Politechnika Wrocławska

Politechnika Wrocławska
Wydział Mechaniczny

Zarządzanie i Inżynieria Produkcji (ZIP)
Organizacja Produkcji (OP)

Uczenie maszynowe jako narzędzie doskonalenia procesów wytwórczych

Machine Learning as a Tool for Manufacturing Process Improvement

Praca dyplomowa magisterska

Autor: Mateusz A. Tabor
Promotor: Dr inż. Kamil Musiał

Wrocław, 2026

Spis treści

1

Charakterystyka algorytmów uczenia maszynowego

1.1 Uczenie nadzorowane (Supervised Learning)

Uczenie nadzorowane to technika, w której algorytm otrzymuje gotowy zestaw danych z wcześniej określonymi etykietami i uczy się na ich podstawie predyktować etykiety dla nowych nieznanych danych. Dane uczące zawierają zmienne wejściowe oraz zmienną predykowaną (etykietę). Standardowy proces uczenia nadzorowanego obejmuje podział zbioru danych na zbiór treningowy, walidacyjny oraz testowy. Trening polega na dopasowaniu parametrów modelu do danych treningowych poprzez minimalizację funkcji straty, która mierzy różnicę między przewidywaniami modelu a rzeczywistymi etykietami. Po zakończeniu treningu danych model zostaje poddany ocenie na zbiorze walidacyjnym w celu doboru hiperparametrów oraz monitorowaniu uczenia maszynowego, aby zapobiec przeuczeniu modelu. Ostateczna ocena odbywa się na zbiorze testowym na danych nieznanych dla modelu i na podstawie tego modelu określana wartość dokładności, precyzji, recall, F1 (dla klasyfikacji) lub MSE, MAE (dla regresji)(Bishop, 2006; Hastie et al., 2009).

1.1.1 Regresja liniowa (Linear Regression)

Regresja liniowa to podstawowa technika statystyczna i uczenia maszynowego stosowana do modelowania zależności między zmienną zależną (predykowaną) a jedną lub więcej zmiennymi niezależnymi (cechami). Celem regresji liniowej jest znalezienie liniowej funkcji, która najlepiej opisuje związek między zmiennymi, umożliwiając przewidywanie wartości zmiennej zależnej na podstawie wartości zmiennych niezależnych (Hastie et al., 2009; James et al., 2013).

Regresja liniowa prosta. W przypadku pojedynczej zmiennej niezależnej model można opisać równaniem (James et al., 2013):

$$y = \alpha + \beta x + \varepsilon, \tag{1.1}$$

gdzie y to zmienna zależna, x to zmienna niezależna, α to wyraz wolny, β to współczynnik nachylenia linii regresji, a ε to składnik losowy (błąd modelu).

Regresja wieloraka. Regresja wieloraka jest rozszerzeniem regresji prostej, poprzez używanie wielu zmiennych niezależnych. Zakłada się liniową zależność między zmienną zależną a kombinacją liniową zmiennych niezależnych (Hastie et al., 2009):

$$y = \alpha + \sum_{i=1}^{p} \beta_i x_i + \varepsilon, \tag{1.2}$$

gdzie y to zmienna zależna, α to wyraz wolny, x_i to zmienne niezależne, β_i to współczynniki regresji określające wpływ danej zmiennej na zmienną zależną, a ε to składnik losowy (błąd modelu).

Celem algorytmu regresji (prostej i wielorakiej) jest znalezienie optymalnych wartości współczynników β_i, które minimalizują błąd predykcji. Współczynniki można dobrać za pomocą różnych metod, jednak najczęściej stosuje się metodę najmniejszych kwadratów (OLS — Ordinary Least Squares), która minimalizuje sumę kwadratów różnic między rzeczywistymi a przewidywanymi wartościami zmiennej zależnej (Hastie et al., 2009).

Estymator OLS (macierzowy zapis).

$$\widehat{\boldsymbol{\beta}} = (X^\top X)^{-1} X^\top \mathbf{y} \tag{1.3}$$

Wzór (1.3) to macierzowy zapis estymatora OLS (Hastie et al., 2009; James et al., 2013). Wyjaśnienie:

- X — macierz projektująca (design matrix) o wymiarach $n \times p$ (lub $n \times (p+1)$, jeśli dodano kolumnę jedynek dla wyrazu wolnego),
- $\mathbf{y}$ — wektor obserwacji o wymiarze $n \times 1$,
- $\widehat{\boldsymbol{\beta}}$ — wektor estymowanych współczynników o wymiarze $p \times 1$.

Aby wyrażenie było poprawne, macierz $X^\top X$ musi być odwracalna (brak doskonałej multikolinearności). Przy klasycznych założeniach (m.in. $E[\varepsilon] = 0$, $\mathrm{Var}(\varepsilon) = \sigma^2 I$) estymator jest nieobciążony, a jego wariancja wynosi $\mathrm{Var}(\widehat{\boldsymbol{\beta}}) = \sigma^2 (X^\top X)^{-1}$. Gdy $X^\top X$ jest źle uwarunkowana lub nieodwracalna, stosuje się regularyzację (np. Ridge) lub metody numeryczne (gradient descent, SVD) (James et al., 2013; Hastie et al., 2009).

Inne metody estymacji współczynników regresji liniowej:

- Metoda gradientu prostego (Gradient Descent) — iteracyjna metoda optymalizacji minimalizująca funkcję straty poprzez aktualizację współczynników w kierunku przeciwnym do gradientu (Bishop, 2006; Murphy, 2012).
- Metoda najmniejszych modułów (Least Absolute Deviations) — minimalizuje sumę bezwzględnych różnic między rzeczywistymi a przewidywanymi wartościami, co czyni ją bardziej odporną na wartości odstające (Birkes and Dodge, 1993).
- Metoda Ridge Regression — wprowadza regularyzację L2, karę za duże wartości współczynników, co pomaga w radzeniu sobie z problemem multikolinearności i przeuczenia modelu (Hoerl and Kennard, 1970; Hastie et al., 2009).
- Metoda Lasso Regression — regularyzacja L1, która może prowadzić do zerowania niektórych współczynników, skutkując modelem o mniejszej liczbie cech (automatyczny wybór cech) (Tibshirani, 1996).
- Metoda Elastic Net — łączy regularyzację L1 i L2, co pozwala na lepsze dostosowanie modelu do danych (Zou and Hastie, 2005).

- Metoda SVD (Singular Value Decomposition) — rozkłada macierz projektującą na składniki, umożliwiając stabilne obliczenie współczynników regresji nawet w przypadku kolinearności cech (Golub and Loan, 1996; Press et al., 2007).
- Metoda Bayesian Regression — wykorzystuje podejście bayesowskie do estymacji współczynników, uwzględniając niepewność i priorytety w modelu (Gelman and Hill, 2006; Bishop, 2006).
- QR Decomposition — rozkłada macierz projektującą na iloczyn macierzy ortogonalnej i górnotrójkątnej, co umożliwia efektywne rozwiązanie układu równań regresji (Golub and Loan, 1996; Press et al., 2007).

1.1.2 Regresja logistyczna (Logistic Regression)

Regresja logistyczna to technika statystyczna i uczenia maszynowego stosowana do modelowania zależności między zmienną zależną a jedną lub więcej zmiennymi niezależnymi, gdy zmienna zależna przyjmuję wartości binarne. Celem regresji logistycznej jest przewidywanie prawdopodobieństwa przynależności do jednej z dwóch klas na podstawie wartości zmiennych niezależnych (Hastie et al., 2009; James et al., 2013).

Model regresji logistycznej. Model regresji logistycznej można zapisać jako (Hastie et al., 2009):

$$P(Y = 1|X) = \sigma(\boldsymbol{\beta}^\top X) = \frac{1}{1 + e^{-\boldsymbol{\beta}^\top X}} \tag{1.4}$$

gdzie:

- $P(Y = 1|X)$ — prawdopodobieństwo, że zmienna zależna Y przyjmuje wartość 1, biorąc pod uwagę zmienne niezależne X,
- $\sigma(z)$ — funkcja sigmoidalna, która przekształca dowolną wartość rzeczywistą z w przedział $(0, 1)$.

Estymacja parametrów. Parametry modelu $\boldsymbol{\beta}$ są estymowane za pomocą metody największej wiarygodności (Maximum Likelihood Estimation, MLE). Celem jest maksymalizacja funkcji wiarygodności (Hastie et al., 2009):

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} P(Y_i|X_i; \boldsymbol{\beta}) \tag{1.5}$$

co jest równoważne minimalizacji funkcji straty (Hastie et al., 2009):

$$J(\boldsymbol{\beta}) = -\sum_{i=1}^{n} [Y_i \log(P(Y_i|X_i; \boldsymbol{\beta})) + (1 - Y_i) \log(1 - P(Y_i|X_i; \boldsymbol{\beta}))] \tag{1.6}$$

1.1.3 k-Najbliższych Sąsiadów (k-Nearest Neighbors)

Algorytm k-Najbliższych Sąsiadów umieszcza dane wejściowe w przestrzeni wielowymiarowej i klasyfikuje je na podstawie etykiet najbliższych sąsiadów w tej przestrzeni. Przestrzeń jest definiowana przez cechy danych, zbiór danych posiadający x cech jest reprezentowany w x-wymiarowej przestrzeni. Algorytm klasyfikując dany obiekt oblicza odległości między nim a wszystkimi innymi obiektami w przestrzeni, a następnie wybiera k najblizszych sąsiadów. Wartość k jest ustalana ustalana przed rozpoczęciem działania algorytmu. Niska wartość parametru k jest bardziej podatna na szumy w danych, podczas gdy wysoka wartość k może prowadzić do nadmiernego uogólnienia modelu (Cover and Hart, 1967).

Algorytm k-najbliższych sąsiadów może wykorzystywać różne metryki do obliczania odległości m.in:

Metryka Euklidesowa: Najpowszechniejsza metryka używana do obliczania odległości między dwoma punktami w przestrzeni wielowymiarowej. Definiowana jest jako:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{1.7}$$

gdzie p i q to dwa punkty w przestrzeni, a n to liczba wymiarów. Wyliczanie odległości metryką euklidesową polega na policzeniu różnicy ogległości w każdym wymiarze dla dwóch punktów, zsumowaniu kwadratów tych różnic, a następnie wyciągnięciu pierwiastka kwadratowego z tej sumy (Duda et al., 2001; Bishop, 2006).

