

# Metody Zespołowe, Ensemble

Igor Wojnicki

April 13, 2024

# Plan prezentacji

Ensemble Learning

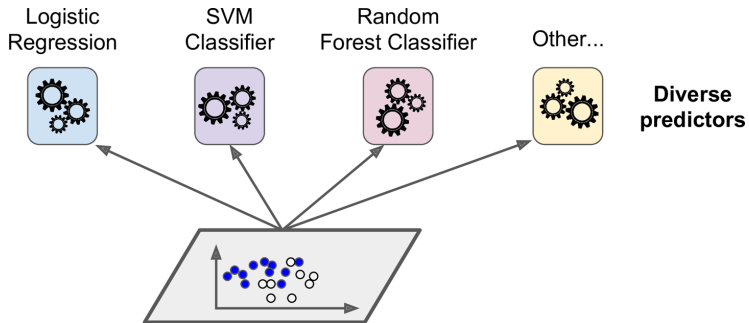
Random Forest

Podsumowanie uczenia nadzorowanego

# Dlaczego Ensemble?

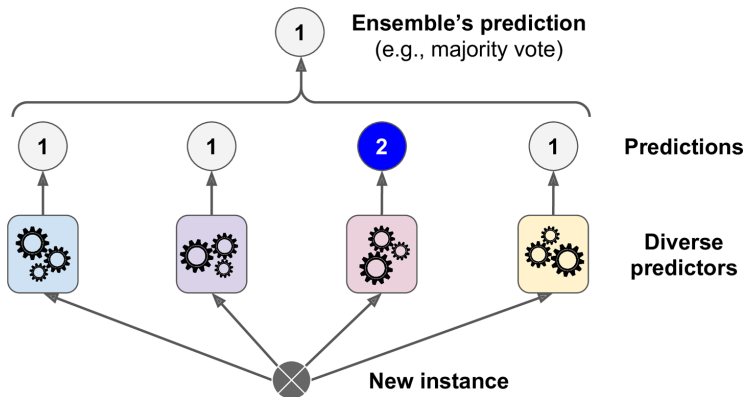
- ▶ *Wisdom of the crowd.*
- ▶ Często używane do porównania wyników.
- ▶ Równoległe.
- ▶ Sekwencyjne.

# Głosowanie



# Strategie głosowania

- ▶ Hard Voting / Majority Vote – Głosowanie Większości / większościowe



- ▶ Co zrobić jak będzie równowaga?

# Ciekawa charakterystyka

- ▶ Ensemble zbudowane z *kiepskich* klasyfikatorów/regresorów, może działać całkiem *dobrze*
- ▶ Pod warunkiem, że klasyfikatory są niezależne.
  - ▶ Zwykle nie są bo uczone na tych samych danych...

## Ensemble, przykład, dane

```
1  from sklearn.model_selection import train_test_split
2  from sklearn.datasets import make_moons
3
4  X, y = make_moons(n_samples=500, noise=0.30,
5                    random_state=42)
6  X_train, X_test, y_train, y_test = \
7      train_test_split(X, y, random_state=42)
```

# Ensemble, przykład, uczenie

```
1  from sklearn.ensemble import RandomForestClassifier
2  from sklearn.ensemble import VotingClassifier
3  from sklearn.linear_model import LogisticRegression
4  from sklearn.svm import SVC
5
6  log_clf = LogisticRegression(solver="lbfgs",
7                               random_state=42)
8  rnd_clf = RandomForestClassifier(n_estimators=100,
9                                  random_state=42)
10 svm_clf = SVC(gamma="scale",
11               random_state=42)
12
13 voting_clf = VotingClassifier(
14     estimators=[('lr', log_clf),
15                 ('rf', rnd_clf),
16                 ('svc', svm_clf)],
17     voting='hard')
```

18



## Ensemble, przykład, predykcja

```
1 print(voting_clf.predict(X_test))
```

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

# Ensemble, porównanie

```
1  from sklearn.metrics import accuracy_score
2
3  for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
4      clf.fit(X_train, y_train)
5      y_pred = clf.predict(X_test)
6      print(clf.__class__.__name__,
7            accuracy_score(y_test, y_pred))
```

