Regresja

Igor Wojnicki

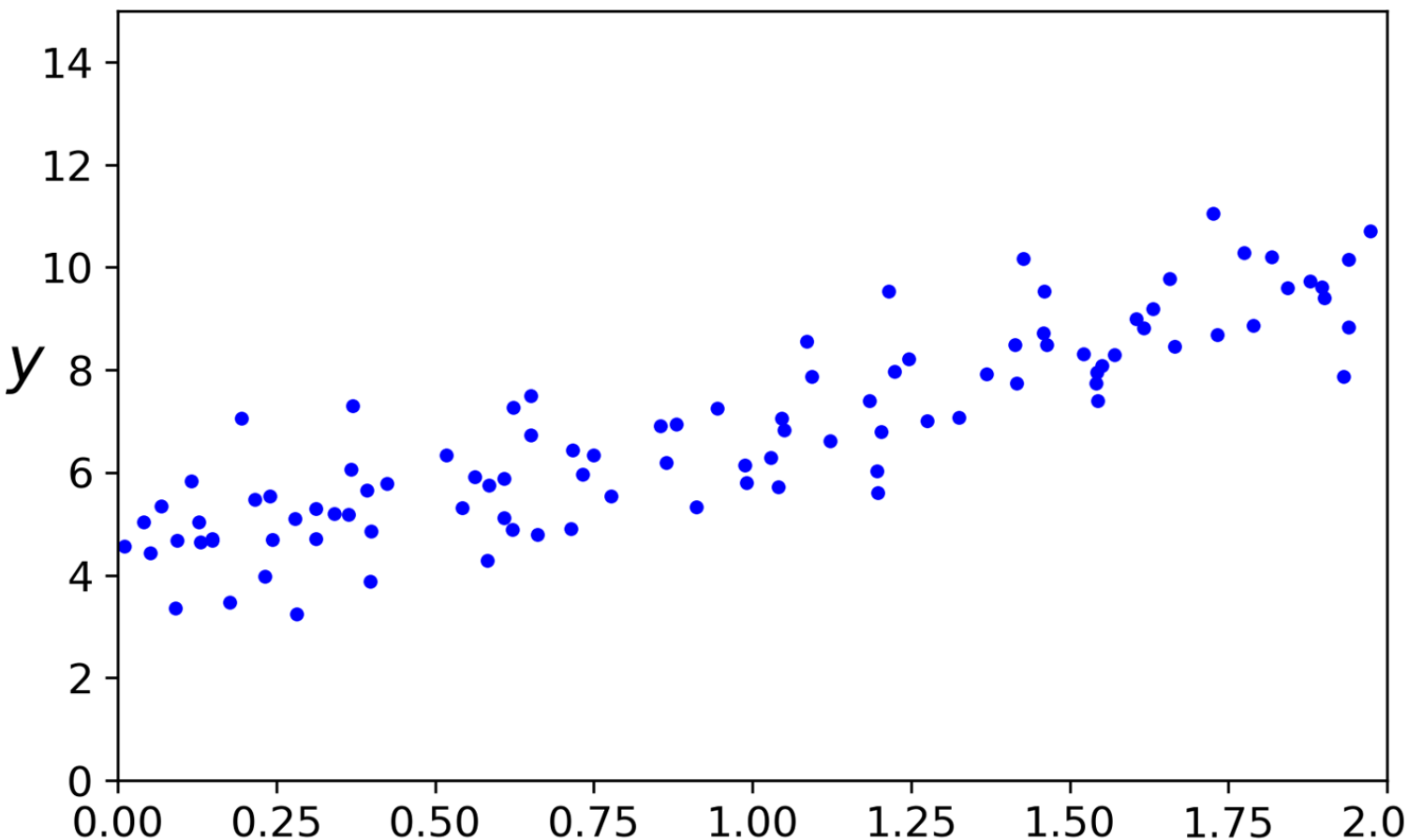
March 17, 2024

Plan prezentacji

Regresja liniowa, Dane

Regresja – trochę nieszczęśliwa nazwa: regresja do średniej (obserwacja biologiczna).

```
1  import numpy as np
2  X = 2 * np.random.rand(100, 1) # wektor pionowy
3  y = 4 + 3 * X + np.random.randn(100, 1)
4                                  # ^- liczba losowa, szum Gaussa
```



Regresja liniowa, Linear Regression, równania

$$\hat{y} = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \ldots + \Theta_n x_n$$

$$\hat{y} = h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{\Theta} \cdot \boldsymbol{x}$$

Rozwiązanie polega na znalezieniu $\boldsymbol{\Theta}$, który minimalizuje funkcję celu (kosztu) Root Mean Square Error (RMSE) dla m instancji:

$$RMSE(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) = \sqrt{\frac{1}{m_i} \sum_{i=1}^{m} (\boldsymbol{\Theta}^T \boldsymbol{x}^{(i)} - y^{(i)})^2}$$

albo MSE (prościej):

$$MSE(\boldsymbol{X}, h_{\boldsymbol{\Theta}}) = \frac{1}{m_i} \sum_{i=1}^{m} (\boldsymbol{\Theta}^T \boldsymbol{x}^{(i)} - y^{(i)})^2$$

- Ktoś rozumie?

Regresja liniowa, równanie normalne

- Równanie: $y = 4 + 3x$
- Model regresji liniowej (h – hipoteza):

$$\hat{y} = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \ldots + \Theta_n x_n$$

$$\hat{y} = h_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{\Theta} \cdot \boldsymbol{x}$$

- Równanie normalne:

$$\hat{\boldsymbol{\Theta}} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

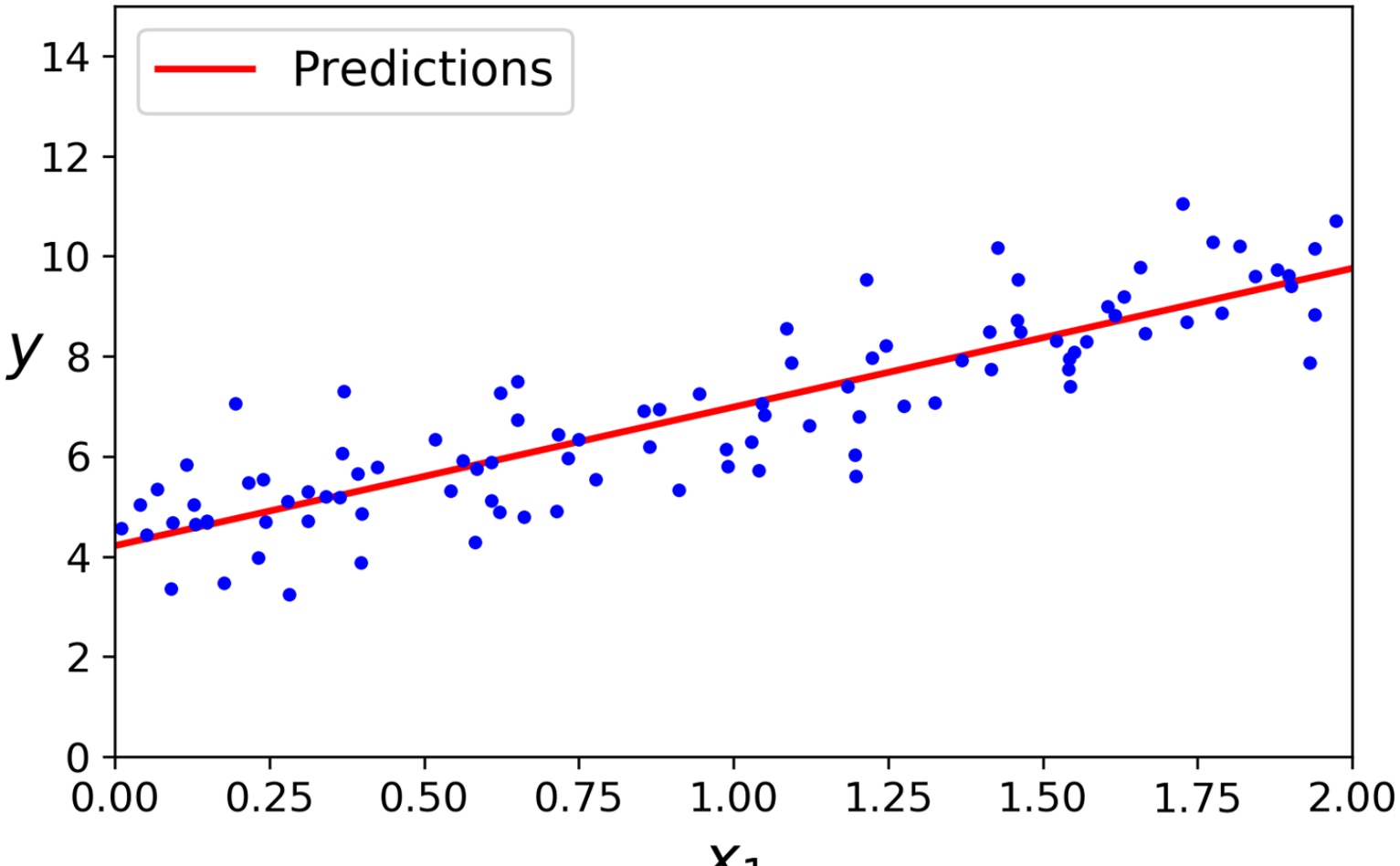
```
1  # dodaj stały komponent X0=1 do każdej instancji/próbki
2  X_b = np.c_[np.ones((100, 1)), X]
3  theta_best = np.linalg.inv(X_b.T.dot(X_b)) \
4                            .dot(X_b.T).dot(y)
5  print(theta_best)
```

```
[[3.9872083 ]
 [2.86708435]]
```