Metryka Manhattan: Metryka Manhattan, nazywana również metryką taksówkową lub L1, mierzy odległość miedzy dwoma punktami jako sumę watości bezwzględnych z różnic współżędnych. Definiowana jest jako:

$$d(p, q) = \sum_{i=1}^{n} |p_i - q_i| \tag{1.8}$$

gdzie p i q to dwa punkty w przestrzeni, a n to liczba wymiarów. Obliczana jest jest różnica wartości p i q dla każdego wymiaru, a następnie sumowane są wartości bezwzględne tych różnic (Duda et al., 2001).

Metryka Kosinusowa: Metryka Kosinusowa mierzy wyznacza odległość między dwoma punktami na podstawie wyliczonego kąta między nimi. Definiowana jest jako:

$$d(p, q) = 1 - \frac{p \cdot q}{\|p\| \|q\|} \tag{1.9}$$

gdzie p i q to dwa punkty w przestrzeni, $p \cdot q$ to iloczyn skalarny wektorów p i q, a $\|p\|$ i $\|q\|$ to normy (długości) tych wektorów. Normy długości wektorów są obliczane jako pierwiastki kwadratowe z sumy kwadratów ich współrzędnych (Manning et al., 2008; Bishop, 2006).

Metryka Minkowskiego: Metryka Minkowskiego jest uogólnieniem metryk Euklidesowej i Manhattan. Umożliwa regulowanie sposobu obliczania odległości poprzez parametr p. Definiowana jest jako:

$$d(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{\frac{1}{p}} \tag{1.10}$$

gdzie p i q to dwa punkty w przestrzeni, n to liczba wymiarów, a p to parametr regulujący sposób obliczania odległości. Dla $p = 1$ metryka Minkowskiego jest równoważna metryce Manhattan, a dla $p = 2$ jest równoważna metryce Euklidesowej (Hastie et al., 2009).

1.1.4 Drzewa decyzyjne (Decision Trees)

Drzewa decyzyjne to algorytm uczenia maszynowego, która służy do podejmowania decyzji na podstawie zestawu reguł, które są reprezentowane w formie drzewa. Drzewo decyzyjne składa się z korzenia, które jest cechą dzielącą dane na grupy, węzłów wewnętrznych, które reprezentują pytania dotyczące cech danych, oraz liści, które reprezentują ostateczne decyzje lub klasyfikacje. Algorytm budowy drzewa decyzyjnego polega na iteracyjnym dzieleniu danych na podzbiory na podstawie cech, które najlepiej rozdzielają dane według określonego kryterium. Drzewo decyzyjne jest budowane, aż wszystkie elementy w podzbiorze należą do tej samej klasy lub nie ma już cech do podziału, osiągnie maksymalną głębokość lub kiedy dalszy podział nie poprawia jakości klasyfikacji (Quinlan, 1986; Breiman et al., 1984).

Drzewo decyzyjne do wyboru najlepszego podziału danych może wykorzystywać różne kryteria, m.in:

Entropia: Entropia jest miarą niepewności lub nieuporządkowania w zbiorze danych. Entropia jest wykorzystywana do oceny jakości podziału danych na podstaiwe cechy. Definiowana jest jako:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i) \tag{1.11}$$

gdzie S to zbiór danych, c to liczba klas, a p_i to proporcja elementów należących do klasy i w zbiorze S. Entropia obliczana jest jako suma iloczynów proporcji klas i logarytmów tych proporcji, a następnie mnożona przez -1 (Shannon, 1948; Quinlan, 1986).

Wskaźnik Gini (Gini Impurity): Wskaźnik Gini mierzy prawdopodobieństwo błędnej klasyfikacji losowo wybranego elementu, gdyby został on oznaczony losowo według rozkładu etykiet w węźle. Im niższy wskaźnik Gini, tym bardziej jednorodny jest węzeł. Wskaźnik Gini dla zbioru S jest definiowany jako:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} p_i^2 \tag{1.12}$$

gdzie c to liczba klas, a p_i to proporcja przykładów w klasie i. Wskaźnik Gini jest używany w algorytmie CART (Classification and Regression Trees) ze względu na swoją prostotę obliczeniową i dobrą wydajność (Breiman et al., 1984).

Zysk informacji (Information Gain): Zysk informacji mierzy redukcję entropii osiągniętą przez podział zbioru danych według danego atrybutu. Jest to różnica między entropią zbioru nadrzędnego a ważoną sumą entropii zbiorów potomnych. Zysk informacji dla atrybutu A w zbiorze S definiuje się jako:

$$\mathrm{IG}(S, A) = H(S) - \sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} H(S_v) \tag{1.13}$$

gdzie Values(A) to zbiór wszystkich możliwych wartości atrybutu A, S_v to podzbiór S, dla którego atrybut A ma wartość v, a $H(S)$ to entropia zbioru. Algorytm ID3 wybiera atrybut o największym zysku informacji (Quinlan, 1986).

Współczynnik zysku (Gain Ratio) Współczynnik zysku jest modyfikacją zysku informacji, która koryguje tendencję do faworyzowania atrybutów o wielu wartościach. Normalizuje zysk informacji przez podzielenie go przez tzw. split information, która mierzy szerokość i jednolitość podziału. Współczynnik zysku dla atrybutu A w zbiorze S definiuje się jako:

$$\mathrm{GainRatio}(S, A) = \frac{\mathrm{IG}(S, A)}{\mathrm{SplitInfo}(S, A)} \tag{1.14}$$

gdzie $\mathrm{IG}(S, A)$ to zysk informacji dla atrybutu A, a $\mathrm{SplitInfo}(S, A)$ to informacja o podziale, definiowana jako:

$$\mathrm{SplitInfo}(S, A) = -\sum_{v \in \mathrm{Values}(A)} \frac{|S_v|}{|S|} \log_2 \left(\frac{|S_v|}{|S|} \right) \tag{1.15}$$

gdzie Values(A) to zbiór wszystkich możliwych wartości atrybutu A, S_v to podzbiór S zawierający elementy, dla których atrybut A ma wartość v, $|S_v|$ to liczba elementów w podzbiorze S_v, a $|S|$ to całkowita liczba elementów w zbiorze S. Informacja o podziale mierzy, jak bardzo atrybut dzieli dane. Im bardziej równomierny podział, tym wyższa wartość SplitInfo, co zmniejsza współczynnik zysku i zapobiega faworyzowaniu atrybutów o wielu unikalnych wartościach. Współczynnik zysku jest używany w algorytmie C4.5 jako ulepszona wersja ID3 (Quinlan, 1993).

Redukcja wariancji (Variance Reduction) Redukcja wariancji jest kryterium stosowanym w drzewach regresyjnych, gdzie celem jest przewidywanie wartości ciągłych, a nie kategorii. Kryterium to wybiera podział, który maksymalnie redukuje wariancję wartości docelowych w węzłach potomnych. Redukcja wariancji dla podziału zbioru S na podzbiory S_{left} i S_{right} definiuje się jako:

$$\mathrm{VarReduction}(S) = \mathrm{Var}(S) - \left(\frac{|S_{\text{left}}|}{|S|} \mathrm{Var}(S_{\text{left}}) + \frac{|S_{\text{right}}|}{|S|} \mathrm{Var}(S_{\text{right}}) \right) \tag{1.16}$$

gdzie $\mathrm{Var}(S)$ oznacza wariancję wartości docelowych w zbiorze S. Algorytm CART dla regresji wykorzystuje to kryterium do budowy drzew regresyjnych (Breiman et al., 1984).

1.1.5 Las losowy (Random Forest)

Las losowy to algorytm uczenia maszynowego, który łączy wiele drzew decyzyjnych w celu poprawy dokładności przewidywań i redukcji przeuczenia. Algorytm ten działa

na zasadzie tworzenia wielu niezależnych drzew decyzyjnych, z których każde jest trenowane na losowym podzbiorze danych i losowym podzbiorze cech. Ostateczna predykcja lasu losowego jest uzyskiwana przez agregację wyników wszystkich drzew. Dla klasyfikacji stosuje się głosowanie większościowe, a dla regresji średnią arytmetyczną (Breiman, 2001).

Las losowy uczy się poprzez losowanie i budowanie wielu drzew decyzyjnych. Każde drzewo dostaje losowy podzbiór danych treningowych oraz losowy podzbiór cech do rozważenia przy każdym podziale węzła. Ten proces losowania danych i cech wprowadza różnorodność między drzewami, co przekłada się na lepszą ogólną wydajność modelu. Poszczególne drzewa dzielone są na podstawie kryteriów omówionych w sekcji dotyczącej drzew decyzyjnych, takich jak entropia, wskaźnik Gini czy zysk informacji (Breiman, 2001).

Dodatkowo przy budownie lasu losowego dobiera się parametry:

Tabela 1.1: Podstawowe hiperparametry lasu losowego

Parametr	**Opis**
Liczba drzew (B)	Liczba drzew decyzyjnych w lesie. Większa liczba zazwyczaj poprawia wydajność, ale zwiększa czas obliczeń.
Maksymalna głębokość	Maksymalna głębokość każdego drzewa. Ograniczenie głębokości może zapobiec przeuczeniu.
Liczba cech (m)	Liczba losowo wybranych cech rozważanych przy każdym podziale. Typowo $m = \sqrt{p}$ dla klasyfikacji lub $m = p/3$ dla regresji. (p to liczba wszystkich cech)
Minimalna liczba próbek	Minimalna liczba próbek wymagana do podziału węzła wewnętrznego lub do utworzenia liścia.
Bootstrap	Czy stosować bootstrap sampling (losowanie ze zwracaniem) do tworzenia podzbiorów treningowych.

Strojenie hiperparametrów lasu losowego, jest zadaniem człowieka, ale najczęściej wykorzystuje się metody automatyczne do testowania.

Przeszukiwanie Siatki (Grid Search) Grid Search polega na przeszukiwaniu wszystkich możliwych kombinacji hiperparametrów z wcześniej zdefiniowanej siatki wartości (James et al., 2013; Hastie et al., 2009). Dla każdej kombinacji algorytm trenuje model i ocenia jego wydajność za pomocą walidacji krzyżowej. Następnie wybiera zestaw parametrów dający najlepsze wyniki według ustalonej metryki (Hastie et al., 2009).

Przeszukiwanie Losowe (Randomized Search) Randomized Search jest wariantem Grid Search, który zamiast sprawdzać wszystkie kombinacje, losowo próbkuje określoną liczbę zestawów hiperparametrów z zadanych rozkładów (James et al., 2013). Liczba iteracji jest definiowana przez programistę.

