Bootcamp Supervised learning

Unsupervised learning

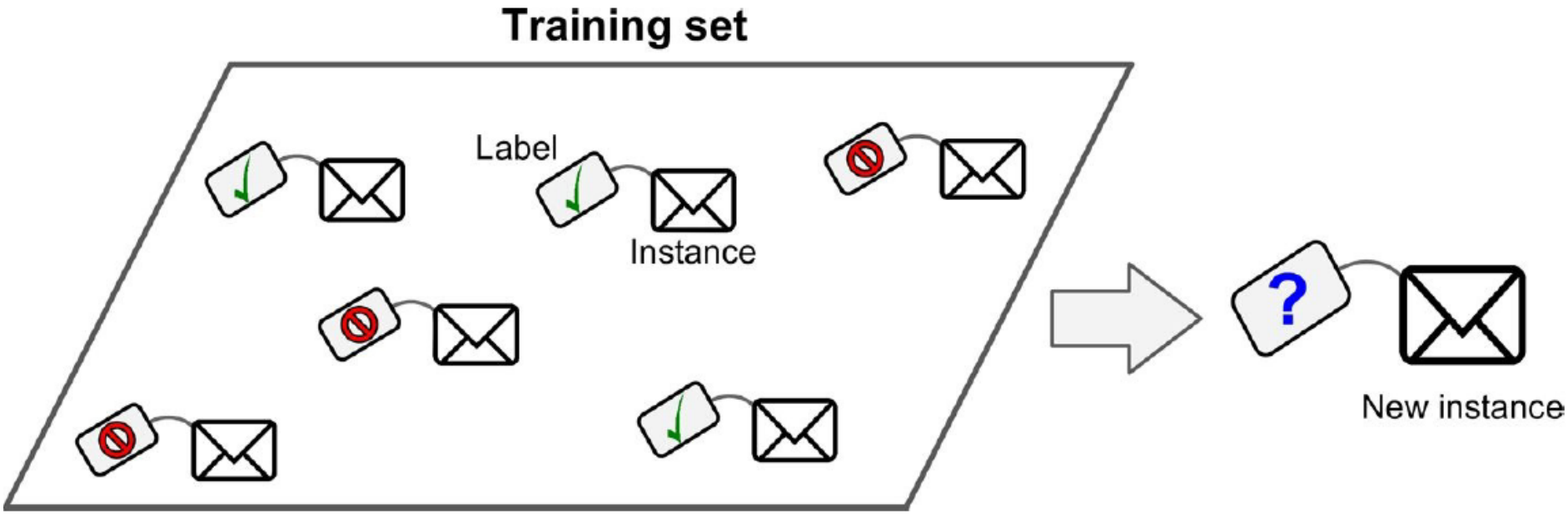
Przemysław Spurek

Supervised Learning

- Filtr spamu możemy rozumieć jako proste zadanie Machine Learning, który może nauczyć się oznaczać, czy dany e-mail to spam, czy normalna wiadomość (nonspam, zwanych również "ham").
- Przykłady, których system używa do uczenia się, nazywa się zestawem treningowym.
- W uczeniu nadzorowanym (Supervised Learning) dane treningowe podawane do algorytmu zawierają rozwiązania zwane etykietami (labels).

Supervised Learning

Typowym nadzorowanym zadaniem - **uczeniem nadzorowanym** jest **klasyfikacja**.
Filtr spamu jest dobrym przykładem: uczymy na wielu emailach wraz z ich klasą (spam/ham) i musi nauczyć się zaklasyfikować nową wiadomość e-mail do jednej z tych klas.



Supervised Learning

- Potrzebna nam jeszcze jest miara oceny. Jednym z przykładów takiej miary jest **Accuracy (ACC)** współczynnik poprawnie sklasyfikowanych przykładów:

$$ACC = \frac{\text{Ilość poprawnie sklasyfikowanych przykładów}}{\text{Ilość wszystkich przykładów}}$$

Logistic Regression

Logistic Regression

- Regresja logistyczna (zwana również **Logistic Regression**) jest powszechnie używana do oszacowania prawdopodobieństwa, że instancja należy do konkretnej klasy (np. Jakie jest prawdopodobieństwo, że dana wiadomość e-mail jest spamem?).
- Jeśli oszacowane prawdopodobieństwo jest większe niż 50%, wówczas model przewiduje, że instancja należy do tej klasy (zwanej klasą dodatnią, oznaczoną "1").
- W ten sposób dostajemy klasyfikator binarny.

`https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z01.ipynb`

Logistic Regression

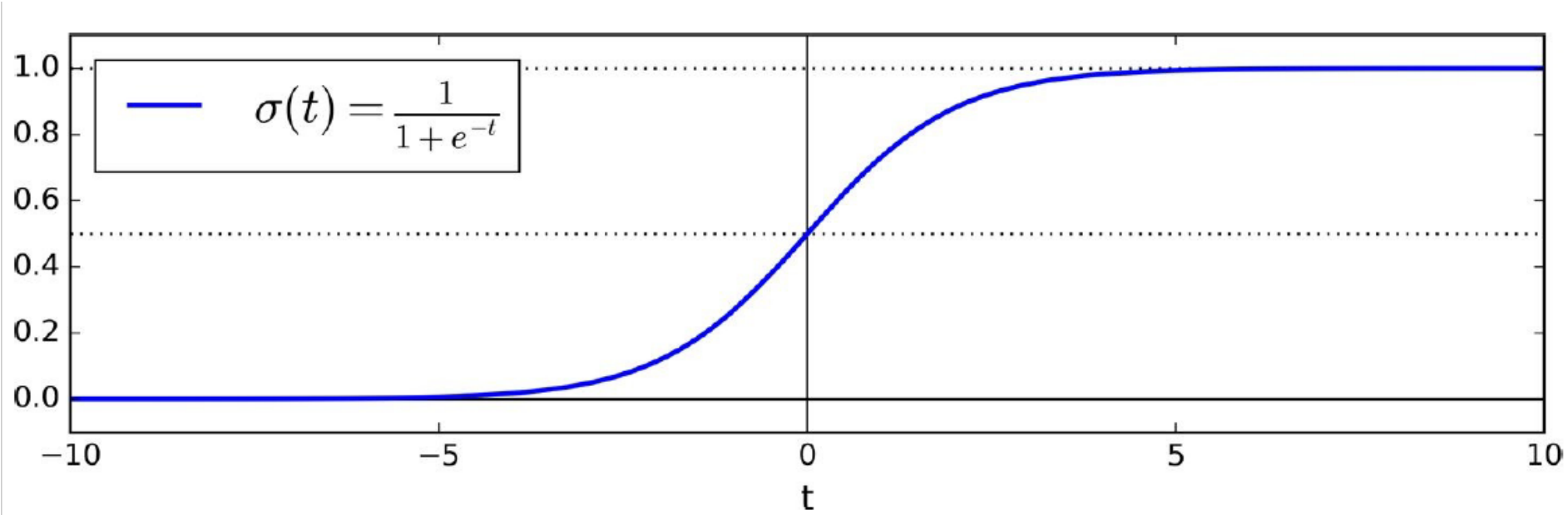
Podobnie jak w modelu regresji liniowej, model **regresji logistycznej** oblicza ważoną sumę elementów wejściowych (plus wartość odchylenia – bias), ale zamiast wyświetlać wynik bezpośrednio, jak ma to miejsce w przypadku modelu regresji liniowej, zwraca **logistic** tego wyniku.

$$\hat{p} = h_{\Theta}(x) = \sigma(\Theta^T x)$$

Logistic - zwana również **logit**, oznaczmy $\sigma(\cdot)$ - jest funkcją sigmoidalną (tzn. w kształcie litery S), która zwraca liczby między 0 a 1.

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Logistic Regression



Logistic Regression

Model regresji logistycznej szacuje prawdopodobieństwo:

$$\hat{p} = h_{\Theta}(x),$$

że instancja x należy do klasy dodatniej, co pozwala dostać etykietę $\hat{y}$.

Należy zauważyć, że $\sigma(t) < 0,5$, gdy $t < 0$, i $\sigma(t) \geq 0,5$, gdy $t \geq 0$, więc model regresji logistycznej przewiduje 1, jeśli $\Theta^T x$ jest dodatnie, a 0 jeśli jest ujemne.

Logistic Regression

- Teraz już wiesz, jak model regresji logistycznej szacuje prawdopodobieństwa i prognozy.
- Ale w jaki sposób jest uczony?
- Celem uczenia jest ustawienie wektora parametrów Θ tak, aby model oszacował wysokie prawdopodobieństwa dla pozytywnych instancji ($y = 1$) i niskie prawdopodobieństwo dla negatywnych instancji ($y = 0$).
- Pomysł ten jest realizowany przez poniższą funkcję kosztów dla pojedynczej instancji x.

$$C(\Theta) = \begin{cases} -\log(\hat{p}), & \text{if } y = 1 \\ -\log(1 - \hat{p}), & \text{if } y = 0 \end{cases}$$

Logistic Regression

$$C(\Theta) = \begin{cases} -\log(\hat{p}), & \text{if } y = 1 \\ -\log(1 - \hat{p}), & \text{if } y = 0 \end{cases}$$

- Taka funkcja kosztu ma sens, ponieważ $-log(t)$ rośnie bardzo szybko, gdy t zbliża się do 0, więc koszt będzie duży, jeśli model oszacuje prawdopodobieństwo bliskie 0 dla pozytywnej instancji, a także będzie bardzo duży, jeśli model oszacuje prawdopodobieństwo bliskie 1 dla negatywnej instancji.
- Z drugiej strony, $-log(t)$ jest bliski 0, gdy t jest bliskie 1, więc koszt będzie bliski 0, jeśli szacowane prawdopodobieństwo jest bliskie 0 dla negatywnej instancji lub bliskie 1 dla pozytywnej instancji.

Logistic Regression

Funkcja kosztów dla całych danych, to po prostu średni koszt wszystkich instancji szkoleniowych.
Można ją napisać w jednym wyrażeniu i nazywa się ją **log loss**:

$$J(\theta) = -\frac{1}{m}\sum_{i=0}^{m} y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)})$$

- `https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z02_LogisticRegression_iris.ipynb`
- `https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z03_LogisticRegression_breast_cancer.ipynb`

Logistic Regression

- Zła wiadomość jest taka, że nie ma jawnego wzoru na rozwiązanie tego problemu i uzyskanie wartości Θ, która minimalizuje tę funkcję kosztu.
- Ale dobrą wiadomością jest to, że ta funkcja kosztu jest wypukła, więc Gradient Descent (lub jakikolwiek inny algorytm optymalizacyjny) gwarantuje znalezienie globalnego minimum.

Klasyfikacja binarna

- Ocena klasyfikatora jest często znacznie trudniejsza niż ocena regresora.
- Dostępnych jest wiele miar wydajności oceny klasyfikatorów.
- Dobrym sposobem oceny modelu jest użycie **cross-validacji** .

Confusion Matrix

- Znacznie lepszym od **Accuracy (ACC)** sposobem oceny wydajności klasyfikatora jest przyjrzenie się **confusion matrix**.
- Każdy rząd **confusion matrix** reprezentuje rzeczywistą klasę, podczas gdy każda kolumna reprezentuje przewidywaną klasę.
- Pierwszy rząd tej macierzy uwzględnia obrazy inne niż 5 (klasa negatywna):
  - ile z nich zostało poprawnie sklasyfikowanych jako non-5 (**true negatives**),
  - ile zostało błędnie sklasyfikowane jako 5s (**false positives**).
- Drugi rząd uwzględnia obrazy 5s (klasa pozytywna):
  - ile zostały błędnie sklasyfikowane jako nie-5 (**false negatives**),
  - ile zostały poprawnie sklasyfikowane jako 5s (**true positives**).

Confusion Matrix

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z03a_cross_val_score.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z04_classification_metrics.ipynb

Confusion Matrix

- **Confusion matrix** dostarcza wielu informacji, ale bardziej przydatne są miary numeryczne.
- **Precision** to dokładność pozytywnych przewidywań – precyzja klasyfikatora:

$$precision = \frac{TP}{TP + FP}$$

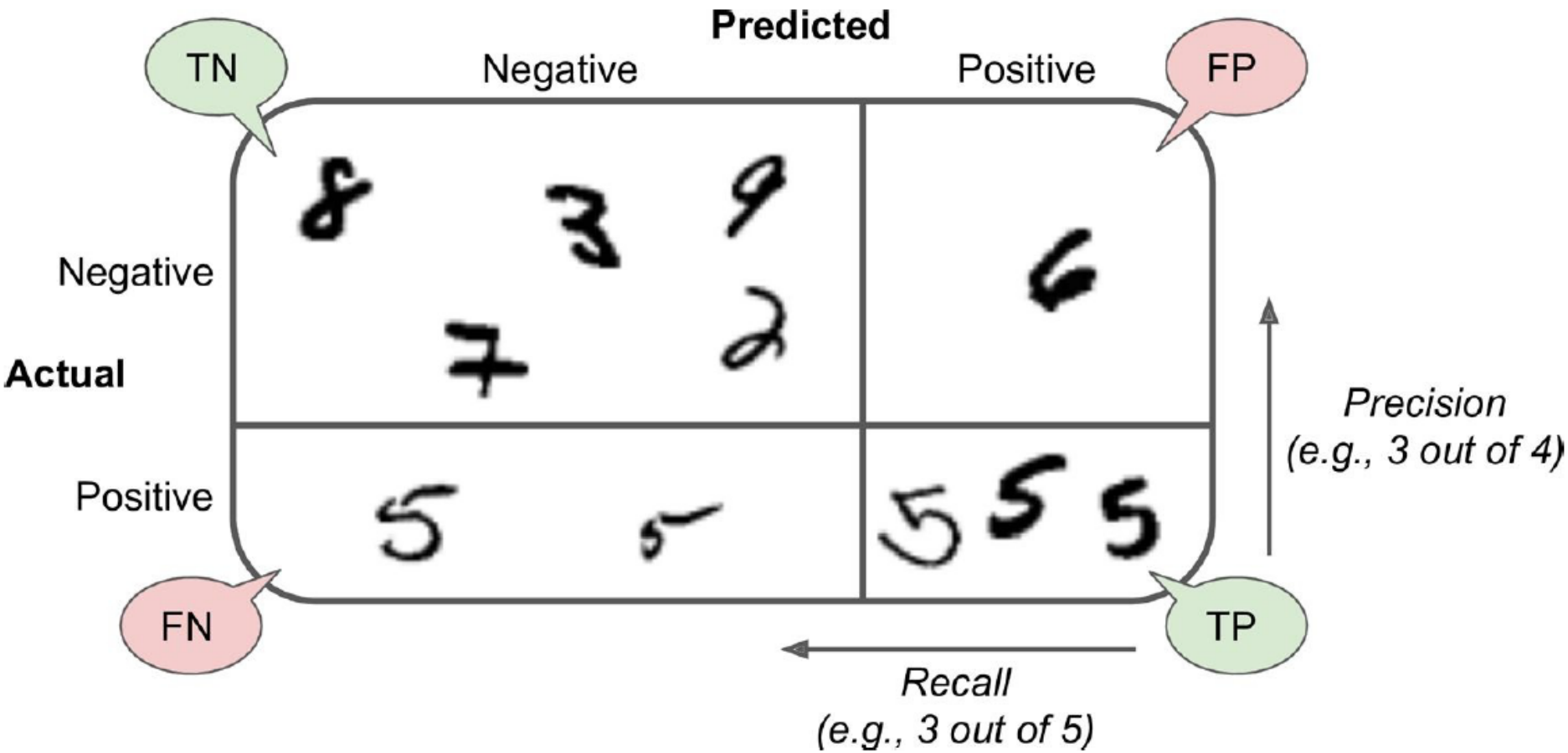
- Prostym sposobem na uzyskanie doskonałej precyzji jest wykonanie jednej pozytywnej prognozy i zapewnienie jej poprawności (precision = $1/1 = 100\%$).
- Nie byłoby to zbyt użyteczne, ponieważ klasyfikator zignorowałby wszystkie, oprócz jednego pozytywnego wystąpienia.