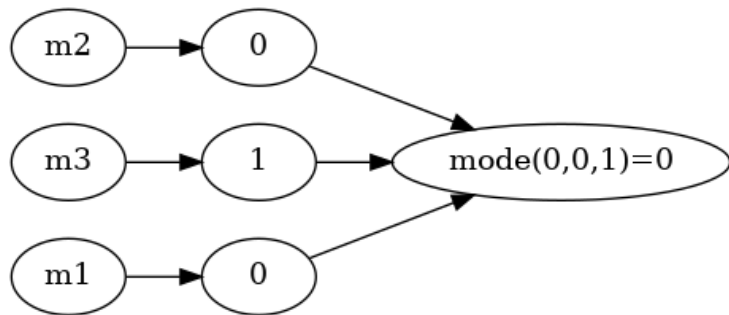
LogisticRegression 0.864

RandomForestClassifier 0.896

SVC 0.896

VotingClassifier 0.912

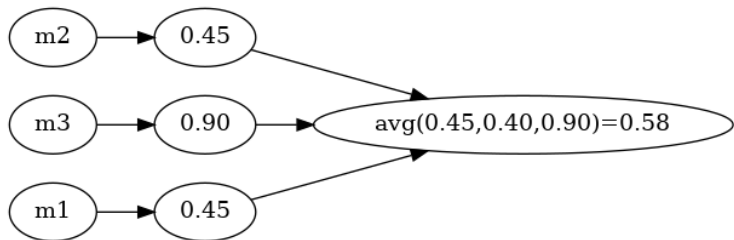
## Strategia głosowania: Hard Voting



$\text{mode}()$  – dominanta.

## Strategia głosowania: Soft Voting

- ▶ Każdy predyktor musi dostarczyć prawdopodobieństwo klasyfikacji `.predict_proba()`
- ▶ Prawdopodobieństwa dla każdej klasy są uśredniane.
- ▶ Wygrywa klasa z największym uśrednionym prawdopodobieństwem.



`avg()` – średnia.

# Ensemble, Soft Voting, uczenie i predykcja

```
1  log_clf = LogisticRegression(solver="lbfgs",
2                                random_state=42)
3  rnd_clf = RandomForestClassifier(n_estimators=100,
4                                   random_state=42)
5  svm_clf = SVC(gamma="scale", probability=True,
6                 random_state=42)
7
8  voting_clf = VotingClassifier(
9      estimators=[('lr', log_clf),
10                  ('rf', rnd_clf),
11                  ('svc', svm_clf)],
12      voting='soft')
13  voting_clf.fit(X_train, y_train)
```

## Ensemble, Soft Voting, porównanie

```
1  from sklearn.metrics import accuracy_score
2
3  for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
4      clf.fit(X_train, y_train)
5      y_pred = clf.predict(X_test)
6      print(clf.__class__.__name__,
7            accuracy_score(y_test, y_pred))
```

LogisticRegression 0.864

RandomForestClassifier 0.896

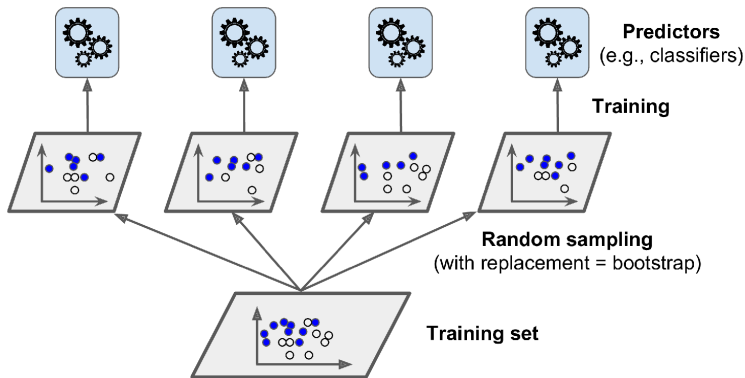
SVC 0.896

VotingClassifier 0.92

# Bagging i Pasting: ensemble inaczej, samplowanie

Samplowanie:

- ▶ z powtórzeniami (with replacement) – Bagging = bootstrap aggregating, bootstrap=True
- ▶ bez powtórzeń (without replacement) – Pasting, bootstrap=False.