Optuna Optuna to framework do optymalizacji hiperparametrów wykorzystujący zaawansowane algorytmy, takie jak Tree-structured Parzen Estimator (TPE). W przeciwieństwie do metod grid i random search, Optuna adaptacyjnie wybiera kolejne kombinacje hi-

perparametrów na podstawie wyników wcześniejszych prób. Uczy się, które obszary przestrzeni są obiecujące i koncentruje tam przeszukiwanie. Framework oferuje elastyczność w definiowaniu przestrzeni parametrów, wbudowane wizualizacje oraz możliwość przycinania nieobiecujących prób (Akiba et al., 2019).

Optymalizacja bayesowska (Bayesian Optimization) Bayesian Optimization buduje probabilistyczny model zastępczy (surrogate model), zazwyczaj Gaussian Process, który aproksymuje funkcję celu (np. dokładność modelu jako funkcję hiperparametrów). Na podstawie tego modelu algorytm wybiera kolejne punkty do próbkowania za pomocą funkcji akwizycji (acquisition function), takiej jak Expected Improvement (EI), która balansuje eksplorację nowych obszarów przestrzeni i eksploatację obiecujących regionów (Snoek et al., 2012).

AutoML (Automated Machine Learning) AutoML automatyzuje cały proces budowy modelu uczenia maszynowego, w tym wybór algorytmu, inżynierię cech, dobór hiperparametrów oraz tworzenie modeli zespołowych. Narzędzia takie jak Auto-sklearn, TPOT czy H2O AutoML wykorzystują kombinację technik optymalizacji (bayesian optimization, evolutionary algorithms) oraz wiedzę z wcześniejszych eksperymentów na podobnych zbiorach (meta-learning) (Feurer et al., 2015; Olson et al., 2016).

1.1.6 Maszyna wektorów nośnych (Support Vector Machine, SVM)

Maszyna wektorów nośnych (SVM) to algorytm uczenia maszynowego stosowany do klasyfikacji i regresji, który działa na zasadzie znajdowania optymalnej hiperpłaszczyzny rozdzielającej dane należące do różnych klas w przestrzeni wielowymiarowej (Bishop, 2006; Hastie et al., 2009). Celem SVM jest maksymalizacja marginesu, czyli odległości między hiperpłaszczyzną a najbliższymi punktami z obu klas, zwanymi wektorami nośnymi (support vectors). Większy margines prowadzi do lepszej generalizacji modelu na nowe, nieznane dane (Bishop, 2006; Hastie et al., 2009).

Zasada działania. Dla liniowo separowalnych danych, SVM znajduje hiperpłaszczyznę definiowaną jako (Bishop, 2006):

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0 \tag{1.17}$$

gdzie $\boldsymbol{w}$ to wektor normalny do hiperpłaszczyzny, $\boldsymbol{x}$ to wektor cech, a b to wyraz wolny. Funkcja decyzyjna klasyfikuje punkt $\boldsymbol{x}$ na podstawie znaku wyrażenia $\boldsymbol{w}^\top \boldsymbol{x} + b$ (Hastie et al., 2009):

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b) \tag{1.18}$$

Optymalna hiperpłaszczyzna jest wyznaczana przez rozwiązanie problemu optymalizacji (Bishop, 2006; Hastie et al., 2009):

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{przy ograniczeniach} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \ \forall i \tag{1.19}$$

gdzie $y_i \in \{-1, +1\}$ to etykiety klas, a $\boldsymbol{x}_i$ to wektory treningowe. Problem ten jest rozwiązywany za pomocą metod programowania kwadratowego lub przez przekształcenie do postaci dualnej z wykorzystaniem mnożników Lagrange'a (Bishop, 2006).

Soft Margin SVM. W praktyce dane często nie są liniowo separowalne lub zawierają szumy. W takich przypadkach stosuje się wariant soft margin SVM, który dopuszcza błędy klasyfikacji poprzez wprowadzenie zmiennych slackowych $\xi_i \geq 0$ (Hastie et al., 2009; James et al., 2013). Problem optymalizacji przyjmuje wtedy postać:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \tag{1.20}$$

przy ograniczeniach:

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \ \forall i \tag{1.21}$$

gdzie $C > 0$ to parametr regularyzacji kontrolujący kompromis między maksymalizacją marginesu a minimalizacją błędów klasyfikacji. Niskie wartości C preferują większy margines (tolerancja na błędy), wysokie wartości C zmuszają model do dokładniejszego dopasowania danych treningowych (Hastie et al., 2009; James et al., 2013).

Sztuczka jądrowa (Kernel Trick). Gdy dane nie są liniowo separowalne w oryginalnej przestrzeni cech, SVM wykorzystuje funkcje jądrowe (kernel functions) do mapowania danych do przestrzeni o wyższym wymiarze, w której separacja liniowa staje się możliwa (Bishop, 2006; Hastie et al., 2009). Najpopularniejsze funkcje jądrowe to:

Jądro liniowe — dla liniowo separowalnych danych (Hastie et al., 2009):

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j \tag{1.22}$$

Jądro wielomianowe — pozwala na nieliniowe granice decyzyjne (Bishop, 2006; Hastie et al., 2009):

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\gamma \boldsymbol{x}_i^\top \boldsymbol{x}_j + r)^d \tag{1.23}$$

gdzie d to stopień wielomianu, γ to parametr skalowania, a r to wyraz wolny.

Jądro radialnej funkcji bazowej (RBF, Gaussian) — najczęściej stosowane, elastyczne (Bishop, 2006; Hastie et al., 2009):

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right) \tag{1.24}$$

gdzie $\gamma > 0$ kontroluje zasięg wpływu pojedynczych punktów treningowych. Małe γ daje szerokie „dzwony"(prostsze modele), duże γ — wąskie (ryzyko przeuczenia) (Hastie et al., 2009).

Jądro sigmoidalne — zachowuje się podobnie do sieci neuronowych (Bishop, 2006):

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\gamma \boldsymbol{x}_i^\top \boldsymbol{x}_j + r) \tag{1.25}$$

Dzięki sztuczce jądrowej SVM może modelować złożone, nieliniowe granice decyzyjne bez jawnego obliczania transformacji do wyższego wymiaru, co jest kosztowne obliczeniowo (Bishop, 2006; Murphy, 2012).

1.1.7 Naiwny Klasyfikator Bayesowski (Naive Bayes)

Naiwny klasyfikator Bayesowski to probabilistyczny algorytm klasyfikacji oparty na twierdzeniu Bayesa z założeniem warunkowej niezależności cech (Bishop, 2006; Murphy, 2012). Algorytm oblicza prawdopodobieństwo przynależności obiektu do każdej z klas

na podstawie wartości jego cech. Po obliczeniu przypisuje obiekt do klasy o najwyższym prawdopodobieństwie po uwzględnieniu danych. Pomimo upraszczającego założenia o niezależności cech, Naive Bayes często osiąga dobre wyniki, szczególnie w zadaniach klasyfikacji tekstu i filtrowania spamu (Bishop, 2006; Murphy, 2012).

Twierdzenie Bayesa. Podstawą algorytmu jest twierdzenie Bayesa, które wyraża prawdopodobieństwo przynależności obiektu do klasy C_k przy danych cechach $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ (Bishop, 2006):

$$P(C_k|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|C_k)P(C_k)}{P(\boldsymbol{x})} \tag{1.26}$$

gdzie $P(C_k|\boldsymbol{x})$ oznacza prawdopodobieństwo a posteriori (po uwzględnieniu danych) przynależności do klasy C_k przy danych cechach $\boldsymbol{x}$, $P(\boldsymbol{x}|C_k)$ to prawdopodobieństwo wystąpienia cech $\boldsymbol{x}$ w klasie C_k (tzw. likelihood), a $P(C_k)$ to prawdopodobieństwo a priori (przed zobaczeniem danych) klasy C_k. Mianownik $P(\boldsymbol{x})$ jest prawdopodobieństwem wystąpienia cech $\boldsymbol{x}$ i pełni rolę stałej normalizującej (Murphy, 2012).

Warianty algorytmu. W zależności od charakteru danych stosuje się różne warianty Naive Bayes (Murphy, 2012):

Gaussian Naive Bayes. Wariant stosowany dla cech ciągłych, zakłada, że wartości każdej cechy w danej klasie mają rozkład normalny (Gaussa) (Bishop, 2006; Murphy, 2012), a parametry rozkładu (średnia μ_k i wariancja σ_k^2) są estymowane z danych treningowych dla każdej cechy i każdej klasy. Gaussowy wariant naiwnego klasyfikatora Bayesowskiego jest szczególnie skuteczny, gdzie cechy są liczbami rzeczywistymi, takimi jak pomiary fizyczne czy dane sensoryczne (Bishop, 2006).

Multinomial Naive Bayes. Wariant przeznaczony dla cech dyskretnych reprezentujących liczby wystąpień zdarzeń, takich jak częstotliwość występowania słów w dokumentach tekstowych (Murphy, 2012). Modeluje prawdopodobieństwo wystąpienia danej liczby zdarzeń zgodnie z rozkładem wielomianowym. Algorytmem jest szczególnie efektywny w zadaniach, gdzie dane są reprezentowane jako wektory liczników. Ten wariant jest często stosowany do klasyfikacji tekstów, filtrowania spamu oraz analizie sentymentu, gdzie każda cecha reprezentuje liczbę wystąpień określonego słowa (Murphy, 2012; Manning et al., 2008).

Bernoulli Naive Bayes. Wariant dla cech binarnych przyjmujących wartości 0 lub 1 (Murphy, 2012). W odróżnieniu od wariantu wielomianowego, który liczy wystąpienia, Bernoulli Naive Bayes modeluje jedynie fakt obecności cechy. Jest stosowany w klasyfikacji dokumentów, gdzie cechy wskazują, czy dane słowo występuje w dokumencie, niezależnie od liczby jego wystąpień (Murphy, 2012; Manning et al., 2008).

1.2 Uczenie nienadzorowane (Unsupervised Learning)

Uczenie nienadzorowane to rodzaj uczenia maszynowego, który sam odkrywa wzorce i struktury danych bez ówcześnie nadachynych etykiek przez człowieka. Alogrytm przetwarza surowe dane wejściowe, a następnie za pomocą metod statystycznych i geometrycz-

nych grupuje je na podstawie podobieństw lub różnic. Po grupowaniu algorytm redukuje liczbę wymiarów danych, zachowując najważniejsze cechy i wzorce. Uczenie nienadzorowane dzieli się na dwie główne kategorie: klasteryzację i redukcję wymiarowości (Bishop, 2006; Hastie et al., 2009).