Confusion Matrix

- Precyzję zwykle stosuje się wraz z inną miarą o nazwie **recall**, zwaną również czułością **sensitivity**.
- **Recall** jest to stosunek pozytywnych instancji, które są poprawnie wykrywane przez klasyfikator:

$$recall = \frac{TP}{TP + FN}$$

Confusion Matrix



F1 score

- Często wygodnie jest łączyć **precision** i **recall** w pojedynczą metrykę, zwaną **F1 score** w szczególności jeśli potrzebujesz prostego sposobu na porównanie dwóch klasyfikatorów.
- Wynik **F1 score** jest średnią harmoniczną **precision** i **recall**.

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2\frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

- Podczas gdy klasyczna średnia traktuje wszystkie wartości w równym stopniu, **średnia harmoniczna** nadaje znacznie większą wagę mniejszym wartością.
- W rezultacie klasyfikator uzyska wysoki wynik **F1 score**, jeśli zarówno **precision** i **recall** będą wysokie.

https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z05_LogisticRegression_breast_cancer_metrics.ipynb

Regularyzacja

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z06_LogisticRegression_regularization.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z07_LogisticRegression_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z08_LogisticRegression_feature_selection_wine.ipynb

Klasyfikatory wieloklasowe

Klasyfikatory wieloklasowe

- Podczas gdy klasyfikatory binarne rozróżniają dwie klasy, klasyfikatory wieloklasowe (Multiclass Classification) mogą rozróżniać więcej niż dwie klasy.
- Niektóre algorytmy (takie jak Random Forest czy naive Bayes classifiers) mogą obsługiwać wiele klas bezpośrednio.
- Inne (takie jak Support Vector Machine lub Linear classifiers) są wyłącznie klasyfikatorami binarnymi.
- Istnieją jednak różne strategie, za pomocą których można wykonywać klasyfikację wieloklasową za pomocą wielu klasyfikatorów binarnych.

Klasyfikatory wieloklasowe - one-versus-all (OvA)

- Jednym ze sposobów stworzenia systemu, który może klasyfikować obrazy cyfr do 10 klas (od 0 do 9), jest nauczenie 10 klasyfikatorów binarnych, po jednym dla każdej cyfry (wykrywacz 0, wykrywacz 1, wykrywacz 2, i tak dalej).
- Następnie, gdy chcesz sklasyfikować obraz, otrzymasz wynik decyzji z każdego klasyfikatora dla tego obrazu i wybierasz klasę, której klasyfikator generuje najwyższy score.
- Nazywa się to strategią **one-versus-all (OvA)** lub **one-versus-the-rest**.

Klasyfikatory wieloklasowe - one-versus-one (OvO)

- Inną strategią jest nauczenie binarnego klasyfikatora dla każdej pary cyfr: jednej dla odróżnienia 0 od 1, drugiej dla odróżnienia 0 od 2 i tak dalej.
- Nazywa się to strategią **one-versus-one (OvO)**.
- Jeśli istnieje N klas, musisz nauczyć

$$N \times (N-1)/2$$

  klasyfikatorów .
- W przypadku problemu MNIST oznacza to uczenie 45 klasyfikatorów binarnych!
- Jeśli chcesz sklasyfikować obraz, musisz odpytać 45 klasyfikatorów i zobaczyć, która klasa wygrywa najwięcej razy.
- Główną zaletą OvO jest to, że każdy klasyfikator uczymy tylko dla dwóch klas (tylko część danych).

https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z09_multinomial_classification.ipynb

Softmax Regression

Softmax Regression

- Model regresji logistycznej można uogólnić tak, aby obsługiwał wiele klas bezpośrednio, bez konieczności uczenia i łączenia wielu klasyfikatorów binarnych.
- Nazywa się to **Regresją Softmax** lub **Multinomial Logistic Regression**.

Softmax Regression

- Pomysł jest dość prosty: gdy poda się instancję x, model regresji Softmax najpierw oblicza wynik $s_k(x)$ dla każdej klasy k, a następnie szacuje prawdopodobieństwo każdej klasy przez zastosowanie funkcji softmax (zwanej również **normalized exponential**).
- Równanie $s_k(x)$ wygląda znajomo, ponieważ jest ono podobne do równania z regresji logistycznej

$$s_k(x) = \Theta_k^T x$$

- Zauważ, że każda klasa ma swój własny wektor parametrów Θ_k.
- Wszystkie te wektory są zwykle przechowywane jako wiersze w macierzy parametrów Θ.

Softmax Regression

Po obliczeniu wyniku każdej klasy dla instancji x, można oszacować prawdopodobieństwo $\hat{p}_k$, że instancja należy do klasy k za pomocą funkcji softmax:

- oblicza wartość funkcji exp na każdym punkcie,
- następnie normalizuje je (dzieląc przez sumę):

$$\hat{p}_k = \sigma(s(x))_k = \frac{\exp s_k(x)}{\sum_{j=1}^{K} \exp s_j(x)}$$

- K to liczba klas.
- $s(x)$ to wektor zawierający wyniki dla każdej klasy dla instancji x.
- $\sigma(s(x))_k$ jest szacowanym prawdopodobieństwem, że instancja x należy do klasy k, biorąc pod uwagę wyniki każdej klasy dla tej instancji.

Softmax Regression

Podobnie jak w Regresji logistycznej, klasyfikator Regresji Softmax przewiduje klasę jako tą, która posiada najwyższe prawdopodobieństwo (która jest po prostu klasą o najwyższym wyniku):

$$\hat{y} = \text{argmax}_k(\sigma(s(x))_k) = \text{argmax}_k(s_k(x)).$$

Softmax Regression

Softmax Regression minimalizacja funkcji kosztowej zwaną **entropią krzyżową**:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$$

gdzie $y_k^{(i)}$ jest równe 1, jeśli klasa docelowa dla i-tego wystąpienia wyniesie k, a w przeciwnym razie jest równa 0.

- **Entropia krzyżowa** penalizuje model, gdy szacuje niskie prawdopodobieństwo dla klasy docelowej.
- **Entropia krzyżowa** jest często używana do mierzenia, jak dobrze zestaw oszacowanych prawdopodobieństw klas jest zgodny z klasami docelowymi.

UWAGA

Entropia krzyżowa między dwoma rozkładami prawdopodobieństw p i q jest zdefiniowana jako:

$$H^x(p, q) = -\sum_x p(x) \log(q(x)),$$

(gdy rozkłady są dyskretne).

https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z10_Softmax_Regression.ipynb

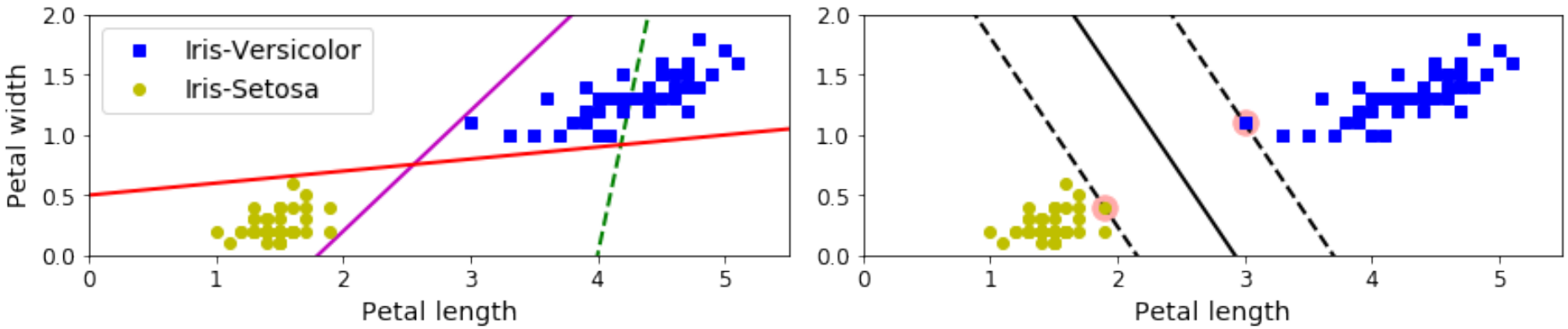
Support Vector Machines (SVM)

Support Vector Machines

- Support Vector Machines (SVM) jest bardzo wydajnym i wszechstronnym modelem uczenia maszynowego, zdolnym do wykonywania liniowej lub nieliniowej klasyfikacji, regresji, a nawet wykrywania elementów odstających.
- SVM są szczególnie dobrze przystosowane do klasyfikacji złożonych, ale małych lub średnich zbiorów danych.

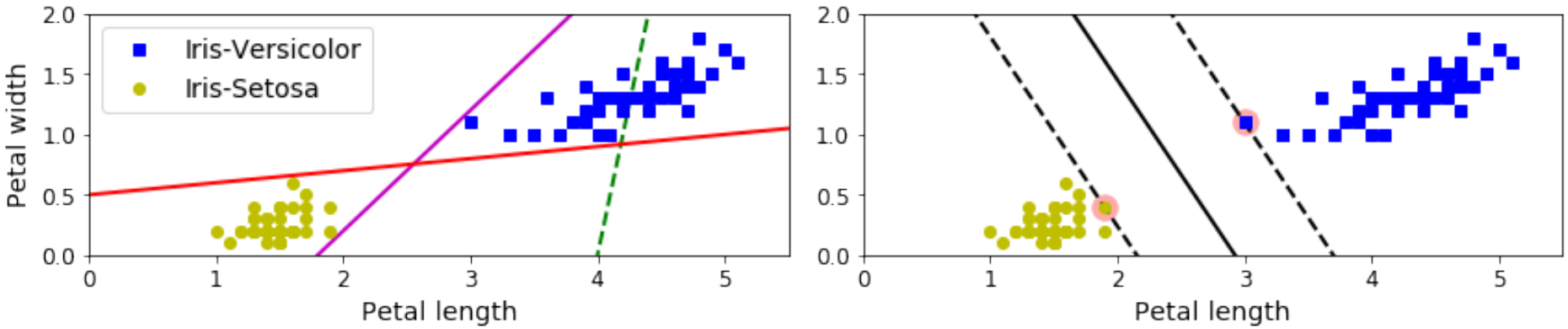
Linear SVM Classification

- Podstawową ideę SVM najlepiej wyjaśnić na danych iris. Rozważmy dwie klasy, które łatwo rozdzielić za pomocą linii prostej (zbiór jest liniowo separowalny).
- Lewy wykres pokazuje granice decyzji trzech możliwych klasyfikatorów liniowych.



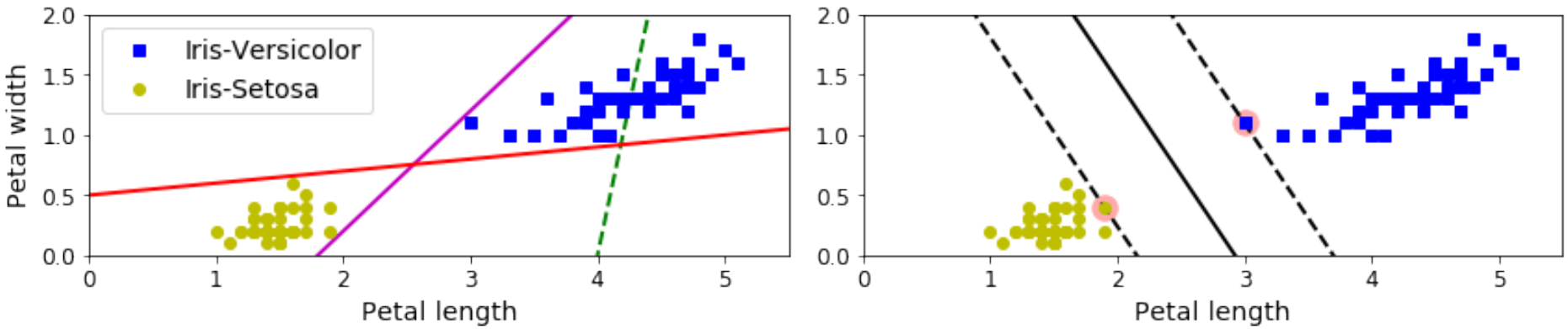
Support Vector Machines (SVM)

- Model, którego granica decyzyjna jest reprezentowana przez linię przerywaną, jest tak zły, że nawet nie rozdziela prawidłowo klas.
- Pozostałe dwa modele doskonale sprawdzają się na tym zestawie treningowym, ale granice ich decyzji są tak bliskie przypadkom, że modele te prawdopodobnie nie będą działać tak dobrze na nowych instancjach.



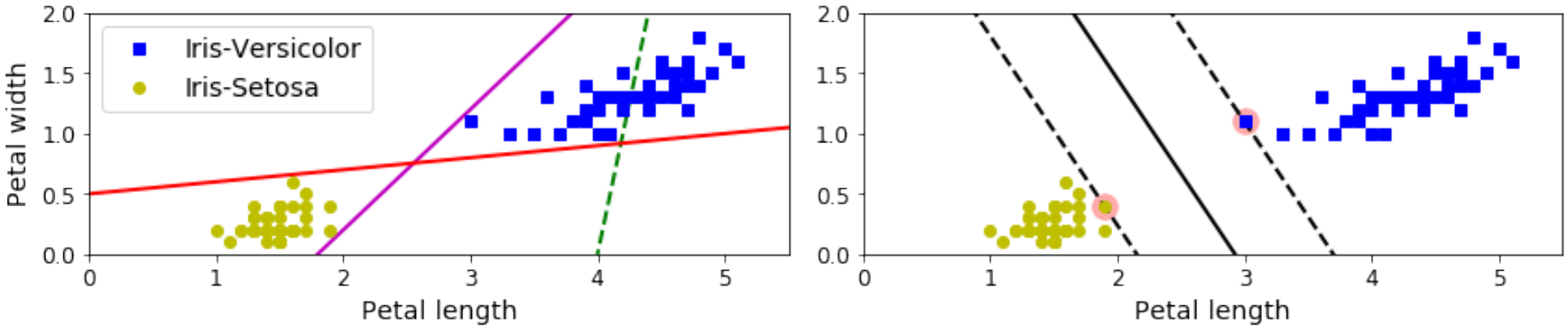
Support Vector Machines (SVM)

- Natomiast linia ciągła na wykresie po prawej stronie reprezentuje granicę decyzyjną klasyfikatora SVM.
- Linia ta nie tylko oddziela dwie klasy, ale także pozostaje jak najdalej od najbliższych instancji szkoleniowych.
- Możesz myśleć o klasyfikatorze SVM jako dopasowującym najszerszy margines (reprezentowaną przez równoległe przerywane linie) pomiędzy klasami.
- Nazywa się to klasyfikacją dużych marginesów.



