ML SVM, KNN, K-Means

ML SVM, KNN, K-Means

Agenda

1. **Architektura SVM**
2. Funkcja kernela i podnoszenie przestrzeni wymiarów
3. Przestrzeń metryczna
4. Parametry KNN
5. Uczenie nienadzorowane na przykładzie K-Means

ML SVM, KNN, K-Means

Agenda

1. Architektura SVM
2. **Funkcja kernela i podnoszenie przestrzeni wymiarów**
3. Przestrzeń metryczna
4. Parametry KNN
5. Uczenie nienadzorowane na przykładzie K-Means

ML SVM, KNN, K-Means

Agenda

1. Architektura SVM
2. Funkcja kernela i podnoszenie przestrzeni wymiarów
3. **Przestrzeń metryczna**
4. Parametry KNN
5. Uczenie nienadzorowane na przykładzie K-Means

ML SVM, KNN, K-Means

Agenda

1. Architektura SVM
2. Funkcja kernela i podnoszenie przestrzeni wymiarów
3. Przestrzeń metryczna
4. **Parametry KNN**
5. Uczenie nienadzorowane na przykładzie K-Means

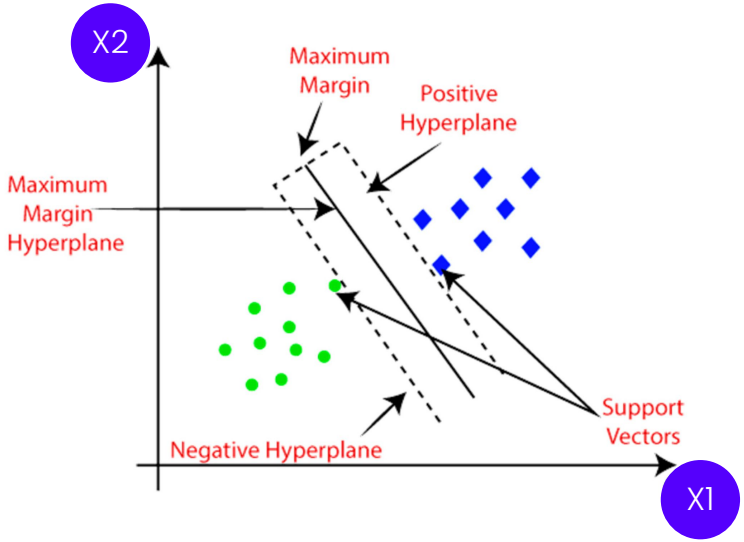
ML SVM, KNN, K-Means

Agenda

1. Architektura SVM
2. Funkcja kernela i podnoszenie przestrzeni wymiarów
3. Przestrzeń metryczna
4. Parametry KNN
5. **Uczenie nienadzorowane na przykładzie K-Means**

ML SVM, KNN, K-Means

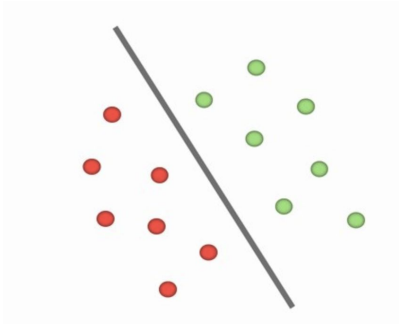
SVM



ML SVM, KNN, K-Means

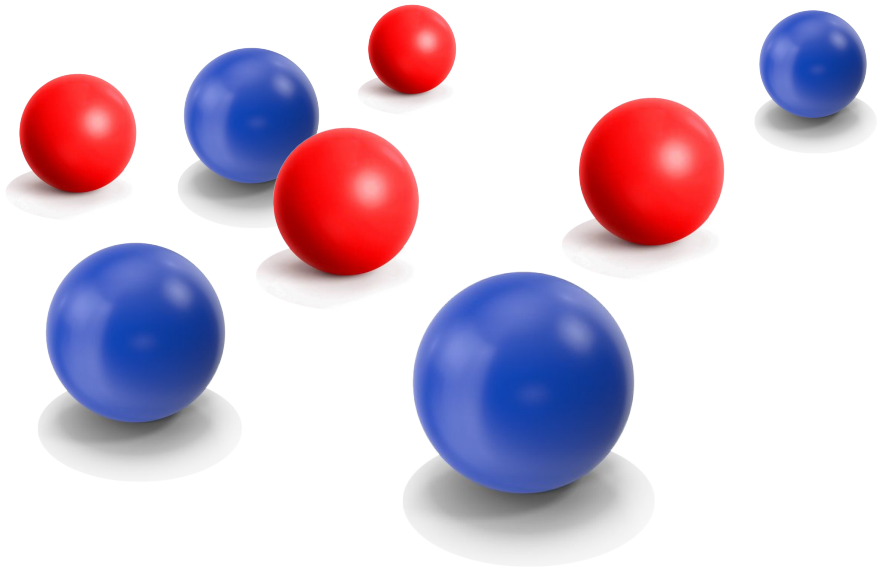
SVM - historia

Opracowane w AT&T Bell Laboratories przez Vladimira Vapnika z kolegami w latach 90 tych -oparte na statystycznych ramach uczenia się lub teorii VC zaproponowanej przez Vapnika (1982, 1995) i Chervonenkisa (1974).



ML SVM, KNN, K-Means

Idea SVM



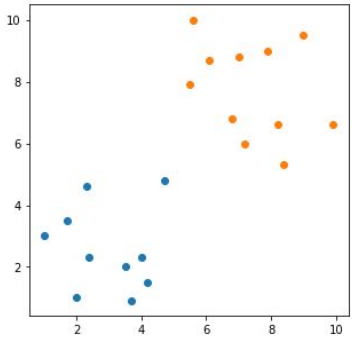
infoShareAcademy.com

ML SVM, KNN, K-Means

Idea działania

```
data = pd.DataFrame({'Kolor': ['CZ', 'CZ', 'CZ', 'CZ', 'CZ', 'CZ', 'CZ', 'CZ', 'CZ', 'CZ', 'NI', 'NI', 'NI', 'NI', 'NI', 'NI', 'NI', 'NI', 'NI', 'NI'],
                    'X': [2, 4, 3.5, 4.2, 1, 4.7, 2.4, 1.7, 2.3, 3.7, 5.6, 7, 9.9, 6.8, 5.5, 8.4, 7.2, 6.1, 9, 8.2, 7.9],
                    'Y': [1, 2.3, 2, 1.5, 3, 4.8, 2.3, 3.5, 4.6, .9, 10, 8.8, 6.6, 6.8, 7.9, 5.3, 6, 8.7, 9.5, 6.6, 9]})
```

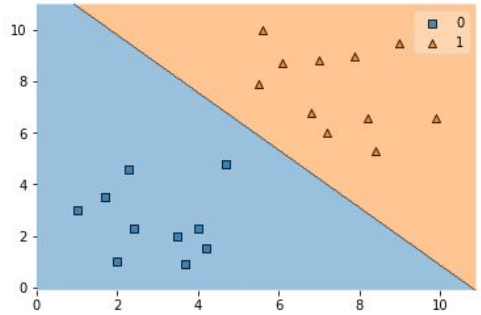
```
plt.figure(figsize=(5,5));
plt.scatter(data[data['Kolor'] =='CZ']['X'], data[data['Kolor'] =='CZ']['Y']);
plt.scatter(data[data['Kolor'] =='NI']['X'], data[data['Kolor'] =='NI']['Y']);
```



ML SVM, KNN, K-Means

Idea działania

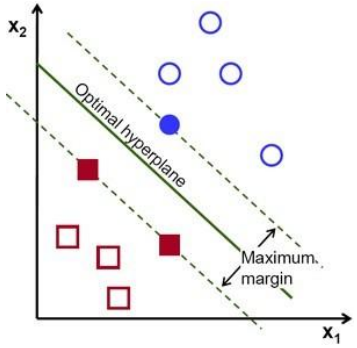
```
encoder = LabelEncoder()
data['Kolor'] = encoder.fit_transform(data['Kolor'])

simple_svm = SVC(kernel='linear')
simple_svm.fit(data[['X', 'Y']], data['Kolor']);

plot_decision_regions(X = data[['X', 'Y']].to_numpy(), y =
data['Kolor'].to_numpy().astype(np.int), clf=simple_svm);
```



ML SVM, KNN, K-Means

hiperplane

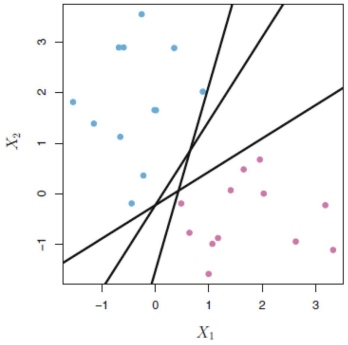
- Linia oddzielajaca klasy to optimal hiperplane – hiperpłaszczyzna.
- Wektory na których się ją “wspiera” to tzw. Support vectors - wektory nośne.
  Są to separowalne punkty w przestrzeni, które są najczęściej skrajnymi elementami klas.
- Maximum margin – maksymalny margines. Celem algorytmu jest uzyskanie największego marginesu pomiędzy klasami. Im większy tym lepiej bo są wtedy maksymalnie “rożne”.