Regresja liniowa, predykcja: $y = 4 + 3x$

```
1  X_new = np.array([[0], [2]])
2  X_new_b = np.c_[np.ones((2, 1)), X_new] # stały komponent d
3  print(y_predict := X_new_b.dot(theta_best))
```

```
[[3.9872083 ]
 [9.72137701]]
```



Regresja, prościej

```
1  from sklearn.linear_model import LinearRegression
2  lin_reg = LinearRegression()
3  lin_reg.fit(X, y)
4  print(lin_reg.intercept_, lin_reg.coef_, "\n",
5        lin_reg.predict(X_new))
```

```
[3.9872083] [[2.86708435]]
 [[3.9872083 ]
 [9.72137701]]
```

- Uwaga: Regresja liniowa jest metodą parametryczną tj. opartą o model a nie instancje – do wykonania predykcji potrzebuje trzymać w pamięci parametry modelu a nie zbiór uczący, liczba parametrów jest znana przed rozpoczęciem procesu uczenia.

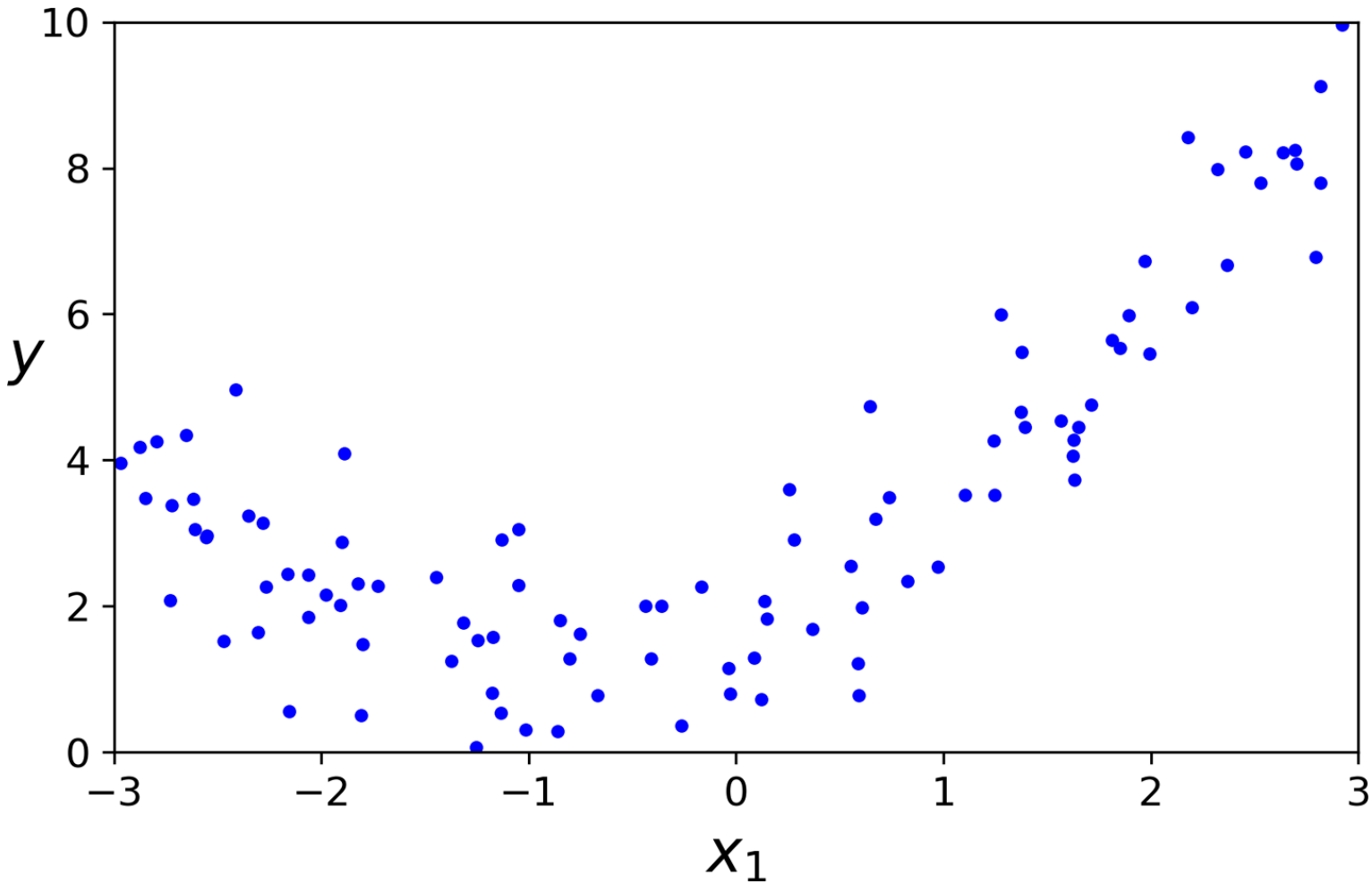
Uwaga: złożoność obliczeniowa

- Macierz wartości cech musi się zmieścić w pamięci.
- Główny problem: odwracanie macierzy $\boldsymbol{X}^T\boldsymbol{X}$ o rozmiarze: $(n+1) \times (n+1)$, złożoność: od $O(n^{2.4})$ do $O(n^3)$
- Scikit Learn używa algorytmu odwracania macierzy Singular Value Decomposition (SVD) o złożoności $O(n^2)$, złożoność dla ilości instancji (próbek) jest liniowa $O(m)$