Support Vector Machines (SVM)

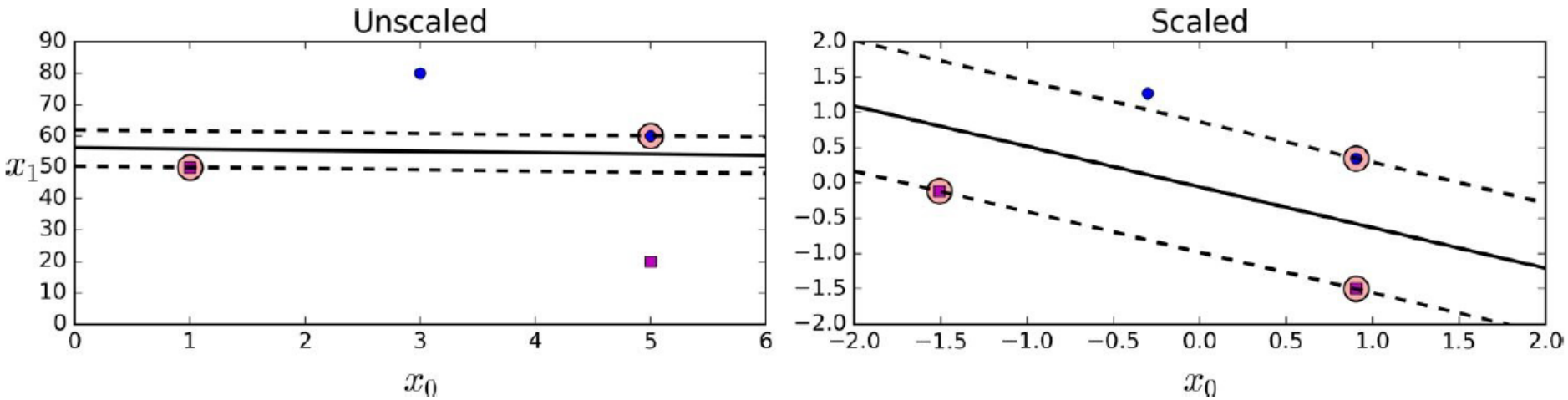
Zauważ, że dodanie większej liczby instancji szkoleniowych "poza marginesem"w ogóle nie wpłynie na granicę decyzyjną: jest w pełni określone przez instancje znajdujące się granicy marginesu. Te przypadki nazywane są **wektorami nośnymi**.



Support Vector Machines (SVM)

SVM są wrażliwe na skalowanie współrzędnych jak widać na rysunku:

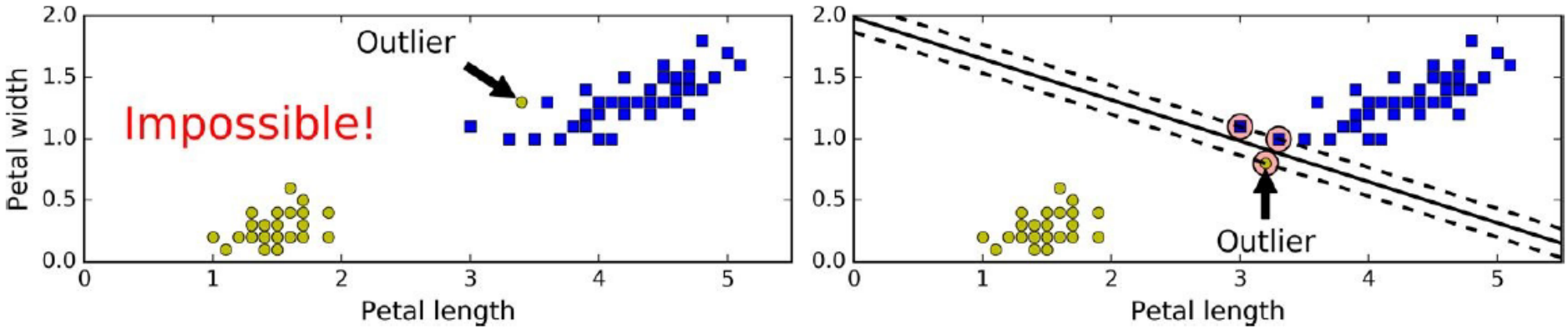
- na lewym wykresie skala pionowa jest znacznie większa niż skala pozioma, więc najszerszy margines jest wyznaczany przez granicę horyzontalną.
- po skalowaniu granice decyzji wyglądają znacznie lepiej (na prawym wykresie).



Support Vector Machines (SVM)

Jeśli ściśle narzucimy, że wszystkie elementy muszą się znajdować poza marginesem i po prawidłowej stronie, to nazywa się to klasyfikacją twardą **hard margin classification**.
Istnieją dwa główne problemy z klasyfikacją twardego marginesu.

- Po pierwsze, działa tylko wtedy, gdy dane są liniowo separowalne.
- Po drugie jest dość wrażliwy na wartości odstające.



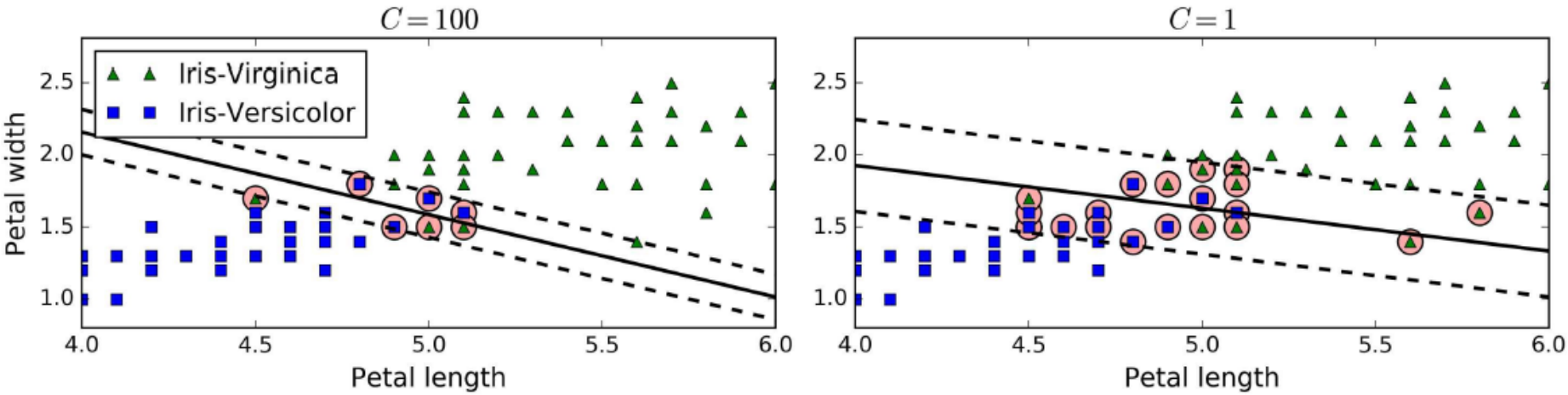
Support Vector Machines (SVM)

- Aby uniknąć tych problemów, lepiej jest użyć bardziej elastycznego modelu.
- Celem jest znalezienie właściwej równowagi między utrzymaniem jak największego marginesu i ograniczeniem naruszenia marginesów (tj. przypadków, które znajdą się wewnątrz marginesu, a nawet po niewłaściwej stronie).
- Nazywa się to **soft margin classification**.

Support Vector Machines (SVM)

W Scikit-Learn możesz kontrolować to obciążenie za pomocą parametru C:

- mniejsza wartość C prowadzi do szerszego marginesu, a co za tym idzie większej ilości naruszeń marginesów.



Support Vector Machines (SVM)

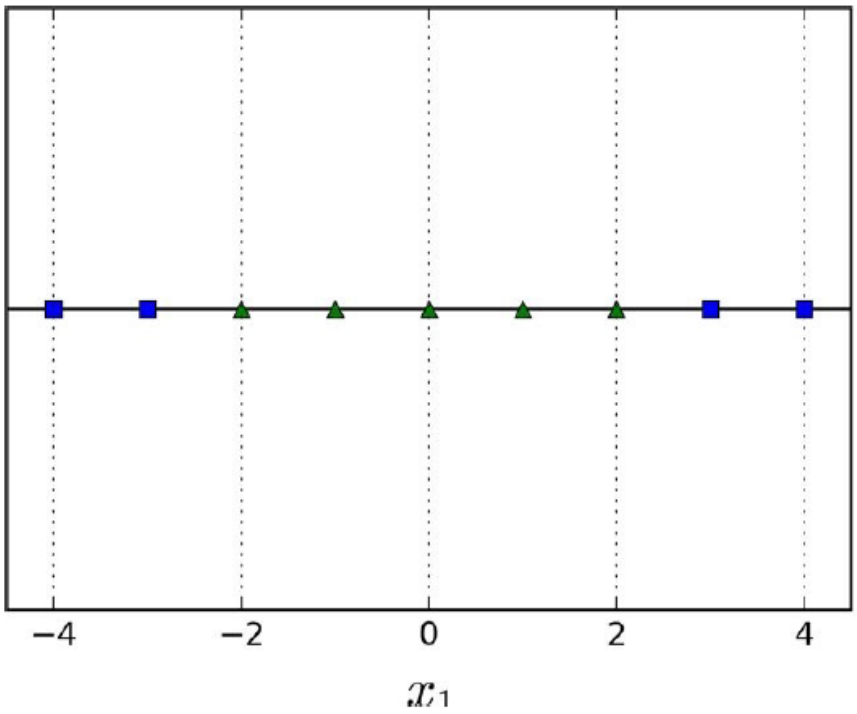
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D01_Z02_linear_SVM.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D02_Z02_linear_SVM.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D02_Z03_SVM_scale.ipynb

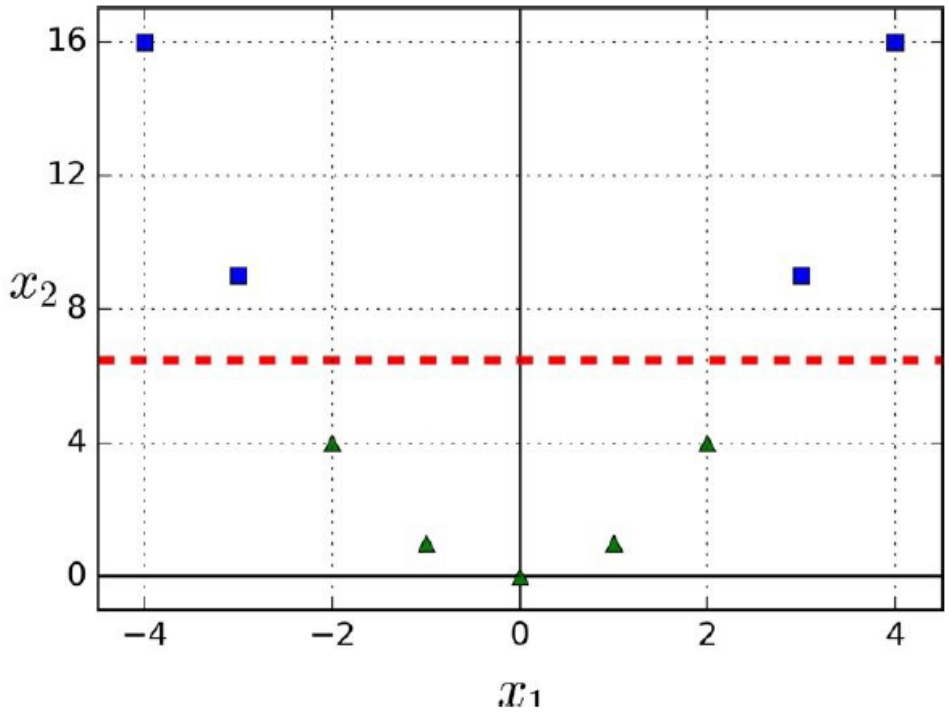
Nonlinear SVM Classification

- Chociaż liniowe klasyfikatory SVM są wydajne i działają zaskakująco dobrze, w wielu przypadkach dane nie są liniowo separowalne.
- Jednym ze sposobów radzenia sobie z nieliniowymi zestawami danych jest dodanie większej liczby współrzędnych, w niektórych przypadkach może to prowadzić do odseparowania liniowego zbioru danych.

Nonlinear SVM Classification

- Rozważ lewy wykres, który reprezentuje prosty zbiór danych z jedną cechą x_1.
- Ten zestaw danych nie jest liniowo separowalny.
- Ale jeśli dodasz drugą wapórzedną $x_2 = x_1^2$, wynikowy zestaw danych 2D będzie idealnie liniowo separowalny.





Nonlinear SVM Classification

Dodawanie funkcji wielomianowych jest proste w implementacji i może współpracować z wszystkimi rodzajami algorytmów uczenia maszynowego (nie tylko z maszynami SVM), ale przy:

- niskim stopniu wielomianu nie radzi sobie z bardzo złożonymi zestawami danych,
- a przy wysokim stopniu wielomianu tworzy ogromną liczbę funkcję, co sprawia, że model jest zbyt wolny.

Kernel trick

- Na szczęście w SVM-ach można zastosować niemal cudowną technikę zwaną **kernel trick**.
- Pozwala uzyskać taki sam wynik, jak w przypadku dodania wielu funkcji wielomianowych, nawet w przypadku wielomianów o bardzo wysokim stopniu, bez konieczności ich dodawania.
- Nie ma więc kombinatorycznej eksplozji liczby funkcji, ponieważ w rzeczywistości nie dodaje się żadnych funkcji.

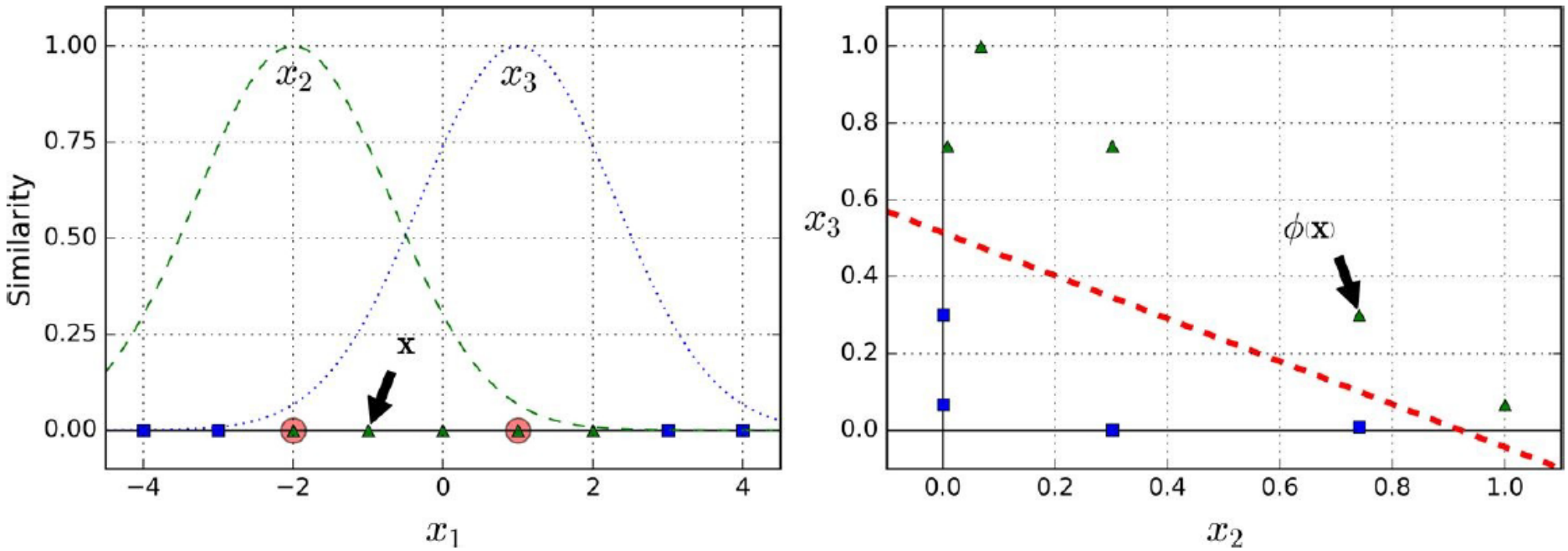
Kernel trick

Określmy funkcję podobieństwa za pomocą **Gaussian Radial Basis Function (RBF)** z parametrem γ:

$$\phi(x, y) = \exp(-\gamma \|x - y\|^2)$$

Kernel trick

- Weźmy zbiór danych (1D) i dodajmy do niego dwa punkty $x_2 = -2$ i $x_3 = 1$ (patrz lewy wykres).
- Następnie określmy funkcję podobieństwa Gaussian Radial Basis Function (RBF) z $\gamma = 0.3$.

