# Support Vector Machines

Igor Wojnicki

March 24, 2024

# Plan prezentacji

Klasyfikacja

Poszukiwanie hiperparametrów

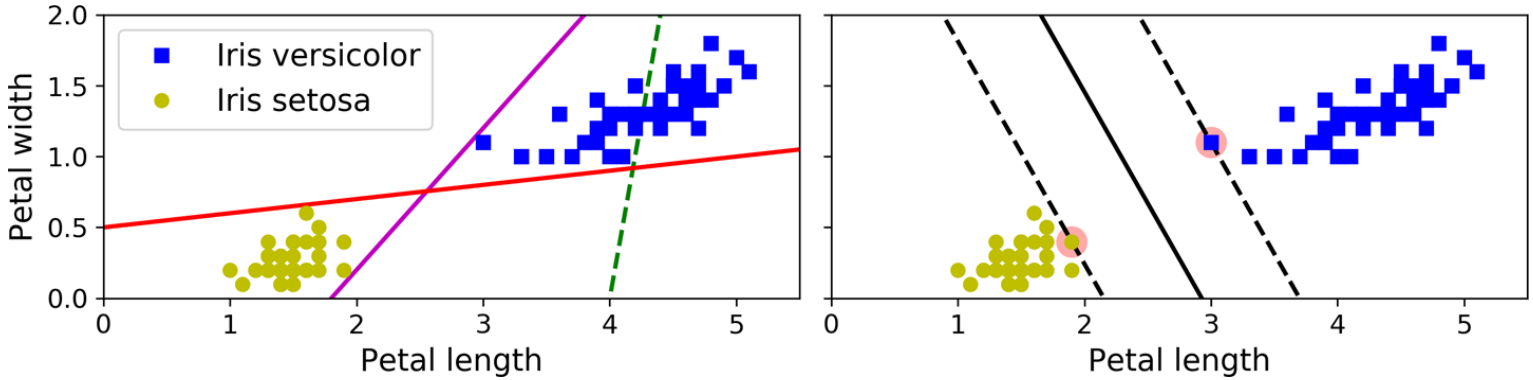
Regresja

# Jak działa SVM?

Maszyna Wektorów Nośnych

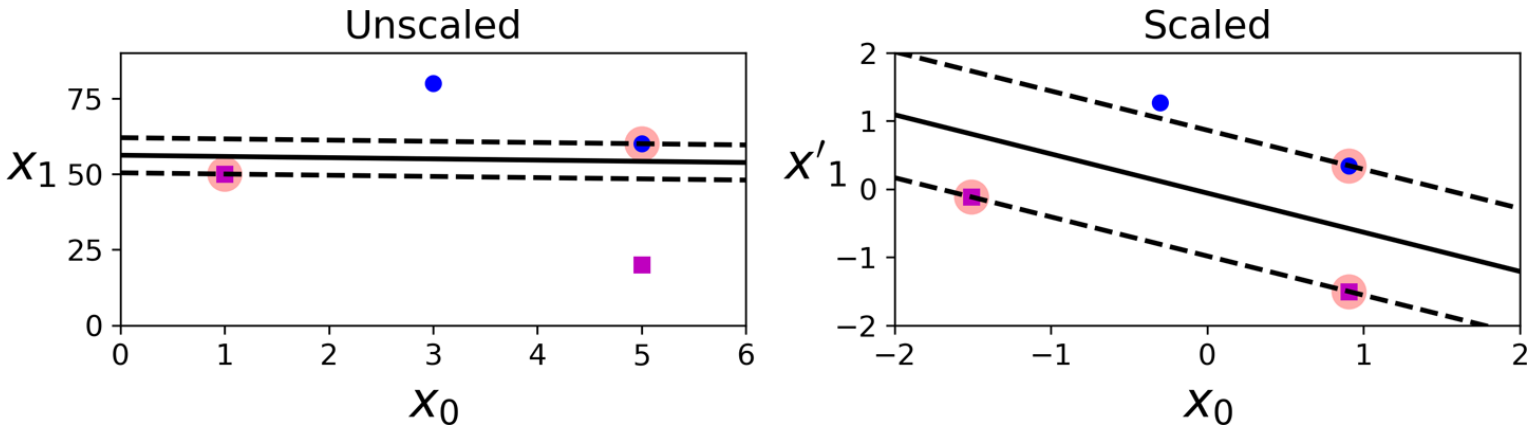
https://www.youtube.com/watch?v=_PwhiWxHK8o

https://www.youtube.com/watch?v=eHsErlPJWUU




- liniowa separacja (. . . i nie tylko)
- liniowe klasyfikatory
- margines klasyfikacji: utworzenie jak najszerszej ulicy pomiędzy klasami
- dodanie nowych instancji poza "ulicą" nie wpływa na klasyfikacje
- granica klas jest oparta (supported) o instancje leżące na granicy "ulicy"
- instancje te nazywane są *support vectors* (wektory nośne)

