu. Na tym zbiorze można przeprowadzić predykcję klas i określić skuteczność z jaką model przypisywał klasy nowym próbkom.

Reprezentacja wyników

Macierz pomyłek (ang. Confusion Matrix)

Tabela zawierająca wyniki skuteczności modelu na zbiorze trenującym lub testowym. Pokazuje liczbę pomyłek i która klasa została błędnie wybrana zamiast oczekiwanej. Można dzięki niej obserwować które klasy są dobrze rozpoznane a które mylone przez model.

Zbiory danych

Źródło

Do uczenia i testowania modeli wykorzystamy zbiory danych udostępnione przez Uniwersytet Kaliforni (University of Calofornia, Irvine). Na ogólnodostępnej stronie internetowej zamieszczono 466 zbiorów danych dla zadania klasyfikacji. Link do strony

Wybór zbiorów

Spośród wielu dostępnych zbiorów prowadzimy selekcję na podstawie trzech czynników:

1. Liczba atrybutów - zależy nam na przetestowaniu modelu dla zbiorów z niewielką liczbą (3-7) i dla zbiorów ze średnią (10-20) i bardzo dużą liczbą atrybutów (100+).
2. Liczba przykładów - Należy przetestować model w przypadku gdy mamy do czynienia z dużą ilością danych i z niewielką ilością danych.
3. Rodzaj atrybutów - atrybuty mogą przyjmować wartości liczb rzeczywistych, albo być skategoryzowane. W przypadku sklasyfikowanych atrybutów należy przekształcić je na liczby całkowite lub zmiennoprzecinkowe.

Wybrane zbiory testowe do projektu

1. Adult Data Set - binarna klasyfikacja, predykcja zarobków czy powyżej 50 tys.$
   Liczba klas: 2
   Liczba atrybutów: 14
   Liczba przykładów: 48842
2. Car Evaluation Data Set - podział na klasy, (słaby, średni, dobry, b.dobry)
   Liczba klas: 4
   Liczba atrybutów: 6
   Liczba przykładów: 1728
3. Divorce Predictors Data Set - predykcja rozwodów, binarne atrybuty i klasyfikacja
   Liczba klas: 2
   Liczba atrybutów: 54
   Liczba przykładów: 170
4. Seeds Data Set - trzy rodzaje zbóż, rozpoznawanie po wymiarach nasienia
   Liczba klas: 3
   Liczba atrybutów: 7
   Liczba przykładów: 210
5. Musk (Version 2) Data Set - rozpoznawanie cząstek na podstawie molekułów
   Liczba klas: 2
   Liczba atrybutów: 168
   Liczba przykładów: 6598
6. Leaf Data Set - rozpoznawanie 40 gatunków roślin po liściach
   Liczba klas: 40
   Liczba atrybutów: 16
   Liczba przykładów: 340

Implementacja i opis plików

Modele

W folderze models zamieściliśmy implementację drzewa losowego, oraz implementację algorytmu ID3. W lesie losowym wykorzystaliśmy klasyfikator SVM z biblioteki sklearn oraz naszą implementację ID3.

Moduł do przeprowadzania eksperymentów

W folderze experiments znajduje się moduł do wielokrotnego przeprowadzania eksperymentów dla zadanych parametrów lasu losowego (`experimentModule.py`). Moduł jako argumenty przyjmuje nazwę zbioru danych, liczbę eksperymentów do przeprowadzenia oraz jaką część zbiory danych przeznaczy na zbiór testowy. Moduł n razy tworzy model lasu losowego, trenuje zbiorem treningowym i sprawdza skuteczność na zbiorze testowym. Zwraca zagregowane wyniki w postaci skuteczności i macierzy pomyłek.

Pomocnicze funkcje

W pliku `functions.py` w folderze experiments zamieściliśmy często powtarzane operacje przy procesie eksperymentów. Między innymi funkcje do rysowania macierzy pomyłek, odczytywania, zapisywania danych do plików oraz agregacji danych.

Zbiory danych

W folderze datasets umieściliśmy pliki ze zbiorami danych formacie csv. Dane pobrane ze strony UCI wymagały przekonwertowania na csv co wykonaliśmy ręcznie za pomocą arkuszy kalkulacyjnych. Wartości w kolumnie “target” w każdym z plików oznaczają przypisaną klasę dla danego wiersza. Dodatkowo w pliku `features.json` sporządziliśmy listę nazw kolumn atrybutów i listę możliwych klas dla zbiorów danych.