Klasteryzacja (Clustering) Klasteryzacja to technika uczenia nienadzorowanego, która polega na grupowaniu podobnych obiektów w zbiory zwane klastrami. Celem klasteryzacji jest identyfikacja naturalnych struktur w danych, gdzie obiekty w tym samym klastrze są bardziej podobne do siebie niż do obiektów z innych klastrów. Algorytmy klasteryzacji wykorzystują różne metryki odległości do oceny podobieństwa między obiektami, takie jak metryka euklidesowa, Manhattan czy kosinusowa, które zostały omówione w sekcji dotyczącej algorytmu k-najbliższych sąsiadów (Duda et al., 2001; Hastie et al., 2009).

Redukcja wymiarowości (Dimensionality Reduction) Redukcja wymiarowości to technika uczenia nienadzorowanego, która polega na zmniejszeniu liczby cech w zbiorze danych przy jednoczesnym zachowaniu jak największej ilości istotnej informacji. Celem redukcji wymiarowości jest uproszczenie modelu, poprawa wydajności obliczeniowej oraz eliminacja szumów i nadmiarowości w danych. Popularne metody redukcji wymiarowości to analiza głównych składowych (PCA) oraz t-SNE (t-distributed Stochastic Neighbor Embedding) (Bishop, 2006; Murphy, 2012).

1.2.1 k-Średnich (k-Means)

Algorytm k-średnich to jedna z metod klasteryzacji, która grupuje dane wejściowe w k klastrów na podstawie podobieństwa między obiektami (Bishop, 2006; Hastie et al., 2009). Algorytm działa iteracyjnie, przypisując każdy obiekt do najbliższego centroidu (środka klastra) i aktualizując położenie centroidów na podstawie średnich wartości obiektów w każdym klastrze. Proces ten powtarza się, aż do osiągnięcia zbieżności, czyli gdy przypisania obiektów do klastrów przestają się zmieniać (Bishop, 2006; Hastie et al., 2009).

$$J = \sum_{j=1}^{k} \sum_{\boldsymbol{x}_i \in C_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2 \tag{1.27}$$

gdzie J to całkowita suma kwadratów błędów, które są minimalizowane iteracyjnie przez algorytm (Hastie et al., 2009), j to iteracja, k to liczba klastrów, $\boldsymbol{x}_i$ to wektor cech obiektu i, C_j to zbiór obiektów przypisanych do klastra j, a $\boldsymbol{\mu}_j$ to centroid klastra j.

W algorytmie k-średnich dobiera sie następujące parametry:

Tabela 1.2: Podstawowe parametry algorytmu k-średnich

Parametr	**Opis**
Liczba klastrów (k)	Liczba klastrów, na które mają być podzielone dane. Wybór odpowiedniej wartości k jest kluczowy dla jakości klasteryzacji (np. metoda łokcia).
Inicjalizacja centroidów	Metoda wyboru początkowych pozycji centroidów (np.metoda k-means++).
Maksymalna liczba iteracji	Maksymalna liczba iteracji, które algorytm wykona przed zatrzymaniem.
Metryka odległości	Metryka używana do obliczania odległości między obiektami a centroidami, (np. metryka euklidesowa, Manhattan czy kosinusowa, które zostały opisane w sekcji dotyczącej algorytmu k-najbliższych sąsiadów).

Metoda łokcia (Elbow Method). Metoda łokcia polega na uruchomieniu algorytmu k-średnich dla różnych wartości k i obliczeniu sumy kwadratów błędów (SSE) dla każdej wartości (Hastie et al., 2009). Następnie wykreśla się wykres SSE w funkcji k i szuka punktu, w którym dalsze zwiększanie k prowadzi do niewielkiej redukcji SSE, tworząc charakterystyczny "łokieć"na wykresie.

Metoda k-means++. Metoda wyboru centroidów k-means++ polega na wyborze początkowych centroidów, aby przyspieszyć zbieżność algorytmu i poprawić jakość klasteryzacji (Arthur and Vassilvitskii, 2007). Pierwszy centroid jest wybierany losowo z danych, a kolejne centroidy są wybierane z prawdopodobieństwem proporcjonalnym do kwadratu odległości od najbliższego już wybranego centroidu. Ta metoda zmniejsza ryzyko złego rozmieszczenia początkowych centroidów, co może prowadzić do gorszych wyników klasteryzacji (Arthur and Vassilvitskii, 2007).

1.2.2 Hierarchiczna klasteryzacja (Hierarchical Clustering)

Hierarchiczna klasteryzacja to metoda klasteryzacji, która tworzy hierarchię klastrów poprzez iteracyjne łączenie lub dzielenie grup obiektów na podstawie ich podobieństwa (Hastie et al., 2009; Bishop, 2006). Istnieją dwa główne podejścia do hierarchicznej klasteryzacji: aglomeracyjne (bottom-up) i dzielące (top-down). W podejściu aglomeracyjnym każdy obiekt zaczyna jako oddzielny klaster, a następnie iteracyjnie łączy się najbliższe klastry, aż do osiągnięcia jednej grupy lub określonej liczby klastrów. W podejściu dzielącym cały zbiór danych zaczyna jako jeden klaster, który jest następnie dzielony na mniejsze klastry na podstawie podobieństwa obiektów (Hastie et al., 2009; Bishop, 2006).

Hierarchiczna klasteryzacja wykorzystuje różne metryki odległości do oceny podobieństwa między obiektami lub klastrami, takie jak metryka euklidesowa, Manhattan czy kosinusowa, które zostały omówione w sekcji dotyczącej algorytmu k-najbliższych sąsiadów (Duda et al., 2001; Hastie et al., 2009). W hierarchicznej klasteryzacji stosuje się również różne metody łączenia klastrów, takie jak (Hastie et al., 2009):

Metoda pojedynczego łączenia (Single Linkage). Metoda pojedynczego łączenia definiuje odległość między dwoma klastrami jako minimalną odległość między dowolnymi dwoma obiektami z tych klastrów (Hastie et al., 2009):

$$d(C_i, C_j) = \min_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} d(\boldsymbol{x}, \boldsymbol{y}) \tag{1.28}$$

gdzie C_i i C_j to dwa klastry, a $d(\boldsymbol{x}, \boldsymbol{y})$ to odległość między obiektami $\boldsymbol{x}$ i $\boldsymbol{y}$. Jest to podejście zachłanne, które może prowadzić do tworzenia długich, cienkich klastrów (tzw. efekt łańcucha) (Hastie et al., 2009).

Metoda pełnego łączenia (Complete Linkage). Metoda pełnego łączenia definiuje odległość między dwoma klastrami jako maksymalną odległość między dowolnymi dwoma obiektami z tych klastrów (Hastie et al., 2009):

$$d(C_i, C_j) = \max_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} d(\boldsymbol{x}, \boldsymbol{y}) \tag{1.29}$$

To podejście prowadzi do tworzenia bardziej zwartych klastrów, ale może być wrażliwe na odległe punkty (outliers) (Hastie et al., 2009).

Metoda średniego łączenia (Average Linkage). Metoda średniego łączenia definiuje odległość między dwoma klastrami jako średnią odległość między wszystkimi parami obiektów z tych klastrów (Hastie et al., 2009):

$$d(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{\boldsymbol{x} \in C_i} \sum_{\boldsymbol{y} \in C_j} d(\boldsymbol{x}, \boldsymbol{y}) \tag{1.30}$$

gdzie $|C_i|$ i $|C_j|$ to liczby obiektów w klastrach C_i i C_j. To podejście stanowi kompromis między metodą pojedynczego i pełnego łączenia, tworząc bardziej zrównoważone klastry (Hastie et al., 2009).

1.2.3 DBSCAN (Density-Based Spatial Clustering)

DBSCAN to algorytm klasteryzacji oparty na gęstości, który grupuje razem punkty znajdujące się blisko siebie w przestrzeni cech, definiując klastry jako obszary o wysokiej gęstości punktów oddzielone od siebie obszarami o niskiej gęstości (Ester et al., 1996). Algorytm DBSCAN identyfikuje klastry na podstawie dwóch głównych parametrów: promienia sąsiedztwa ε (epsilon) oraz minimalnej liczby punktów $minPts$ wymaganej do utworzenia gęstego regionu. Punkty są klasyfikowane jako rdzeniowe, brzegowe lub szumowe w zależności od liczby sąsiadów w promieniu ε (Ester et al., 1996).

Sąsiedztwo ε. Dla punktu $\boldsymbol{p}$ sąsiedztwo ε definiowane jest jako zbiór punktów (Ester et al., 1996):

$$N_\varepsilon(\boldsymbol{p}) = \{\boldsymbol{q} \in D \mid d(\boldsymbol{p}, \boldsymbol{q}) \leq \varepsilon\} \tag{1.31}$$

gdzie D to zbiór wszystkich punktów, a $d(\boldsymbol{p}, \boldsymbol{q})$ to odległość między punktami $\boldsymbol{p}$ i $\boldsymbol{q}$.

Punkt rdzeniowy (core point). Punkt $\boldsymbol{p}$ jest punktem rdzeniowym, jeśli liczba punktów w jego sąsiedztwie ε wynosi co najmniej $minPts$ (Ester et al., 1996):

$$|N_\varepsilon(\boldsymbol{p})| \geq minPts \tag{1.32}$$

Bezpośrednia osiągalność gęstościowa. Punkt $\boldsymbol{q}$ jest bezpośrednio osiągalny gęstościowo z punktu $\boldsymbol{p}$, jeśli $\boldsymbol{p}$ jest punktem rdzeniowym i $\boldsymbol{q} \in N_\varepsilon(\boldsymbol{p})$ (Ester et al., 1996).

Osiągalność gęstościowa. Punkt $\boldsymbol{q}$ jest osiągalny gęstościowo z punktu $\boldsymbol{p}$, jeśli istnieje łańcuch punktów $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n$, gdzie $\boldsymbol{p}_1 = \boldsymbol{p}$ i $\boldsymbol{p}_n = \boldsymbol{q}$, taki że każdy kolejny punkt jest bezpośrednio osiągalny gęstościowo z poprzedniego (Ester et al., 1996).