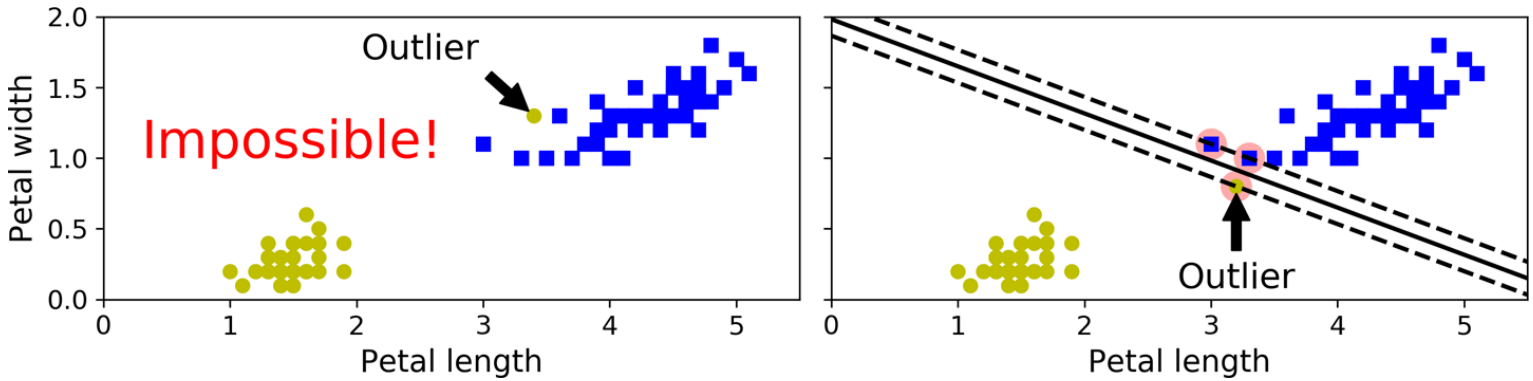
# Uwaga na skalę cech




- Klasyfikacja na oryginalnych wartościach cech może być marna...

# Problemy z klasyfikacją SVM, elementy/wartości odstające, outliers

- Hard Margin Classification
- Gdzie powinna być droga?
- Jaka szeroka?

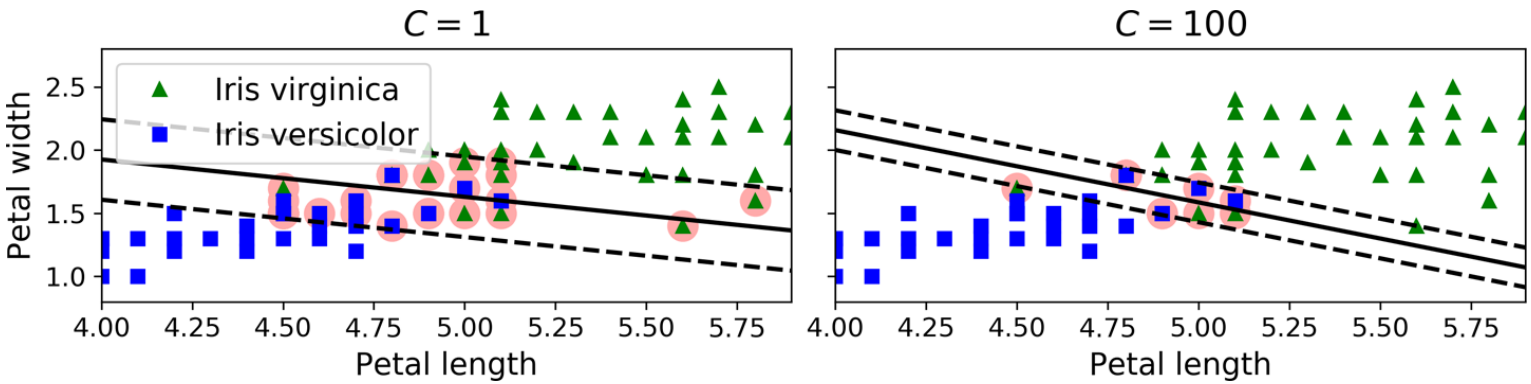



Uwaga:

- Działa tylko jeżeli możemy liniowo rozdzielić klasy.
- Nie radzi sobie z wartościami odstającymi (outliers).