- ▶ Wygrywa dominanta. Dobrze się skaluje. Dowolny klasyfikator.

# Bagging

```
1  from sklearn.ensemble import BaggingClassifier
2  from sklearn.tree import DecisionTreeClassifier
3
4  bag_clf = BaggingClassifier(
5      DecisionTreeClassifier(), n_estimators=500,
6      max_samples=100, bootstrap=True, random_state=42)
7  bag_clf.fit(X_train, y_train)
8  y_pred = bag_clf.predict(X_test)
9
10 from sklearn.metrics import accuracy_score
11 print(accuracy_score(y_test, y_pred))
```

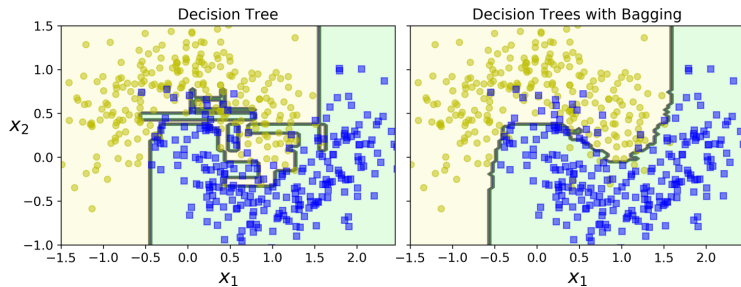
0.904

`max_samples` – liczba instancji do wylosowania, bezwzględna (tutaj: 100 instancji) albo procentowo [0, 1]

► Uwaga: 1 vs. 1.0 w Pythonie, liczba instancji: 1 vs. 100%.



# Bagging, porównanie



# Bagging, ewaluacja

- ▶ Niektóre instancje są używane wielokrotnie, nawet dla tego samego predyktora.
- ▶ Niektóre instancje nie są w ogóle używane: *out-of-bag* – 37%.
- ▶ Można ich użyć do ewaluacji, jako zbiór testujący.
- ▶ *scikit-learn* może to zrobić automatycznie do określenia dokładności (*accuracy*).
- ▶ Uwaga: predyktorów musi być dużo.

```
1 bag_clf = BaggingClassifier(  
2     DecisionTreeClassifier(), n_estimators=500,  
3     bootstrap=True, oob_score=True, random_state=40)  
4 bag_clf.fit(X_train, y_train)  
5 print(bag_clf.oob_score_)
```

0.9013333333333333

## Bagging, ewaluacja wszystkich instancji

```
1 print(bag_clf.oob_decision_function_[:5])
```

```
[[0.31746032 0.68253968]
 [0.34117647 0.65882353]
 [1.         0.         ]
 [0.         1.         ]
 [0.         1.         ]]
```

# Wybór/sampling cech

Umożliwia uczenie predyktora na podstawie losowo wybranych cech.

- ▶ z/bez powtórzeń: `bootstrap_features=True/False`
- ▶ `max_features` – liczba cech do wylosowania,
  - ▶ bezwzględna np. 100, albo
  - ▶ procentowo  $[0, 1]$  np. 0.7
- ▶ Random Patches: wybór instancji i cech
  - ▶ `max_samples=0.5`, `max_features=0.7`
- ▶ Random Subspaces: wszystkie instancje i wybór cech
  - ▶ `bootstrap=False`, `max_samples=1.0`,  
`bootstrap_features=True`, `max_features=0.7`