Plan prezentacji

Regresja wielomianowa, Dane

```
1  m = 100
2  X = 6 * np.random.rand(m, 1) - 3
3  y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```



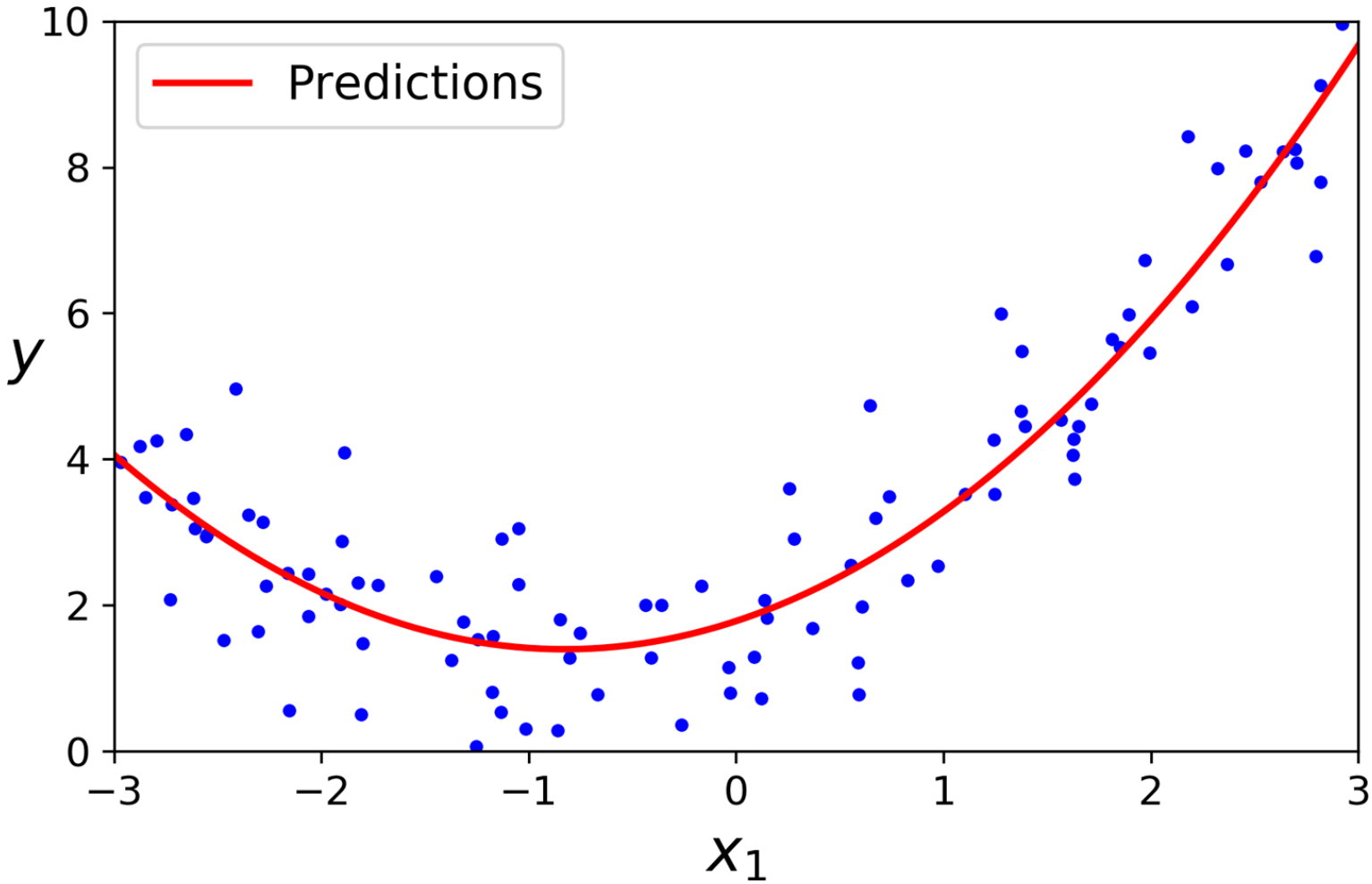
Regresja wielomianowa, Polynomial Regression

```
1  from sklearn.preprocessing import PolynomialFeatures
2  poly_features=PolynomialFeatures(degree=2,include_bias=False
3  X_poly = poly_features.fit_transform(X)
4  print(X[0], X_poly[0]) # (-0.31479346)^2=0.09909492
5  lin_reg = LinearRegression()
6  lin_reg.fit(X_poly, y)
7  print(lin_reg.intercept_, lin_reg.coef_)
8  print(lin_reg.predict(
9      poly_features.fit_transform([[0],[2]])))
10 print(lin_reg.coef_[0][1] * 2**2 + lin_reg.coef_[0][0] * 2
11       + lin_reg.intercept_[0])
```

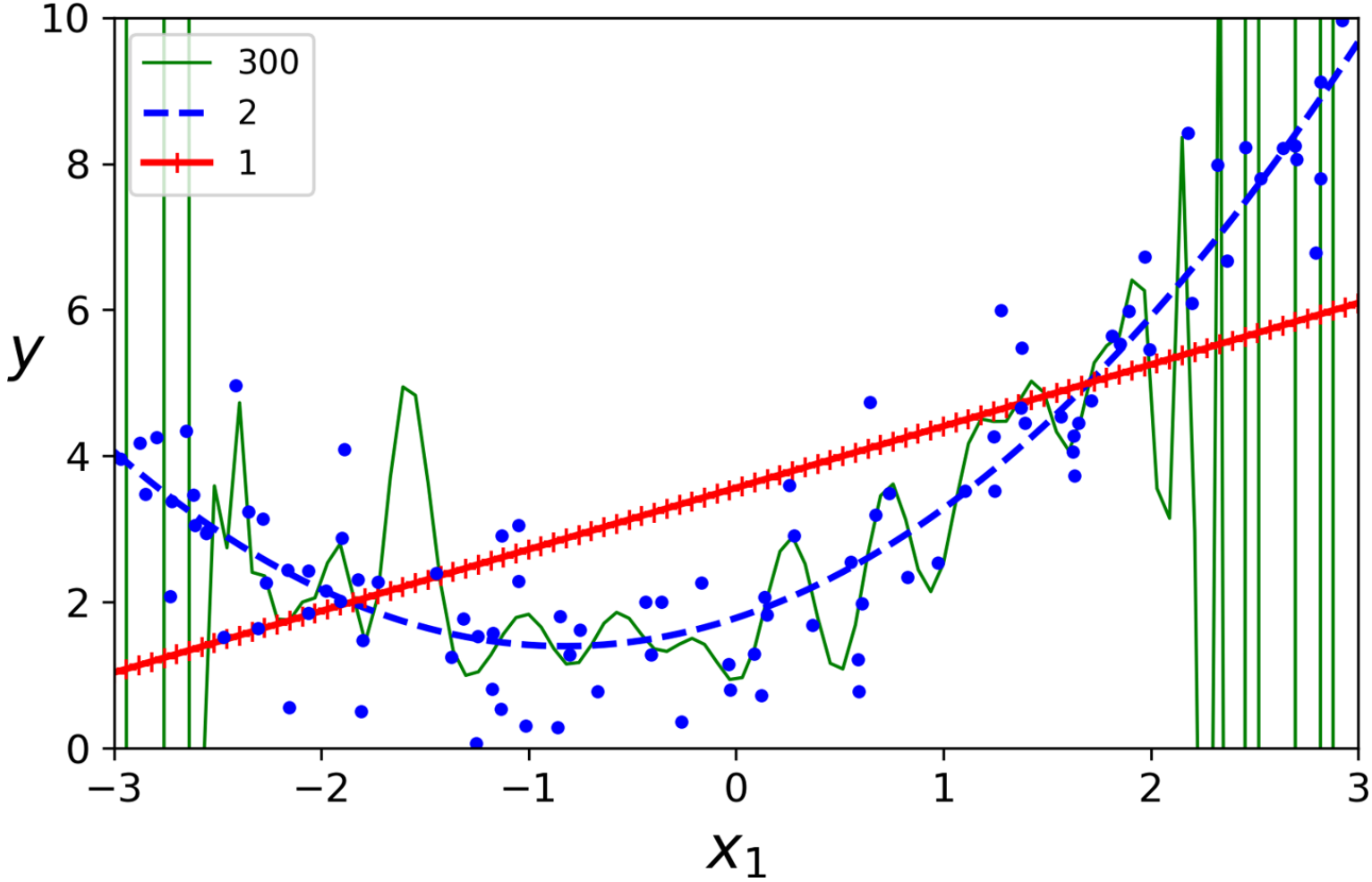
```
[-0.31479346] [-0.31479346  0.09909492]
[2.10826468] [[1.06225693 0.45372695]]
[[2.10826468]
 [6.04768632]]
6.047686318001189
```

- x^2 jako dodatkowa cecha, równanie: $y = 0.5 * x^2 + x + 2$

Regresja wielomianowa



Podsumowanie, Regresja Wielomianowa



- Regularyzacja modelu wielomianowego, tj. ograniczenie modelu celem minimalizacji przeuczenia (overfitting): zmniejszenie stopnia wielomianu.

Plan prezentacji

K-Najbliższych Sąsiadów

- KNN: k-Nearest Neighbors
- Znajdź k najbliższych sąsiadów
  - regresja: średnia wartość dla w/w sąsiadów.
  - klasyfikacja: większościowa etykieta wśród w/w sąsiadów.
- Najbliższych:
  - odległość euklidesowa,
  - podobieństwo kosinusowe,
  - odległość Czebyszewa, Hamminga...

K-Najbliższych Sąsiadów, przykład