# Soft Margin Classification

- Równowaga:
  - maksymalna szerokością "drogi"
  - ograniczenie naruszeń marginesu klasyfikacji




- hiperparametr $C$ (domyślnie równy 1.0); współczynnik regularyzacji
- jeżeli *overfitting* należy zmniejszyć $C$

# Zbiór danych: Irysy

```
1  from sklearn import datasets
2  iris = datasets.load_iris()
3  print(iris.keys(),"\n" ,
4        iris.DESCR)
5
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'featu
 .. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
```

# SVM, uczenie

```
1  import numpy as np
2  from sklearn.pipeline import Pipeline
3  from sklearn.preprocessing import StandardScaler
4  from sklearn.svm import LinearSVC
5  iris = datasets.load_iris()
6  X = iris["data"][:, (2, 3)]  # długość i szerokość płatka
7  y = (iris["target"] == 2).astype(np.int8)  # Iris virginica
8  svm_clf = Pipeline([
9          ("scaler", StandardScaler()),
10         ("linear_svc", LinearSVC(C=1,
11                                  loss="hinge",
12                                  random_state=42)),
13     ])
14 svm_clf.fit(X, y)
```

- automatyczne skalowanie – uwaga: zastosowanie potoku (Pipeline)
- funkcja kary: ujemna odległość od granicy

# SVM, klasyfikacja

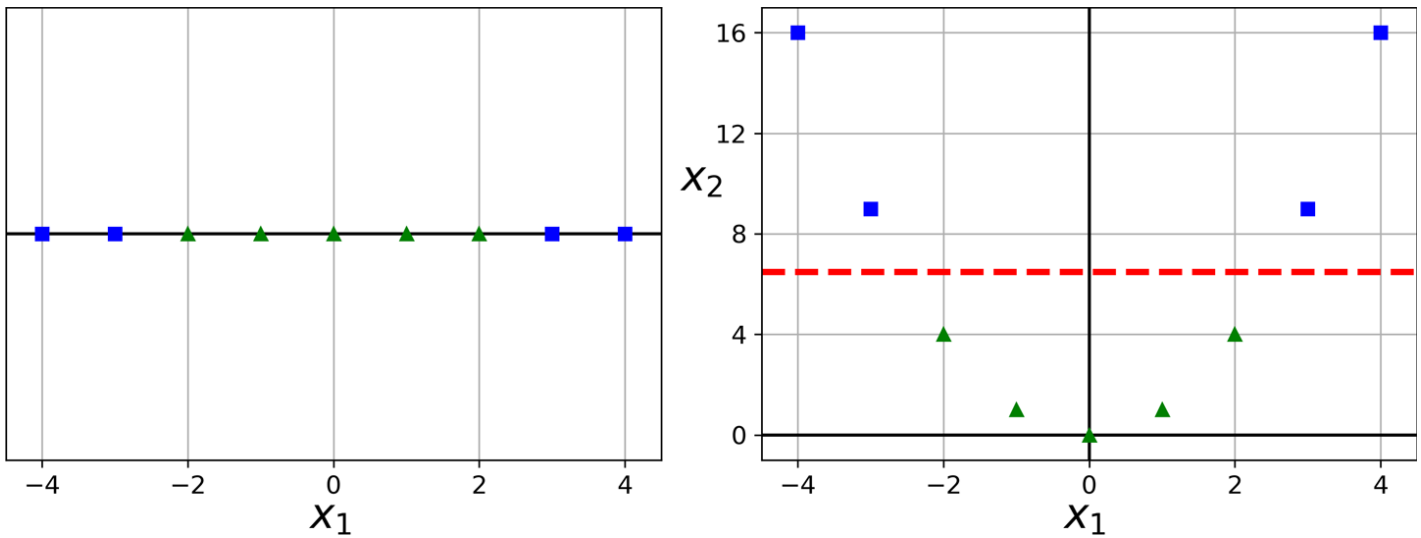
```
1  print(svm_clf.predict([[5.5, 1.7],
2                         [4.5, 1.7]]))
```

```
[1 0]
```

- kwiatek o długości płatka 5.5 cm i szerokości 1.7 cm to *Iris virginica*
- kwiatek o długości płatka 4.5 cm i szerokości 1.7 cm to nie jest *Iris virginica*
- brak informacji o prawdopodobieństwie, w porównaniu z Regresją Logistyczną.