Wyniki eksperymentów

W folderze results zamieściliśmy wyniki przeprowadzonych eksperymentów, zagregowane pliki json i pickle dla każdej metody eksperymentów. W podfolderach dla konkretnych zbiorów danych załączyliśmy macierz pomyłek dla klasyfikatora z największą skutecznością w danym eksperymencie.

Eksperymenty i analiza

W notatniku `experiments.ipynb` przeprowadziliśmy eksperymenty i badania parametrów lasu losowego oraz klasyfikatora SVM dla każdego ze zbiorów danych. Wyniki z tych eksperymentów zapisaliśmy w folderze results. Następnie w notatniku `analysis.ipynb` przeprowadziliśmy analizę zebranych danych i przedstawiliśmy spostrzeżenia i wnioski.

Przeprowadzone eksperymenty

Dla każdego ze zbiorów danych badaliśmy poniższe parametry lasu losowego:

- n_clfs - liczbę klasyfikatorów
- rows_prc - procent wierszy ze zbioru treningowego dla pojedynczego klasyfikatora
- svm_prc - procent udziału klasyfikatora typu SVM wśród wszystkich klasyfikatorów
- kernel - jądro klasyfikatorów SVM

Procent atrybutów wybranych do pojedynczego klasyfikatora uzależniliśmy od liczby atrybutów n w zbiorze danych i wyznaczyliśmy na stałym poziomie obliczając $\sqrt{n}$

Grid search

Dla analizowanych parametrów wybraliśmy równomiernie rozłożone wartości. Przeprowadziliśmy eksperymenty dla wszystkich możliwych permutacji poniższych parametrów:

```
n_clfs   = [30, 60, 90, 120]
rows_prc = [10, 30, 50, 70, 90]
svm_prc  = [10, 30, 50, 70, 90]
kernel   = ["linear", "poly", "rbf", "sigmoid"]
```

Randomised search

W losowy sposób dobraliśmy wartości analizowanych parametrów, aby nie ominąć potencjalnie dobrych kombinacji które mogą znajdować się “pomiędzy” siatką wartości przy experymencie typu grid search. Poniżej zakresy z których losowaliśmy wartości:

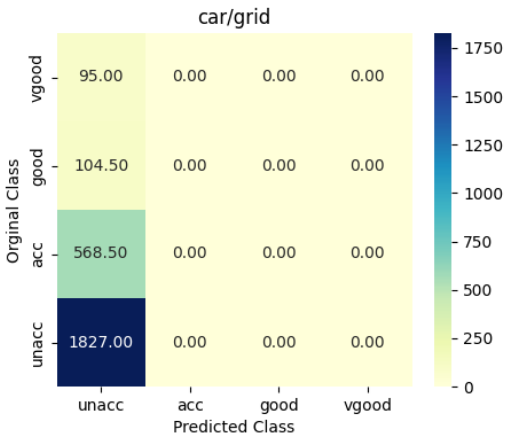
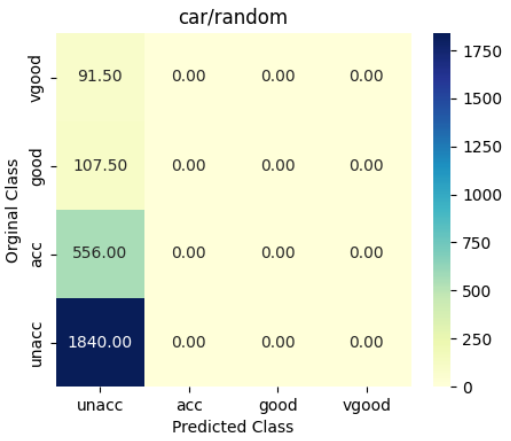
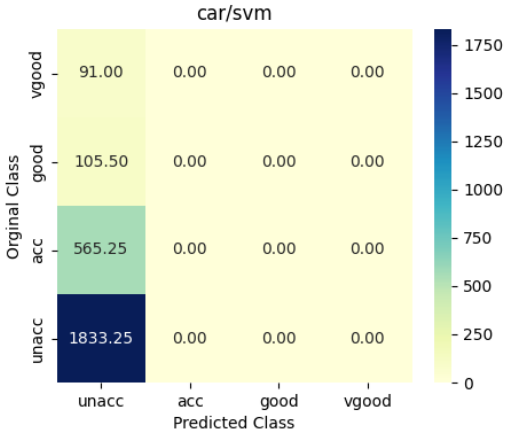
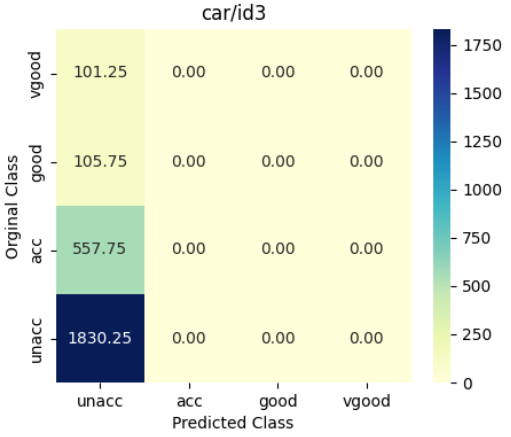
```
n_clfs   = [30, 100]
rows_prc = [20, 100]
svm_prc  = [20, 100]
kernel   = ["linear", "poly", "rbf", "sigmoid"]
```