```
1  import sklearn.neighbors
2  knn_reg = sklearn.neighbors.KNeighborsRegressor(
3      n_neighbors=3)
4  knn_reg.fit(X, y)
5  print(knn_reg.predict(X_new))
```

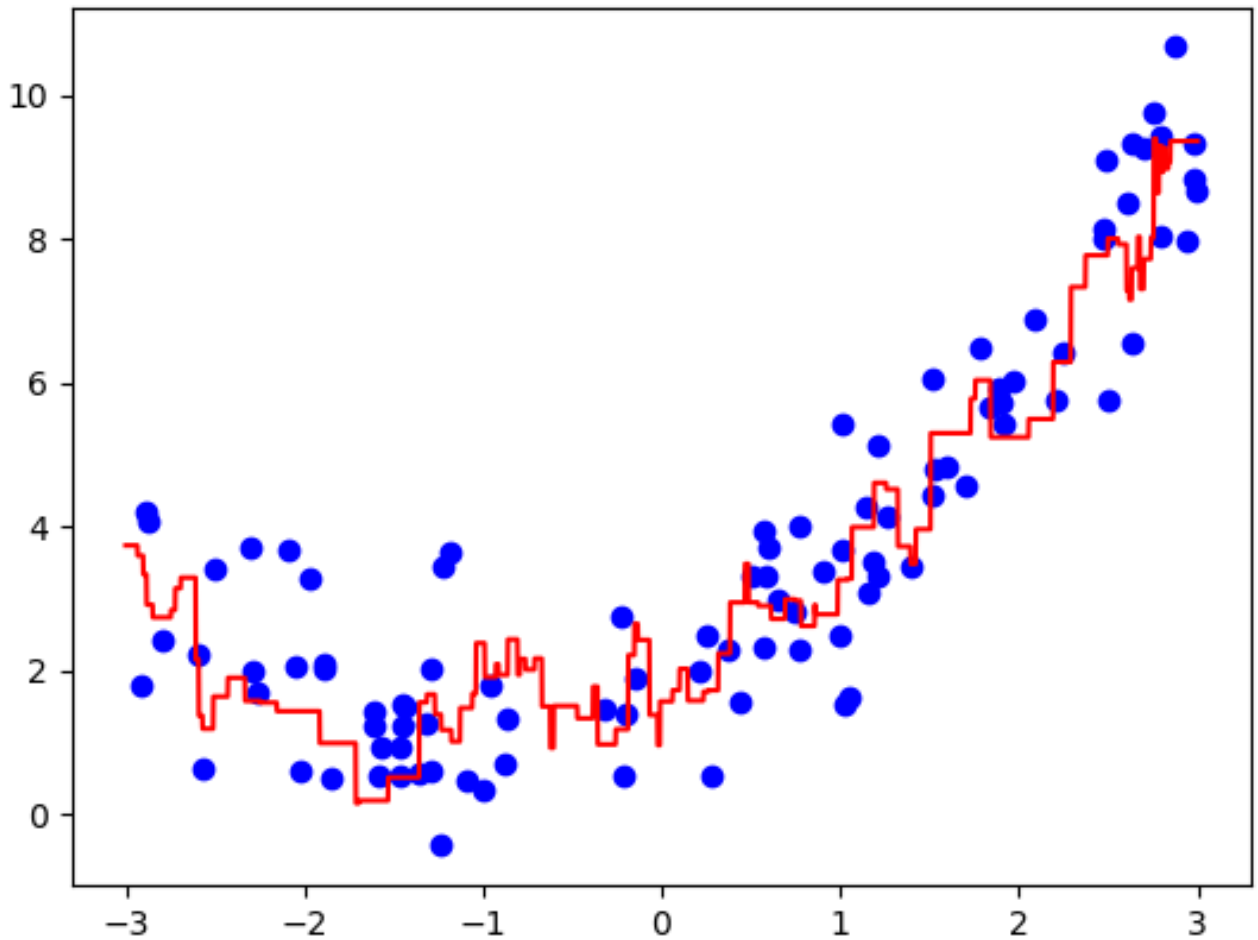
```
[[4.34961254]
 [9.7583373 ]]
```

- Uwaga: KNN jest metodą nieparametryczną tj. opartą o instancje, a nie model – do wykonania predykcji potrzebuje trzymać w pamięci cały zbiór uczący.

Jak wygląda regresja KNN?