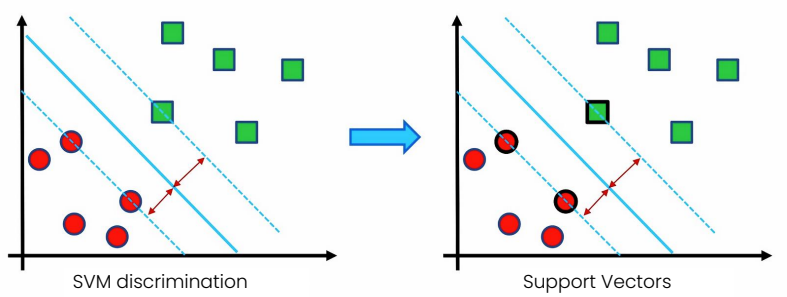


ML SVM, KNN, K-Means

hiperplane

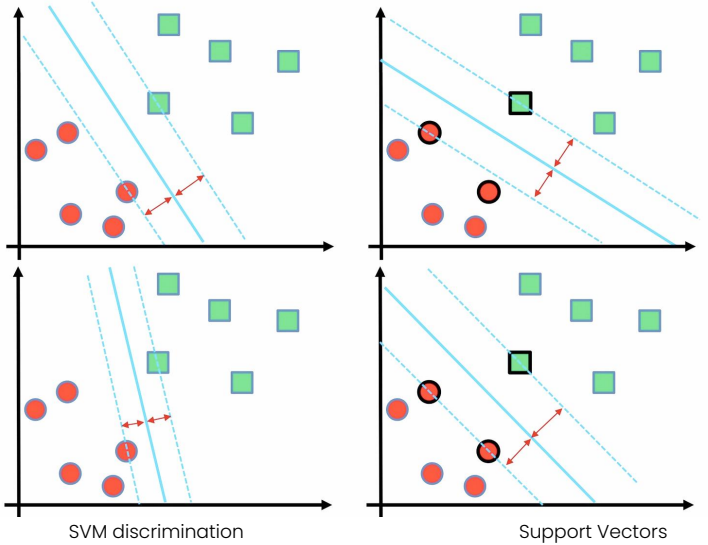
Możliwe jest narysowanie wielu hiperpłaszczyzn (rys. po prawej) ale algorytm szuka optymalnej.





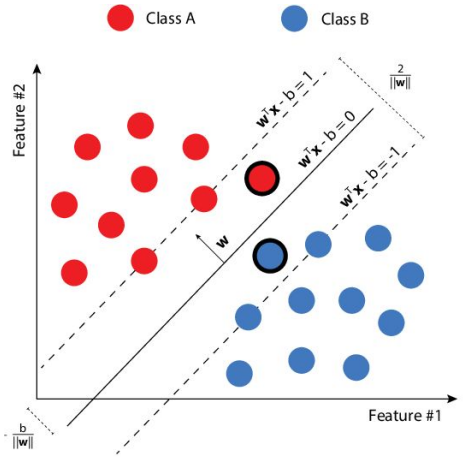
ML SVM, KNN, K-Means

hiperplane



ML SVM, KNN, K-Means

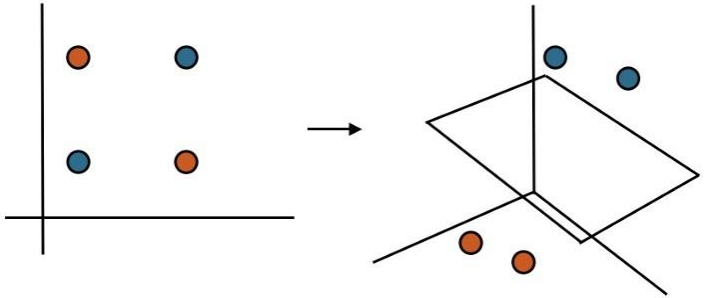
Miękki margines



ML SVM, KNN, K-Means

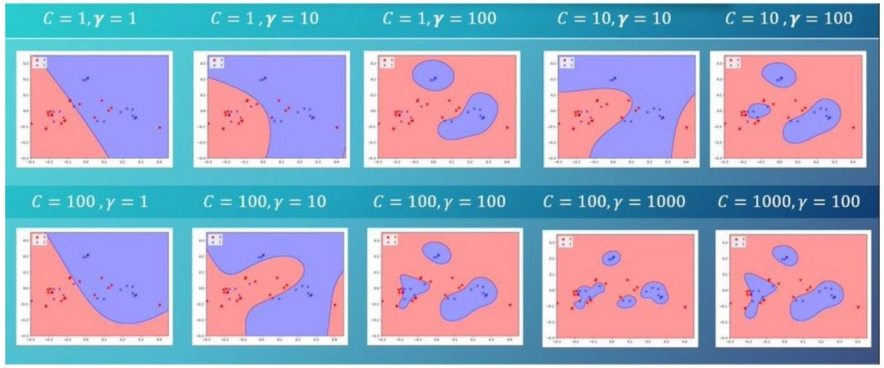
SVM

Maszyna wektorów nośnych klasyfikuje dane wykorzystując niejawne przekształcenie zbioru treningowego do przestrzeni cech wyższego wymiaru. W nowej przestrzeni cech dopasowywana jest optymalna hiperpłaszczyzna rozdzielająca dwie klasy danych i jednocześnie maksymalizuje margines pomiędzy hiperpłaszczyzną, a punktami znajdującymi się najbliżej niej, nazywanymi wektorami nośnymi.



ML SVM, KNN, K-Means

C oraz gamma



ML SVM, KNN, K-Means

SVM - regresja

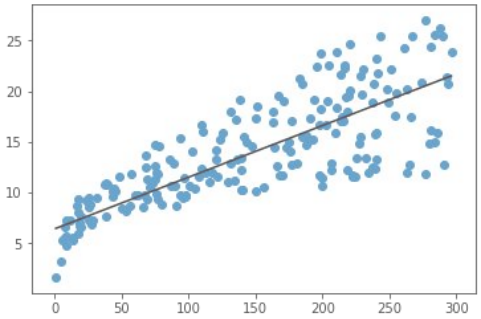
```
df = pd.read_csv("data/Advertising.csv")
df = df.drop("Unnamed: 0", axis=1)
X = df['TV']
y = df['Sales']
df.head()
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

ML SVM, KNN, K-Means

SVM - regresja