# Plan prezentacji

Ensemble Learning

Random Forest

Podsumowanie uczenia nadzorowanego

# Random Forest

- ▶ Droga na skróty, zamiast  
BaggingClassifier+DecisionTreeClassifier
- ▶ Bagging + Drzewa decyzyjne, max\_samples=1.0
- ▶ Hiperparametry z BaggingClassifier i  
DecisionTreeClassifier.
- ▶ Dodatkowo: nie wyszukuje cech do podziału węzła spośród  
wszystkich, ale z losowo wybranego podzbioru – większa  
różnorodność.

```
1 from sklearn.ensemble import RandomForestClassifier
2
3 rnd_clf = RandomForestClassifier(n_estimators=500,
4                                 max_leaf_nodes=16,
5                                 random_state=42)
6 rnd_clf.fit(X_train, y_train)
7
8 y_pred_rf = rnd_clf.predict(X_test)
```

## Pomiar ważności cech

```
1  from sklearn.datasets import load_iris
2  iris = load_iris()
3  rnd_clf = RandomForestClassifier(n_estimators=500,
4                                  random_state=42)
5  rnd_clf.fit(iris["data"], iris["target"])
6  for name, score in zip(iris["feature_names"],
7                          rnd_clf.feature_importances_):
8      print(name, score)
9
10 print(rnd_clf.feature_importances_)
```

```
sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682
[0.11249225 0.02311929 0.44103046 0.423358   ]
```

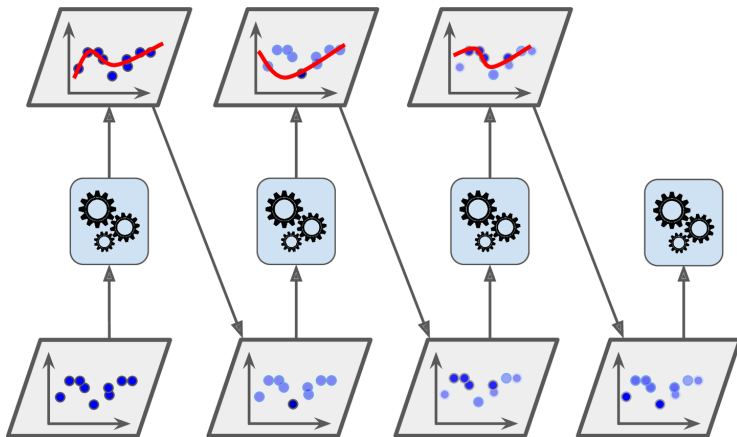
► Sumują się do 1.0

# Boosting

Kilka kiepskich modeli  $\rightarrow$  jeden dobry model.



# AdaBoost



- ▶ Technika sekwencyjna, por. zrównoleglanie z Bagging/Pasting?
- ▶ Każdej instancji i predyktorowi przypisywana jest waga.
- ▶ Wagi są modyfikowane.
- ▶ Po nauczaniu wykorzystywane są wszystkie predyktory

# AdaBoost

1. Początkowa waga każdej instancji:  $1/m$ ,  $m$  – liczba instancji.
  - ▶ Większa waga = instancja częściej używana
2. Uczenie predyktora.
3. Ocena predyktora + określenie wagi predyktora (lepszy  $\rightarrow$  większa waga)
4. Wagi zwiększane dla błędnie sklasyfikowanych instancji.
5. Jeżeli jeszcze jeden predyktor to idź do: 2
6. Użyj wszystkich predyktorów (ensemble) z wagami proporcjonalnymi do oceny (głosowanie większościowe z wagami predyktorów).

## AdaBoost, przykład

```
1  from sklearn.model_selection import cross_val_score
2  from sklearn.datasets import load_iris
3  from sklearn.ensemble import AdaBoostClassifier
4
5  X, y = load_iris(return_X_y=True)
6  clf = AdaBoostClassifier(n_estimators=100)
7  scores = cross_val_score(clf, X, y, cv=5)
8  print(scores.mean())
```

0.9466666666666665

- ▶ Domyślnie klasyfikatorem jest:  
DecisionTreeClassifier(max\_depth=1)

## AdaBoost, przykład

```
1 X, y = load_iris(return_X_y=True)
2 clf = AdaBoostClassifier(SVC(kernel="linear",
3                             probability=True),
4                             n_estimators=100)
5 scores = cross_val_score(clf, X, y, cv=5)
6 print(scores.mean())
```

0.9733333333333334

# Gradient Boosting

- ▶ Sekwencyjny.
- ▶ Następny predyktor stara się poprawić błędy poprzedniego.