```
1  import matplotlib.pyplot as plt
2  plt.clf()
3  plt.scatter(X, y, c="blue")
4  X_new1 = np.arange(-3, 3, 0.001).reshape(-1,1)
5  plt.plot(X_new1, knn_reg.predict(X_new1), c="red")
6  f = "knn.png"
7  plt.savefig(f)
8  print(f)
```

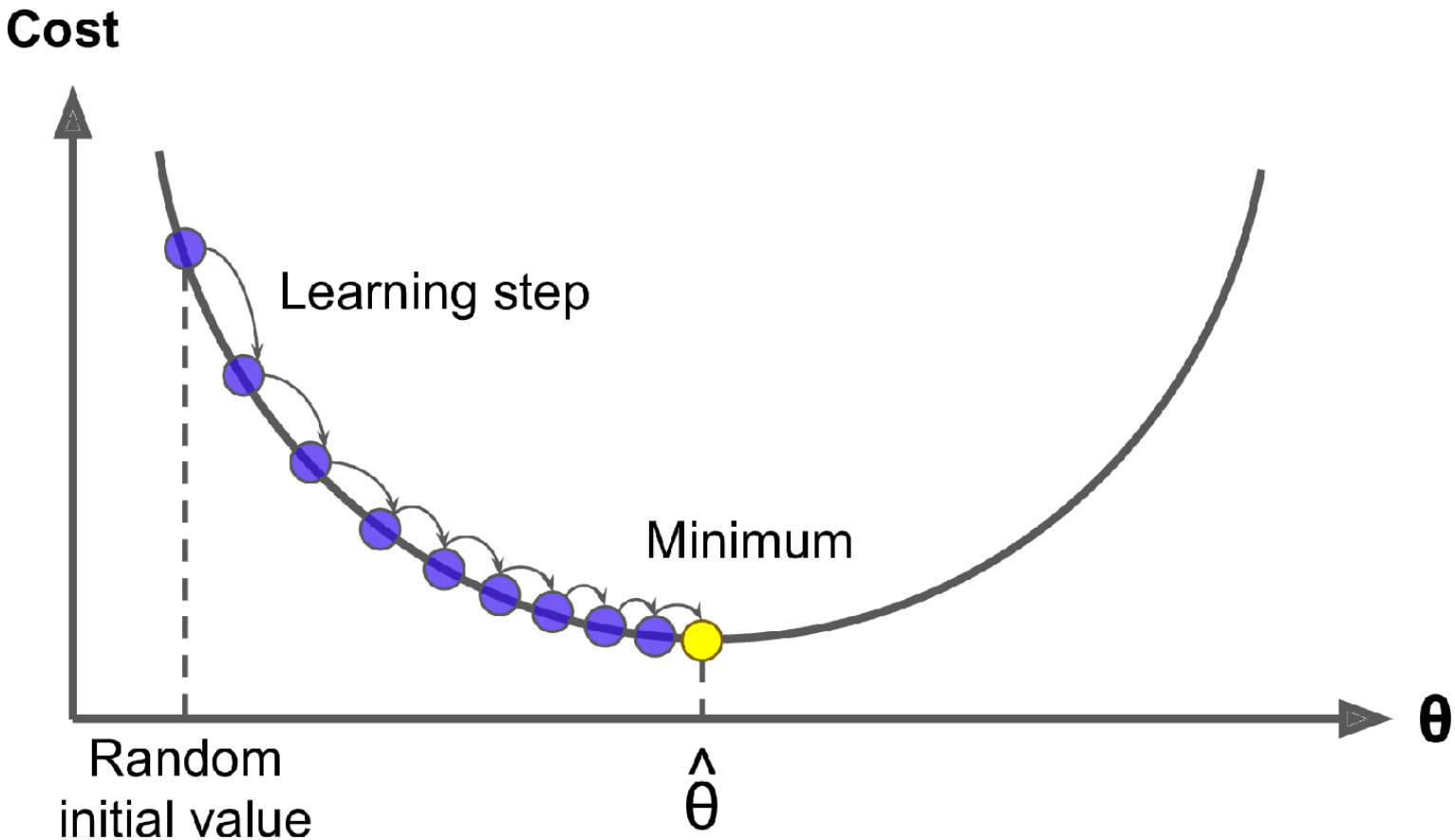
Regresja KNN



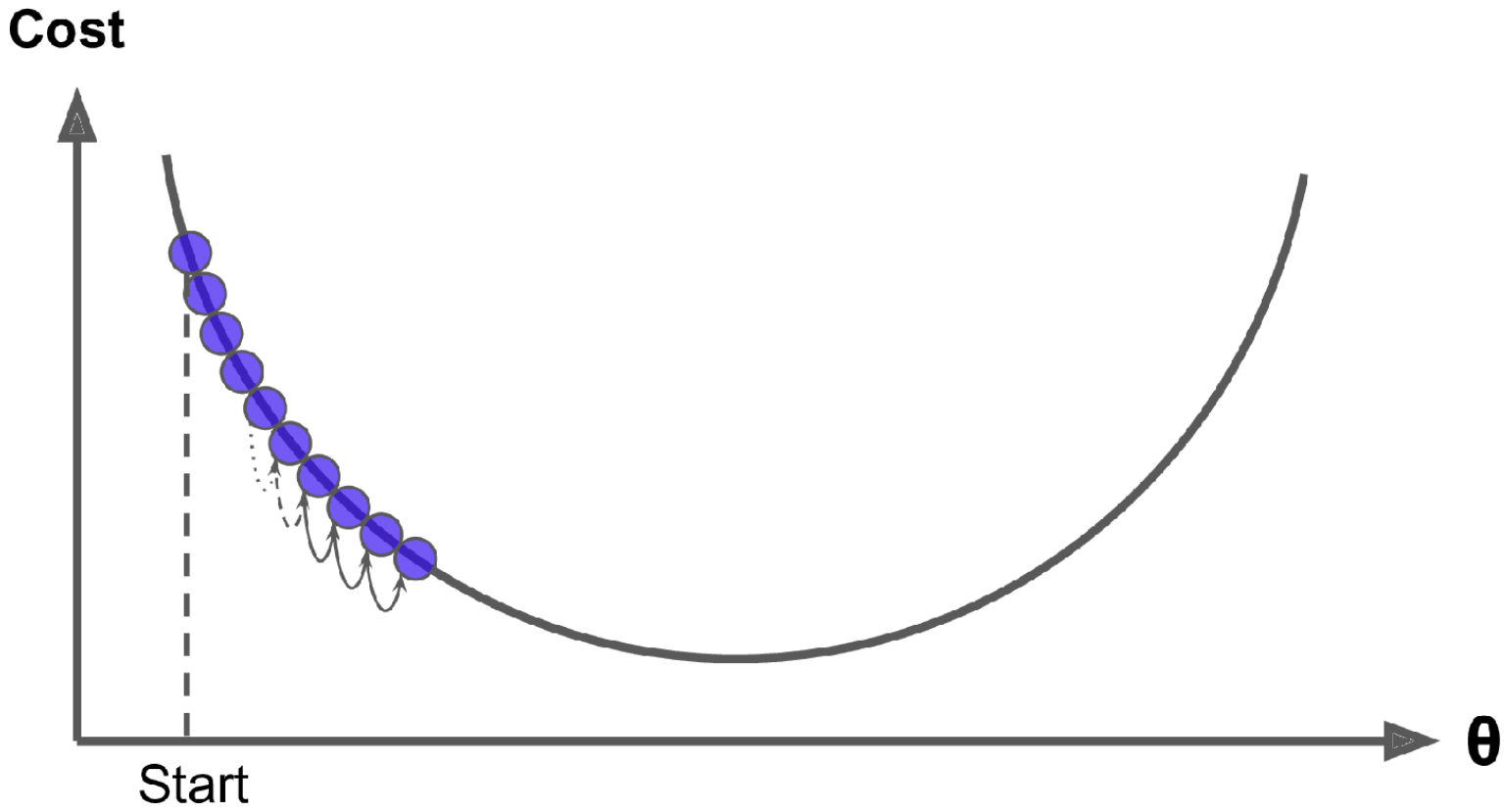
Plan prezentacji

Metoda Gradientu Prostego, Gradient Descent

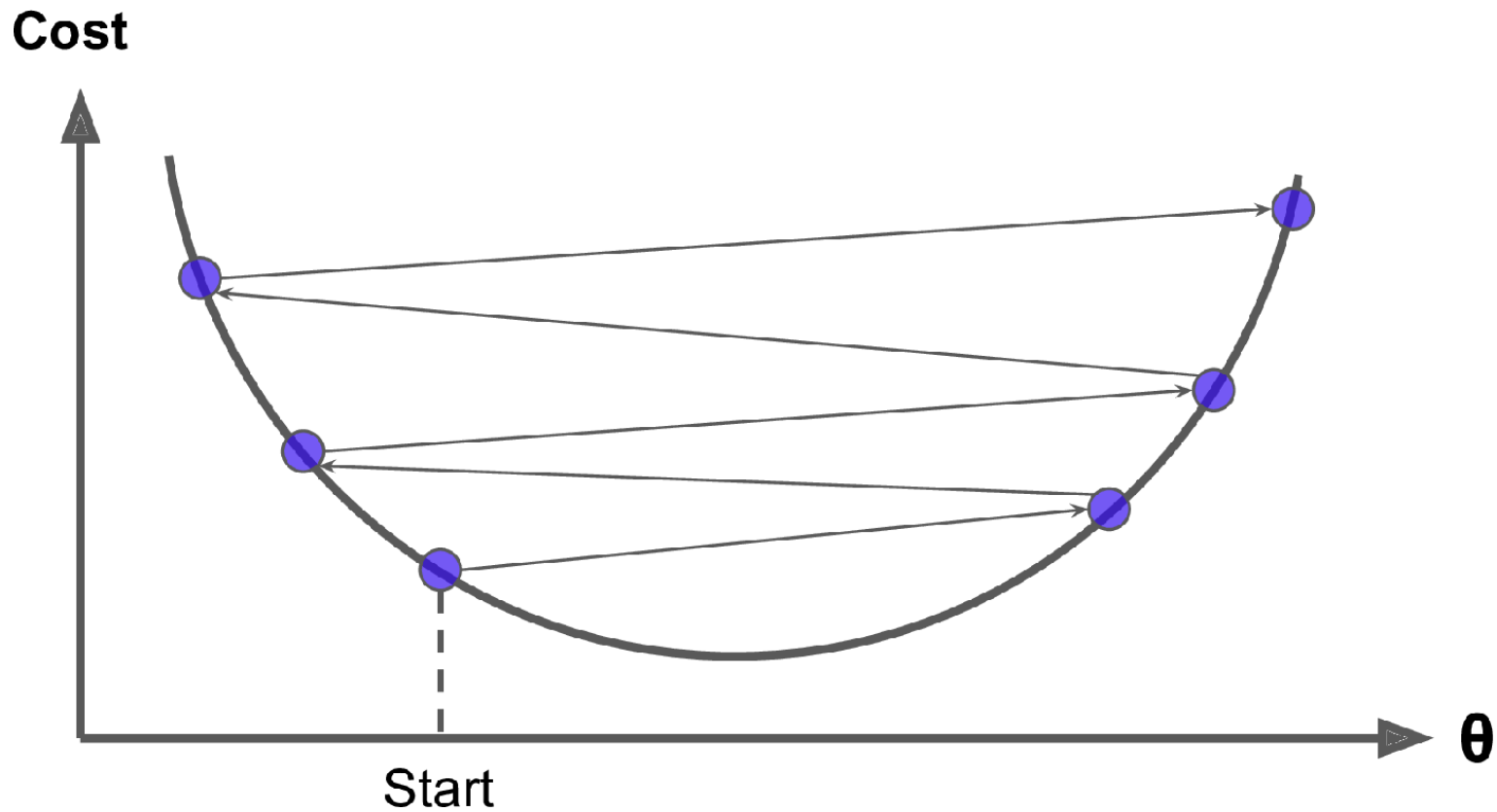
Minimalizacja funkcji celu (kosztu) MSE. Czyli co zrobić jeżeli nie ma równania normalnego.



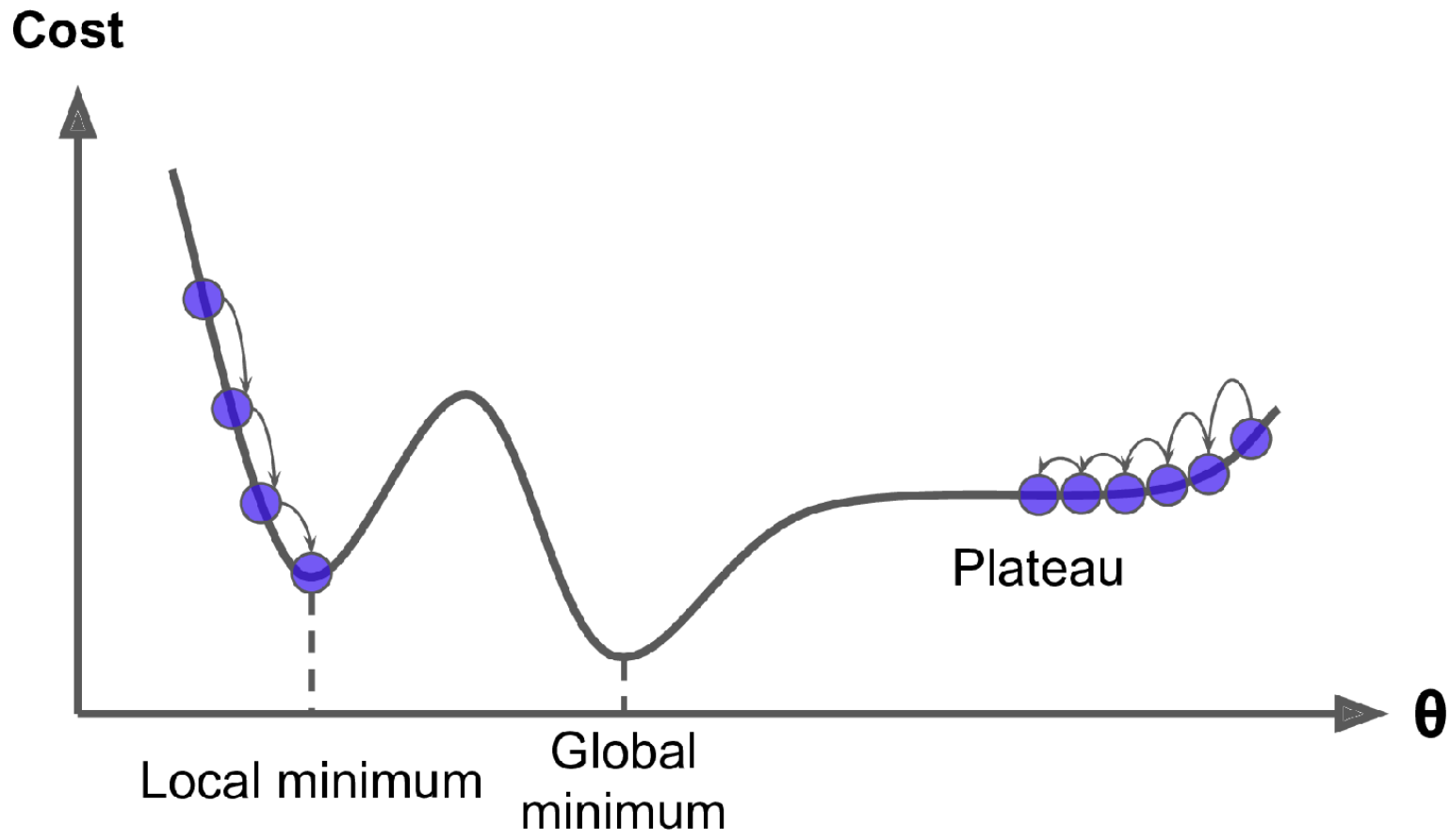
Gradient descent, za mały krok (learning rate)



Gradient descent, za duży krok (learning rate)



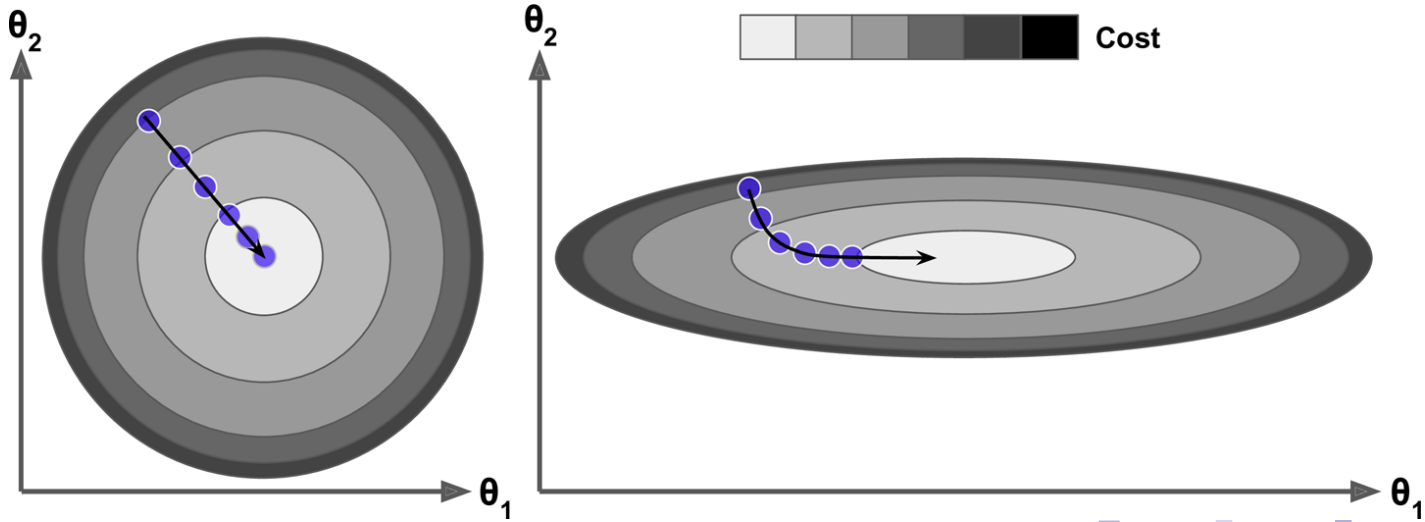
Gradient descent, lokalne minimum



Na szczęście MSE dla regresji liniowej jest wypukła :) Ale w ogólnym przypadku można nie znaleźć minimum!

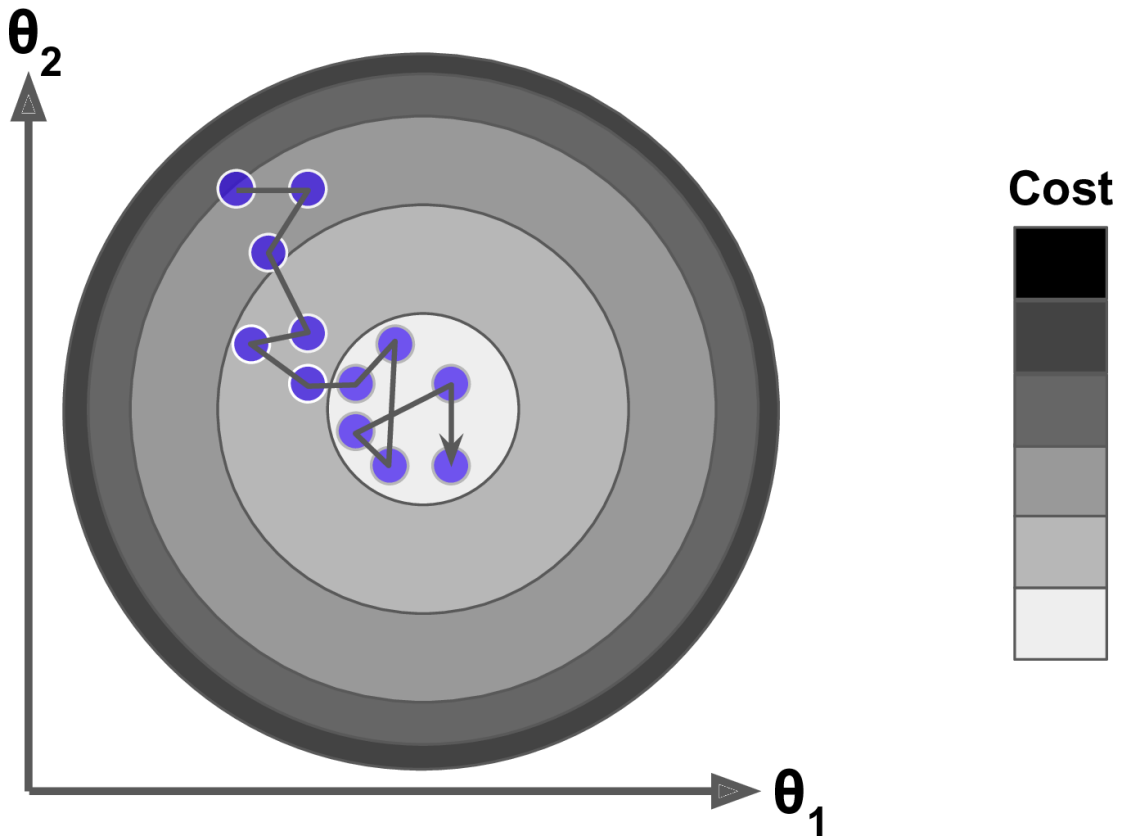
Gradient Descent, problem

- W każdej iteracji trzeba obliczyć gradient funkcji celu (kosztu) na podstawie całego zbioru uczącego :(
- Dla regresji liniowej nie ma sensu, ten sam wynik co dla *równania normalnego*.
  - W ogólnym przypadku gdy nie ma równania liniowego metody gradientowe mają większy sens.
- Rozwiązanie: Stochastic Gradient Descent
- Uwaga: skala cech powinna być taka sama, aby kształt funkcji celu był symetryczny; jeżeli tak nie jest algorytm może nie zmierzać do minimum.



Stochastic Gradient Descent

- Zamiast używać całego zbioru uczącego, wybierz losowo instancję ze zbioru uczącego i policz dla niej gradient.
- Nie trzeba trzymać całego zbioru uczącego w pamięci: możliwość implementacji out-of-core.
- Nieregularny, "skacze" po znalezionych wartościach – z drugiej strony pozwala to na wyjście z minimum lokalnego.



Stochastic Gradient Descent, regresja liniowa

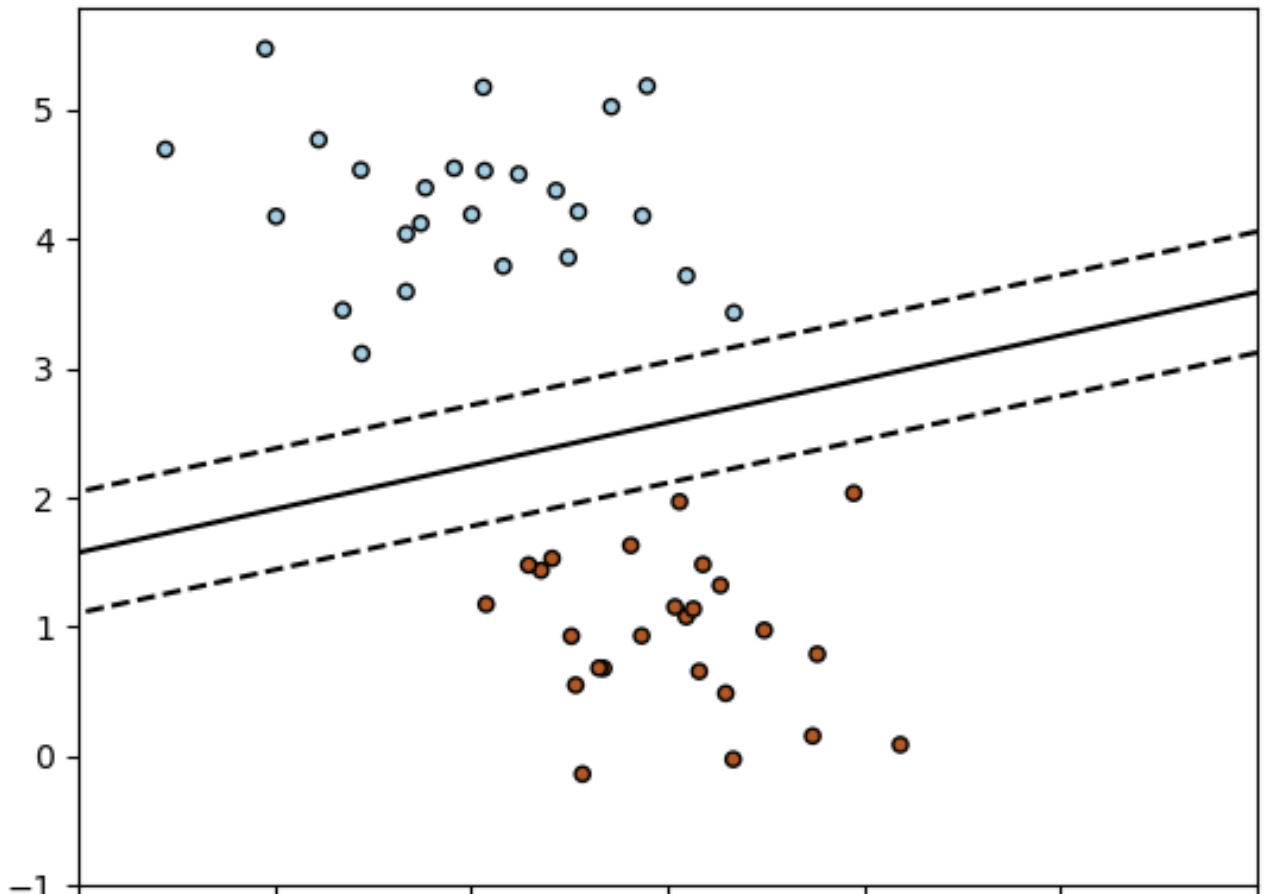
```
1  from sklearn.linear_model import SGDRegressor
2  sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3,
3                         penalty=None, eta0=0.1)
4  sgd_reg.fit(X, y.ravel())
5  #                  ^- zmiana wektora pionowego na poziomy
6  #                     (spłaszczona tablica)
7  print(sgd_reg.intercept_, sgd_reg.coef_)
```