# SVM, klasyfikacja nieliniowa

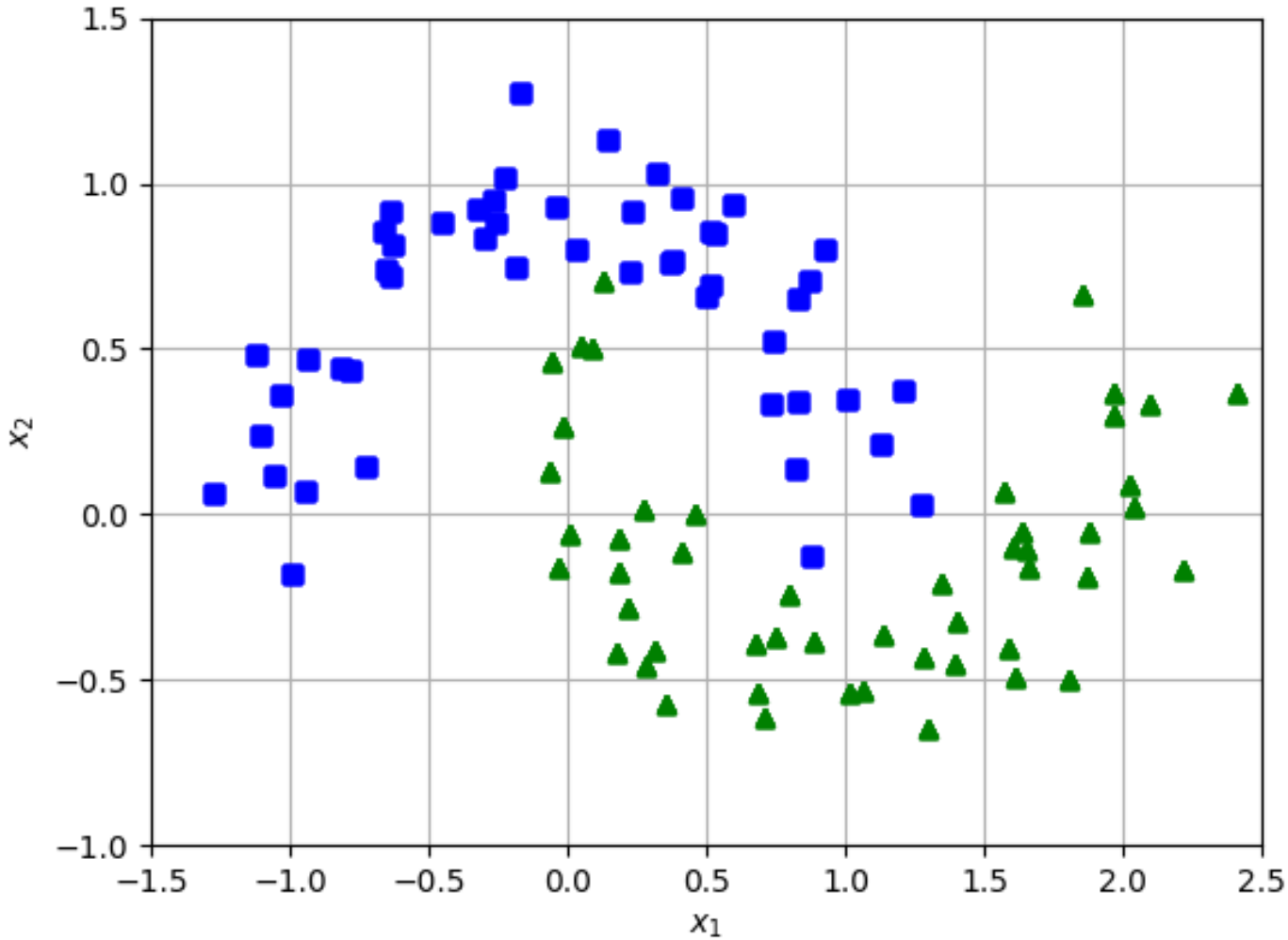
- Co jeżeli instancji nie można odseparować hiperpłaszczyzną?
- Dodać więcej cech.



# SVM, nieliniowa, dane

```
1  from sklearn.datasets import make_moons
2  import matplotlib.pyplot as plt
3
4  X, y = make_moons(n_samples=100, noise=0.15,
5                    random_state=42)
6
7  def plot_dataset(X, y, axes, file):
8      plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
9      plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
10     plt.axis(axes)
11     plt.grid(True, which='both')
12     plt.xlabel("$x_1$")
13     plt.ylabel("$x_2$")
14     plt.savefig(f)
15
16 f = "moons_dataset.png"
17 plot_dataset(X, y, [-1.5, 2.5, -1, 1.5], f)
18 print(f)
```