Las losowy SVM oraz las losowy ID3

Przetestowaliśmy także skuteczność modelu dla przypadków gdy cały las składa się tylko z jednego z rodzajów klasyfikatorów. Dla obu klasyfikatorów metodą randomised search szukaliśmy najlepszych zestawów parametrów dla każdego zbioru danych.

Analiza wyników

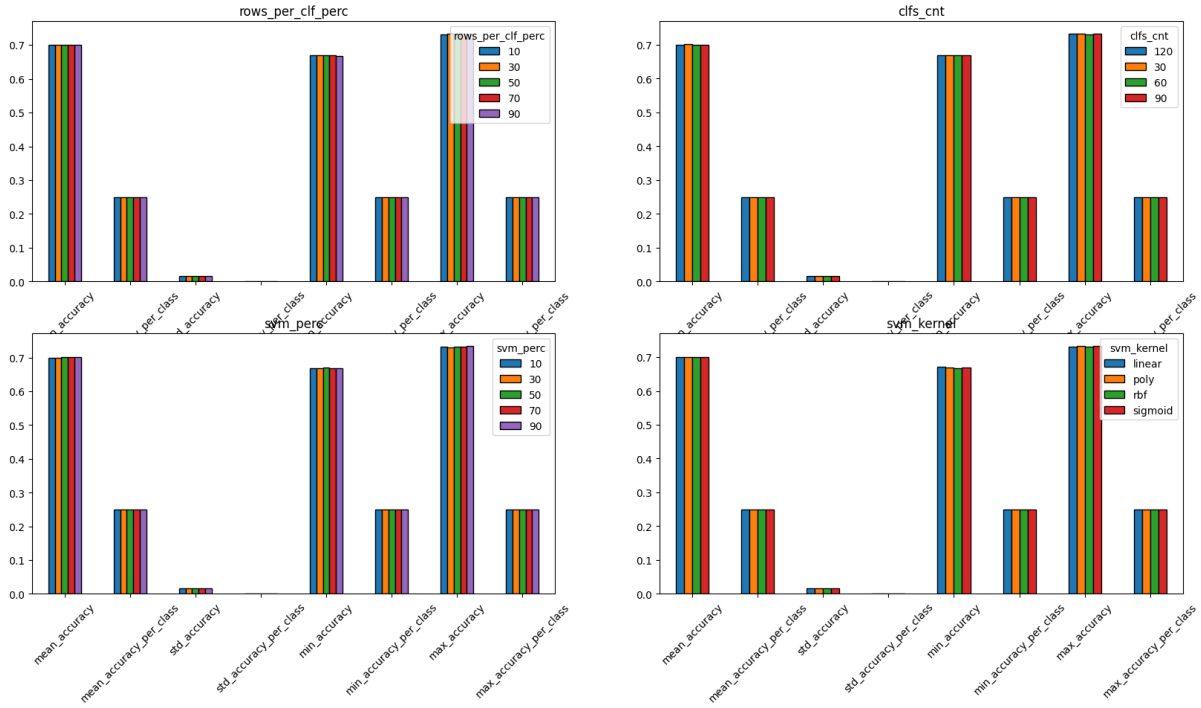
Zbiór danych cars

Grid search	Randomised search
	
SVM search	**ID3 search**
	