Klasyfikacja punktów w DBSCAN. Algorytm DBSCAN klasyfikuje każdy punkt w zbiorze danych do jednej z trzech kategorii (Ester et al., 1996):

Punkt rdzeniowy (core point): punkt $\boldsymbol{p}$ jest rdzeniowy, jeśli ma co najmniej $minPts$ sąsiadów w promieniu ε, tj. $|N_\varepsilon(\boldsymbol{p})| \geq minPts$. Punkty rdzeniowe stanowią centrum klastrów i inicjują ich tworzenie.

Punkt brzegowy (border point): punkt należący do klastra, ale niebędący punktem rdzeniowym. Punkt brzegowy leży w sąsiedztwie ε co najmniej jednego punktu rdzeniowego, ale sam ma mniej niż $minPts$ sąsiadów. Punkty brzegowe znajdują się na peryferiach klastrów i są przypisywane do klastra poprzez bezpośrednią osiągalność gęstościową z punktu rdzeniowego.

Punkt szumowy (noise point): punkt, który nie jest ani rdzeniowy, ani brzegowy. Punkt szumowy nie leży w sąsiedztwie ε żadnego punktu rdzeniowego i nie należy do żadnego klastra. Punkty szumowe reprezentują obserwacje odstające (outliers) lub szum w danych.

Dzięki tej klasyfikacji DBSCAN automatycznie identyfikuje i odrzuca punkty odstające, co czyni go odpornym na szum w danych (Ester et al., 1996).

1.2.4 PCA (Principal Component Analysis)

Analiza głównych składowych (PCA) to technika redukcji wymiarowości, która przekształca oryginalne cechy danych w nowy zestaw nieskorelowanych zmiennych zwanych głównymi składowymi (Murphy, 2012; Hastie et al., 2009). Główne składowe są liniowymi kombinacjami oryginalnych cech i są uporządkowane według wariancji, którą wyjaśniają w danych. Pierwsza główna składowa wyjaśnia największą część wariancji, druga główna składowa wyjaśnia drugą co do wielkości część wariancji, i tak dalej. PCA jest szeroko stosowana do wizualizacji danych, kompresji danych oraz usuwania szumów (Murphy, 2012; Hastie et al., 2009).

Macierz kowariancji. Algorytm PCA rozpoczyna się od wycentrowania danych (odjęcia średniej od każdej cechy) i obliczenia macierzy kowariancji, która opisuje zależności między cechami (Bishop, 2006; Hastie et al., 2009):

$$\Sigma = \frac{1}{n} X^\top X \tag{1.33}$$

gdzie X to macierz danych o wymiarach $n \times p$ (po wycentrowaniu), gdzie n to liczba próbek, a p to liczba cech. Macierz Σ ma wymiary $p \times p$.

Rozkład własny (Eigendecomposition). Kolejnym krokiem jest znalezienie wektorów własnych i wartości własnych macierzy kowariancji poprzez rozwiązanie równania (Bishop, 2006; Hastie et al., 2009):

$$\Sigma \boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \tag{1.34}$$

gdzie $\boldsymbol{v}_i$ to i-ty wektor własny określający kierunek i-tej głównej składowej, a λ_i to odpowiadająca mu wartość własna reprezentująca wariancję wyjaśnianą przez tę składową. Wektory własne są ortogonalne, co zapewnia nieskorelowanie głównych składowych (Hastie et al., 2009).

Transformacja danych. Po uporządkowaniu wektorów własnych według malejących wartości własnych ($\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p$), dane są transformowane do nowej przestrzeni głównych składowych (Hastie et al., 2009):

$$Z = XW \tag{1.35}$$

gdzie W to macierz o wymiarach $p \times k$ zawierająca k pierwszych wektorów własnych jako kolumny, a Z to macierz danych w nowej przestrzeni o wymiarach $n \times k$. Wybór liczby k składowych do zachowania zależy od wymaganej ilości wyjaśnianej wariancji.

Wariancja wyjaśniona. Proporcja wariancji wyjaśnianej przez i-tą główną składową wynosi (Hastie et al., 2009):

$$\text{Var}_i = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j} \tag{1.36}$$

Suma wariancji wyjaśnianych przez pierwsze k składowych określa, jaki procent całkowitej zmienności danych został zachowany po redukcji wymiarowości. W praktyce często wybiera się k takie, aby zachować 90-95% wariancji (Bishop, 2006; Hastie et al., 2009).

1.2.5 t-SNE (t-distributed Stochastic Neighbor Embedding)

t-SNE to technika redukcji wymiarowości, która przekształca dane wysokowymiarowe w przestrzeń niskowymiarową (zazwyczaj 2D lub 3D) w taki sposób, aby zachować lokalne struktury danych (van der Maaten and Hinton, 2008). Algorytm t-SNE modeluje podobieństwa między punktami w oryginalnej przestrzeni jako prawdopodobieństwa warunkowe, a następnie optymalizuje rozmieszczenie punktów w przestrzeni niskowymiarowej, minimalizując różnicę między tymi prawdopodobieństwami za pomocą dywergencji Kullbacka-Leiblera (van der Maaten and Hinton, 2008).

Podobieństwa w przestrzeni wysokowymiarowej. W przestrzeni wysokowymiarowej podobieństwo między punktami $\boldsymbol{x}_i$ i $\boldsymbol{x}_j$ jest modelowane jako prawdopodobieństwo warunkowe (van der Maaten and Hinton, 2008):

$$p_{j|i} = \frac{\exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2 / 2\sigma_i^2)} \tag{1.37}$$

gdzie σ_i to szerokość jądra Gaussa dla punktu $\boldsymbol{x}_i$. Prawdopodobieństwo symetryczne definiuje się jako:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{1.38}$$

gdzie n to liczba punktów danych.

Podobieństwa w przestrzeni niskowymiarowej. W przestrzeni niskowymiarowej podobieństwo między punktami $\boldsymbol{y}_i$ i $\boldsymbol{y}_j$ jest modelowane za pomocą rozkładu t-Studenta z jednym stopniem swobody (van der Maaten and Hinton, 2008):

$$q_{ij} = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)^{-1}} \tag{1.39}$$

Optymalizacja. Algorytm t-SNE minimalizuje dywergencję Kullbacka-Leiblera między rozkładami P i Q (van der Maaten and Hinton, 2008):

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{1.40}$$

Optymalizacja jest przeprowadzana za pomocą gradientu prostego lub metod opartych na momencie, co pozwala na znalezienie układu punktów $\boldsymbol{y}_i$ w przestrzeni niskowymiarowej, który najlepiej zachowuje lokalne struktury danych z przestrzeni wysokowymiarowej (van der Maaten and Hinton, 2008).

Parametry t-SNE. Algorytm t-SNE posiada kilka kluczowych parametrów, które wpływają na jakość i charakter wynikowej wizualizacji (van der Maaten and Hinton, 2008):

- **Perplexity:** Określa liczbę najbliższych sąsiadów branych pod uwagę przy obliczaniu podobieństw w przestrzeni wysokowymiarowej. Typowe wartości to od 5 do 50. Wyższe wartości prowadzą do bardziej globalnych struktur, podczas gdy niższe wartości skupiają się na lokalnych strukturach.
- **Liczba iteracji:** Określa, ile razy algorytm będzie aktualizował położenia punktów w przestrzeni niskowymiarowej. Większa liczba iteracji może prowadzić do lepszej konwergencji, ale zwiększa czas obliczeń.
- **Współczynnik uczenia (learning rate):** Kontroluje szybkość aktualizacji położeń punktów podczas optymalizacji. Zbyt wysoki współczynnik może prowadzić do niestabilności, podczas gdy zbyt niski może spowolnić zbieżność.

Dobranie powyższych parametrów jest głównym czynnikiem wpływajacym na czytelność i jakość wizualizacji (van der Maaten and Hinton, 2008).

1.2.6 UMAP (Uniform Manifold Approximation and Projection)

UMAP to technika redukcji wymiarowości i wizualizacji danych, która opiera się na teorii topologii i geometrii różniczkowej (McInnes et al., 2018). UMAP tworzy niskowymiarową reprezentację danych wysokowymiarowych, zachowując zarówno lokalne, jak i globalne struktury danych. Algorytm UMAP składa się z dwóch głównych etapów: konstrukcji grafu sąsiedztwa w przestrzeni wysokowymiarowej oraz optymalizacji rozmieszczenia punktów w przestrzeni niskowymiarowej (McInnes et al., 2018).

Konstrukcja grafu sąsiedztwa. W pierwszym etapie UMAP buduje graf sąsiedztwa, w którym każdy punkt danych jest połączony z jego k najbliższymi sąsiadami (McInnes et al., 2018). Podobieństwo między punktami $\boldsymbol{x}_i$ i $\boldsymbol{x}_j$ jest modelowane za pomocą funkcji ważonej:

$$w_{ij} = \exp\left(-\frac{d(\boldsymbol{x}_i, \boldsymbol{x}_j) - \rho_i}{\sigma_i}\right) \tag{1.41}$$

gdzie $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to odległość między punktami, ρ_i to odległość do najbliższego sąsiada punktu $\boldsymbol{x}_i$, a σ_i to skalowanie lokalne kontrolujące gęstość sąsiedztwa.

Optymalizacja w przestrzeni niskowymiarowej. W drugim etapie UMAP optymalizuje rozmieszczenie punktów $\boldsymbol{y}_i$ w przestrzeni niskowymiarowej, minimalizując funkcję kosztu opartą na różnicy między grafem sąsiedztwa w przestrzeni wysokowymiarowej a grafem w przestrzeni niskowymiarowej (McInnes et al., 2018):

$$C = \sum_{i \neq j} \left(w_{ij} \log \frac{w_{ij}}{q_{ij}} + (1 - w_{ij}) \log \frac{1 - w_{ij}}{1 - q_{ij}} \right) \tag{1.42}$$

gdzie q_{ij} to podobieństwo między punktami $\boldsymbol{y}_i$ i $\boldsymbol{y}_j$ w przestrzeni niskowymiarowej, modelowane za pomocą funkcji:

$$q_{ij} = \frac{1}{1 + a\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^{2b}} \tag{1.43}$$

gdzie a i b to parametry kontrolujące kształt funkcji podobieństwa. Optymalizacja jest przeprowadzana za pomocą metod gradientu prostego lub jego wariantów, co pozwala na znalezienie układu punktów $\boldsymbol{y}_i$ w przestrzeni niskowymiarowej, który najlepiej zachowuje struktury danych z przestrzeni wysokowymiarowej (McInnes et al., 2018). **Parametry**

UMAP. Algorytm UMAP posiada kilka kluczowych parametrów, które wpływają na jakość i charakter wynikowej wizualizacji (McInnes et al., 2018):

- **Liczba sąsiadów (n_neighbors):** Określa liczbę najbliższych sąsiadów branych pod uwagę przy budowie grafu sąsiedztwa. Typowe wartości to od 5 do 50. Wyższe wartości prowadzą do bardziej globalnych struktur, podczas gdy niższe wartości skupiają się na lokalnych strukturach.
- **Minimalna odległość (min_dist):** Kontroluje, jak blisko punkty mogą być rozmieszczone w przestrzeni niskowymiarowej. Niższe wartości prowadzą do bardziej skondensowanych klastrów, podczas gdy wyższe wartości rozpraszają punkty bardziej równomiernie.