# Moon dataset



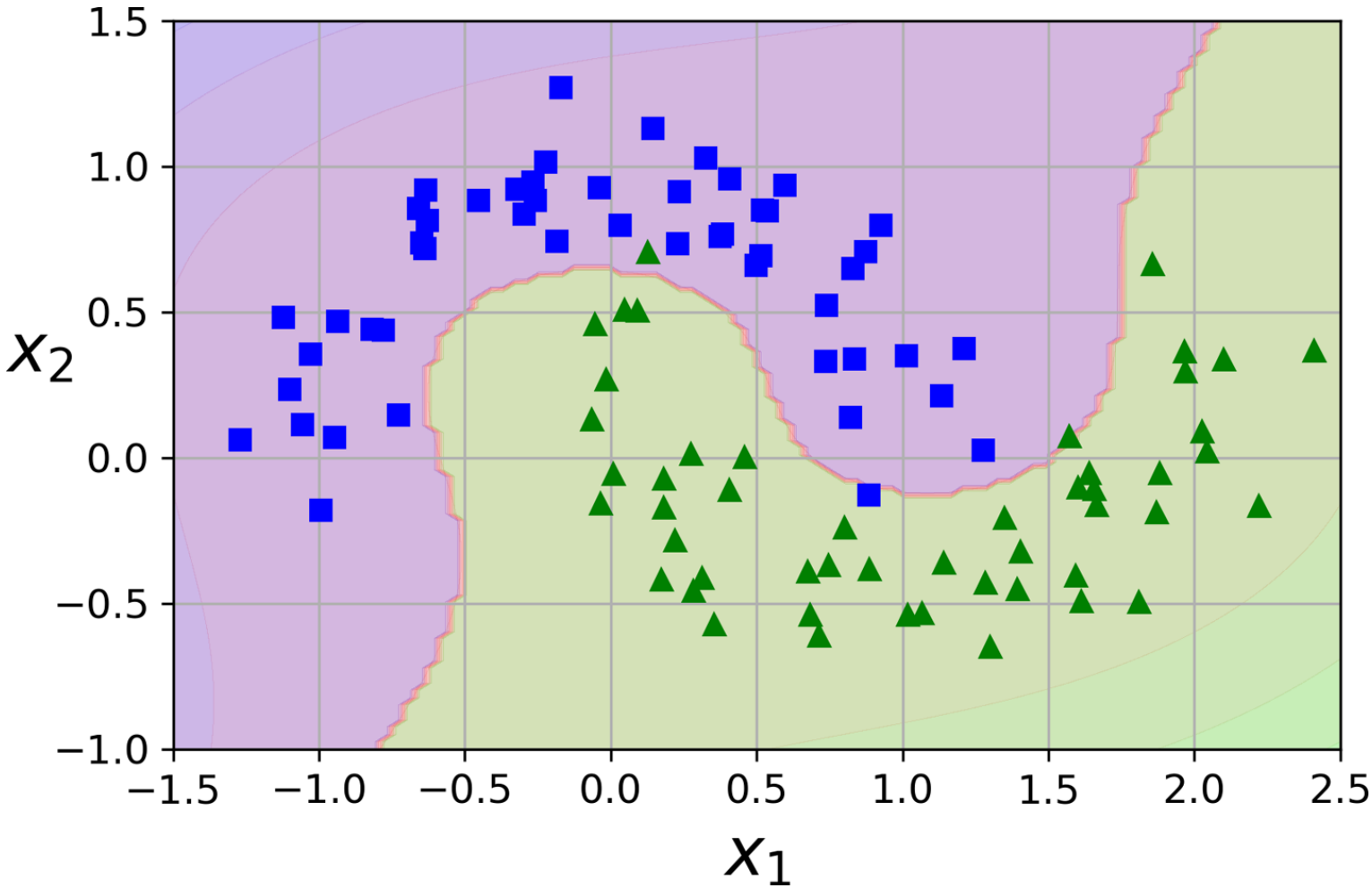
# SVM, nieliniowa, uczenie

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3,
                                             include_bias=False)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge",
                              max_iter=3000, # zbieżność
                              random_state=42))
    ])

polynomial_svm_clf.fit(X, y)
```

# SVM, nielioniowa, klasyfikacja

```
1  print(polynomial_svm_clf.predict([[0.5, 0],
2                                    [1, 0]]))
```

```
[1 0]
```



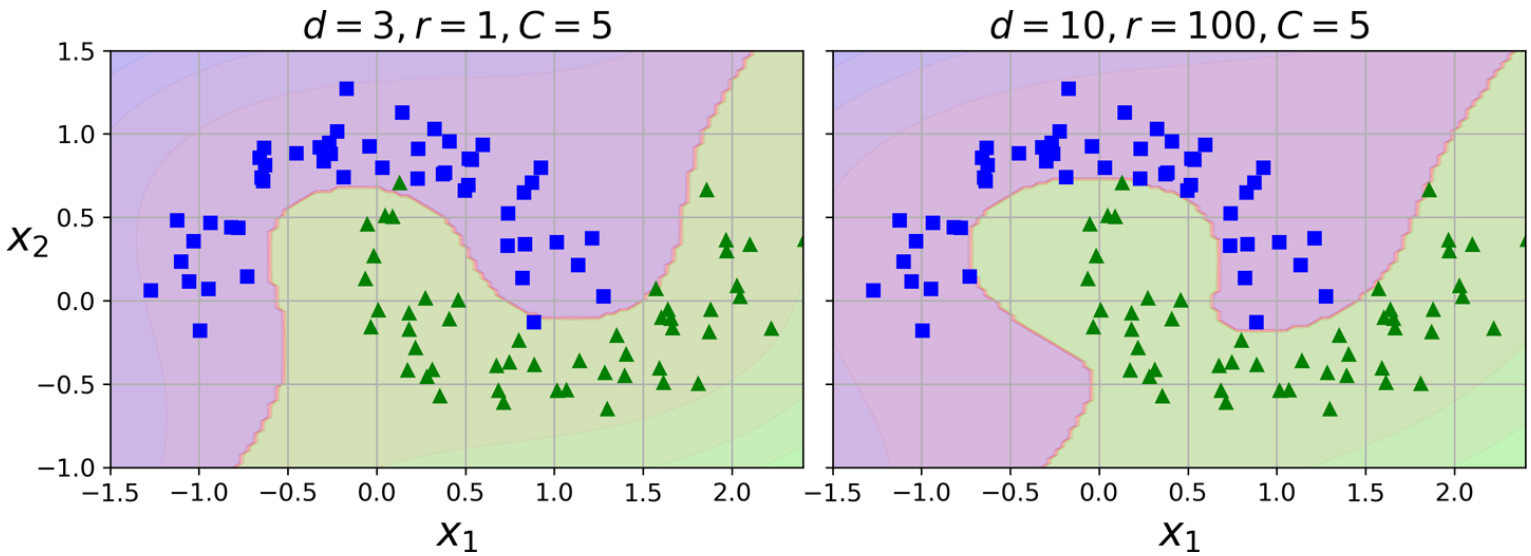
# SVM, nieliniowa klasyfikacja, raz jeszcze