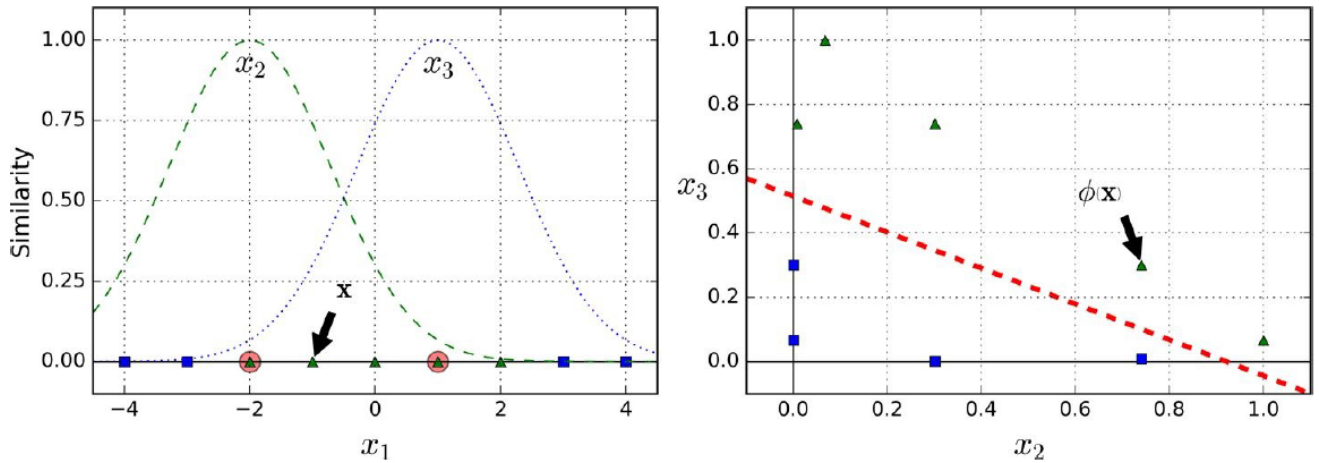


Kernel trick

- Teraz jesteśmy gotowi wyznaczyć nową funkcję.
- Na przykład spójrzmy na instancję $x = -1$: znajduje się ona w odległości 1 od pierwszego punktu orientacyjnego, a 2 od drugiego punktu orientacyjnego.
- Dlatego jego nowe cechy to:

$$x2 = \exp(-0.3 \cdot 1^2) \approx 0.74$$

$$x3 = \exp(-0.3 \cdot 2^2) \approx 0.30$$



Support Vector Machines (SVM)

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D02_Z04_polinomina_SVM.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D02_Z05_polinomina_SVM.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D02_Z06_rbf_SVM.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D02_Z07_rbf_SVM.ipynb

SVM - wzory

Użyjemy innej konwencji, która jest wygodniejsza (i bardziej powszechna) gdy mamy do czynienia z SVM:

- b będzie oznaczać bias,
- w będzie oznaczać wagę.

SVM - wzory

Liniowy model SVM przewiduje klasę dla nowego x po prostu obliczając funkcję decyzyjną:

$$w^T x + b = w_1 x_1 + \ldots + w_n x_n + b.$$

jeśli wynik jest dodatni, przewidywana klasa $\bar{y}$ jest klasą dodatnią 1, a w przeciwnym przypadku klasą ujemną 0.

$$\hat{y} = \begin{cases} 0, & \text{if } w^T x_b < 0 \\ 1, & \text{if } w^T x_b \geq 0 \end{cases}$$

SVM - wzory

- **Płaszczyzny równoległe do granicy decyzyjnej, dla których funkcja decyzyjna jest równa 1 lub -1 nazywamy marginesami.**
- Są one równoległe i znajdują się w równej odległości od granicy decyzji, tworząc wokół niej margines.
- Uczenie liniowego klasyfikatora SVM oznacza znalezienie wartości wag w i biasu b, które sprawiają, że margines jest tak szeroki, jak to możliwe, przy jednoczesnym uniknięciu naruszenia marginesu (twardego marginesu) lub ich ograniczenia (miękkiego marginesu).

SVM - wzory

- Wektor współczynników $w = [w_1, w_2, \ldots w_n]^T$ nazywamy wektorem normalnym płaszczyzny.
- Margines powinien być jak największy.
- Szukamy takich współczynników płaszczyzny dyskryminacyjnej, aby osiągnąć maksymalny margines.
- Takich rozwiązań jest nieskończenie wiele, ponieważ jest nieskończenie wiele wektorów normalnych do danej płaszczyzny (różnią się długością).
- **Im mniejszy wektor wag w, tym większy margines**

SVM - wzory

UWAGA.

Im mniejszy wektor wag w, tym większy margines

- Dlatego chcemy zminimalizować $\|w\|$, aby uzyskać duży margines.
- Jeśli jednak chcemy również uniknąć naruszenia marginesu (twardego marginesu), potrzebujemy funkcji decyzyjnej, która jest większa niż 1 dla wszystkich pozytywnych instancji szkoleniowych i mniejszej niż -1 dla negatywnych instancji szkoleniowych. Jeśli zdefiniujemy:
  - $t^{(i)} = -1$ dla negatywnych instancji (jeśli $y^{(i)} = 0$),
  - $t^{(i)} = 1$ dla pozytywnych instancji (jeśli $y^{(i)} = 1$),

  wówczas otzrymamy ograniczenie jako:

$$t^{(i)}(w^T x^{(i)} + b) \geq 1$$

  dla wszystkich przypadków.

SVM - wzory

Możemy zatem wyrazić funkcję kosztu SVM o twardym marginesie jako problem optymalizacyjny:

$$\text{minimize } _{w,b} \frac{1}{2}\|w\|^2 = \frac{1}{2} w^T w$$

$$\text{subject to } t^{(i)}(w^T x^{(i)} + b) \geq 1$$

SVM - wzory

Aby uzyskać funkcję kosztu dla miękkiego marginesu, musimy wprowadzić zmienną $\varepsilon^{(i)} \geq 0$ dla każdej instancji.
Ogólnie mamy:

- gdy $\varepsilon^{(i)} = 0$, $\mathbf{x}^{(i)}$ jest klasyfikowany poprawnie i leży poza marginesem lub na jego granicy,
- gdy $0 < \varepsilon^{(i)} < 1$, $\mathbf{x}^{(i)}$ jest klasyfikowany poprawnie, ale leży w marginesie,
- gdy $\varepsilon^{(i)} > 1$, $\mathbf{x}^{(i)}$ jest klasyfikowany błędnie.

SVM - wzory

Mamy obecnie dwa sprzeczne cele:

- zmniejszanie $\varepsilon^{(i)}$ jak to tylko możliwe, w celu zmniejszenia naruszenia marginesu,
- uczynienie $\|w\|$ tak małym jak to możliwe, w celu zwiększenia marginesu.

Tutaj pojawia się hiperparametr C: pozwala nam to określić kompromis między tymi dwoma celami.

SVM - wzory

Daje nam to problem optymalizacyjny:

$$\text{minimize } _{w,b,\varepsilon} = \frac{1}{2} w^T w + C \sum_{i=1}^{m} \varepsilon^{(i)}$$

$$\text{subject to } t^{(i)}(w^T x^{(i)} + b) \geq 1 - \varepsilon^{(i)}$$

$$\text{and subject to } \varepsilon^{(i)} \geq 0 \text{ for } i = 1, \ldots, m$$

SVM - wzory (dual problem)

- Biorąc pod uwagę ograniczony problem optymalizacji, znany jako **primal problem**, można wyrazić inny ale blisko związany problem, zwany **dual problem**.
- Rozwiązanie problemu dualnego zazwyczaj daje ograniczenie dolne.
- W pewnych warunkach może nawet mieć te same rozwiązania, co pierwotny problem. Na szczęście problem SVM spełnia te warunki,

SVM - wzory (dual problem)

$$\text{minimize}_{\alpha} = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}x^{(i)T}x^{(j)} + \sum_{i=1}^{m}\alpha^{(i)}$$

$$\text{subject to } \alpha_i \geq 0 \text{ for } i = 1, \ldots, m$$

Po znalezieniu wektora, który minimalizuje to równanie, można obliczyć $\hat{w}$ i $\hat{b}$, minimalizujące podstawowy problem SVM za pomocą:

$$\hat{w} = \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}x^{(i)}$$

$$\hat{b} = \frac{1}{n_s}\sum_{i=1;\ \hat{\alpha}^{(i)}>0}^{m}(1 - t^{(i)}(w^T x^{(i)}))$$

SVM - wzory (Kernelized SVM)

Definiujemy kernele:

- Linear
$$K(x, y) = x^T y$$
- RBF
$$K(x, y) = \exp(-\gamma \|x - y\|^2)$$
- ...

i zamieniamy w powyższym równaniu:

$$x^{(i)T} x^{(j)}$$

na

$$K(x^{(i)} x^{(j)})$$

Model Evaluation and Improvement

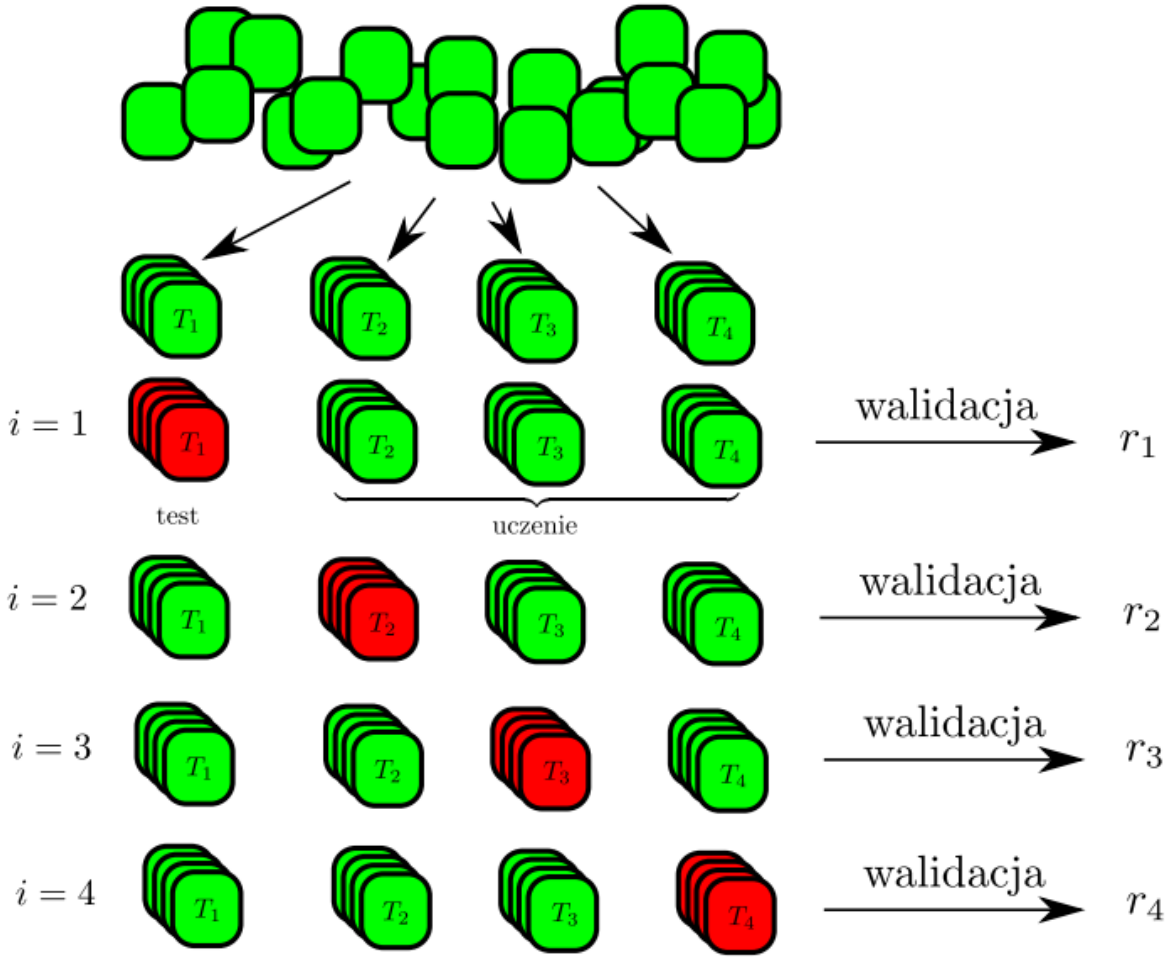
Model Evaluation and Improvement

- **Cross-validation** to statystyczna metoda oceny modelu, która jest bardziej stabilna i dokładna, niż przy użyciu podziału na train/test.
- Podczas **Cross-validation** dane są wielokrotnie dzielone, a na podziałach uczymy wiele modeli.
- Najczęściej używaną wersją **Cross-validation** jest **k-fold cross-validation**, gdzie k określa się zwykle 5 lub 10.
- Podczas wykonywania **five-fold cross-validation,** dane są najpierw dzielone na pięć części (jeśli się da) równej wielkości, zwanych **foldami**.