```
srv_reg = SVR(kernel='linear').fit(X.to_numpy().reshape(-1,1), y)

def reg(x):
        y = srv_reg.predict(x)
        return y

xs = np.arange(min(X),max(X),0.01)
plt.plot(xs,reg(xs.reshape(-1,1)),color="black");
plt.scatter(X,y);
```



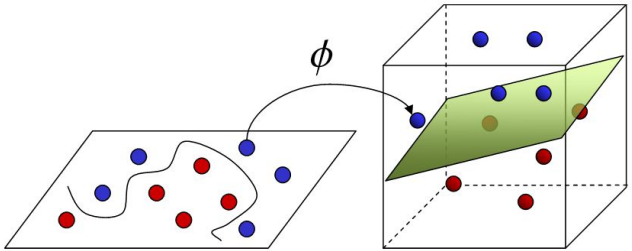
Zadanie 15.1 (instrukcja)

1. Korzystając ze zbioru danych "Iris" dostępnego w bibliotece scikit-learn, zadanie klasyfikacji ma na celu przewidzenie gatunku irysa (Setosa, Versicolor, lub Virginica) na podstawie cech takich jak długość i szerokość płatków oraz długość i szerokość działek kielicha.
2. Dla tego zadania, skorzystamy z zbioru danych "Boston House Prices" dostępnego w scikit-learn. Celem jest przewidzenie cen domów w Bostonie na podstawie różnych cech, takich jak liczba pokoi, współczynnik przestępczości w okolicy, czy odległość do zatrudnienia.

ML SVM, KNN, K-Means

kernel trick

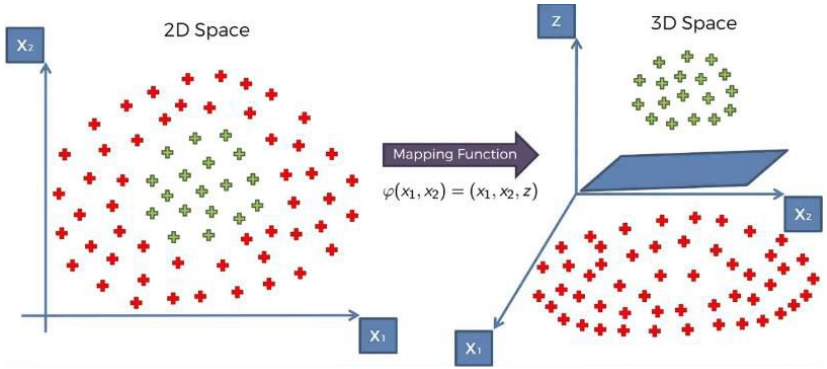
Jądro - funkcja matematyczna służąca do przekształcania danych wejściowych w inną formę.
Typowe funkcje jądra obejmują liniowe, nieliniowe, wielomianowe itp.



Input Space Feature Space

ML SVM, KNN, K-Means

kernel trick

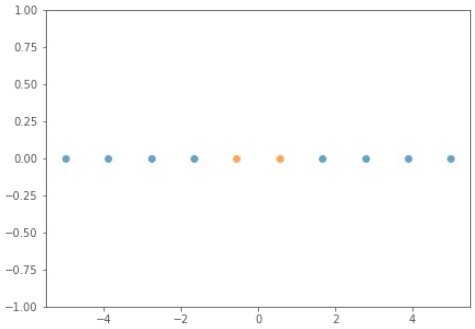


ML SVM, KNN, K-Means

Jak odseparować od siebie klasy, które wydają się być nie do odseparowania?

Przykład liniowy:

```
X = np.linspace(-5, 5, 10)
y = np.zeros(10)
labs = np.array([0, 0, 0, 0, 1, 1, 0, 0, 0, 0])
plt.figure(figsize=(7,5))
plt.scatter(X[labs!=1], y[labs!=1])
plt.scatter(X[labs==1], y[labs==1])
plt.ylim(bottom=-1, top=1);
```

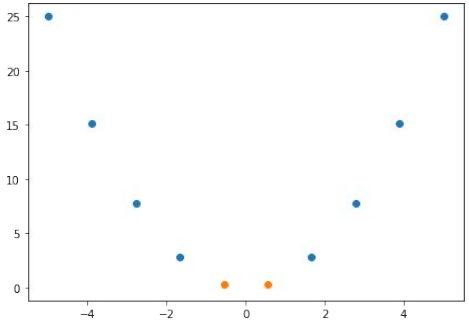


ML SVM, KNN, K-Means

Jak odseparować od siebie klasy, które wydają się być nie do odseparowania?

x_sq = X2**

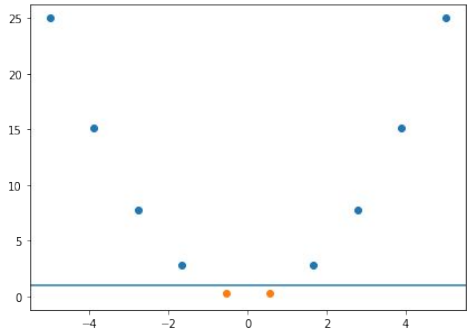
```
plt.figure(figsize=(7,5))
plt.scatter(X[labs!=1], x_sq[labs!=1])
plt.scatter(X[labs==1], x_sq[labs==1]);
```



ML SVM, KNN, K-Means

Jak odseparować od siebie klasy, które wydają się być nie do odseparowania?

```
plt.figure(figsize=(7,5))
plt.scatter(X[labs!=1], x_sq[labs!=1])
plt.scatter(X[labs==1], x_sq[labs==1])
plt.axhline(y=1, );
```

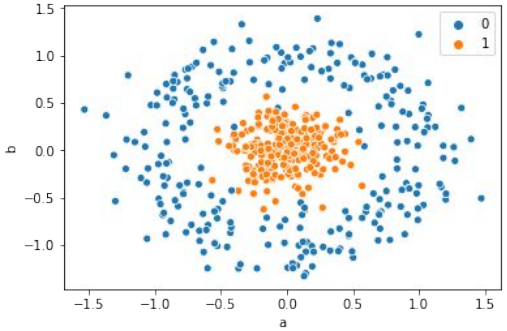


infoShareAcademy.com

ML SVM, KNN, K-Means

Typy kerneli

```
df = datasets.make_circles(n_samples=500,random_state=0, noise =.2, factor=.1)
X = pd.DataFrame(df[0], columns=['a', 'b'])
y = df[1]
sns.scatterplot(X.a, X.b, hue = y);
```

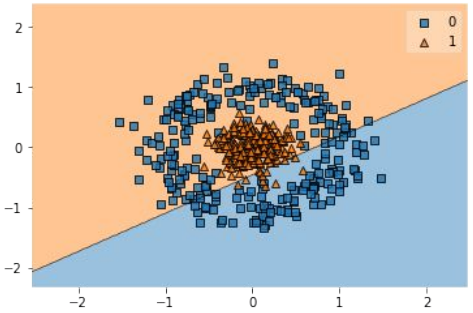


ML SVM, KNN, K-Means

Typy kerneli

Liniowy:

```
lin_svm = SVC(kernel='linear',gamma = 'scale')
lin_svm.fit(X,y)
plot_decision_regions(X.to_numpy(),y, clf=lin_svm);
```



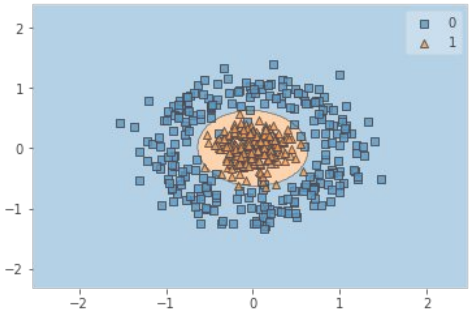
ML SVM, KNN, K-Means

Typy kerneli

Wielomianowy:

Degree=2

```
svm_poly2 = SVC(kernel='poly', degree=2,gamma
= 'scale').fit(X,y)
plot_decision_regions(X.to_numpy(),y, clf=svm_poly2);
```



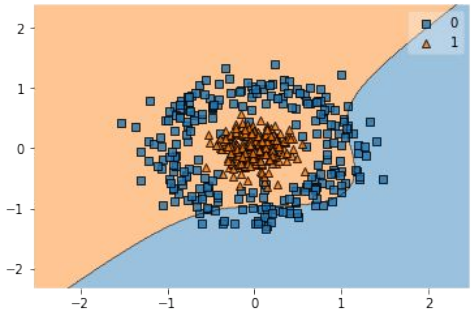
ML SVM, KNN, K-Means

Typy kerneli

Wielomianowy:

Degree=3

```
svm_poly3 = SVC(kernel='poly', degree=3,gamma = 'scale').fit(X,y)
plot_decision_regions(X.to_numpy(),y, clf=svm_poly3);
```

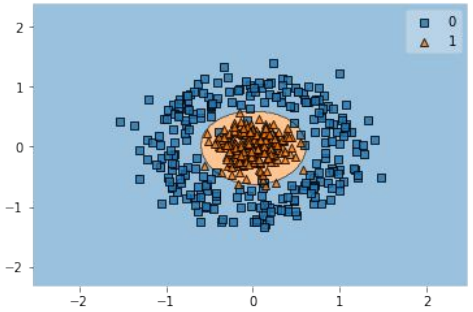


ML SVM, KNN, K-Means

Typy kerneli

Radial (rbf):

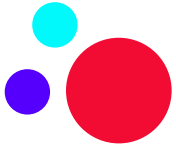
```
svm_rad = SVC(kernel='rbf',gamma = 'scale').fit(X,y)
plot_decision_regions(X.to_numpy(),y, clf=svm_rad);
```



Zadanie 15.2 (instrukcja)

1. Dla zbioru danych poniżej stwórz modele SVM z kernelem liniowym, radial i polynominal w kilku wariantach stopni wielomianu. Narysuj wykresy oraz policz skuteczność modeli. Który był najskuteczniejszy ?