```
1  from sklearn.svm import SVC
2
3  poly_kernel_svm_clf = Pipeline([
4      ("scaler", StandardScaler()),
5      ("svm_clf", SVC(kernel="poly", degree=3,
6                      coef0=1, C=5))])
7  poly_kernel_svm_clf.fit(X, y)
8  print(poly_kernel_svm_clf.predict([[0.5, 0],
9                                     [1, 0]]))
```

```
[1 0]
```

- wydajniej dla wyższych stopni niż dodawanie cech, wolniejsze uczenie niż *LinearSVC* przy dużej liczbie instancji,
- są również inne *kernels*, inna implementacja niż *LinearSVC*
- wielomian wyższego stopnia $\rightarrow$ overfitting
- wielomian niższego stopnia $\rightarrow$ underfitting
- hiperparametr *coef0* – im większy tym większy wpływ wielomianów wysokiego stopnia

# SVM, nieliniowa klasyfikacja, raz jeszcze




r = *coef0*, im większy tym większy wpływ wielomianów wysokiego stopnia

# Złożonośc obliczeniowa

- $m$ – liczba instancji, $n$ – liczba cech
- LinearSVC, $O(m * n)$,
- SVC, $O(m^2 * n)$ – $O(m^3 * n)$

# Plan prezentacji

Klasyfikacja

Poszukiwanie hiperparametrów

Regresja

# Przegląd zupełny

GridSearchCV, tutorial, konwencja nazw parametrów w potoku (Pipeline): `nazwa__parametr`.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    "svm_clf__degree": range(1, 6),
    "svm_clf__C" : [0.01, 0.1, 1, 10, 100, 1000]
}
search = GridSearchCV(poly_kernel_svm_clf,
                      param_grid,
                      scoring="accuracy",
                      n_jobs=-1)
search.fit(X, y)
print(f"Cross-validated accuracy = {search.best_score_}")
print(search.best_params_)
```

```
Cross-validated accuracy = 0.9733333333333334
{'svm_clf__C': 1000, 'svm_clf__degree': 4}
```

# Przegląd losowy

RandomizedSearchCV

```
1  from sklearn.model_selection import RandomizedSearchCV
2  param_grid = {
3      "svm_clf__degree": range(1, 6),
4      "svm_clf__C" : [0.01, 0.1, 1, 10, 100, 1000]
5  }
6  search = RandomizedSearchCV(poly_kernel_svm_clf,
7                              param_grid,
8                              scoring="accuracy",
9                              n_iter=10, n_jobs=-1)
10 search.fit(X, y)
11 print(f"Cross-validated score = {search.best_score_}")
12 print(search.best_params_)
```

```
Cross-validated score = 0.9666666666666668
{'svm_clf__degree': 5, 'svm_clf__C': 1000}
```

`n_iter` ile zestawów parametrów ma być wybieranych losowo (samplowanych)