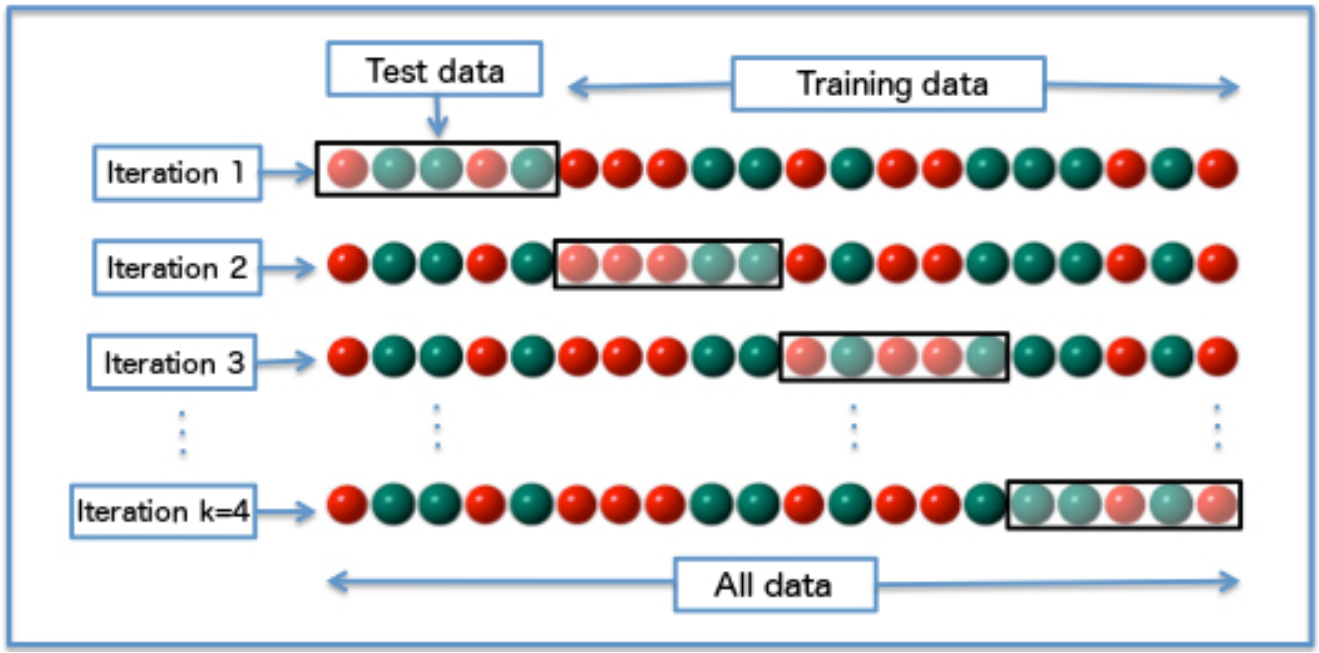
Model Evaluation and Improvement

- Następnie uczona jest sekwencja modeli.
- Pierwszy model jest uczony przy użyciu pierwszego **foldama** jako zestawu testowego, a pozostałe **foldy** (2-5) są używane jako zestaw treningowy.
- Model jest budowany z wykorzystaniem danych w **foldów** 2-5, a następnie dokładność jest oceniana w **foldzie** 1.
- Następnie budowany jest inny model, tym razem przy użyciu **folda** 2 jako zestawu testowego i danych w **foldach** 1, 3, 4 i 5 jako zestawu treningowy.
- Ten proces powtarza się, stosując **foldy** 3, 4 i 5 jako zestawy testowe.
- Dla każdego z tych pięciu rozdziałów danych na train/test obliczamy dokładność modelu.

Model Evaluation and Improvement



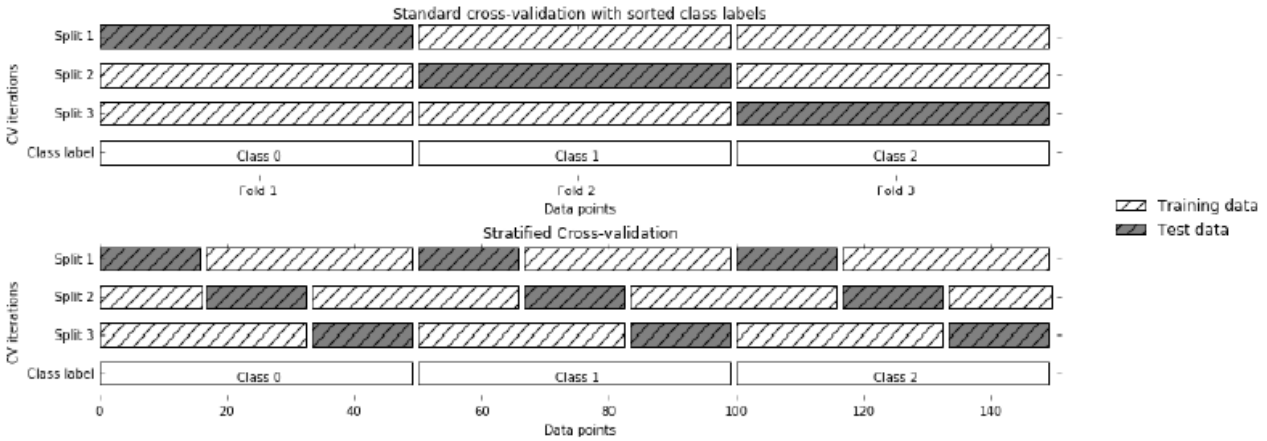
Model Evaluation and Improvement



Stratified k-Fold Cross-Validation

- Ponieważ prosta strategia **k-Fold Cross-Validation** może doprowadzić do podziału, w którym w jednym foldzie znajduje się tylko jedna klasa;
- `scikit-learn` nie używa **k-Fold Cross-Validation** do klasyfikacji, ale raczej używa **stratified k-fold cross-validation**,
- w **stratified k-fold cross-validation** dzielimy dane tak, aby proporcje między klasami były takie same w każdym foldzie, jak w całym zbiorze danych.

Stratified k-Fold Cross-Validation



Na przykład, jeśli 90% twoich próbek należy do klasy A, a 10% twoich próbek należy do klasy B, to wszystkie foldy zawierają 90% próbek należących do klasy A, a 10% próbek należących do klasy B.

Leave-one-out cross-validation

- Inną często używaną metodą sprawdzania poprawności modelu jest metoda **Leave-one-out cross-validation**.
- Możesz myśleć o **Leave-one-out cross-validation** jako o klasycznej metodzie **k-Fold Cross-Validation**, gdzie każdy fold jest pojedynczą próbką.
- Dla każdego podziału wybierasz jeden punkt danych, który ma być zestawem testowym.
- Może to być bardzo czasochłonne, szczególnie w przypadku dużych zestawów danych, ale czasami zapewnia lepsze wyniki.

Cross-validation with groups

- Innym bardzo powszechnym podejściem do cross-validacji jest sytuacja, gdy w danych są bardzo powiązane grupy.
- Załóżmy, że chcesz zbudować system rozpoznawania emocji na podstawie zdjęć twarzy i zbierasz zestaw danych przedstawiający zdjęcia 100 osób, w których każda osoba jest wielokrotnie użyta, pokazując różne emocje.
- Celem jest zbudowanie klasyfikatora, który może poprawnie identyfikować emocje osób nie znajdujących się w zbiorze danych.
- Możesz użyć **Stratified k-Fold Cross-Validation** aby zmierzyć wydajność klasyfikatora.

Cross-validation with groups

- Jest jednak prawdopodobne, że zdjęcia tej samej osoby będą znajdować się zarówno w train jak i w test.
- Klasyfikator będzie znacznie łatwiej wykrywać emocje na twarzy, która jest częścią zestawu uczacego, w porównaniu do zupełnie nowych twarzy.
- Aby dokładnie ocenić uogólnienie na nowe twarze, musimy w związku z tym upewnić się, że zestawy uczące i testowe zawierają obrazy różnych osób.

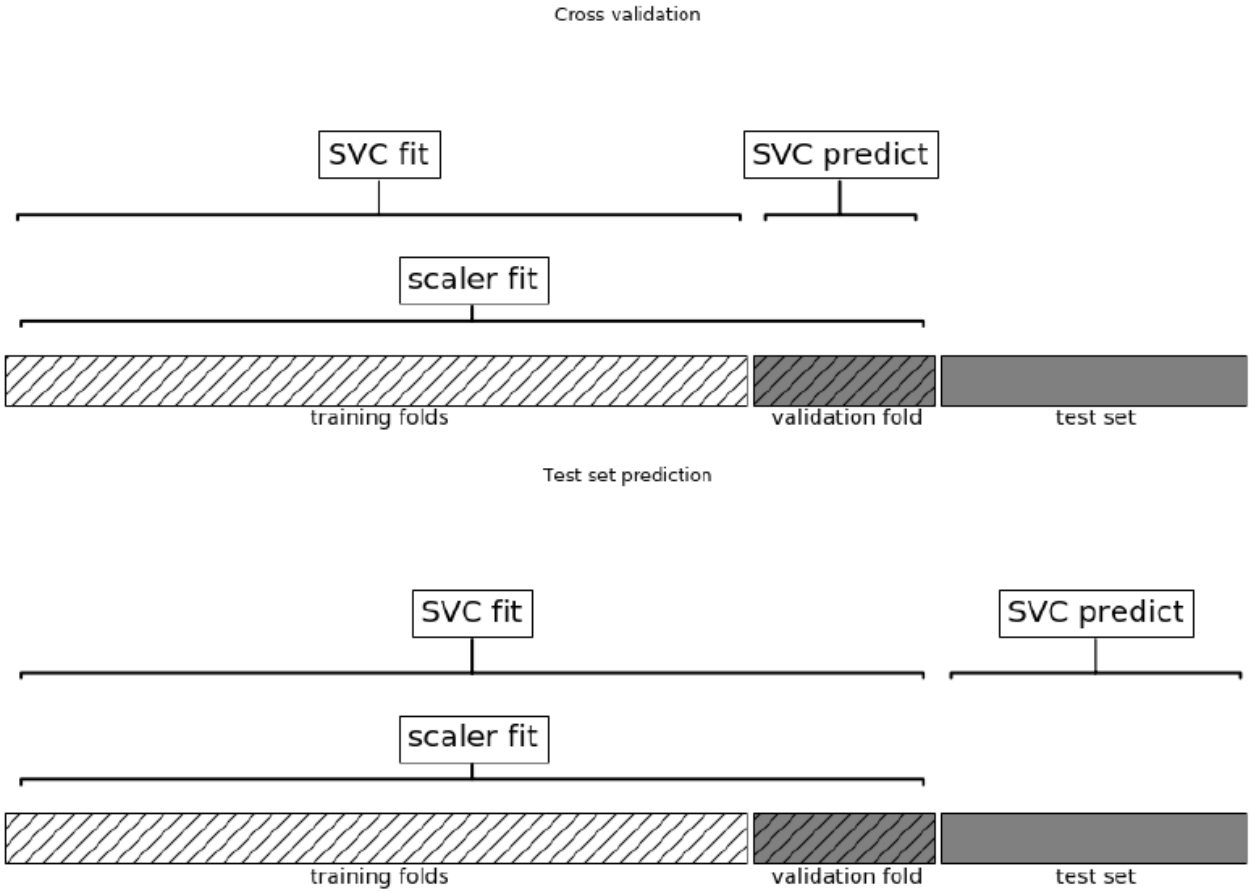
Cross-validation with groups

- Aby to osiągnąć, możemy użyć **GroupKFold**, który przyjmuje tablicę grup jako argument, który możemy wykorzystać do wskazania, która osoba jest na obrazie.
- Tablica grup wskazuje grupy w danych, które nie powinny być rozdzielane podczas tworzenia zestawów treningowego i testowego.

Cross-validation with groups

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z01_Stratified_GridSearchCV.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z02_Stratified_GridSearchCV.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z03_Stratified_GridSearchCV.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z04_double_ross_validation.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z05_double_ross_validation.ipynb

StandardScaler



StandardScaler

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z06_StandardScaler.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z07_Pipeline.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z08_Pipeline_GridSearchCV.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z09_Pipeline_GridSearchCV.ipynb

Dealing with missing data

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z10_missing_data.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z11_missing_data.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z12_missing_MNIST.ipynb

Categorical data

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z14_categorical_data.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D03_Z15_titanic.ipynb

Ocena Klasyfikatorów II

Precision/recall tradeoff

- Wynik **F1 score** faworyzuje klasyfikatory, które mają podobną **precision** i **recall**.

Precision/recall tradeoff

- Wynik **F1 score** faworyzuje klasyfikatory, które mają podobną **precision** i **recall**.
- To nie zawsze jest to, czego potrzebujesz: w niektórych kontekstach dbamy głównie o **precision**, a w innych kontekstach naprawdę zależy nam na **recall**.

Precision/recall tradeoff

- Wynik **F1 score** faworyzuje klasyfikatory, które mają podobną **precision** i **recall**.
- To nie zawsze jest to, czego potrzebujesz: w niektórych kontekstach dbamy głównie o **precision**, a w innych kontekstach naprawdę zależy nam na **recall**.
- Na przykład, jeśli wytrenowałeś klasyfikator do wykrywania filmów, które są bezpieczne dla dzieci, prawdopodobnie wolałbyś klasyfikator, który odrzuca wiele dobrych filmów (**low recall**), ale zachowuje tylko te bezpieczne (**high precision**).

Precision/recall tradeoff

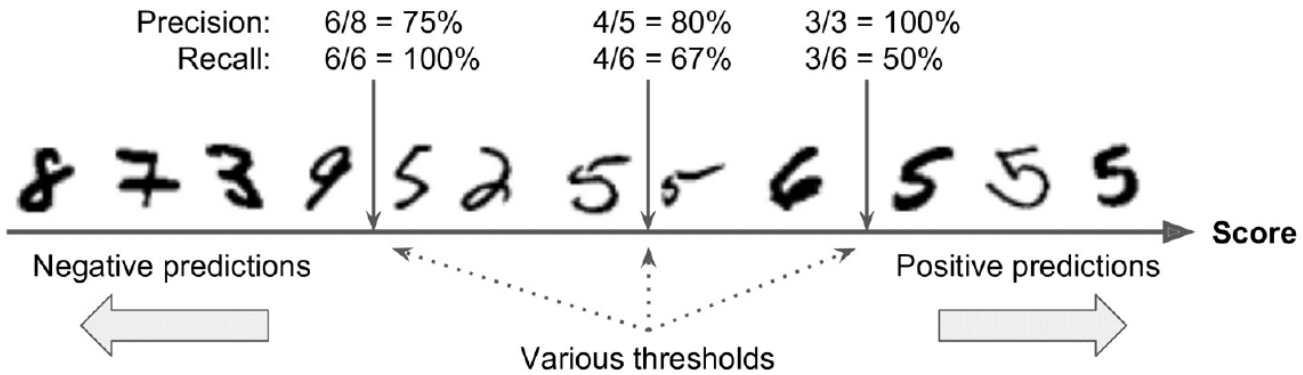
- Wynik **F1 score** faworyzuje klasyfikatory, które mają podobną **precision** i **recall**.
- To nie zawsze jest to, czego potrzebujesz: w niektórych kontekstach dbamy głównie o **precision**, a w innych kontekstach naprawdę zależy nam na **recall**.
- Na przykład, jeśli wytrenowałeś klasyfikator do wykrywania filmów, które są bezpieczne dla dzieci, prawdopodobnie wolałbyś klasyfikator, który odrzuca wiele dobrych filmów (**low recall**), ale zachowuje tylko te bezpieczne (**high precision**).
- Z drugiej strony, przypuśćmy, że uczymy klasyfikator, który wykrywa kieszonkowców na kamerach z monitoringu: prawdopodobnie dobrze jest jeśli klasyfikator ma niskie **precision** 30%, o ile ma 99% **recall** (na pewno strażnicy dostaną kilka fałszywych alarmów, ale prawie wszyscy złodzieje zostaną złapani).

Precision/recall tradeoff

- Niestety nie możesz minimalizować ich obu: zwiększenie **precision** zmniejsza **recall** i na odwrót. Nazywa się to **precision/recall tradeoff**.