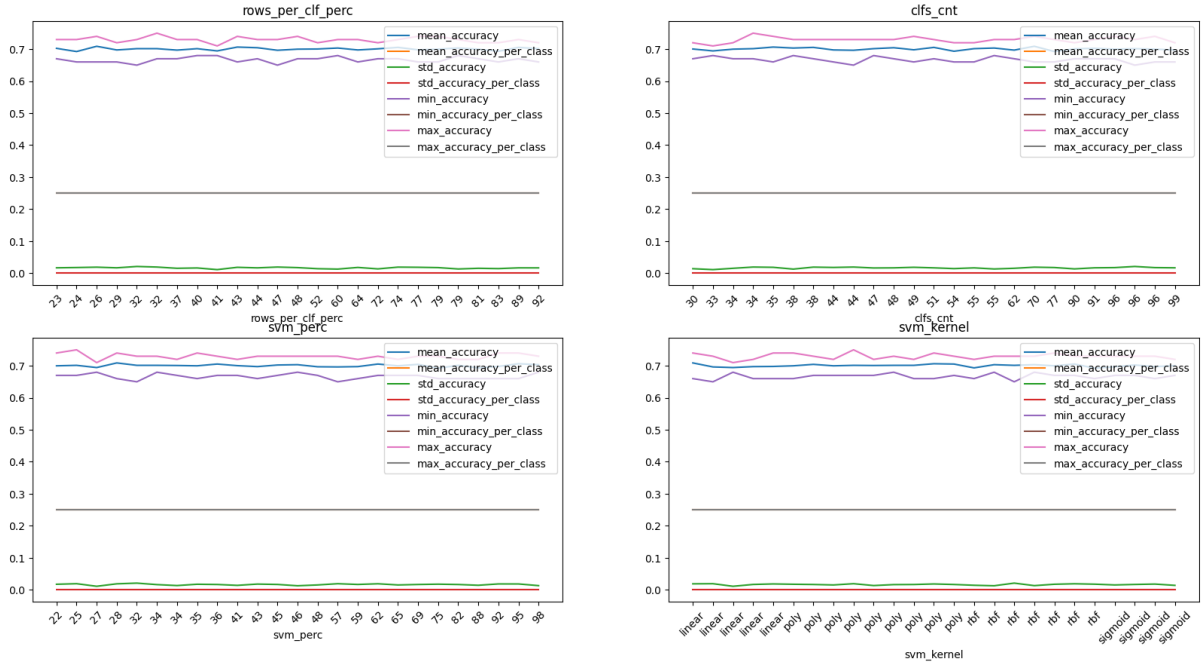
Wyniki

Okazuje się, że przez niezbalansowane klasy modele nauczyły się jedynie przewidywać najliczniejszą z klas. Z tego powodu nie jesteśmy w stanie porównać różnych zestawów hiperparametrów do siebie

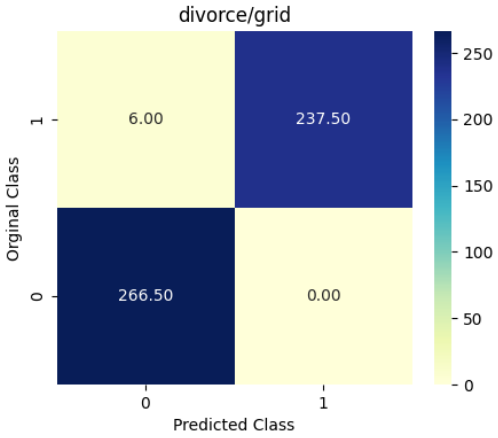
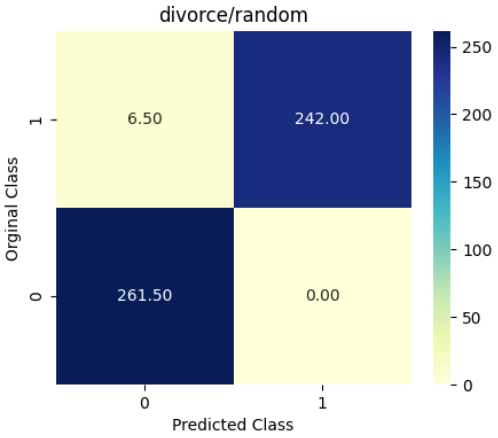
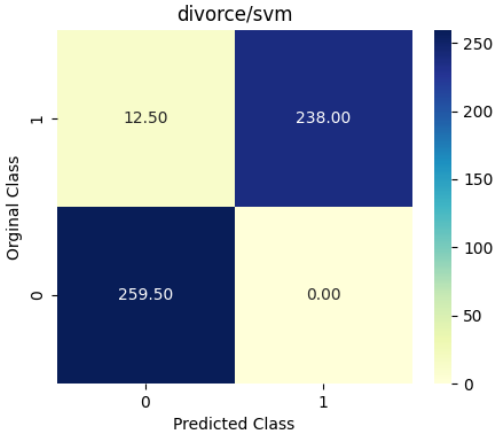
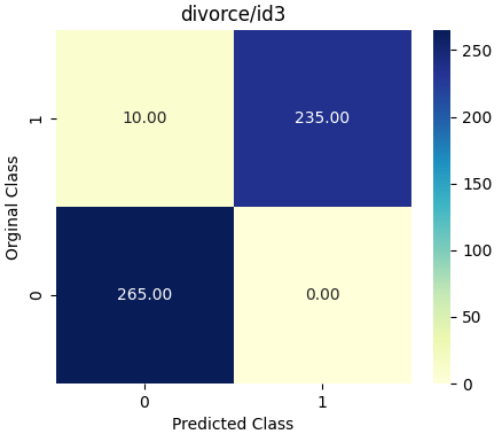
Grid search