# Plan prezentacji

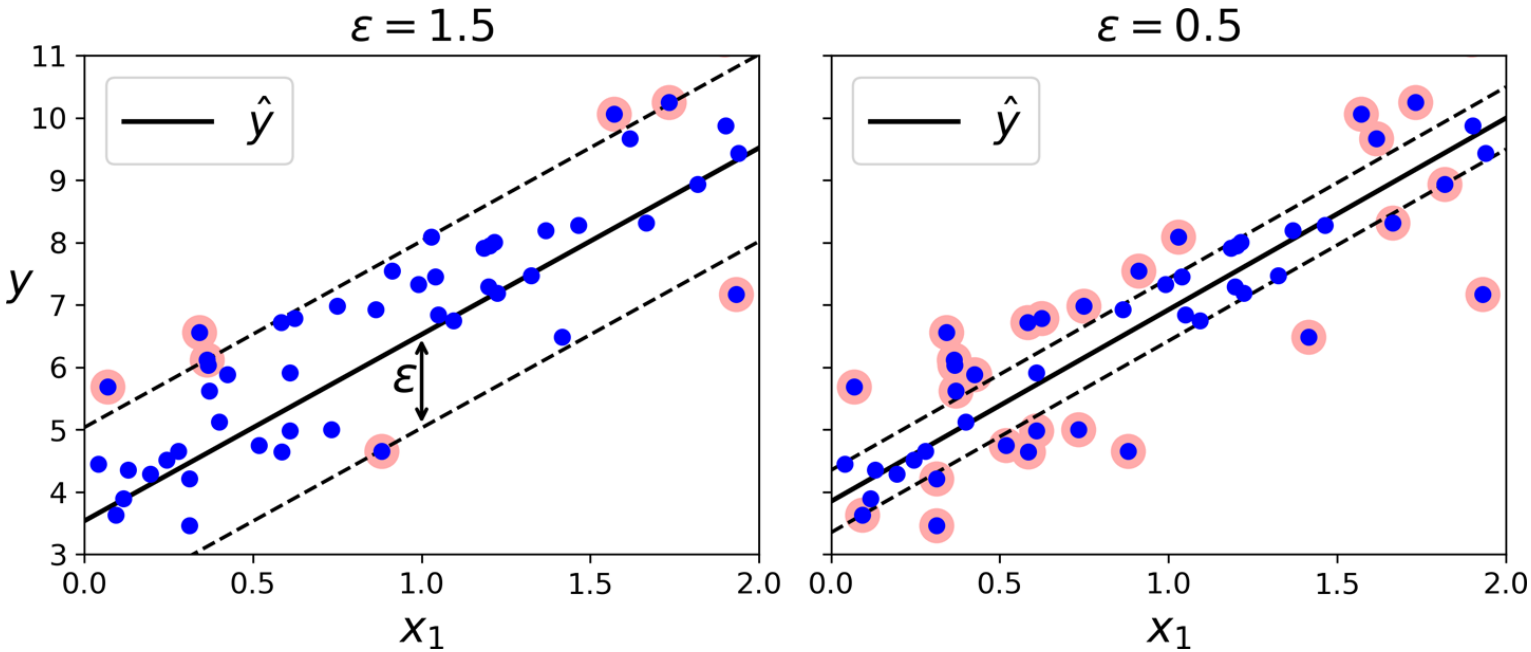
Klasyfikacja

Poszukiwanie hiperparametrów

Regresja

# SVM, regresja liniowa

- Dopasowanie jak największej liczby instancji w "ulicy" minimalizując naruszenie marginesów $\epsilon$.



# SVM, regresja liniowa, dane

```
np.random.seed(42)
m = 50
X = 2 * np.random.rand(m, 1)
y = (4 + 3 * X + np.random.randn(m, 1)).ravel()
```

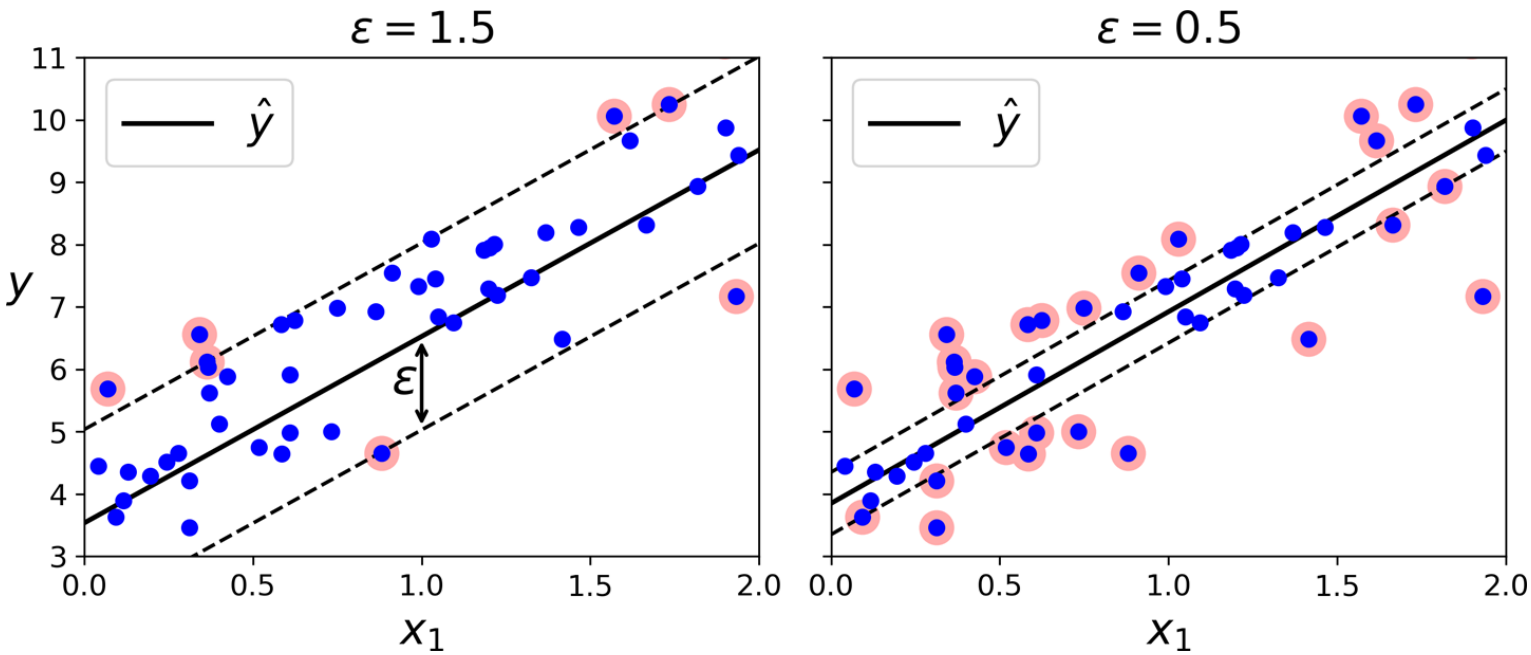
# SVM, regresja liniowa, uczenie

```
1  from sklearn.svm import LinearSVR
2
3  svm_reg = LinearSVR(epsilon=1.5, random_state=42)
4  svm_reg.fit(X, y)
```