- **Liczba wymiarów docelowych (n_components):** Określa liczbę wymiarów w przestrzeni niskowymiarowej (zazwyczaj 2 lub 3).
- **Liczba iteracji (n_epochs):** Określa, ile razy algorytm będzie aktualizował położenia punktów w przestrzeni niskowymiarowej. Większa liczba iteracji może prowadzić do lepszej konwergencji, ale zwiększa czas obliczeń.

Dopasowanie parametrów takich jak liczba sąsiadów (n_neighbors), minimalna odległość (min_dist), liczba wymiarów docelowych i epochs przesądza o równowadze między zachowaniem lokalnych struktur danych a globalnej topologii w wizualizacjach UMAP (McInnes et al., 2018).

1.3 Uczenie półnadzorowane (Semi-Supervised Learning)

Uczenie półnadzorowane to podejście w uczeniu maszynowego, które łączy cechy uczenia nadzorowanego i nienadzorowanego, wykorzystując zarówno dane oznaczone etykietami, jak i dane bez etykiet do trenowania modelu. Ten rodzaj uczenia wykorzystuje niewielki zbiór danych oznaczonych do nauki podstawowej struktury problemu. W kolejnym etapie wykorzystuje duży zbiór danych nieoznaczonych do poprawy uogólnienia modelu i lepszego zrozumienia rozkładu danych. Skuteczność tych metod opiera się na założeniu, że struktura danych nieoznaczonych dostarcza informacji o warunkowym rozkładzie etykiet (Chapelle et al., 2006; Zhu and Goldberg, 2009).

Założenia uczenia półnadzorowanego. Skuteczność uczenia półnadzorowanego opiera się na kilku kluczowych założeniach dotyczących struktury danych (Chapelle et al., 2006; Zhu and Goldberg, 2009):

Założenie płynności (Smoothness Assumption): Jeśli dwa punkty $\boldsymbol{x}_1$ i $\boldsymbol{x}_2$ w przestrzeni cech są blisko siebie, to ich odpowiadające etykiety y_1 i y_2 powinny być podobne. Oznacza to, że funkcja decyzyjna powinna być płynna w obszarach o wysokiej gęstości danych.

Założenie klastrów (Cluster Assumption): Dane mają tendencję do tworzenia odrębnych klastrów, a punkty w tym samym klastrze prawdopodobnie należą do tej samej klasy. Granice decyzyjne powinny przebiegać przez obszary o niskiej gęstości danych, dzieląc różne klastry.

Założenie rozmaitości (Manifold Assumption): Dane wysokowymiarowe leżą w przybliżeniu na niskowymiarowej rozmaitości. Punkty, które są bliskie na tej rozmaitości, powinny mieć podobne etykiety, nawet jeśli są daleko od siebie w oryginalnej przestrzeni wysokowymiarowej.

1.3.1 Self-Training (Samouczenie)

Self-training to jedna z najprostszych metod uczenia półnadzorowanego, która działa w sposób iteracyjny, wykorzystując własne predykcje modelu do etykietowania danych nieoznaczonych. Algorytm rozpoczyna od wytrenowania modelu na małym zbiorze danych

oznaczonych, a następnie używa tego modelu do predykcji etykiet dla danych nieoznaczonych. Dane nieoznaczone, dla których model jest najbardziej pewny swoich predykcji, są dodawane do zbioru treningowego wraz z przewidywanymi etykietami. Model jest następnie ponownie trenowany na powiększonym zbiorze danych, a proces powtarza się iteracyjnie, aż do wyczerpania danych nieoznaczonych lub osiągnięcia kryterium stopu (Zhu and Goldberg, 2009).

Algorytm self-training rozpoczyna się od wytrenowania klasyfikatora na małym zbiorze oznaczonych danych. Następnie klasyfikator jest wykorzystywany do przewidywania etykiet dla danych nieoznaczonych. Dla każdej predykcji obliczany jest poziom pewności, który określa, jak pewny jest model swojej decyzji. Wybierany jest podzbiór predykcji, dla których poziom pewności przekracza ustalony próg, i predykcje te wraz z przewidywanymi etykietami są dodawane do zbioru treningowego. Klasyfikator jest następnie trenowany ponownie na powiększonym zbiorze danych. Proces powtarza się iteracyjnie — kroki przewidywania, wyboru pewnych predykcji i retrenowania — aż do osiągnięcia kryterium stopu, którym może być wyczerpanie danych nieoznaczonych, brak nowych pewnych predykcji lub osiągnięcie maksymalnej liczby iteracji (Zhu and Goldberg, 2009).

Głównym ryzykiem self-training jest kumulacja błędów — jeśli model popełni błąd w przewidywaniu etykiety i dodaje ją do zbioru treningowego, błędne etykiety mogą propagować się i wzmacniać w kolejnych iteracjach, prowadząc do degradacji wydajności (Zhu and Goldberg, 2009).

1.3.2 Co-Training

Co-training to metoda uczenia półnadzorowanego zaproponowana dla sytuacji, gdy dane można naturalnie opisać za pomocą dwóch różnych, niezależnych zestawów cech. Każdy zestaw cech powinien być wystarczający do trenowania dobrego klasyfikatora, a widoki powinny być warunkowo niezależne przy danej etykiecie klasy. Dwa klasyfikatory są trenowane oddzielnie na różnych zestawach cech, a następnie każdy klasyfikator etykietuje dane nieoznaczone dla drugiego klasyfikatora, wykorzystując najbardziej pewne predykcje. Ta wzajemna nauka pozwala klasyfikatorom uczyć się od siebie nawzajem, wykorzystując komplementarną informację z różnych perspektyw danych (Zhu and Goldberg, 2009).

Algorytm co-training (Zhu and Goldberg, 2009): Co-training rozpoczyna się od podziału cech na dwa niezależne widoki $\boldsymbol{x}^{(1)}$ i $\boldsymbol{x}^{(2)}$. Następnie trenowane są dwa klasyfikatory f_1 i f_2 na małym zbiorze danych oznaczonych, każdy wykorzystując tylko swój widok danych. W kolejnym kroku każdy klasyfikator f_i przewiduje etykiety dla wszystkich danych nieoznaczonych. Klasyfikator f_1 wybiera n najbardziej pewnych pozytywnych i n najbardziej pewnych negatywnych przykładów i dodaje je do zbioru treningowego dla f_2, podczas gdy klasyfikator f_2 wykonuje analogiczną operację, dodając swoje najbardziej pewne predykcje do zbioru treningowego dla f_1. Oba klasyfikatory są następnie trenowane ponownie na powiększonych zbiorach danych. Proces powtarza się przez ustaloną liczbę iteracji lub do momentu wyczerpania danych nieoznaczonych (Zhu and Goldberg, 2009).

1.3.3 Transductive SVM (TSVM)

Transductive Support Vector Machine to rozszerzenie standardowego SVM na uczenie półnadzorowane, które maksymalizuje margines zarówno dla danych oznaczonych, jak i nieoznaczonych. TSVM poszukuje hiperpłaszczyzny, która nie tylko poprawnie klasyfikuje dane oznaczone z maksymalnym marginesem, ale również dzieli dane nieoznaczone tak, aby granica decyzyjna przechodziła przez obszary o niskiej gęstości punktów. Algorytm realizuje założenie klastrów (cluster assumption), dążąc do sytuacji, w której jak najmniej danych nieoznaczonych znajduje się blisko granicy decyzyjnej (Chapelle et al., 2006).

Funkcja celu TSVM. Problem optymalizacji dla TSVM można zapisać jako (Chapelle et al., 2006):

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*,\boldsymbol{y}^*} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{l}\xi_i + C^*\sum_{j=1}^{u}\xi_j^* \tag{1.44}$$

przy ograniczeniach dla danych oznaczonych:

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, l \tag{1.45}$$

i dla danych nieoznaczonych z przewidywanymi etykietami $y_j^* \in \{-1, +1\}$:

$$y_j^*(\boldsymbol{w}^\top \boldsymbol{x}_j + b) \geq 1 - \xi_j^*, \quad \xi_j^* \geq 0, \quad j = 1, \ldots, u \tag{1.46}$$

gdzie C i C^* to parametry regularyzacji dla danych oznaczonych i nieoznaczonych odpowiednio.

Problem jest trudny obliczeniowo (NP-trudny), ponieważ wymaga optymalizacji również po etykietach danych nieoznaczonych y_j^*. W praktyce stosuje się heurystyki i przybliżone algorytmy optymalizacji (Chapelle et al., 2006).

1.3.4 Graph-Based Methods (Metody grafowe)

Metody grafowe w uczeniu półnadzorowanym konstruują graf, w którym węzły reprezentują wszystkie punkty danych (zarówno oznaczone, jak i nieoznaczone), a krawędzie reprezentują podobieństwo między nimi. Założenie jest takie, że etykiety powinny zmieniać się płynnie wzdłuż grafu — punkty połączone silnymi krawędziami (wysokie podobieństwo) powinny mieć podobne etykiety. Etykiety są propagowane od węzłów oznaczonych do nieoznaczonych zgodnie z tą zasadą, wykorzystując strukturę grafu do wnioskowania o etykietach (Zhu and Goldberg, 2009).

Konstrukcja grafu. Najpopularniejsze metody konstrukcji grafu to (Zhu and Goldberg, 2009):

- **k-nearest neighbor graph** — każdy punkt jest połączony z k najbliższymi sąsiadami.
- **ε-neighborhood graph** — punkty są połączone, jeśli odległość między nimi jest mniejsza niż ε.

- **Fully connected graph** — wszystkie punkty są połączone, a wagi krawędzi maleją z odległością.

Macierz podobieństwa (wag krawędzi) W często definiuje się za pomocą jądra Gaussa (Zhu and Goldberg, 2009):

$$W_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right) \tag{1.47}$$

gdzie σ kontroluje szerokość jądra.