```
df = datasets.make_moons(n_samples=500, random_state=0, noise = .09)
X = pd.DataFrame(df[0], columns=['a', 'b'])
y = df[1]
sns.scatterplot(X.a, X.b, hue = y);
```



ML SVM, KNN, K-Means

Klasyfikacja wieloklasowa

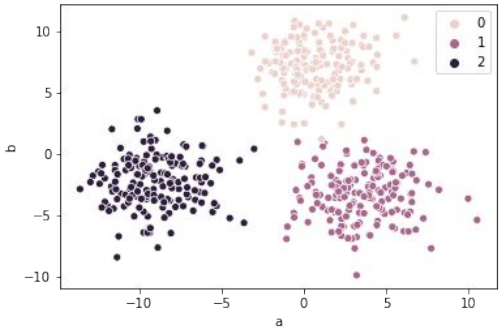
Dwa główne podejścia:

- one vs one (stosowany domyślnie w sklearn),
- one vs all (aby go użyć należy posłużyć się modelem sklearn.multiclass.OneVsRestClassifier).

ML SVM, KNN, K-Means

Klasyfikacja wieloklasowa

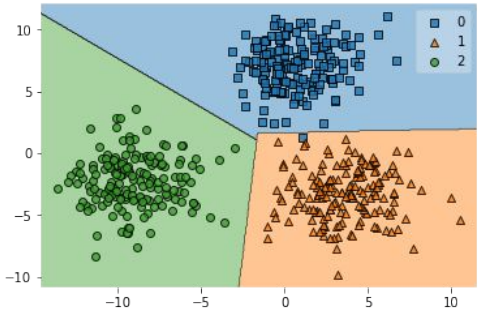
```
df = datasets.make_blobs(n_samples=500, random_state=67,
centers=3,cluster_std=2)
X = pd.DataFrame(df[0], columns=['a', 'b'])
y = df[1]
sns.scatterplot(X.a, X.b, hue = y);
```



ML SVM, KNN, K-Means

Klasyfikacja wieloklasowa

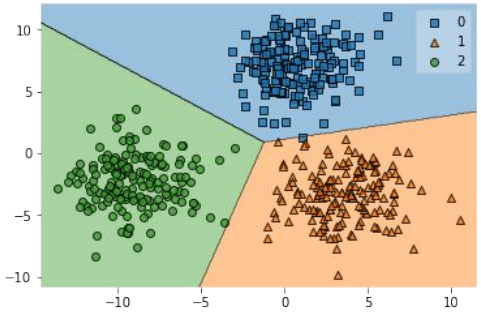
```
lin_ovo = SVC(kernel='linear')
lin_ovo.fit(X,y)
plot_decision_regions(X.to_numpy(),y, clf=lin_ovo);
```



ML SVM, KNN, K-Means

Klasyfikacja wieloklasowa

```
lin_ovr = OneVsRestClassifier(SVC(kernel='linear')).fit(X,y)
plot_decision_regions(X.to_numpy(),y, clf=lin_ovr);
```



Zadanie 15.3 (instrukcja)

Rozważmy zbiór danych Iris, dostępny w bibliotece scikit-learn. Zadaniem jest zastosowanie dwóch różnych podejść do klasyfikacji wieloklasowej, tj. One-vs-One (OvO) i One-vs-Rest (OvR), za pomocą klasyfikatora SVM z kernela liniowego.

1. One-vs-One (OvO):
   - Użyj klasyfikatora SVM z kernela liniowego w podejściu One-vs-One do klasyfikacji trzech klas (setosa, versicolor, virginica) na podstawie cech kwiatów z datasetu Iris.
   - Podziel zbiór danych na zbiór treningowy i testowy.
   - Wytrenuj model SVM za pomocą podejścia OvO.
   - Dokonaj predykcji dla danych testowych.
   - Oceń dokładność modelu.

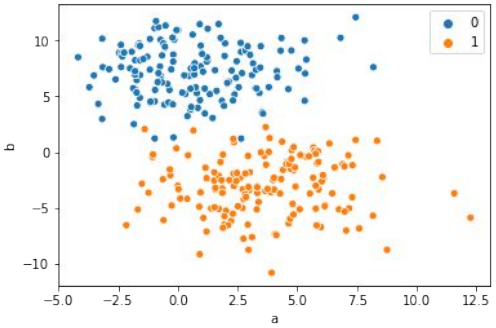
2. One-vs-Rest (OvR):
   - Użyj klasyfikatora SVM z kernela liniowego w podejściu One-vs-Rest do klasyfikacji trzech klas na podstawie tych samych danych Iris.
   - Podziel zbiór danych na zbiór treningowy i testowy.
   - Wytrenuj model SVM za pomocą podejścia OvR.
   - Dokonaj predykcji dla danych testowych.
   - Oceń dokładność modelu.

ML SVM, KNN, K-Means

Hiperparametry

Regularyzacja (C):

```
df = datasets.make_blobs(n_samples=300, random_state=67,
centers=2,cluster_std=2.5)
X = pd.DataFrame(df[0], columns=['a', 'b'])
y = df[1]
sns.scatterplot(X.a, X.b, hue = y);
```

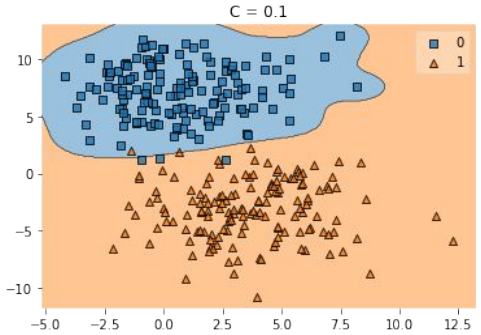


ML SVM, KNN, K-Means

Hiperparametry

Regularyzacja (C):

```
svm = SVC(kernel='rbf',gamma='auto',C=0.1)
svm.fit(X,y)
plt.title('C = 0.1')
plot_decision_regions(X.to_numpy(),y, clf=svm);
```

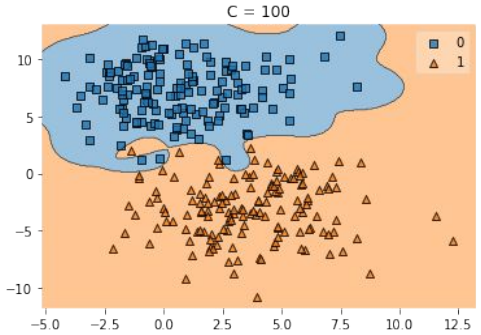


ML SVM, KNN, K-Means

Hiperparametry

Regularyzacja (C):