# SVM, regresja liniowa, predykcja

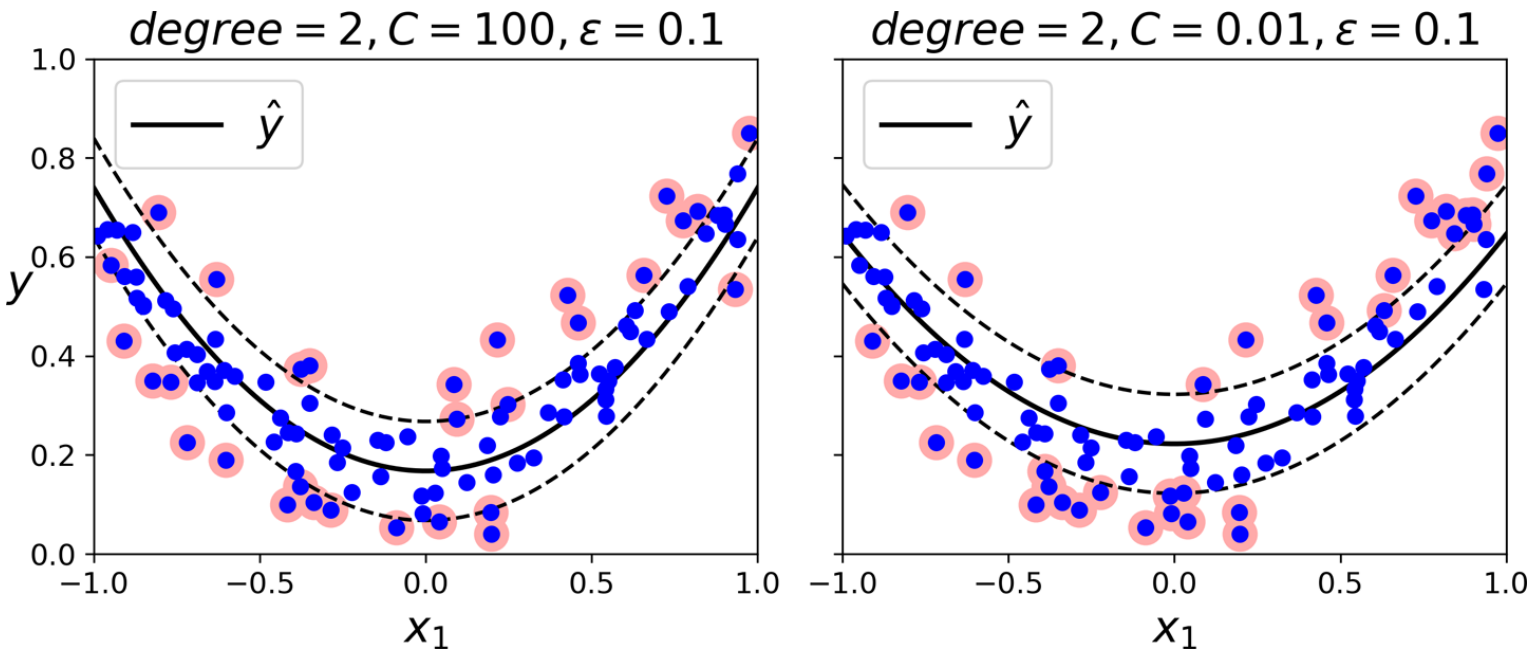
```
1  print(svm_reg.predict([[1],[2]]))
```

```
[6.52640746 9.51919121]
```



# SVM, regresja nielinowa, dane

```
np.random.seed(42)
m = 100
X = 2 * np.random.rand(m, 1) - 1
y = (0.2 + 0.1 * X + 0.5 * X**2 +
     np.random.randn(m, 1)/10).ravel()
```



# SVM, regresja nielinowa, uczenie

```
1  from sklearn.svm import SVR
2
3  svm_poly_reg = SVR(kernel="poly", degree=2,
4                     C=100, epsilon=0.1)
5  svm_poly_reg.fit(X, y)
```

# SVM, regresja nielinowa, predykcja

```
1  print(svm_poly_reg.predict([[0],[-1]]))
```

```
[0.16764293 0.73995101]
```