```
[3.9031561] [2.82281154]
```

- 1000 epok (epoch) tj. iteracji lub jeżeli różnica mniejsza niż 0.001 w stosunku do poprzedniej epoki
- learning rate = 0.1 (rozmiar kroku / współczynnika uczenia).

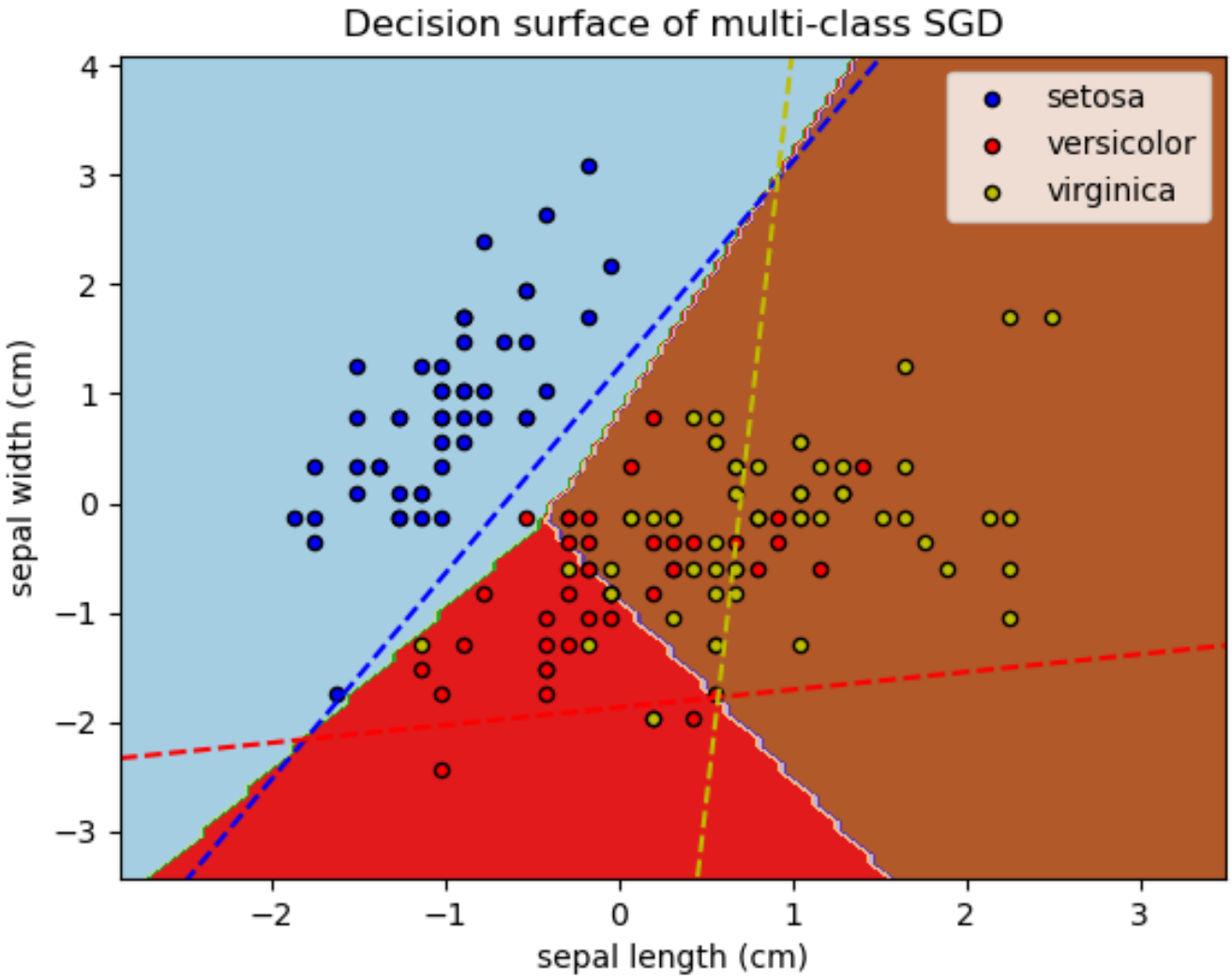
Stochastic Gradient Descent, klasyfikacja binarna, raz jeszcze

Funkcja liniowa dzieląca instancje należące do różnych klas.
`https://scikit-learn.org/stable/modules/sgd.html#classification`



Stochastic Gradient Descent, wiele klas

Funkcja liniowa dla każdej klasy oddzielająca instancje tej klasy od reszty, strategia OvR (One-vs-Rest).



Plan prezentacji

Regresja logistyczna, Logistic Regression

- Uwaga: to nie jest regresja, ale klasyfikacja!
- Obliczenie prawdopodobieństwa czy instancja należy do określonej klasy.
- Binarny klasyfikator: jeżeli prawdopodobieństwo $> 50\%$ to należy do klasy.

Regresja logistyczna, dane