```
svm = SVC(kernel='rbf',gamma='auto',C=100)
svm.fit(X,y)
plt.title('C = 100')
plot_decision_regions(X.to_numpy(),y, clf=svm);
```

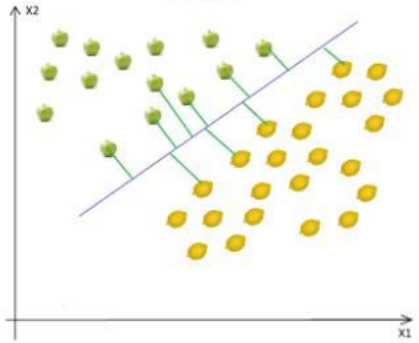


ML SVM, KNN, K-Means

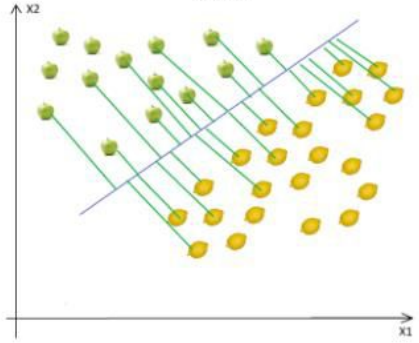
Hiperparametry

Gamma γ:

High Gamma



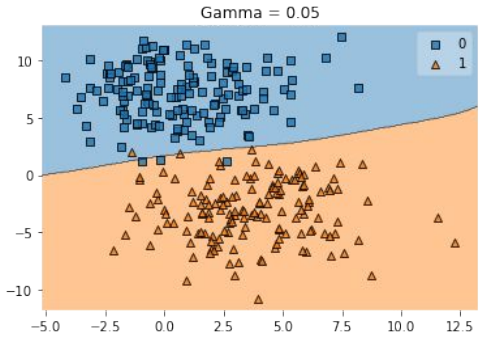
Low Gamma



ML SVM, KNN, K-Means

Hiperparametry

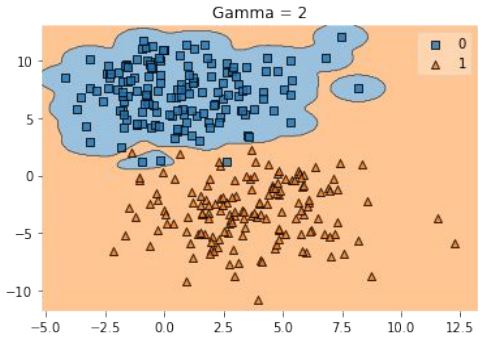
```
svm = SVC(kernel='rbf',gamma=0.05)
svm.fit(X,y)
plt.title('Gamma = 0.05')
plot_decision_regions(X.to_numpy(),y, clf=svm);
```



ML SVM, KNN, K-Means

Hiperparametry

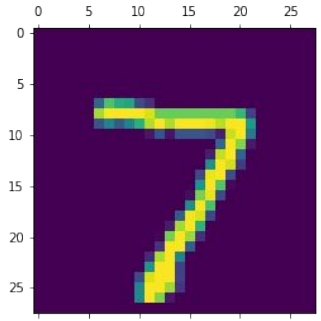
```
svm = SVC(kernel='rbf',gamma=2)
svm.fit(X,y)
plt.title('Gamma = 2')
plot_decision_regions(X.to_numpy(),y, clf=svm);
```



ML SVM, KNN, K-Means

Klasyfikacja obrazów

```
mnist = pd.read_csv('data/mnist_test.csv')
mnist = mnist.iloc[0:2000]

Y = mnist['label']
X = mnist.drop(columns = 'label').to_numpy()

plt.matshow(X[0].reshape(28,28));
```



ML SVM, KNN, K-Means

Klasyfikacja obrazów

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size =
0.3,random_state=50)
```

Regresja logistyczna:

```
lin = LogisticRegression().fit(X_train,Y_train)
pred = lin.predict(X_test)
print('Accuracy:',accuracy_score(Y_test,pred))
```

Accuracy: 0.81

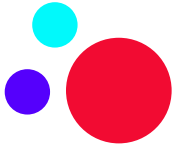
SVM:

```
svm = SVC(kernel='poly',gamma='auto',degree=3).fit(X_train,Y_train)
pred = svm.predict(X_test)
print('Accuracy:',accuracy_score(Y_test,pred))
```

Accuracy: 0.89

Zadanie 15.4 (instrukcja)

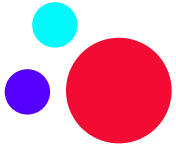
Zbiór mnist (pierwszych 2000 obserwacji) podziel na zbiory: treningowy walidacyjny, testowy w stosunku 60%, 20% 20%. Następnie spróbuj znaleźć najlepszy model SVM. Przetestuj różne kernele i ich hiperparametry (C, gamma) sprawdzając skuteczność (accuracy score) na zbiorze walidacyjnym. Wybierz najlepszy model i na końcu policz skuteczność tego modelu na zbiorze testowym.



ML SVM, KNN, K-Means

SVM - zalety

- Wszechstronny do funkcji jądra specyficznych dla użytkownika.
- Wydajna pamięć.
- Skuteczny w przypadkach, gdy liczba wymiarów jest większa niż liczba próbek (więcej zmiennych jak obserwacji).
- Rozwiązuje problemy liniowe i nieliniowe.



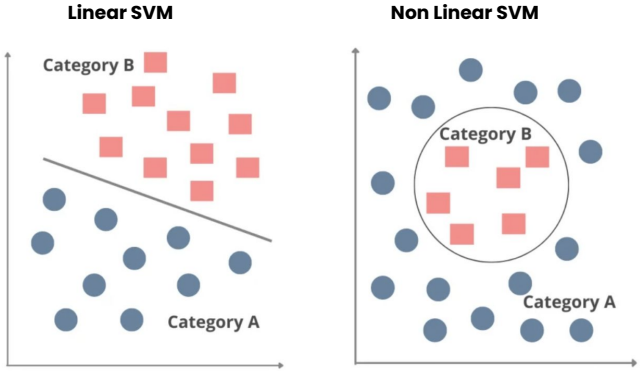
ML SVM, KNN, K-Means

SVM - wady

- Podatny na błędy i nadmierne dopasowanie w przypadku zaszumionych danych (np. nakładające się funkcje dla różnych etykiet) oraz outliery.
- Długi czas obliczeń w przypadku bardzo dużych zbiorów danych.
- Nie daje probabilistycznego wyjaśnienia wyników.

ML SVM, KNN, K-Means

SVM - podsumowanie

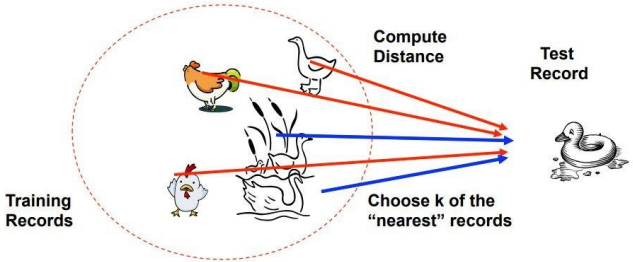


ML SVM, KNN, K-Means

KNN

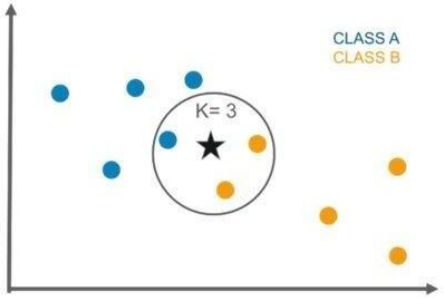
Intuicja tzw. Nearest Neighbor Classifiers (bardzo prosta):

-"If it walks like a duck, quacks like a duck, then it's probably a duck".



ML SVM, KNN, K-Means

KNN



ML SVM, KNN, K-Means

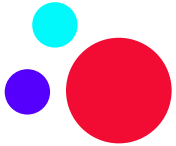
Przestrzeń metryczna

Niech X będzie niepustym zbiorem, np. $\mathbb{R}$ (oś liczbowa), $\mathbb{R}2$ (układ współrzędnych na płaszczyźnie), $\mathbb{R}3$(układ współrzędnych w przestrzeni).

Metrykę w zbiorze X nazywamy funkcję $d: X \times X \rightarrow [0,\infty)$ spełniającą dla dowolnych elementów a, b, c ze zbioru X następujące warunki:

- Identyczność: $d(a,b) = 0 \Leftrightarrow a = b$
- Symetria: $d(a,b) = d(b,a)$
- Nierówność trójkąta: $d(a,b) \leq d(a,c)+d(c,b)$

Mówimy wtedy, że para (X, d) jest przestrzenią metryczną.



ML SVM, KNN, K-Means

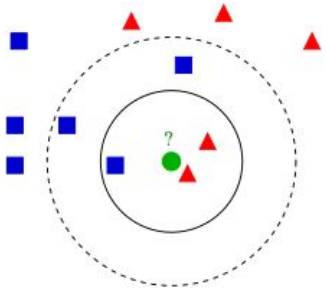
Pojęcie kuli

Kula w przestrzeni metrycznej (X,d) o środku w punkcie O i promieniu r to zbiór wszystkich elementów, których odległość od środka jest mniejsza od długości promienia. Gdy dodamy do niej zbiór wszystkich punktów odległych dokładnie o r, otrzymamy kulę domkniętą.