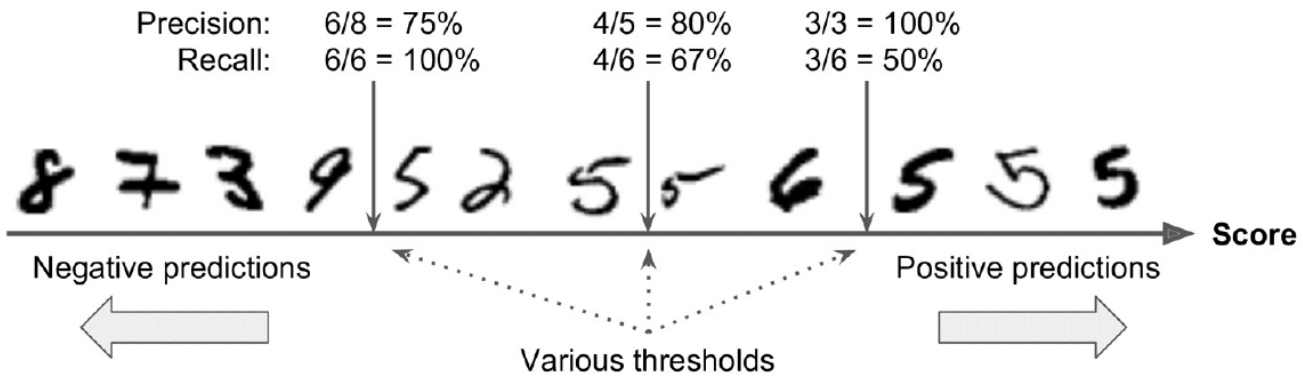
Precision/recall tradeoff

- Aby zrozumieć ten kompromis przyjrzyjmy się, w jaki sposób klasyfikator podejmuje decyzje dotyczące klasyfikacji.
- Dla każdego punktu oblicza funkcję decyzyjną, a jeśli ten wynik jest większy niż z góry ustawiony próg, przydziela ten element do klasy pozytywnej lub negatywnej.
- Rysunek pokazuje kilka cyfr umieszczonych od najniższego wyniku po lewej do najwyższego wyniku po prawej stronie.



Precision/recall tradeoff

- Przypuśćmy, że próg decyzyjny znajduje się przy centralnej strzałce (pomiędzy dwoma piątkami):
  - mamy 4 prawdziwe piątki po prawej stronie tego progu i jedną fałszywą. Dlatego przy tym progu **precision** wynosi 80% (4 na 5).
  - Ale z 6 rzeczywistych 5, klasyfikator wykrywa tylko 4, więc **recall** wynosi 67% (4 z 6).
- Teraz, jeśli podniesiesz próg (przesuń go do strzałki po prawej), szósty wynik false positive staje się true negative, tym samym zwiększając **precision** (do 100% w tym przypadku), ale jeden wynik true positive staje się false negative, zmniejszając **recall** do 50%.
- I odwrotnie, obniżenie progu zwiększa **recall** i zmniejsza **precision**.



ROC Curve

- **Receiver operating characteristic (ROC)** jest kolejnym powszechnym narzędziem stosowanym w klasyfikatorach binarnych.
- Jest to bardzo podobne do krzywej **precision/recall curve**, ale zamiast opisywać relację między precision, a recall opisuje:

sensitivity (recall) vs 1-specificity

gdzie:

$$specificity = \frac{TN}{FP + TN}$$

$$recall = \frac{TP}{TP + FN}$$

ROC Curve

- Ponownie jak wcześniej: im wyższe **recall**, tym więcej fałszywych alarmów **specificity**.
- Linia przerywana przedstawia krzywą ROC czysto losowego klasyfikatora.
- Dobry klasyfikator daje krzywą jak najdalszą od tej.

ROC Curve

Jednym ze sposobów porównywania klasyfikatorów jest pomiar area under the curve (AUC).

- Idealny klasyfikator będzie miał ROC AUC równe **1**.
- Podczas gdy klasyfikator czysto losowy będzie miał ROC AUC równe **0.5**.

`http://arogozhnikov.github.io/2015/10/05/roc-curve.html`

ROC Curve

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D04_Z01_classification_metrics_II.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D04_Z02_roc_curve.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D04_Z03_roc_curve.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D04_Z04_roc_curve.ipynb

K-Nearest Neighbors

K-Nearest Neighbors

- Algorytm k-najbliższego sąsiada **K-Nearest Neighbors** (KNN) jest szczególnie interesujący, ponieważ różni się zasadniczo od algorytmów uczenia maszynowego, które omówiliśmy do tej pory.
- KNN jest typowym przykładem leniwego ucznia (**lazy learner**).
- Nazywa się on **lazy** nie ze względu na jego pozorną prostotę, ale dlatego, że nie uczy się funkcji dyskryminacyjnej z danych treningowych, ale zapamiętuje zestaw danych treningowych.

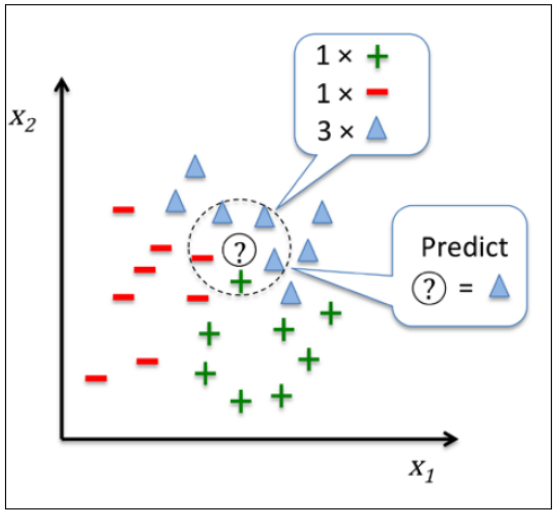
K-Nearest Neighbors

Sam algorytm KNN jest dość prosty i można go podsumować w następujący sposób:

- 1. Wybierz liczbę k i metrykę odległości.
- 2. Znajdź najbliższych sąsiadów próbki, którą chcemy sklasyfikować.
- 3. Przypisuj etykietę klasy większością głosów.

K-Nearest Neighbors

Poniższy rysunek ilustruje, w jaki sposób nowemu punktowi danych przypisuje się etykietę klasy oznaczonej trójkątem, opartą na głosowaniu większościowym spośród pięciu najbliższych sąsiadów.



Na podstawie wybranej metryki odległości algorytm KNN znajduje k próbek w zbiorze treningowym, które są najbliższe (najbardziej podobne) do punktu, który chcemy sklasyfikować. Etykieta klasy nowego punktu danych jest określana większością głosów spośród jego najbliższych sąsiadów.

K-Nearest Neighbors

- Właściwy wybór k ma kluczowe znaczenie dla znalezienia właściwej równowagi między over-fitting, a under-fitting.
- Musimy również upewnić się, że wybieramy metrykę odległości odpowiednią dla funkcji w zbiorze danych.
- Często stosuje się prostą miarę odległości Euklidesowej dla próbek o wartościach rzeczywistych, na przykład kwiaty w naszym zestawie danych Iris, które mają cechy mierzone w centymetrach.
- Jeśli jednak używamy miary odległości euklidesowej, ważne jest również standaryzowanie danych, aby każda cecha w równym stopniu przyczyniała się do odległości.

K-Nearest Neighbors

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D05_Z01_KNN.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D05_Z02_KNN_distance.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D05_Z03_KNN.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D05_Z04_KNN.ipynb

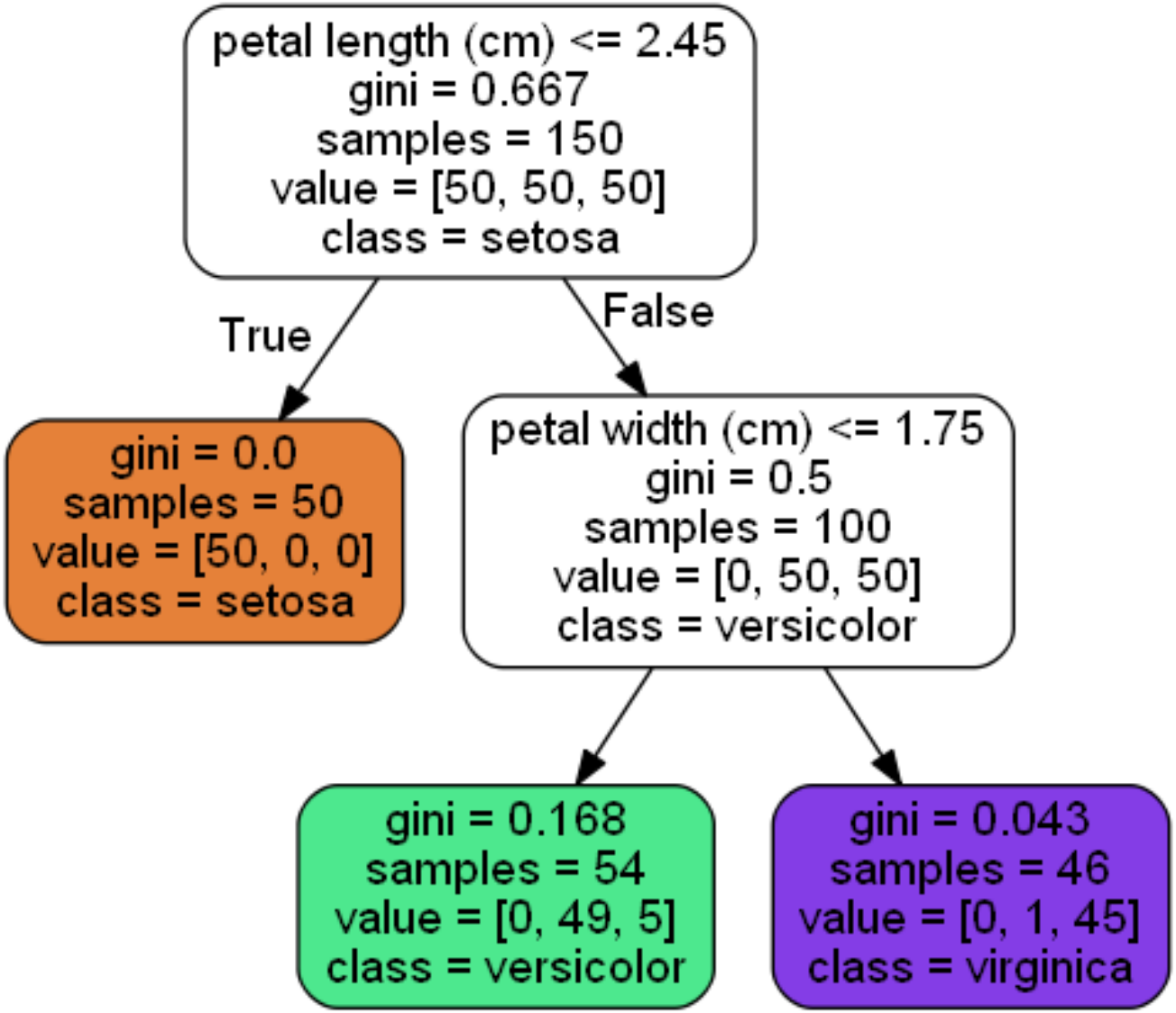
Decision Trees

Decision Trees

- Podobnie jak SVM, drzewa decyzyjne są wszechstronnymi algorytmami uczenia maszynowego, które mogą wykonywać zadania klasyfikacji i regresji.
- Aby zrozumieć drzewa decyzyjne, po prostu zbudujmy je i spójrzmy jak wygląda predykcja.

`https://github.com/przem85/bootcamp/blob/master/supervised_learning/D06_Z01_decision_trees.ipynb`
`http://www.r2d3.us/visual-intro-to-machine-learning-part-1/`

Decision Trees



Decision Trees

- Zobaczmy, jak drzewo przedstawione na rysunku tworzy prognozy.
- Załóżmy, że mamy nową instancję x.
- Zaczynasz od węzła głównego (depth 0, at the top):
  - Ten węzeł pyta, czy długość płatka kwiatu jest mniejsza niż 2,45 cm.
  - Jeśli tak, to przesuń się w dół do lewego węzła (depth 1, left).
  - W tym przypadku jest to węzeł liści (tzn. nie ma żadnych węzłów), więc nie zadaje żadnych pytań.
  - Wystarczy spojrzeć na przewidywaną klasę dla tego węzła, a drzewo decyzji przewiduje, że kwiatek jest Iris-Setosa (class=setosa).

Decision Trees

- Załóżmy teraz, że używamy innego x, ale tym razem długość płatka jest większa niż 2,45 cm.
- Musisz przejść w dół do prawego węzła podrzędnego (depth 1, right), który nie jest węzłem liścia, więc zadaje kolejne pytanie
- Czy szerokość płatka jest mniejsza niż 1,75 cm? Jeśli tak, to twój kwiat najprawdopodobniej jest Iris-Versicolor (depth 2, left).
- Jeśli nie to prawdopodobnie jest to Iris-Virginica (depth 2, right).
- To naprawdę takie proste.

Decision Trees

Atrybut węzła zawiera:

- Liczbę instancji szkoleniowych.
  Na przykład 100 przypadków treningu ma długość płatka większą niż 2,45 cm (depth 1, right), spośród których 54 mają szerokość płatka mniejszą niż 1,75 cm (depth 2, left).
- Atrybut **value** węzła informuje o liczbie instancji szkoleniowych dla każdej klasy, do której odnosi się ten węzeł. Np. dolny prawy węzeł dotyczy 0 punktów Iris-Setosa, 1 Iris-Versicolor i 45 Iris-Virginica.

Na koniec atrybut gini węzła mierzy jego "pure":

- Węzeł jest "pure"(**gini=0**)), jeśli wszystkie instancje szkoleniowe należą do tej samej klasy.
- Na przykład, ponieważ lewy węzeł odnosi się tylko do instancji szkoleniowych Iris-Setosa, jest on czysty, a jego wynik gini wynosi 0.

Równanie pokazuje, w jaki sposób algorytm treningowy oblicza wynik G_i

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

gdzie:

- $p_{i,k}$ jest stosunkiem wystąpień klasy k pomiędzy instancjami szkoleniowymi w i-tym węźle.

Na przykład, lewy dolny węzeł ma wynik gini równy:

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0,168.$$

Decision Trees

Scikit-Learn używa algorytmu CART, który generuje tylko drzewa binarne (tzn. pytania mają tylko odpowiedzi tak / nie). Jednak inne algorytmy, takie jak ID3, mogą generować drzewa decyzyjne z węzłami, które mają więcej niż dwoje dzieci.

Decision Trees

Drzewo decyzyjne mogą również oszacować prawdopodobieństwo, że instancja należy do konkretnej klasy k:

- najpierw przechodzi przez drzewo, aby znaleźć węzeł liścia dla tej instancji,
- a następnie zwraca stosunek instancji szkoleniowych klasy k w tym węźle.

Załóżmy na przykład, że znalazłeś kwiat, którego płatki mają 5 cm długości i 1,5 cm szerokości. Odpowiedni węzeł liści jest węzłem (depth-2 left node), więc drzewo decyzyjne powinno wyprowadzić następujące prawdopodobieństwa:

- 0% dla Iris-Setosa (0/54),
- 90,7% dla Iris-Versicolor (49/54),
- 9,3% dla Iris-Virginica (5/54).

Oczywiście, drzewo decyzyjne przewidzi klasę Iris-Versicolor (klasa 1), ponieważ ma największe prawdopodobieństwo.

Decision Trees

- Scikit-Learn wykorzystuje algorytm **Classification And Regression Tree (CART)** do stworzenia drzew decyzyjnych (zwanych również drzewkami 'rosnącymi').
- Pomysł jest naprawdę prosty:
  - Algorytm najpierw dzieli zestaw treningowy na dwie podgrupy za pomocą pojedynczej cechy k i progu t_k (np. "Długość płatka $< 2{,}45$ cm").
  - Jak wybiera k i t_k? Szuka pary (k, t_k), która produkuje najczystsze podzbiory (ważone przez ich wielkości).