## Gradient Boosting, na piechotę

```
1  from sklearn.tree import DecisionTreeRegressor
2  X = np.random.rand(100, 1) - 0.5
3  y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
4
5  tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=0)
6  tree_reg1.fit(X, y)
7  y2 = y - tree_reg1.predict(X)
8  tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=0)
9  tree_reg2.fit(X, y2)
10 y3 = y2 - tree_reg2.predict(X)
11 tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=0)
12 tree_reg3.fit(X, y3)
13
14 X_new = np.array([[0.8]])
15 y_pred = sum(tree.predict(X_new)
16               for tree in (tree_reg1, tree_reg2, tree_reg3))
17 print(y_pred)

[0.73641434]
```

## Gradient Boosting, prościej

```
1 from sklearn.ensemble import GradientBoostingRegressor
2 gbrt = GradientBoostingRegressor(max_depth=2,
3                                   n_estimators=3,
4                                   learning_rate=1.0,
5                                   random_state=42)
6 gbrt.fit(X, y)
7 print(gbrt.predict(X_new))
```

[0.73641434]

- ▶ Taki sam wynik, jak poprzednio :)
- ▶ `learning_rate`: szybkość uczenia, domyślnie 0.1 (zmniejszenie udziału kolejnego estymatora do 10%), tutaj przyspieszamy, bo tylko 3 estymatory.

## Gradient Boosting, szybkość uczenia

```
1  gbrt = GradientBoostingRegressor(max_depth=2,  
2                                     n_estimators=3,  
3                                     learning_rate=1.0)  
4  gbrt.fit(X, y)  
5  print(gbrt.predict(X_new))
```

[0.73641434]

```
1  gbrt = GradientBoostingRegressor(max_depth=2,  
2                                     n_estimators=3,  
3                                     learning_rate=0.1)  
4  gbrt.fit(X, y)  
5  print(gbrt.predict(X_new))
```

[0.38517789]



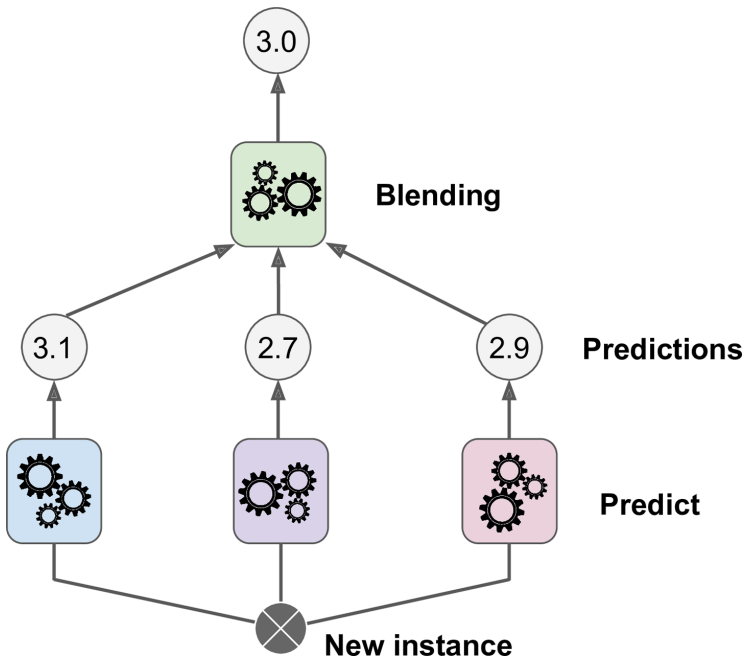
# Gradient Boosting

```
1  gbrt = GradientBoostingRegressor(max_depth=2,  
2                                     n_estimators=100,  
3                                     learning_rate=0.1)  
4  gbrt.fit(X, y)  
5  print(gbrt.predict(X_new))
```