Przykładowo kulą w $\mathbb{R}$ może być odcinek, w $\mathbb{R}2$ koło, a w $\mathbb{R}3$ kula.

ML SVM, KNN, K-Means

Przykłady metryk



ML SVM, KNN, K-Means

Metryka euklidesowa

Metryka euklidesowa – jest to „naturalny” sposób mierzenia odległości w przestrzeniach.

$$d_e(x, y) = \sqrt{(y_1 - x_1)^2 + ... + (y_n - x_n)^2}$$

Kula w tej przestrzeni to wszystkie
takie punkty x spełniające nierówność:

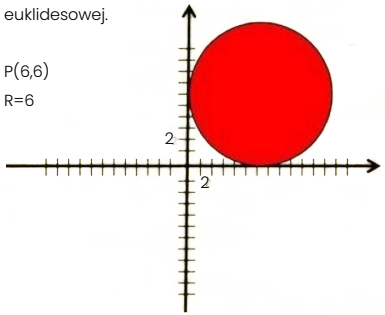
$$d_e(x, x_0) \leq r^2$$

W tym przypadku to:

$$(x - x_0)^2 + (y - y_0)^2 + ... \leq r^2$$

Kula w metryce
euklidesowej.

P(6,6)
R=6



ML SVM, KNN, K-Means

Metryka taksówkowa

Metryka taksówkowa, zwana również Manhattan lub miejską.

Między dwoma punktami poruszamy się tylko prosto wschód-zachód i północ-południe.

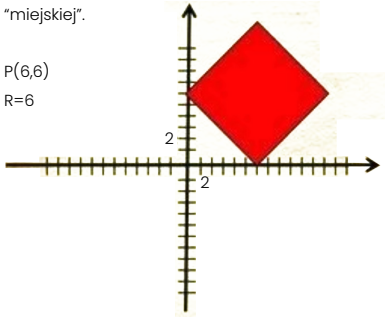
$$d_e(x,y) = |x_1 - y_1| + \ldots + |x_n - y_n|$$

$$|x_1 - x_2| + |y_1 - y_2| \leqslant r.$$

Kula w metryce "miejskiej".

P(6,6)

R=6



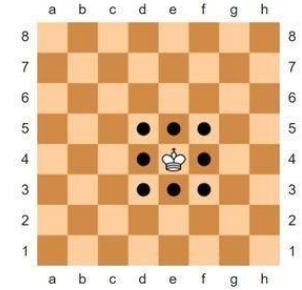
infoShareAcademy.com

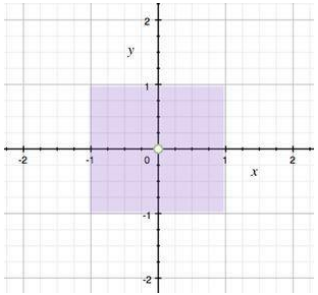
ML SVM, KNN, K-Means

Metryka maksimum

Metryka maksimum, zwana również Czebyszewa, nieskończoność, szachową.

$$d_e(x, y) = \max_{k=1,..n} |x_k - y_k|$$



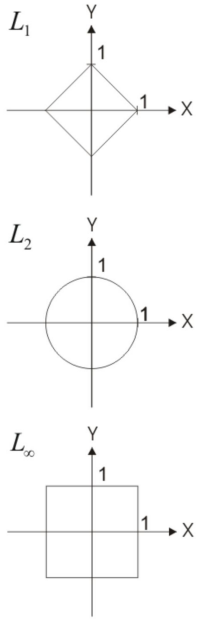


ML SVM, KNN, K-Means

Metryka Minkowskiego

Jest to uogólniona miara między punktami w przestrzeni euklidesowej. Nazywana również metryką L_m.

$$L_m(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^m \right)^{\frac{1}{m}}$$



ML SVM, KNN, K-Means

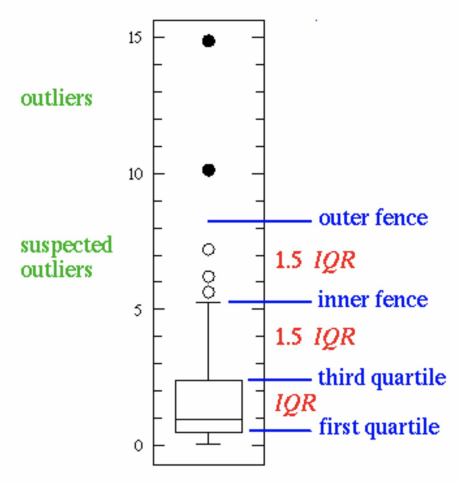
Inne metryki

- Odległość na powierzchni kuli (haversine) - używana do mierzenia odległości punktów na powierzchni Ziemii: https://en.wikipedia.org/wiki/Haversine_formula

- Odległość Levenshteina - opisuje ile przekształceń potrzeba by z jednego napisu otrzymać inny. Używana przy porównywaniu słów, tekstów: https://en.wikipedia.org/wiki/Levenshtein_distance

ML SVM, KNN, K-Means

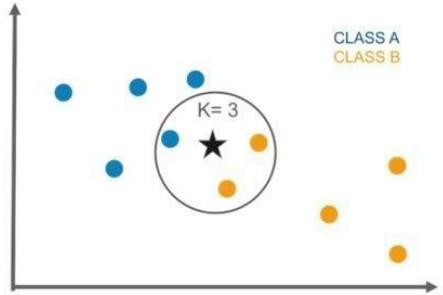
Wartości odstające



ML SVM, KNN, K-Means

K najbliższych sąsiadów

KNN jest metodą uczenia maszynowego z nadzorem.
Polega na wyszukaniu obserwacji będących w najbliższym sąsiedztwie w przestrzeni.
Może być używana zarówno do klasyfikacji jak i regresji.

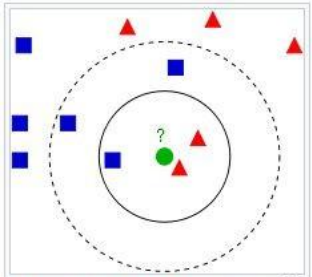


ML SVM, KNN, K-Means

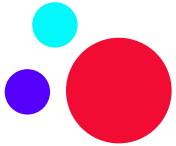
Algorytm KNN

1. Zapamiętujemy położenie wszystkich punktów w zbiorze uczącym.
2. Dla obserwacji, której dokonujemy predykcji wyliczamy odległości do wszystkich punktów ze zbioru uczącego.
3. Wybieramy K obserwacji znajdujących się najbliżej tej obserwacji.
4. W przypadku klasyfikacji predykcją będzie klasa najczęściej występująca
5. W przypadku regresji, będzie to średnia wartość zmiennej zależnej z K najbliższych obserwacji.

Możemy też opcjonalnie również użyć odległości jako wag.



W przypadku k=3 (mniejszy okrąg), zielona kropka zostanie zakwalifikowana do czerwonych trójkątów. W przypadku k=5 (większy okrąg) – do niebieskich kwadratów

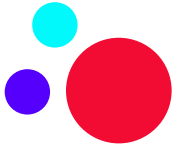


ML SVM, KNN, K-Means

Rekomendacje

Algorytm ten stosuje się nie tylko do regresji czy klasyfikacji - dzięki niemu możemy znaleźć najbliższe sąsiedztwo w przestrzeni wielowymiarowej.

Można zastosować go do różnego rodzaju rekomendacji.



ML SVM, KNN, K-Means

KNN - implementacja

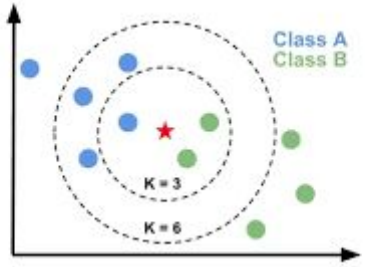
```
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

knn_classifier = KNeighborsClassifier(n_neighbors=3)

knn_classifier.fit(X_train, y_train)

y_pred = knn_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
```

Zadanie 15.5 (instrukcja)

Twoim zadaniem jest zastosowanie algorytmu K-Nearest Neighbors (KNN) do klasyfikacji cech win na podstawie dostępnych pomiarów. W tym przypadku użyj zbioru danych Wine, który zawiera trzy klasy win (klasyfikacja na podstawie trzech różnych odmian win).