Label Propagation. Algorytm label propagation propaguje etykiety przez graf iteracyjnie. Niech Y będzie macierzą etykiet rozmiaru $n \times c$, gdzie n to liczba punktów, a c to liczba klas. Dla danych oznaczonych $Y_{il} = 1$ jeśli punkt i należy do klasy l, w przeciwnym razie $Y_{il} = 0$.

Macierz przejścia P definiuje się jako znormalizowaną macierz wag (Zhu and Goldberg, 2009):

$$P_{ij} = \frac{W_{ij}}{\sum_{k=1}^{n} W_{ik}} \tag{1.48}$$

Algorytm iteracyjnie aktualizuje etykiety według (Zhu and Goldberg, 2009):

$$Y^{(t+1)} = PY^{(t)} \tag{1.49}$$

gdzie po każdej iteracji etykiety danych oznaczonych są resetowane do oryginalnych wartości, aby zachować pewność co do prawdziwych etykiet.

Label Spreading. Label spreading to wariant label propagation, który pozwala na miękką zmianę również etykiet danych oznaczonych (z mniejszą wagą). Algorytm minimalizuje funkcję kosztu (Zhu and Goldberg, 2009):

$$E(Y) = \frac{1}{2}\sum_{i,j=1}^{n} W_{ij}\left\|\frac{Y_i}{\sqrt{D_{ii}}} - \frac{Y_j}{\sqrt{D_{jj}}}\right\|^2 + \mu\sum_{i=1}^{l}\|Y_i - Y_i^0\|^2 \tag{1.50}$$

gdzie D to macierz diagonalna stopni węzłów ($D_{ii} = \sum_j W_{ij}$), Y_i^0 to oryginalne etykiety danych oznaczonych, a μ kontroluje siłę przywiązania do oryginalnych etykiet. Rozwiązanie w stanie równowagi można znaleźć analitycznie.

1.3.5 Generative Models (Modele generatywne)

Modele generatywne w uczeniu półnadzorowanym modelują łączny rozkład prawdopodobieństwa $P(\boldsymbol{x}, y)$ dla cech i etykiet. Kluczowa idea polega na tym, że dane nieoznaczone pomagają w lepszej estymacji rozkładu marginalnego $P(\boldsymbol{x})$, co z kolei poprawia estymację rozkładu warunkowego $P(y|\boldsymbol{x})$ poprzez współdzielenie parametrów. Najczęściej stosuje się mieszaniny rozkładów Gaussa (Gaussian Mixture Models), gdzie zakłada się, że każda klasa jest generowana z mieszaniny komponentów gaussowskich (Chapelle et al., 2006; Murphy, 2012).

Semi-Supervised GMM. Model zakłada, że dane są generowane z mieszaniny K rozkładów Gaussa (Murphy, 2012):

$$P(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) \tag{1.51}$$

gdzie π_k to wagi komponentów ($\sum_k \pi_k = 1$), $\boldsymbol{\mu}_k$ to średnie, a Σ_k to macierze kowariancji.

Dla danych oznaczonych log-likelihood wynosi (Chapelle et al., 2006):

$$\log L_{\text{labeled}} = \sum_{i=1}^{l} \log P(y_i, \boldsymbol{x}_i) = \sum_{i=1}^{l} \log \sum_{k \in C_{y_i}} \pi_k \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k) \tag{1.52}$$

gdzie C_{y_i} to zbiór komponentów odpowiadających klasie y_i.

Dla danych nieoznaczonych (Chapelle et al., 2006):

$$\log L_{\text{unlabeled}} = \sum_{j=1}^{u} \log P(\boldsymbol{x}_j) = \sum_{j=1}^{u} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_j|\boldsymbol{\mu}_k, \Sigma_k) \tag{1.53}$$

Całkowita log-likelihood:

$$\log L = \log L_{\text{labeled}} + \log L_{\text{unlabeled}} \tag{1.54}$$

Parametry są estymowane za pomocą algorytmu EM (Expectation-Maximization), który w kroku E oblicza prawdopodobieństwa przynależności punktów do komponentów, a w kroku M aktualizuje parametry modelu na podstawie tych prawdopodobieństw (Murphy, 2012).

1.3.6 Entropy Minimization (Minimalizacja entropii)

Minimalizacja entropii to podejście oparte na zasadzie, że dobry model powinien być pewny swoich predykcji również dla danych nieoznaczonych. Entropia rozkładu predykcji mierzy niepewność modelu — niska entropia oznacza pewne predykcje (rozkład skoncentrowany na jednej klasie), wysoka entropia oznacza niepewność (rozkład rozłożony równomiernie). Metoda dodaje do funkcji straty składnik karny, który minimalizuje entropię predykcji dla danych nieoznaczonych, zmuszając model do dokonywania pewnych klasyfikacji (Chapelle et al., 2006).

Funkcja straty. Całkowita funkcja straty składa się z nadzorowanego składnika dla danych oznaczonych i składnika minimalizacji entropii dla danych nieoznaczonych (Chapelle et al., 2006):

$$L = L_{\text{supervised}} + \lambda L_{\text{entropy}} \tag{1.55}$$

gdzie:

$$L_{\text{supervised}} = \frac{1}{l} \sum_{i=1}^{l} \mathcal{L}(f(\boldsymbol{x}_i), y_i) \tag{1.56}$$

to standardowa strata klasyfikacji dla danych oznaczonych, a:

$$L_{\text{entropy}} = -\frac{1}{u}\sum_{j=1}^{u}\sum_{k=1}^{c} p_k^{(j)} \log p_k^{(j)} \tag{1.57}$$

to entropia predykcji dla danych nieoznaczonych, gdzie $p_k^{(j)} = P(y = k|\boldsymbol{x}_j)$ to przewidywane prawdopodobieństwo klasy k dla punktu j. Parametr λ kontroluje wagę składnika entropijnego.

Minimalizacja entropii jest szczególnie efektywna w połączeniu z innymi metodami półnadzorowanymi i jest często używana jako regularyzator w głębokich sieciach neuronowych (Chapelle et al., 2006).

Zastosowania uczenia półnadzorowanego. Uczenie półnadzorowane znajduje zastosowanie w wielu dziedzinach, gdzie etykietowanie danych jest kosztowne (Chapelle et al., 2006; Zhu and Goldberg, 2009; Hastie et al., 2009):

- **Klasyfikacja tekstów i dokumentów** — gdzie ręczne oznaczanie dokumentów jest czasochłonne, ale dostępne są ogromne zbiory tekstów nieoznaczonych.
- **Rozpoznawanie obrazów i wizja komputerowa** — wykorzystanie milionów zdjęć bez etykiet do poprawy klasyfikatorów wytrenowanych na małych zbiorach oznaczonych.
- **Bioinformatyka i medycyna** — analiza sekwencji DNA/RNA lub obrazów medycznych, gdzie eksperymentalne potwierdzenie etykiet jest bardzo drogie.
- **Kontrola jakości w produkcji** — w procesach wytwórczych, gdzie dostępne są ogromne ilości danych z czujników (nieoznaczonych), ale tylko nieliczne przypadki defektów są zidentyfikowane i oznaczone.
- **Systemy rekomendacyjne** — gdzie mamy dużo danych o interakcjach użytkowników, ale mało jawnych ocen.

Uczenie półnadzorowane jest szczególnie wartościowe w rzeczywistych zastosowaniach przemysłowych, gdzie koszt eksperckich etykiet jest wysoki, a dane nieoznaczone są generowane automatycznie przez systemy monitorujące i czujniki (Chapelle et al., 2006; Zhu and Goldberg, 2009).

1.4 Uczenie ze wzmocnieniem (Reinforcement Learning)

Uczenie ze wzmocnieniem to paradygmat uczenia maszynowego, w którym agent uczy się podejmować decyzje poprzez interakcję ze środowiskiem w celu maksymalizacji skumulowanej nagrody. W przeciwieństwie do uczenia nadzorowanego, gdzie model uczy się na podstawie par wejście-wyjście, w uczeniu ze wzmocnieniem agent otrzymuje jedynie sygnały zwrotne w postaci nagród lub kar za wykonane akcje. Agent musi samodzielnie odkryć, które akcje prowadzą do największych nagród, ucząc się poprzez eksperymentowanie i doświadczenie. Uczenie ze wzmocnieniem jest szczególnie efektywne w problemach

sekwencyjnego podejmowania decyzji, gdzie długoterminowe konsekwencje akcji są ważniejsze niż natychmiastowe nagrody (Sutton and Barto, 2018).

Podstawowe elementy uczenia ze wzmocnieniem. System uczenia ze wzmocnieniem składa się z kilku kluczowych elementów (Sutton and Barto, 2018):

Agent: System podejmujący decyzje, który uczy się optymalnej strategii działania poprzez interakcję ze środowiskiem.

Środowisko (Environment): Wszystko to, z czym agent wchodzi w interakcję i co znajduje się poza jego bezpośrednią kontrolą. Środowisko odbiera akcje agenta i zwraca nowe stany oraz nagrody.

Stan (State): Reprezentacja bieżącej sytuacji w środowisku, oznaczana jako $s \in \mathcal{S}$, gdzie $\mathcal{S}$ to przestrzeń wszystkich możliwych stanów.

Akcja (Action): Decyzja podejmowana przez agenta w danym stanie, oznaczana jako $a \in \mathcal{A}$, gdzie $\mathcal{A}$ to przestrzeń wszystkich możliwych akcji.

Nagroda (Reward): Sygnał numeryczny $r \in \mathbb{R}$ otrzymywany przez agenta od środowiska po wykonaniu akcji, wskazujący na natychmiastową wartość tej akcji.

Polityka (Policy): Strategia agenta określająca, jaką akcję wybrać w danym stanie, oznaczana jako $\pi(a|s)$ dla polityki stochastycznej lub $\pi(s)$ dla polityki deterministycznej.

Proces decyzyjny Markova (Markov Decision Process). Formalne podstawy uczenia ze wzmocnieniem opierają się na procesie decyzyjnym Markova (MDP), który definiuje się jako krotkę $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ (Sutton and Barto, 2018), gdzie:

- $\mathcal{S}$ — zbiór stanów,
- $\mathcal{A}$ — zbiór akcji,
- $P(s'|s, a)$ — prawdopodobieństwo przejścia do stanu s' po wykonaniu akcji a w stanie s,
- $R(s, a, s')$ — funkcja nagrody otrzymana za przejście ze stanu s do s' poprzez akcję a,
- $\gamma \in [0, 1]$ — współczynnik dyskontowania określający wartość przyszłych nagród względem natychmiastowych.