Randomised search



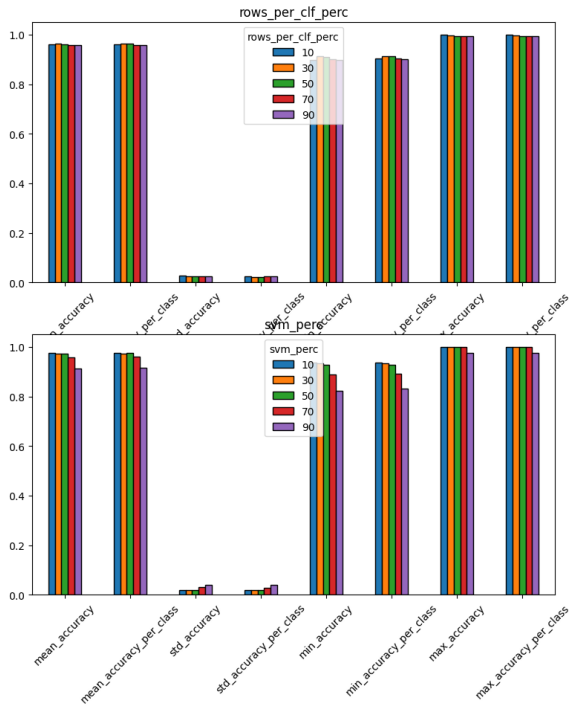
Zbiór danych divorce

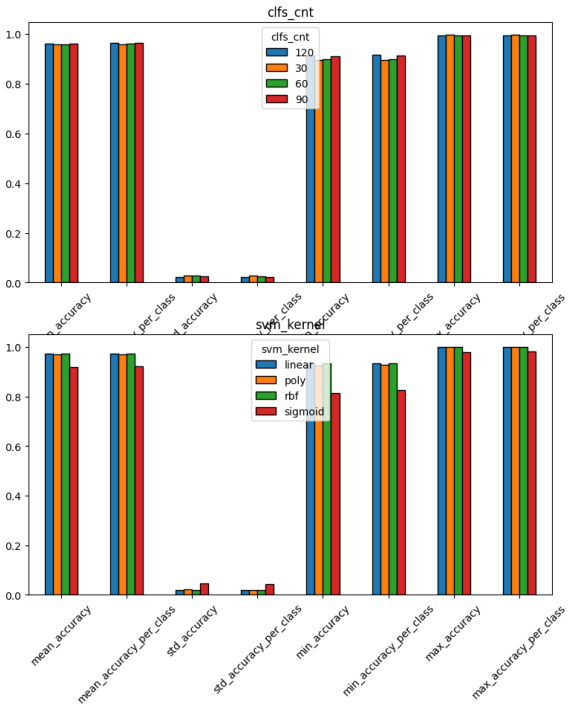
Grid search	Randomised search
	
SVM search	ID3 search
	

Wyniki

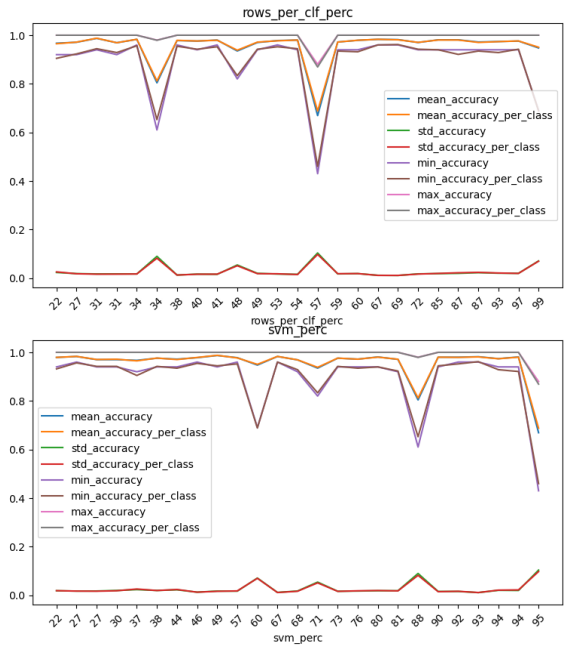
W przypadku tego datasetu widoczne jest, że niezależnie od wszystkich hiperparametrów lasu losowego poza svm_kernel modele performują równie dobrze. Dla svm_kernel = sigmoid mamy znaczny spadek performance, zarówno dla zwykłego accuracy jak i dla accuracy per klasa. Powyższe wnioski potwierdza przeszukiwanie