```
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
```

Regresja logistyczna, uczenie

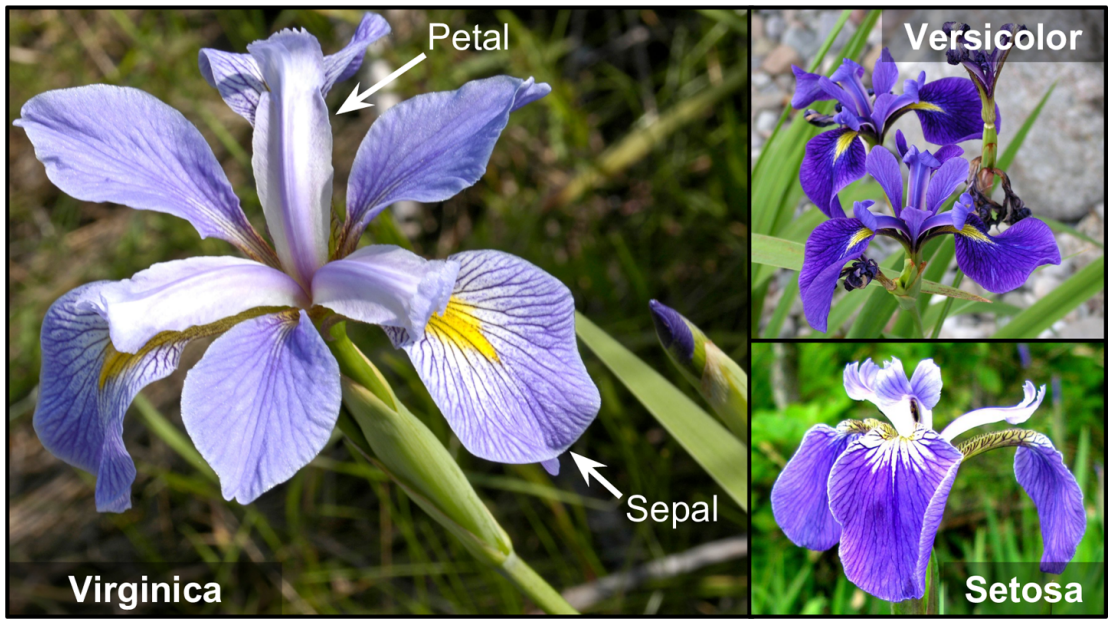
```
X = iris["data"][:, 3:]  # szerokośc płatka
# 1 jeżeli Iris virginica, 0 w p.p.
y = (iris["target"] == 2).astype(np.uint8)
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="lbfgs",
                             random_state=42)
log_reg.fit(X, y)
```

Regresja logistyczna, predykcja

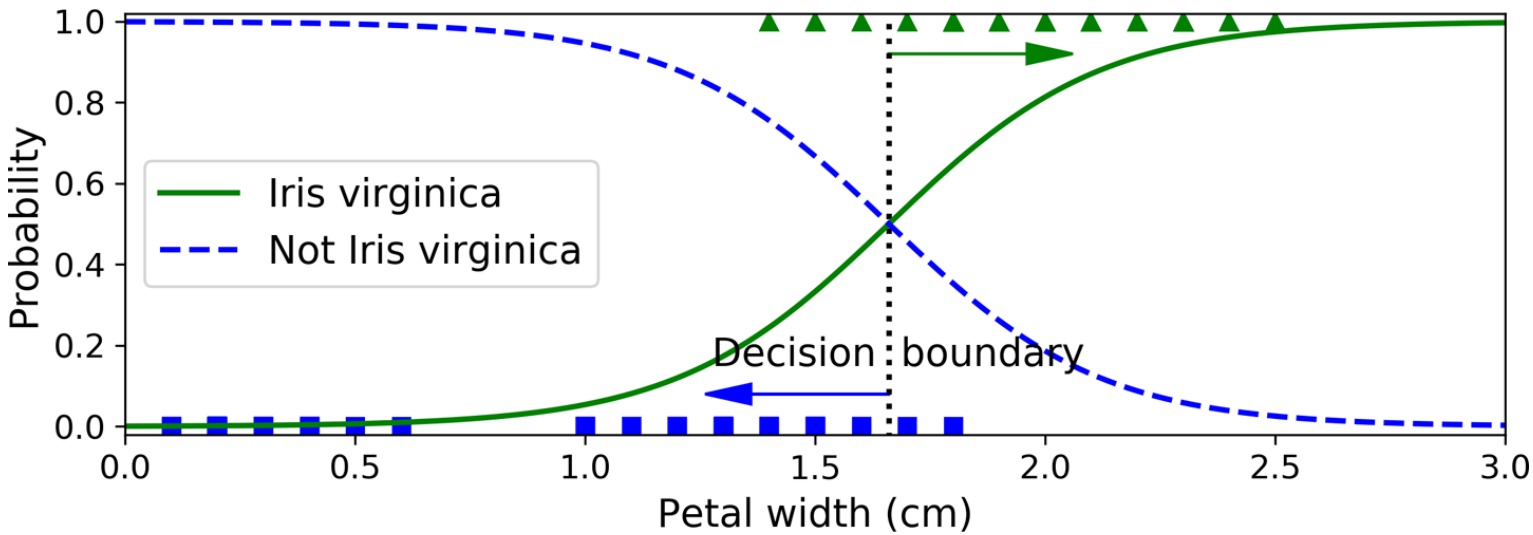
```
1  petal_width = [[1.7], [1.5]]
2  print(log_reg.predict(petal_width),"\n",
3        log_reg.predict_proba(petal_width))
```