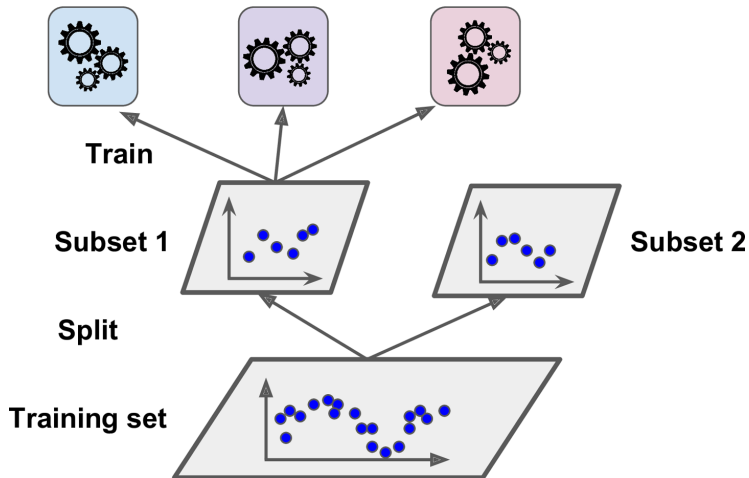
[0.72796928]

- ▶ niska prędkość uczenia – potrzeba więcej regresorów.

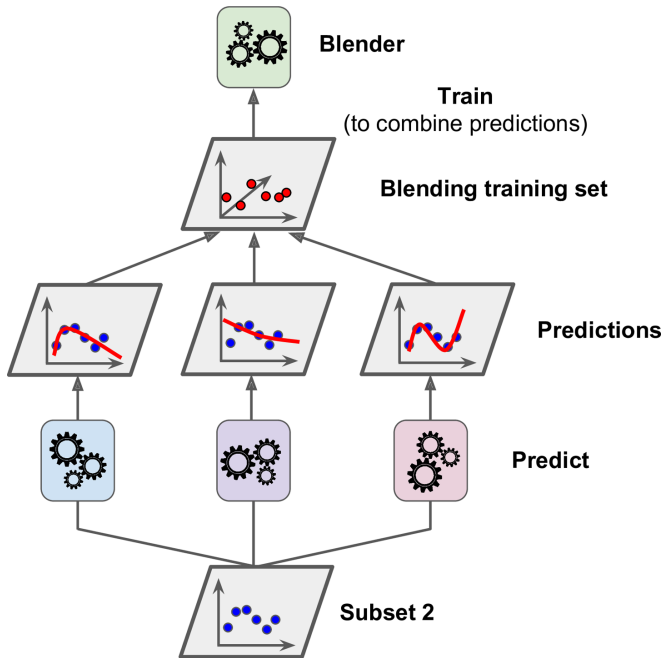
## Stacking - blender: osobny model do agregacji predykcji



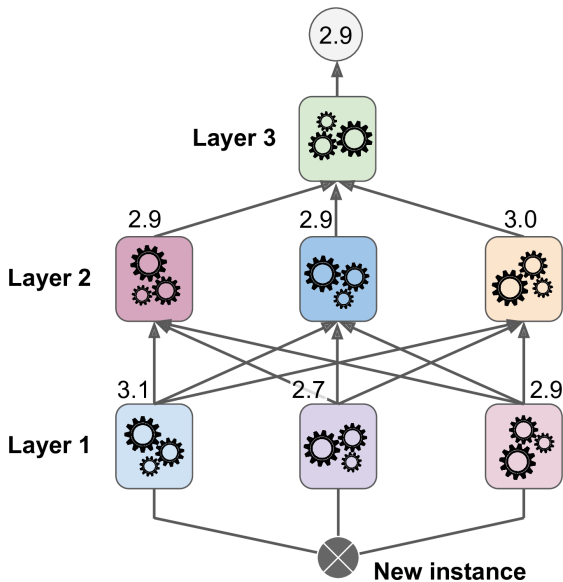
# Stacking, uczenie modeli



# Stacking, uczenie blendera



# Stacking, wiele blenderów



## Stacking, *scikit-learn*

- ▶ `sklearn.ensemble.StackingClassifier`
- ▶ `sklearn.ensemble.StackingRegressor`

Używana jest walidacja krzyżowa do trenowania blendera.