typu GridSearch: dla wszystkich hiperparametrów metryki są bardzo stabilne, jednakże dla hiperparametru svm_kernel widzimy, że sigmoid ma zauważalnie gorsze wyniki niż pozostałe opcje. Dodatkowo dla eksperymentu z lasami składającymi się wyłącznie z klasyfikatorów typu SVM widzimy to wyraźnie dla tego typu kernela.

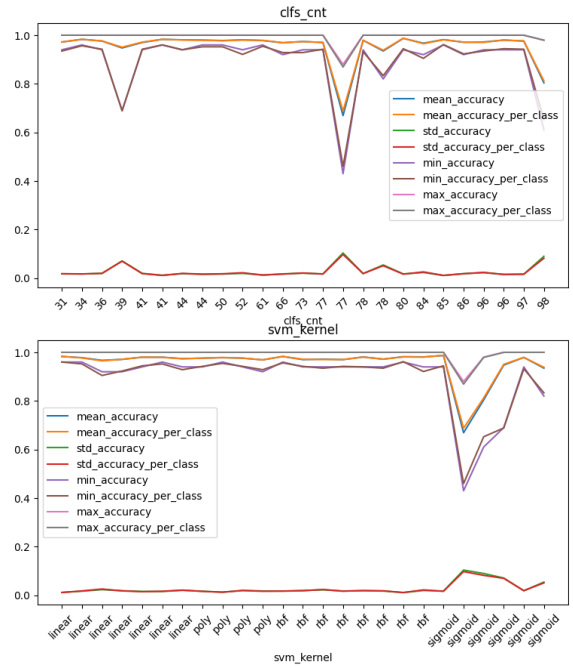
Grid search



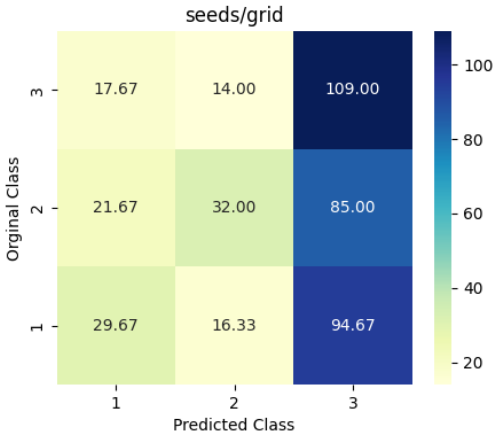
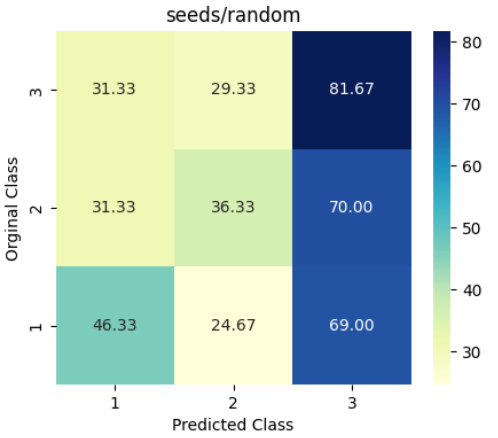
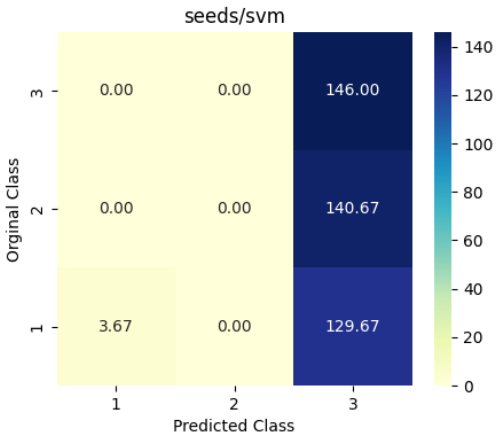
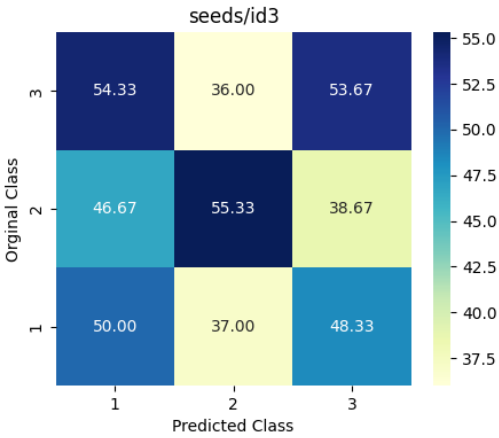