Funkcja kosztu, którą algorytm próbuje zminimalizować, jest podana przez:

$$J(k, t_k) = \frac{m_{left}}{n} G_{left} + \frac{m_{right}}{n} G_{right}$$

gdzie:

$G_{left/right}$ impurity dla prawego i lewego podzbioru,

$m_{left/right}$ ilość elementów w prawym i lewym podzbiorze.

Decision Trees

- Po pomyślnym podzieleniu zestawu treningowego na dwie części, dzieli otrzymane podzbiory przy użyciu tej samej logiki na następne podzbiory.
- Zatrzymuje się po osiągnięciu maksymalnej głębokości (zdefiniowanej przez max_depth) lub jeśli nie może znaleźć podziału, który zmniejszy impurity.

Decision Trees

Jak widać, CART jest algorytmem zachłannym **greedy algorithm**: wyszukuje on optymalny podział tylko na najwyższym poziomie, a następnie powtarza proces na każdym kroku. Nie sprawdza, czy podział doprowadzi do najniższego wyniku.

Decision Trees

- Domyślnie używana jest miara Gini (impurity measure), ale można wybrać **entropię**, ustawiając parametr criterion = 'entropy'.
- Entropia zestawu wynosi zero, gdy zawiera instancje tylko jednej klasy.

$$H_i = - \sum_{k=1,\ p_{i,k} \neq 0}^{n} p_{i,k} \log(p_{i,k})$$

Więc czy powinieneś użyć Gini czy entropii?

Prawda jest taka, że w większości przypadków nie robi to dużej różnicy i oba podejści prowadzą do podobnych drzew.

Gini ma tendencję do izolowania najliczniejszych klas w dwóch gałęzi drzewa, podczas gdy entropia ma tendencję do wytwarzania nieco bardziej zrównoważonych drzew.

Decision Trees

- Drzewa decyzyjne zawierają bardzo niewiele założeń dotyczących danych uczących (w przeciwieństwie do modeli liniowych, które oczywiście zakładają, że dane są liniowe).
- Jeśli nie dodamy dodatkowych ograniczeeń drzewo decyzyjne zowerfituje.
- Aby uniknąć przeuczenia musisz ograniczyć swobodę drzewa decyzyjnego podczas treningu.
- Jak już wiesz, nazywa się to regularyzacją.
- Hiperparametry regularyzacji zależą od zastosowanego algorytmu, ale zazwyczaj można przynajmniej ograniczyć maksymalną głębokość drzewa decyzyjnego. W Scikit-Learn kontrolowane jest to przez **max_depth** (domyślna wartość to None, co oznacza nieograniczoną liczbę).
- Zmniejszenie max_depth spowoduje regularyzację modelu, a tym samym zmniejszy ryzyko przeuczenia.

Decision Trees

Klasa DecisionTreeClassifier ma kilka innych parametrów, które podobnie ograniczają kształt drzewa decyzyjnego:

- **min_samples_split** (minimalna liczba próbek, które węzeł musi mieć przed podzieleniem),
- **min_samples_leaf** (minimalna liczba próbek, które musi posiadać węzeł liści),
- **min_weight_fraction_leaf** (to samo, co **min_samples_leaf**, ale wyrażone jako ułamek całkowitej liczby ważonych instancji),
- **max_leaf_nodes** (maksymalna liczba węzłów liści),
- **max_features** (maksymalna liczba współrzędnych, które są oceniane pod kątem dzielenia w każdym węźle).

Decision Trees

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D06_Z01_decision_trees.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D06_Z02_decision_trees.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D06_Z03_decision_trees_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D06_Z04_decision_trees_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D06_Z05_decision_trees_feature_selection.ipynb

Ensemble Learning and Random Forests

Random Forests

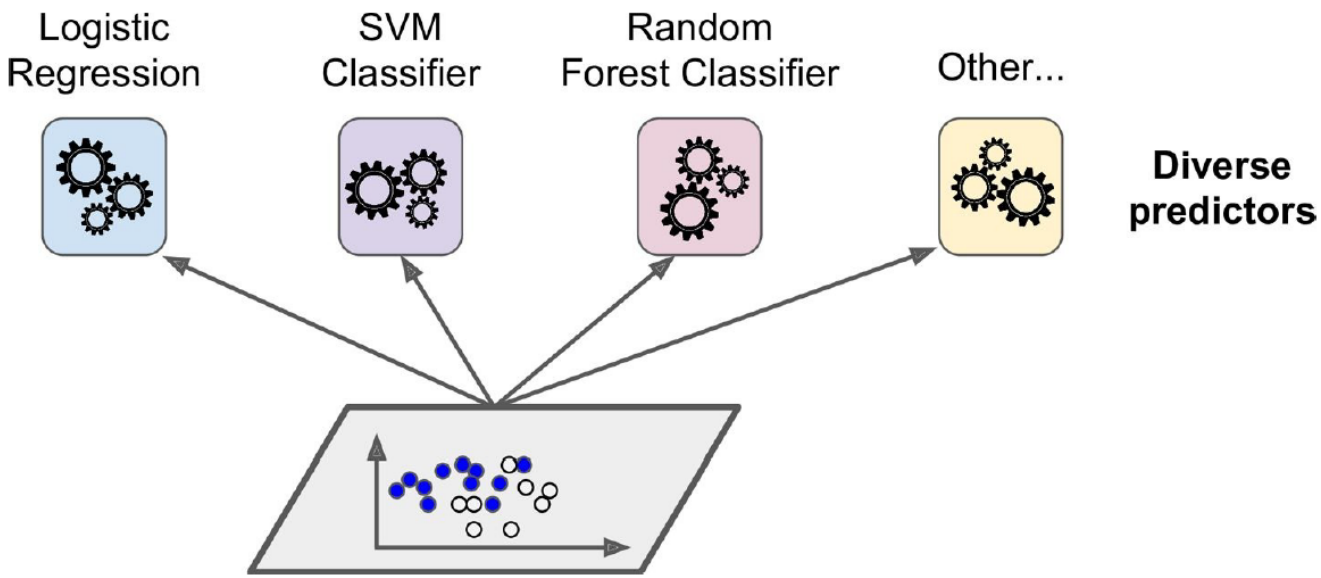
- Załóżmy, że zadajesz złożone pytanie tysiącom przypadkowych osób, a następnie zsumujesz ich odpowiedzi.
- W wielu przypadkach okaże się, że ta zagregowana odpowiedź jest lepsza niż odpowiedź eksperta.
- Nazywa się to mądrością tłumu (**wisdom of the crowd**).
- Podobnie, jeśli zagregujesz przewidywania grupy predyktorów (takich jak klasyfikatory lub regresory) często uzyskasz lepsze prognozy niż w przypadku najlepszego indywidualnego predyktora.
- Grupę predyktorów nazywana jest **Komitetem**, tak więc ta technika nosi nazwę **Ensemble Learning**, na polski mozna to przetłumaczyć jako **komitet klasyfikatorów**.

Random Forests

- Na przykład możesz nauczyć grupę drzew decyzyjnych, każdy na innym losowym podzbiorze zestawu treningowego.
- Aby dokonać predykcji, wystarczy uzyskać prognozy wszystkich pojedynczych drzew, a następnie wybrać tą, która uzyska najwięcej głosów.
- Taki komitet drzew decyzyjnych nazywa się Lasem Losowym (Random Forest) i pomimo swojej prostoty jest to jeden z najpotężniejszych algorytmów uczenia maszynowego dostępnych obecnie.

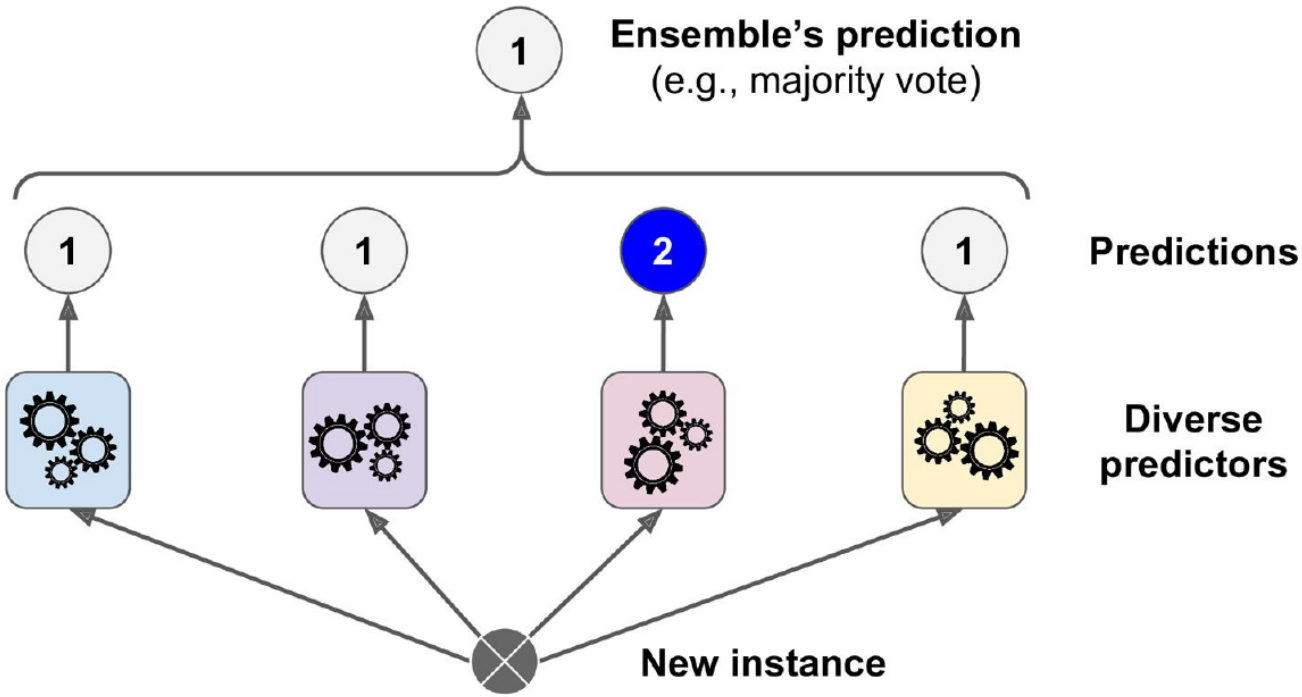
Random Forests

Załóżmy, że wytrenowałeś kilka klasyfikatorów, z których każdy osiągnął dokładność około 80%.



Random Forests

Bardzo prostym sposobem na stworzenie jeszcze lepszego klasyfikatora jest agregacja prognoz każdego klasyfikatora i przewidywanie klasy, która uzyska najwięcej głosów. Ten klasyfikator głosowania większością głosów nazywa się **hard voting classifier**.



Random Forests

- Nieco zaskakujące jest to, że **voting classifier** osiąga często wyższą dokładność niż najlepszy klasyfikator w komitecie.
- W rzeczywistości, nawet jeśli każdy klasyfikator jest słaby (**weak learner**) (co oznacza, że jest tylko nieznacznie lepszy niż losowy), komitet nadal może być silnym klasyfikatorem **strong learner** (osiągającym wysoką dokładność).
- Oczywiście pod warunkiem, że mamy wystarczającą liczbę słabych klasyfikatorów i są one wystarczająco różnorodne.

Random Forests

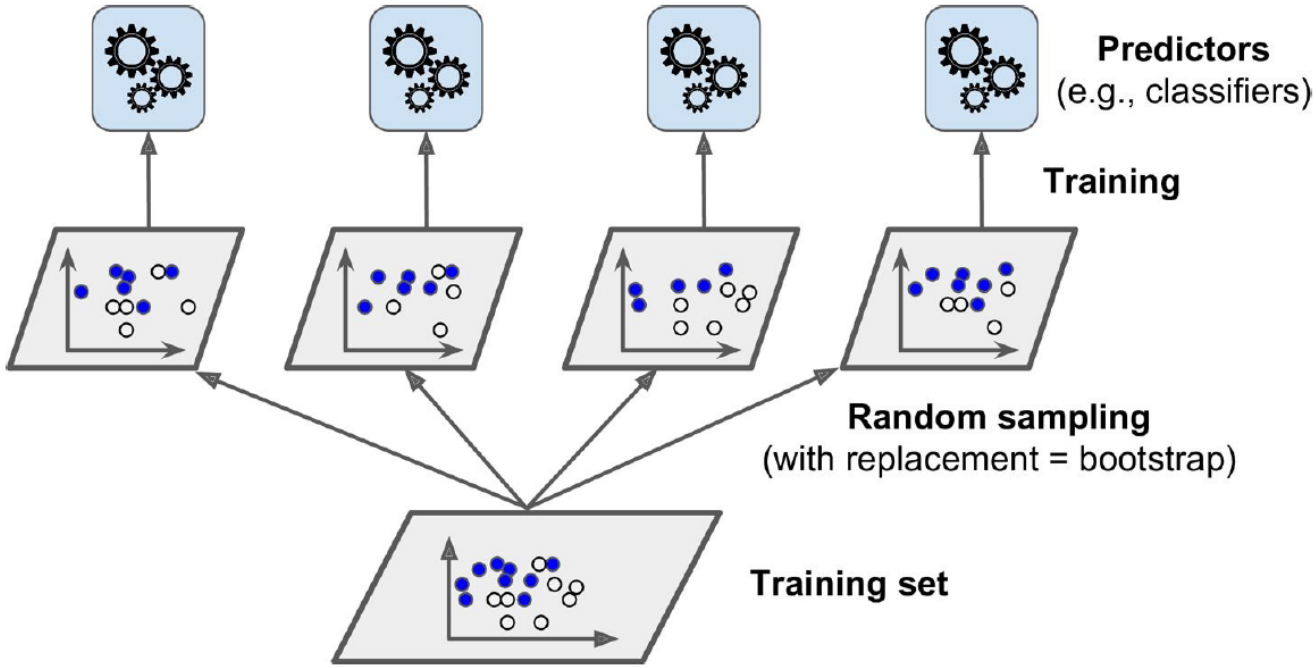
https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z01_ensemble.ipynb

Bagging and Pasting

- Jednym ze sposobów uzyskania zróżnicowanego zestawu klasyfikatorów jest użycie bardzo różnych algorytmów uczenia, co właśnie omówiliśmy.
- Innym podejściem jest użycie tego samego algorytmu dla każdego predyktora, ale uczenie ich na różnych losowych podzbiorach zbioru treningowego.
- Gdy pobieranie próbek odbywa się ze zwracaniem, ta metoda nazywa się **bagging** (skrót od bootstrap aggregating).
- Gdy pobieranie próbek odbywa się bez zwracania, nazywa się to **pasting**.

Bagging and Pasting

Innymi słowy, zarówno **bagging** i **pasting** pozwala na próbkowanie instancji kilka razy dla wielu predyktorów, ale tylko **bagging** umożliwia pobieranie próbek kilka razy dla tego samego predyktora.



Bagging and Pasting

- Gdy wszystkie predyktory zostaną nauczone, komitet może przewidzieć nową etykietę, po prostu agregując przewidywania wszystkich predyktorów.
- Funkcja agregacji jest zwykle jakąś statystyką np najczęstszą prognozą dla klasyfikacji lub średnią dla regresji.
- Wszystkie predyktory mogą być trenowane równolegle.
- Również prognozy mogą być wykonywane równolegle.