ML SVM, KNN, K-Means

KNN - podsumowanie



ML SVM, KNN, K-Means

K-Means

https://vas3k.com/blog/machine_learning/

ML SVM, KNN, K-Means

K-Means

- Uczenie dzielimy na **z nadzorem** i **bez nadzoru:**

Każdy przykład ma określoną etykietkę (label, target):

$$p(y \mid \vec{x})$$

	sepal l	sepal w	petal l	petal w
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

	target
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

Przykłady są tylko zbiorami cech; algorytm "uczy się" rozkładu prawdopodobieństwa, który wygenerował zbiór:

$$p(\vec{x})$$

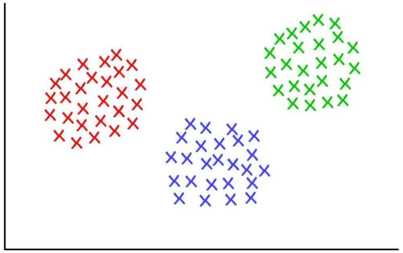
	sepal l	sepal w	petal l	petal w	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

ML SVM, KNN, K-Means

Klasteryzacja

Jest to proces dzielenia danych na grupy (klastry) bazując na wzorcach w danych.

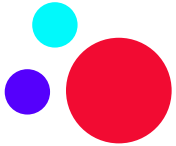
Podział odbywa się w taki sposób, by podobne obiekty należały do tej samej grupy, a różne obiekty do różnych.



ML SVM, KNN, K-Means

Zastosowanie

- Segmentacja użytkowników:
  - dedykowana komunikacja (marketing)
- Segmentacja grup klientów:
  - dedykowane produkty
- Segmentacja obrazów:
  - diagnostyka obrazowa w medycynie
- Grupowanie dokumentów:
  - automatyczny podział podobnych do siebie dokumentów
- Grupowanie produktów:
  - kategorie w sklepach internetowych
- Systemy rekomendacyjne:
  - podobni do Ciebie kupili
  - podobne produkty



ML SVM, KNN, K-Means

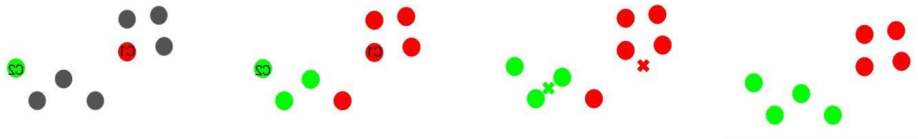
K-Means

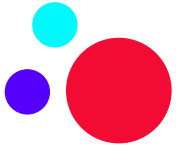
Algorytm k-średnich - często jest algorytmem pierwszego wyboru ze względu na jego prostotę i łatwość interpretacji.

ML SVM, KNN, K-Means

Algorytm K-Means

1. Wybierz liczbę klastrów K.
2. Wybierz k losowych punktów z danych jako tzw. centroidy.
3. Przypisz każdą obserwację do najbliższego centroidu (klastra).
4. Przelicz na nowo centroidy klastrów (środki).
5. Powtórz kroki 3 i 4 dla nowych centroidów do czasu aż:
   - centroidy dla nowoutworzonych klastrów się nie zmieniają,
   - punkty pozostają w tym samym klastrze,
   - osiągnięto maksymalną liczbę iteracji.





ML SVM, KNN, K-Means

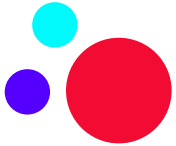
K-Means w sklearn

Trenowanie modelu:

```
from sklearn.cluster import KMeans
model = KMeans(**parametry)
model.fit(dane_treningowe)
```

Aplikacja modelu:

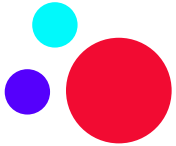
```
model.predict(nowy_data_point)
```



ML SVM, KNN, K-Means

K-Means w sklearn

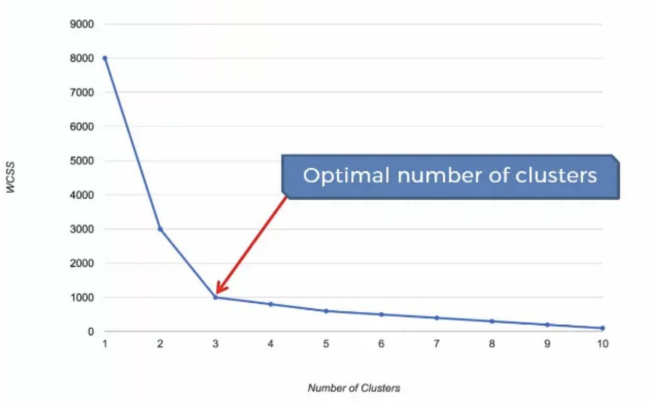
- n_clusters
- n_init
- Init
- tol
- max_iter
- algorithm



ML SVM, KNN, K-Means

Optymalna liczba klastrów

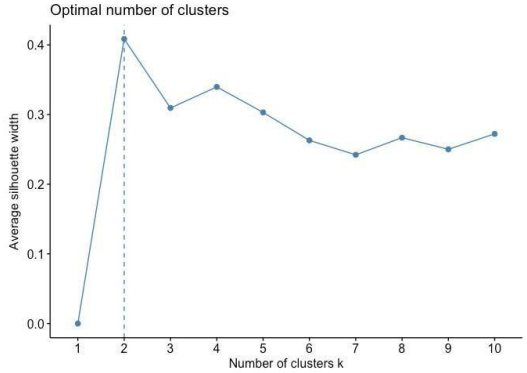
Elbow Method:



ML SVM, KNN, K-Means

Optymalna liczba klastrów

Silhouette score:



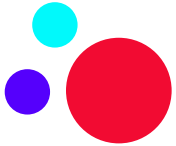
$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j)$$

średnia odległość punktu i wewnątrz klastra

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

średnia odległość punktu i do innego klastra

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1$$



ML SVM, KNN, K-Means

Wykres Silhouette


```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score


X, y = make_blobs(n_samples=500,
                  n_features=2,
                  centers=4,
                  cluster_std=1,
                  center_box=(-10.0, 10.0),
                  shuffle=True,
                  random_state=1)
```

ML SVM, KNN, K-Means

Wykres Silhouette

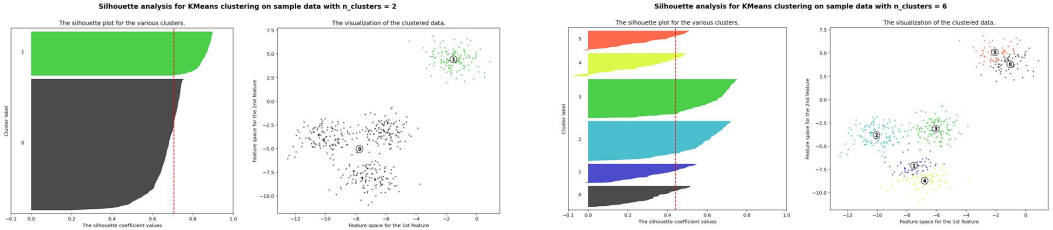
For n_clusters = 2 The average silhouette_score is : 0.7049787496083262

For n_clusters = 3 The average silhouette_score is : 0.5882004012129721

For n_clusters = 4 The average silhouette_score is : 0.6505186632729437

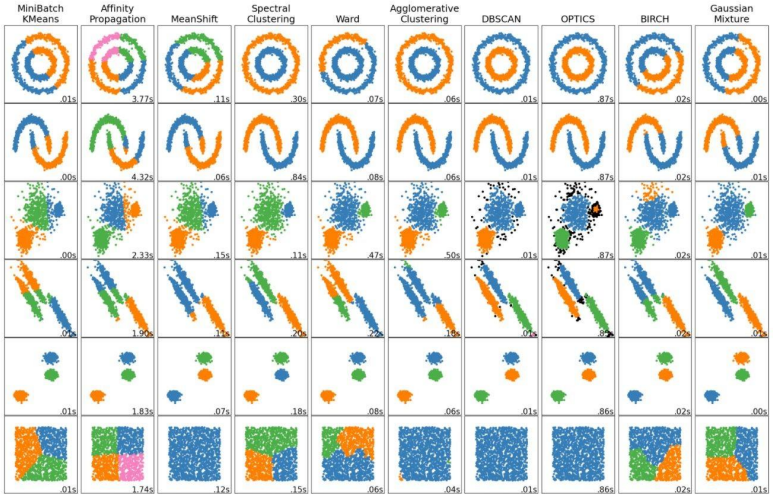
For n_clusters = 5 The average silhouette_score is : 0.5745566973301872