# Plan prezentacji

Ensemble Learning

Random Forest

Podsumowanie uczenia nadzorowanego

# Błąd generalizacji, uogólnienia modelu

## obciążenia, Bias

- ▶ Tendencja do uczenia się tych samych błędów.
- ▶ Różnica pomiędzy wartością docelową, a prognozą modelu.
- ▶ Zbyt prosty model.
- ▶ Underfitting.

## zmienności, Variance

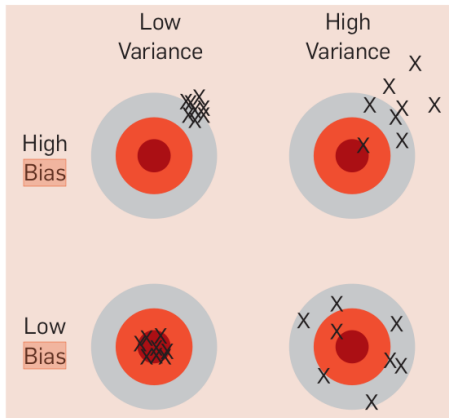
- ▶ Uczenie się losowych zmian/szumu.
- ▶ Nadmierna wrażliwość na małe różnice danych uczących.
- ▶ Zbyt złożony model.
- ▶ Overfitting.

## nieredukowalny, Irreducible error

- ▶ Wynika z zaszumienia danych.
- ▶ Trzeba oczyścić dane.



# Błąd generalizacji



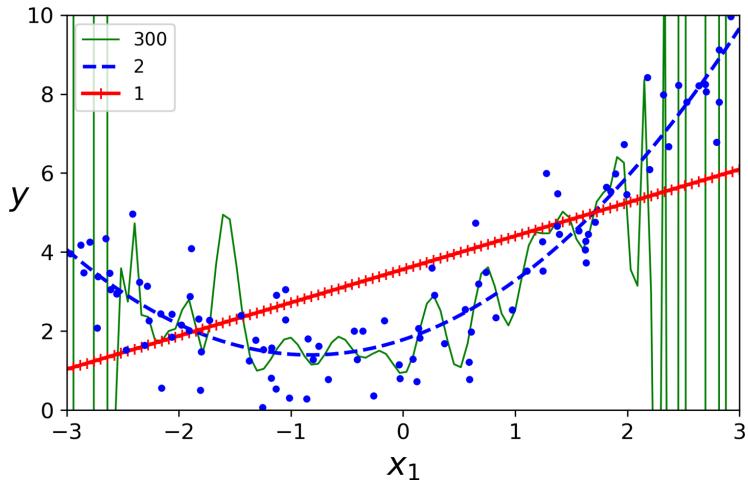
Źródło: Pedro M. Domingos: A few useful things to know about machine learning. *Commun. ACM* 55(10): 78-87 (2012)

# Bias-Variance tradeoff

Wikipedia: Bias-variance tradeoff

- ▶ Zwiększenie złożoności modelu zwiększa jego wariancję i zmniejsza jego obciążenie.
- ▶ Zmniejszenie złożoności modelu zwiększa jego obciążenie i zmniejsza jego wariancję.
- ▶ Kiedy model staje się zbyt skomplikowany może dojść do przeuczenia
  - ▶ model uczy się też odchyleń nie mających wpływu na realny trend.
- ▶ Regularyzacja: ograniczenie modelu – zmniejszenie overfitting – np. redukcja stopnia wielomianu, ograniczenie wartości parametrów
  - ▶ również osiągalna za pomocą hiperparametrów dango modelu, np. ograniczenie głębokości drzewa decyzyjnego.

# Bias-Variance tradeoff



# Podsumowanie dot. metryk

- ▶ Confusion Matrix
  - ▶ Dokładność (Accuracy)
  - ▶ Precyzja (Precision)
  - ▶ Czułość (Recall)
  - ▶ F1
- ▶ RMSE/MSE

# Różne regresory