Bagging and Pasting

- Klasa BaggingClassifier obsługuje również próbkowanie współrzędnych.
- Jest to kontrolowane przez dwa hiperparametry:
  - max_features,
  - bootstrap_features.
- Działają one w taki sam sposób jak:
  - max_samples (maksymalna liczba współrzędnych, które są oceniane pod kątem dzielenia w każdym węźle),
  - bootstrap (jeśli chcesz użyć pasting zamiast bagging, po prostu ustaw bootstrap=False),

  ale dla próbkowania cech zamiast próbkowania instancji.
- W związku z tym każdy predyktor zostanie nauczony na losowym podzbiorze cech wejściowych.

Bagging and Pasting

- Jest to szczególnie przydatne, gdy mamy do czynienia z wejściami wielowymiarowymi (takimi jak obrazy).
- Próbkowanie obu instancji szkoleniowych i funkcji współrzędnych nazywa się metodą **Random Patches method**.
- Utrzymywanie wszystkich instancji szkoleniowych (tj. bootstrap=False oraz max_samples=1.0), ale próbkowanie współrzednych (tj. bootstrap_features=True i/lub max_features mniejszy niż 1.0) nazywa się **Random Subspaces method**.

Bagging and Pasting

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z01_ensemble.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z02_ensemble.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z03_ensemble.ipynb

Random Forests

Random Forests

Jak już wspomniano, **Random Forest** jest komitetem drzew decyzyjnych.

- Zwykle uczonym metodą **bagging** (lub czasami **pasting**).
- Zazwyczaj z ustawionymi max_samples na rozmiar zestawu uczącego (max_samples można ustawić na wartość zmiennoprzecinkową w zakresie od 0.0 do 1.0, w którym to przypadku maksymalna liczba instancji do próbkowania jest równa rozmiarowi zestawu treningowego razy max_samples.).
- Zamiast budować BaggingClassifier i przekazywać mu DecisionTreeClassifier, możesz zamiast tego użyć klasy RandomForestClassifier, która jest wygodniejsza i zoptymalizowana dla DecisionTrees (podobnie, istnieje klasa RandomForestRegressor dla zadań regresji).

Random Forests

Z kilkoma wyjątkami RandomForestClassifier ma:

- wszystkie hiperparametry DecisionTreeClassifier (aby kontrolować jak wyglądają drzewa),
- wszystkie hiperparametry klasy BaggingClassifier, aby kontrolować komitet klasyfikatorów.

Random Forests

- Jeśli spojrzysz na jedno drzewo decyzyjne, ważne współrzędne prawdopodobnie pojawią się bliżej korzenia drzewa (root), podczas gdy nieistotne współrzędne będą często wyświetlane bliżej liści (lub wcale).
- W związku z tym można oszacować znaczenie danej współrzednej, obliczając średnią głębokość na jakiej pojawia się ona na wszystkich drzewach w lesie.
- Scikit-Learn oblicza to automatycznie dla każdej funkcji po treningu.
- Możesz uzyskać dostęp do wyniku za pomocą zmiennej feature_importances_.

Random Forests

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z04_random_forests.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z05_random_forests.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z06_random_forests_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z08_random_forests_feature_selection.ipynb

Boosting

Boosting

- Boosting (oryginalnie nazywany hypothesis boosting) odnosi się do dowolnej metody opartej na komitecie klasyfikatorów, która może połączyć kilka słabych klasyfikatorów w jeden silny.
- Ogólną ideą większości metod typu **boosting** jest sekwencyjne trenowanie predyktorów, z których każdy próbuje skorygować swójego poprzednika.
- Dostępnych jest wiele metod typu **boosting**, ale zdecydowanie najbardziej popularne są AdaBoost (Adaptive Boosting) oraz Gradient Boosting.

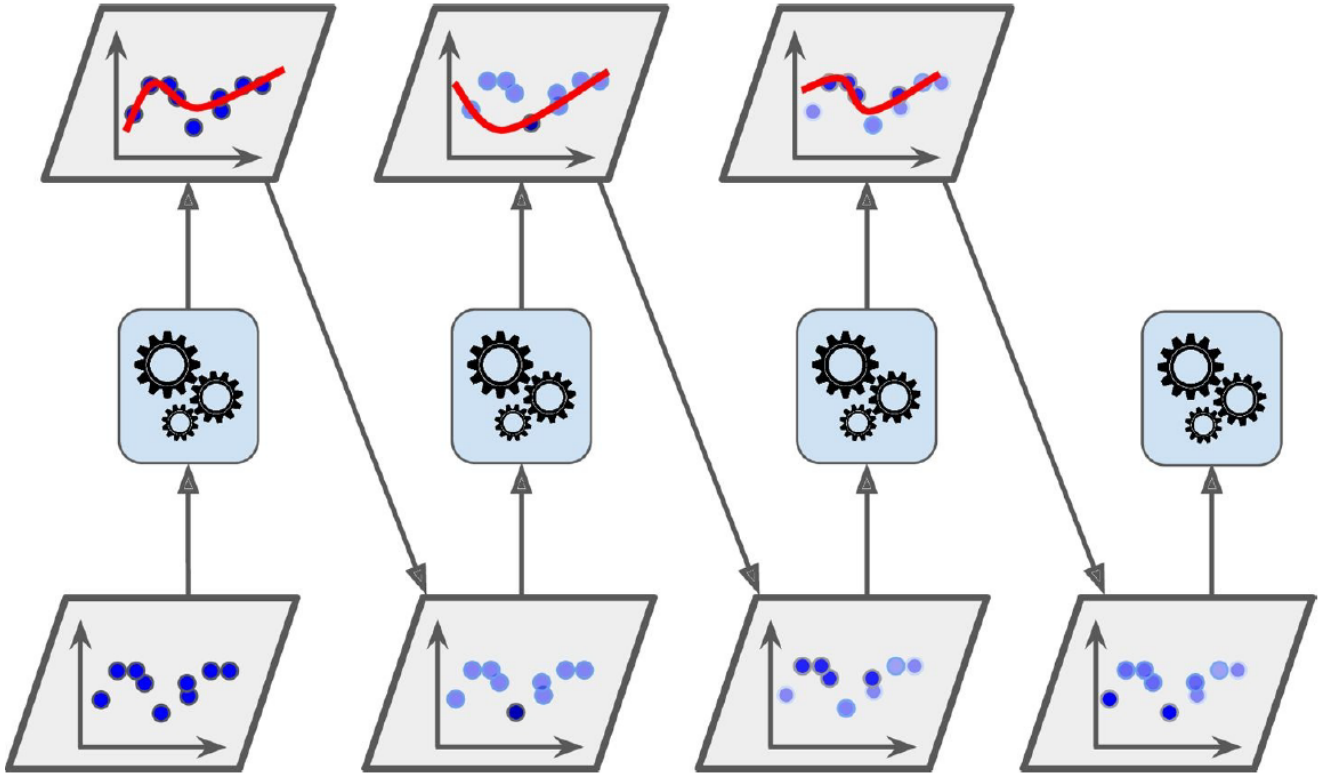
AdaBoost

- Jednym ze sposobów na to, by nowy czynnik predykcyjny poprawił swojego poprzednika, jest zwrócenie większej uwagi na instancje szkoleniowe, które zostały **underfitted** przez poprzednika.
- Powoduje to nowe predyktory koncentrujące się coraz bardziej na trudnych przypadkach.
- Jest to technika stosowana przez **AdaBoost**.

AdaBoost

- Na przykład, aby zbudować klasyfikator **AdaBoost**, pierwszy klasyfikator bazowy (taki jak drzewo decyzyjne) jest uczony i wykorzystywany do tworzenia prognoz na zbiorze treningowym.
- Względna waga nieprawidłowo sklasyfikowanych instancji szkoleniowych jest następnie zwiększana.
- Drugi klasyfikator jest uczony przy użyciu zaktualizowanych wag (wagi do losowania punktów) i ponownie sprawdza wynik na zbiorze treningowym.
- I tak dalej.

AdaBoost



AdaBoost

- Gdy wszystkie predyktory zostaną nauczone, komitet tworzy przewidywania bardzo podobne do bagging i pasting, z wyjątkiem tego, że predyktory mają różne wagi w zależności od ogólnej dokładności ważonego zestawu treningowego.

AdaBoost

Przyjrzyjmy się bliżej algorytmowi AdaBoost.

- Każda waga instancji $w^{(i)}$ jest początkowo ustawiona na $\frac{1}{m}$.
- Pierwszy predyktor jest uczony, a jego ważony współczynnik błędu r_1 jest obliczany na zbiorze treningowym zgodnie ze wzorem:

$$r_j = \frac{\sum_{i=1,\ \hat{y}_j^{(i)} \neq y^{(i)}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

  gdzie: $\hat{y}_j^{(i)}$ to predykcja j-klasyfikatora na i-tym przykładzie.
- Wagę predyktora α_j oblicza się za pomocą równania:

$$\alpha_j = \eta \log\left(\frac{1 - r_j}{r_j}\right)$$

  gdzie η jest współczynnikiem uczenia się - learning rate (domyślnie 1).

AdaBoost

- Im dokładniejszy jest predyktor, tym wyższa będzie jego waga.
- Jeśli jest to po prostu klasyfikator losowy, to jego waga będzie bliska zeru.
- Jeśli klasyfikator jest zły (to znaczy gorszy niż losowy), wówczas jego waga będzie ujemna.
- Następnie wagi instancji są aktualizowane za pomocą równania:

$$w^{(i)} = \begin{cases} w^{(i)}, & \text{if } y_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j), & \text{if } y_j^{(i)} \neq y^{(i)} \end{cases}$$

- Wagi błędnie sklasyfikowanych instancji są zwiększane.
- Następnie wszystkie wagi instancji są znormalizowane (tj. dzielone przez $\sum w^{(i)}$).

AdaBoost

- Na koniec, nowy predyktor jest trenowany przy użyciu zaktualizowanych wag, a cały proces jest powtarzany (obliczana jest nowa waga predyktora, wagi instancji są aktualizowane, następnie przygotowywany jest następny predyktor itd.).
- Algorytm zatrzymuje się, gdy zostanie osiągnięta określona liczba predyktorów lub gdy zostanie znaleziony idealny predyktor.
- Aby przewidzieć etykietę AdaBoost po prostu oblicza przewidywania wszystkich predyktorów i waży je za pomocą wag predykcyjnych α_j.
- Przewidywana klasa to ta, która otrzymuje większość głosów ważonych:

$$\hat{y}_i = \underset{\{k\}}{\text{argmax}} \sum_{j=1,\ \hat{y}_i=k}^{N} \alpha_i$$

AdaBoost

- Scikit-Learn używa wieloklasowej wersji AdaBoost nazywanej SAMME (Stagewise Additive Modeling using a Multiclass Exponential loss function).
- Jeżeli mamy tylko dwie klasy to SAMME daje taki sam wynik jak AdaBoost.

AdaBoost

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z09_AdaBoost.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z10_AdaBoost.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z11_AdaBoost.ipynb

Gradient Boosting

- Innym bardzo popularnym algorytmem **Boostingu** jest **Gradient Boosting**.
- Podobnie jak AdaBoost, **Gradient Boosting** działa poprzez sekwencyjne dodawanie predyktorów do komitetu, z których każdy koryguje swojego poprzednik.
- Jednak zamiast modyfikować wagi punktów tak jak w AdaBoost, **Gradient Boosting** próbuje dopasować nowy predykator do błędu poprzedniego predyktora.

`https://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html` `https://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html`

Gradient Boosting

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z14_Gradient_Boosted_Regression_Trees.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z15_Gradient_Boosting.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z16_Gradient_Boosting.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z18_gradient_boosting_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z19_gradient_boosting.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z20_gradient_boosting.ipynb

Gradient Boosting

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D07_Z17.ipynb

Automatic Feature Selection

Automatic Feature Selection

- Ale skąd możesz wiedzieć, które współrzędne są najważniejsze?
- Istnieją trzy podstawowe strategie:
  - univariate statistics,
  - model-based selection,
  - iterative selection

Univariate Statistics

- Metodami statystycznymi możemy sprawdzić zależność między każdą cechą (współrzędną), a etykietą.
- Następnie możemy wybrać te współrzędne, które sa najbardziej zależne.
- Kluczową właściwością tych testów jest to, że są one jednowymiarowe, co oznacza, że uwzględniają tylko każdą współrzedną osobno.
- W związku z tym współrzędna zostanie odrzucona jeśli jest zależna z inną współrzędną.
- Testy jednowymiarowe są często bardzo szybkie do obliczenia i nie wymagają budowania modelu.
- Z drugiej strony, są całkowicie niezależne od modelu, który chcesz zastosować po wybraniu współrzednych.

Model-Based Feature Selection

- Wybór współrzędnych oparty na **Model-Based Feature Selection** wykorzystuje nadzorowany model uczenia maszynowego do oceny ważności każdej współrzędnej.
- Nadzorowany model, który jest używany do wyboru cech, nie musi być tym samym modelem, który chcemy mauczyć.
- **Model-Based Feature Selection** wprowadza miarę ważności dla każdej współrzędnej, tak aby można było uszeregować je względem tej miary.
- Drzewa decyzyjne i modele oparte na drzewach decyzyjnych zapewniają atrybut feature_importances, który bezpośrednio koduje znaczenie każdej funkcji.
- Modele liniowe z karą L1 uczą się rzadkich współczynników, może to być postrzegane jako forma wyboru cech dla samego modelu, ale może być również wykorzystana jako etap wstępnego przetwarzania danych dla innego modelu.
- W przeciwieństwie do selekcji jednowymiarowej, wybór oparty na modelu uwzględnia wszystkie współrzędne jednocześnie, a zatem może

Iterative Feature Selection

- W testach jednoczynnikowych nie używaliśmy modelu, podczas gdy w **Model-Based Feature Selection** używaliśmy pojedynczego modelu do wyboru współrzędnych.
- W **Iterative Feature Selection** tworzona jest seria modeli z różną liczbą współrzednych.
- Istnieją dwie podstawowe metody:
  - począwszy od 1 współrzędnej, dodajemy kolejne jeden po drugim, aż zostanie osiągnięty warunek stopu,
  - począwszy od wszystkich współrzędnych usuwamy elementy jeden po drugim, aż do osiągnięcia pewnego warunku stopu.
- Ponieważ zbudowana została seria modeli, to podejście to jest o wiele bardziej kosztowne obliczeniowo niż metody, które omówiliśmy wcześniej.

Iterative Feature Selection

Jedną z metod iteracyjnych jest **recursive feature elimination (RFE)**:

- Rozpoczyna ona od wszystkich współrzednych, buduje model i odrzuca najmniej ważną współrzedną.
- Następnie tworzony jest nowy model przy użyciu wszystkich współrzędnych, oprócz odrzuconych i tak dalej, aż pozostanie tylko określona liczba współrzędnych.
- Aby to zadziałało, model użyty do selekcji musi w jakiś sposób określić znaczenie współrzędnych, tak jak miało to miejsce w przypadku **Model-Based Feature Selection**.

Feature Selection

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D08_Z01_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D08_Z02_feature_selection.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D08_Z03_Wi_Fi.ipynb

Unbalanced data

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D08_Z04_unbalanced_data.ipynb
- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D08_Z05_unbalanced_data.ipynb

Randomized Search

- https://github.com/przem85/bootcamp/blob/master/supervised_learning/D08_Z06_RandomizedSearch.ipynb