For n_clusters = 6 The average silhouette_score is : 0.4387644975296138



ML SVM, KNN, K-Means

Inne metody clusteringu

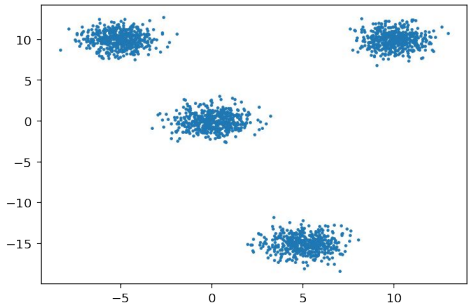


A comparison of the clustering algorithms in scikit-learn

ML SVM, KNN, K-Means

Implementacja

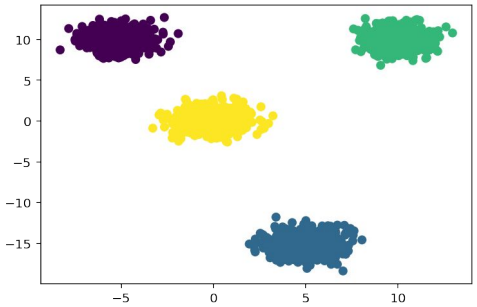
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

data = pd.read_csv("data/synthetic-1.csv").values
```



ML SVM, KNN, K-Means

Implementacja

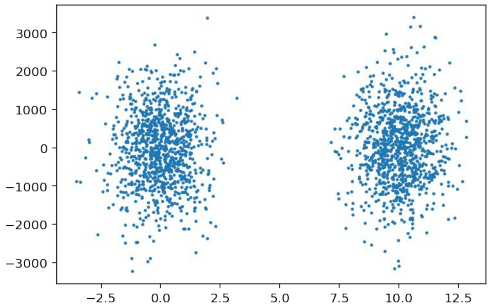
```
from sklearn.cluster import KMeans

ks = KMeans(4, init="random", n_init=10).fit_predict(data)

ks

        array([0, 3, 1, ..., 3, 0, 2])

plt.scatter(data[:,0], data[:,1], c=ks)
```

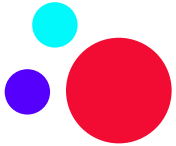


ML SVM, KNN, K-Means

Implementacja

```
data = pd.read_csv("data/scales.csv").values
plt.scatter(data[:, 0], data[:, 1], 2)
```





ML SVM, KNN, K-Means

Implementacja

```
ms = KMeans(2, init="k-means++", n_init=10)
pred = ms.fit(data).predict(data)
```

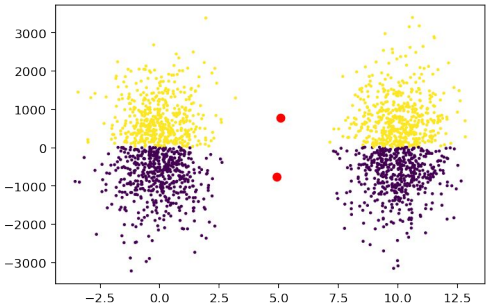
```
ms.cluster_centers_
```

```
array([[   4.90388469, -752.72246929],
       [   5.08170737,  788.29127696]])
```

ML SVM, KNN, K-Means

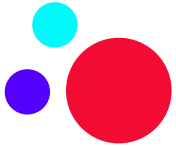
Implementacja

```
plt.scatter(data[:, 0], data[:, 1], 2, c=pred)
plt.scatter(ms.cluster_centers_[:,0], ms.cluster_centers_[:,1], c="red")
```



Zadanie 15.6 (instrukcja)

Dla zbioru danych Iris z biblioteki sklearn. Zastosuj algorytm K-Means do klastrowania tych danych w celu zidentyfikowania naturalnych grup.



ML SVM, KNN, K-Means

Inercja

Inercja (Inertia).

Suma odległości wszystkich punktów w klastrze od centroidu (środka ciężkości własnego klastra - najbliższego) - odległość wewnątrz klastra.

Chcemy minimalizować.

ML SVM, KNN, K-Means

Implementacja

```
ms.inertia_
```

```
684710802.1321297
```

```
from sklearn.preprocessing import scale
```

```
scaled_data = scale(data)
```

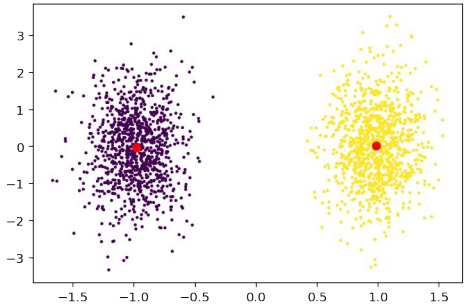
```
scaled_data
```

```
array([[ 0.88075426, -0.1969764 ],
       [-1.08605514,  2.19940887],
       [-0.94236542, -0.19915185],
       ...,
       [-0.8743019 , -0.11168427],
       [ 1.25531152, -0.15825949],
       [ 0.97803738, -3.19015715]])
```

ML SVM, KNN, K-Means

Implementacja

```
ms2 = KMeans(2, init="k-means++", n_init=10)
pred2 = ms2.fit(scaled_data).predict(scaled_data)
plt.scatter(scaled_data[:, 0], scaled_data[:, 1], 2, c=pred2)
plt.scatter(ms2.cluster_centers_[:,0], ms2.cluster_centers_[:,1], c="red")
```



ms2.inertia_

2073.7519150180733

ML SVM, KNN, K-Means

Implementacja

Wybór k:

```
data = pd.read_csv("data/synthetic-1.csv").values

plt.scatter(data[:, 0], data[:, 1], 2)
```

ML SVM, KNN, K-Means

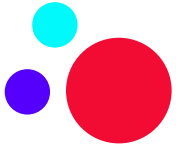
Implementacja

```
k = 4
ms = KMeans(k).fit(data)

ms.score(data)
```

3953.631021978067

```
ms.inertia_
```

3953.6310219780685



ML SVM, KNN, K-Means

Implementacja

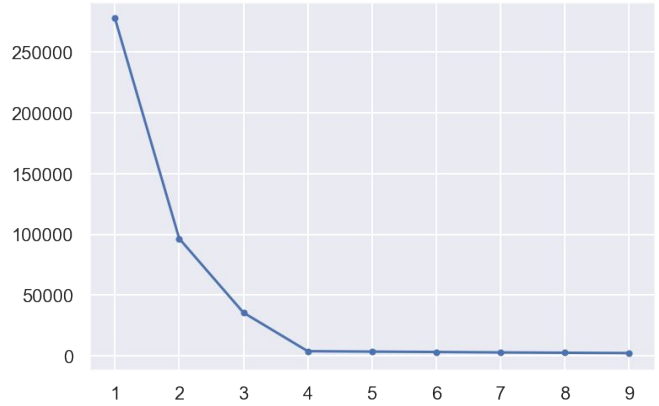

```
[KMeans(k).fit(data).inertia_ for k in range(1, 10)]
```

```
[277695.22957089165,
 96595.36113577684,
 35653.3972019451,
 3953.6310219780685,
 3587.3712225985755,
 3238.3389228632554,
 2932.6509570852913,
 2609.281854683741,
 2400.4554721882973]
```

ML SVM, KNN, K-Means

Implementacja

```
scores = [- KMeans(k).fit(data).score(data) for k in range(1, 10)]
plt.plot(range(1, 10), scores, markersize=3, marker="o")
plt.show()
```



Zadanie 15.7 (instrukcja)

Dla zbioru danych load_digits() znajdź optymalną liczbę klastrów za pomocą algorytmu K-Means i metody „łokcia”.

ML SVM, KNN, K-Means

Podsumowanie