Randomised search





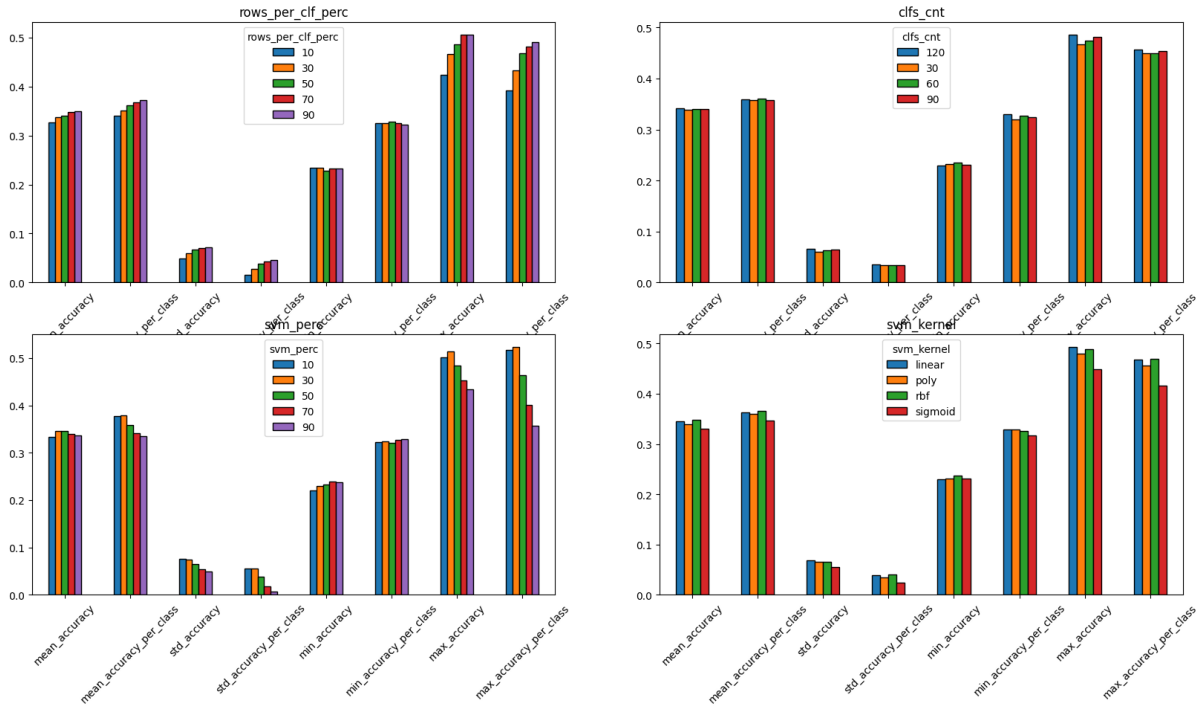
Zbiór danych seeds

Grid search	Randomised search
	
SVM search	**ID3 search**
	

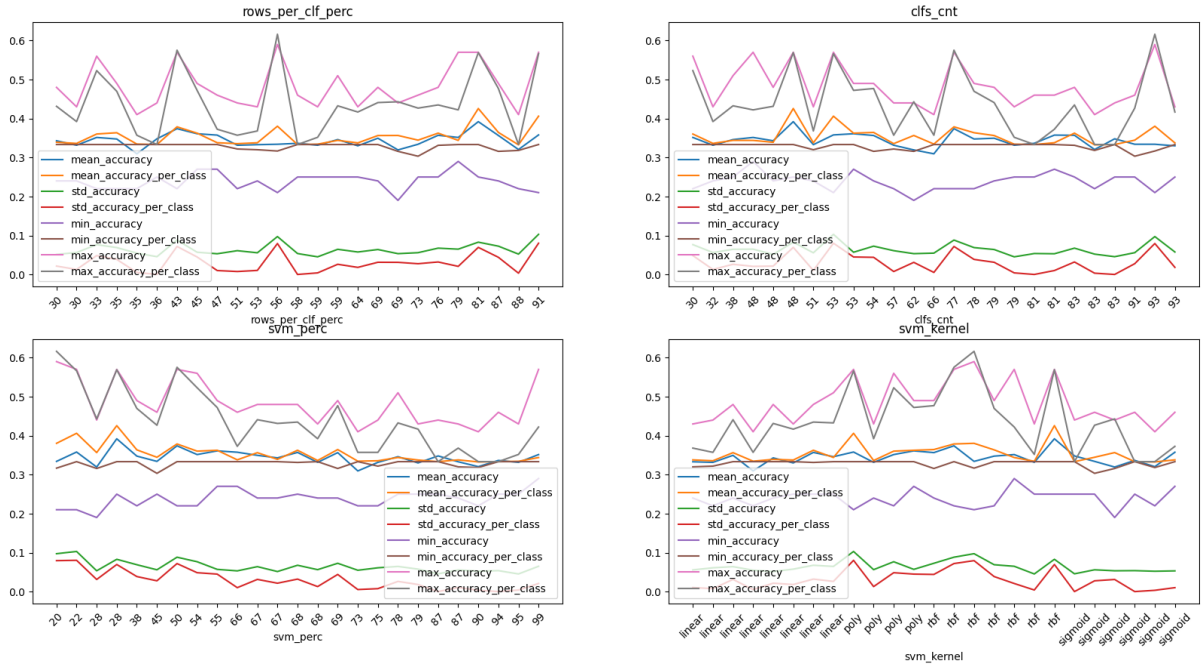
Wyniki

Wyniki na tym zbiorze danych są najbardziej informatywne dotychczas. Widzimy, że im większy udział klasyfikatorów SVM w lesie tym średnio gorsze wyniki (svm_perc). Jesteśmy również w stanie zauważyć, że im większa część datasetu użyta była do wytrenowania pojedynczego klasyfikatora w drzewie tym lepsze wyniki lasu. Ilość klasyfikatorów nie wpływa w znaczący sposób na wyniki.

Grid search



Randomised search



Wnioski i podsumowanie

Pozostałe zbiory danych

Dla zadań rozpoznawania cząstek i predykcji rozwodów modele zacinały się podczas procesu uczenia i nie dało się wytrenować modelu. Nie byliśmy w stanie znaleźć powodu dlaczego akurat te zbiory danych. Niektóre eksperymenty udało nam się na nich przeprowadzić. Zadanie rozpoznawania roślin po liściach przerosło las losowy i nie dawał żadnych rozsądnych wyników, skuteczność na tym zbiorze danych otrzymywaliśmy na poziomie 0.5 %, dlatego odsunęliśmy ten zbiór danych od eksperymentów.

Przewlekłe obliczenia

Przeprowadzenie eksperymentów, czyli wytrenowanie modeli z wieloma różnymi parametrami, oraz przetestowanie ich trwało bardzo długo i wiele razy się zacinało lub przerywało. Przeprowadzenie finalnych eksperymentów z notatnika experiments trwało ponad dobę. Testowanie lasu losowego generuje olbrzymie ilości modeli co przekłada się na długie czasy uczenia.

Podsumowanie

Las losowy w naszych eksperymentach był odporny na parametry i dla większości dawał zbliżone rezultaty. Dzięki temu że las zawiera wiele klasyfikatorów oraz system głosowania nie grozi mu nadmierne dopasowanie do danych. Losowe przydzielanie atrybutów oraz wierszy do klasyfikatora powoduje większą różnorodność modeli i inne predykcje.