```
[1 0]
 [[0.45722097 0.54277903]
 [0.66709636 0.33290364]]
```

- kwiatek o szerokości płatka 1.7 cm jest *Iris virginica*, 54%
- kwiatek o szerokości płatka 1.5 cm nie jest *Iris virginica*, 66%



Regresja logistyczna, jak to działa?



Wiele klas, Softmax, uczenie

```
1  X = iris["data"][:, (2, 3)]  # długość, szerokość płatka
2  y = iris["target"]
3
4  softmax_reg = LogisticRegression(multi_class="multinomial",
5                                   solver="lbfgs",
6                                   C=10, random_state=42)
7  softmax_reg.fit(X, y)
8  print(y) # 3 klasy, Iris: virginica, versicolor, setosa
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

- Bez "multinomial": strategia *one-vs-the-rest*
- "C": regularyzacja modelu, im mniejszy tym bardziej.
- "solver": Large-scale Bound-constrained Optimization Software

Softmax, predykcja

```
petal = [[5, 1.8]]
print(softmax_reg.predict(petal), "\n",
      softmax_reg.predict_proba(petal))
```

```
[2]
 [[2.08323402e-06 1.84645895e-01 8.15352022e-01]]
```

- kwiatek o długości płatka 5 cm i szerokości 1.8 cm jest to *Iris setosa*, 80%