MDP zakłada własność Markova, która stwierdza, że przyszłość zależy tylko od obecnego stanu i akcji, a nie od całej historii (Sutton and Barto, 2018):

$$P(s_{t+1}, r_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = P(s_{t+1}, r_{t+1}|s_t, a_t) \tag{1.58}$$

Funkcja wartości. Celem agenta jest maksymalizacja oczekiwanego skumulowanego zwrotu (return), zdefiniowanego jako zdyskontowana suma przyszłych nagród (Sutton and

Barto, 2018):

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1.59}$$

gdzie γ kontroluje, jak bardzo agent ceni przyszłe nagrody — dla $\gamma = 0$ agent jest myopic (krótkowzroczny), a dla $\gamma \to 1$ agent w pełni uwzględnia przyszłe konsekwencje.

Funkcja wartości stanu (state-value function) dla polityki π definiuje się jako oczekiwany zwrot startując ze stanu s i następując politykę π (Sutton and Barto, 2018):

$$V^\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right] \tag{1.60}$$

Funkcja wartości akcji (action-value function lub Q-function) określa oczekiwany zwrot przy rozpoczęciu w stanie s, wykonaniu akcji a i następowaniu polityki π (Sutton and Barto, 2018):

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right] \tag{1.61}$$

Równanie Bellmana. Funkcje wartości spełniają rekurencyjne równanie Bellmana, które wyraża wartość stanu poprzez natychmiastową nagrodę i zdyskontowaną wartość następnego stanu (Sutton and Barto, 2018):

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s, a) \left[R(s, a, s') + \gamma V^\pi(s') \right] \tag{1.62}$$

Dla funkcji Q równanie Bellmana przyjmuje postać (Sutton and Barto, 2018):

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a) \left[R(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s', a') \right] \tag{1.63}$$

Optymalna funkcja wartości $V^*(s) = \max_\pi V^\pi(s)$ spełnia równanie optymalności Bellmana (Sutton and Barto, 2018):

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a) \left[R(s, a, s') + \gamma V^*(s') \right] \tag{1.64}$$

1.4.1 Q-Learning

Q-Learning to algorytm uczenia ze wzmocnieniem bez modelu (model-free), który uczy się optymalnej funkcji wartości akcji $Q^*(s, a)$ bez potrzeby znajomości dynamiki środowiska $P(s'|s, a)$. Algorytm iteracyjnie aktualizuje oszacowania wartości Q na podstawie doświadczeń zebranych przez agenta w środowisku, wykorzystując zasadę temporal difference (TD) learning. Q-Learning jest algorytmem off-policy, co oznacza, że może uczyć się optymalnej polityki nawet gdy agent podąża za inną, eksploracyjną polityką (Sutton and Barto, 2018).

Reguła aktualizacji Q-Learning. Po wykonaniu akcji a_t w stanie s_t, otrzymaniu nagrody r_{t+1} i przejściu do stanu s_{t+1}, wartość $Q(s_t, a_t)$ jest aktualizowana zgodnie z regułą (Sutton and Barto, 2018):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{1.65}$$

gdzie $\alpha \in (0, 1]$ to współczynnik uczenia kontrolujący szybkość aktualizacji, a $\max_{a'} Q(s_{t+1}, a')$ reprezentuje oszacowanie wartości najlepszej akcji w następnym stanie.

Wyrażenie w nawiasie kwadratowym nazywane jest błędem TD (temporal difference error):

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \tag{1.66}$$

który mierzy różnicę między obecnym oszacowaniem a celem bootstrappowanym z następnego stanu.

Wykazano, że Q-Learning zbiega do optymalnej funkcji wartości akcji Q^* pod warunkiem, że każda para stan-akcja jest odwiedzana nieskończenie wiele razy i współczynnik uczenia spełnia odpowiednie warunki (Sutton and Barto, 2018).

1.4.2 SARSA (State-Action-Reward-State-Action)

SARSA to algorytm uczenia ze wzmocnieniem bez modelu, który w przeciwieństwie do Q-Learning jest algorytmem on-policy — uczy się wartości polityki, którą faktycznie wykonuje. Nazwa algorytmu pochodzi od sekwencji elementów używanych do aktualizacji: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. SARSA jest bardziej konserwatywny niż Q-Learning, ponieważ uwzględnia rzeczywiste akcje podejmowane przez agenta, a nie zakłada zawsze wybór optymalnej akcji (Sutton and Barto, 2018).

Reguła aktualizacji SARSA. Po wykonaniu akcji a_t w stanie s_t, otrzymaniu nagrody r_{t+1}, przejściu do stanu s_{t+1} i wybraniu następnej akcji a_{t+1} zgodnie z polityką, wartość $Q(s_t, a_t)$ jest aktualizowana według reguły (Sutton and Barto, 2018):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{1.67}$$

Kluczowa różnica w porównaniu do Q-Learning polega na tym, że zamiast $\max_{a'} Q(s_{t+1}, a')$ używane jest $Q(s_{t+1}, a_{t+1})$, gdzie a_{t+1} to akcja rzeczywiście wybrana przez agenta zgodnie z jego polityką (np. ε-greedy).

SARSA jest bardziej odpowiedni w środowiskach, gdzie eksploracja nieoptymalnych akcji jest ryzykowna, ponieważ uwzględnia konsekwencje faktycznie podejmowanych akcji eksploracyjnych (Sutton and Barto, 2018).

1.4.3 Deep Q-Network (DQN)

Deep Q-Network to przełomowe rozszerzenie Q-Learning wykorzystujące głębokie sieci neuronowe do aproksymacji funkcji wartości akcji w problemach z dużymi lub ciągłymi przestrzeniami stanów. Klasyczny Q-Learning przechowuje wartości Q w tabeli, co staje się niepraktyczne dla złożonych problemów. DQN reprezentuje funkcję $Q(s, a)$ jako sieć

neuronową z parametrami θ, oznaczaną jako $Q(s, a; \theta)$, która może generalizować wiedzę między podobnymi stanami. Algorytm DQN wprowadza kilka innowacji technicznych niezbędnych do stabilnego trenowania głębokich sieci w kontekście uczenia ze wzmocnieniem (Sutton and Barto, 2018).

Experience Replay. DQN wykorzystuje bufor doświadczeń (experience replay buffer), w którym przechowywane są krotki przejść $(s_t, a_t, r_{t+1}, s_{t+1})$. Podczas treningu losowo próbkowane są mini-batche z tego bufora, co łamie korelację między kolejnymi próbkami i poprawia stabilność uczenia (Sutton and Barto, 2018).

Target Network. DQN wykorzystuje dwie sieci neuronowe: sieć główną $Q(s, a; \theta)$ i sieć docelową $Q(s, a; \theta^-)$. Sieć główna jest aktualizowana w każdym kroku, podczas gdy sieć docelowa jest aktualizowana rzadziej (co C kroków) kopiując wagi z sieci głównej. To rozwiązanie stabilizuje cele uczenia, ponieważ cel y_t w funkcji straty jest obliczany za pomocą stałych (przez pewien czas) wag (Sutton and Barto, 2018).

Funkcja straty DQN. Parametry sieci θ są optymalizowane poprzez minimalizację funkcji straty (Sutton and Barto, 2018):

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(y - Q(s, a; \theta))^2\right] \tag{1.68}$$

gdzie cel y jest obliczany za pomocą sieci docelowej:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \tag{1.69}$$

a $\mathcal{D}$ oznacza rozkład próbkowany z bufora doświadczeń.

1.4.4 Policy Gradient Methods (Metody gradientu polityki)

Metody gradientu polityki to alternatywne podejście do uczenia ze wzmocnieniem, które zamiast uczyć się funkcji wartości, bezpośrednio optymalizują parametryzowaną politykę $\pi(a|s; \theta)$. Polityka może być reprezentowana jako sieć neuronowa, która dla danego stanu s produkuje rozkład prawdopodobieństwa nad akcjami. Metody te są szczególnie efektywne w problemach z ciągłymi lub bardzo dużymi przestrzeniami akcji, gdzie wybór maksimum z funkcji Q byłby trudny obliczeniowo (Sutton and Barto, 2018; Szepesvári, 2010).

Cel optymalizacji. Celem jest maksymalizacja oczekiwanej wartości funkcji wartości stanu początkowego (Sutton and Barto, 2018):

$$J(\theta) = \mathbb{E}_{s_0}\left[V^{\pi_\theta}(s_0)\right] \tag{1.70}$$

lub równoważnie oczekiwanej skumulowanej nagrody pod polityką π_θ.

Twierdzenie o gradiencie polityki. Gradient funkcji celu względem parametrów polityki można wyrazić jako (Sutton and Barto, 2018):

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi(a|s; \theta)\, Q^{\pi_\theta}(s, a)\right] \tag{1.71}$$

gdzie oczekiwanie jest po trajektoriach generowanych przez politykę π_θ.

W praktyce $Q^{\pi_\theta}(s, a)$ jest aproksymowane przez skumulowaną nagrodę z trajektorii (REINFORCE) lub przez nauczoną funkcję wartości (Actor-Critic).

Algorytm REINFORCE. Podstawowy algorytm gradientu polityki, REINFORCE, używa pełnych zwrotów z trajektorii jako nieobciążonych estymatorów $Q^{\pi_\theta}(s, a)$ (Sutton and Barto, 2018):

$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi(a_t|s_t; \theta) \tag{1.72}$$

gdzie G_t to rzeczywisty zwrot otrzymany po akcji a_t.

Metody Actor-Critic. Metody Actor-Critic łączą podejście gradientu polityki (actor) z uczeniem funkcji wartości (critic). Actor aktualizuje parametry polityki θ w kierunku poprawy wydajności, podczas gdy critic uczy się funkcji wartości $V^\pi(s)$ lub $Q^\pi(s, a)$ z parametrami ω, która jest używana do oceny akcji aktora. Zamiast używać pełnych zwrotów G_t, actor-critic wykorzystuje bootstrap-owane oszacowania z critic, co zmniejsza wariancję gradientu (Sutton and Barto, 2018; Szepesvári, 2010).

Aktualizacja critic (dla TD(0)) (Sutton and Barto, 2018):

$$\omega \leftarrow \omega + \beta \delta_t \nabla_\omega V(s_t; \omega) \tag{1.73}$$

gdzie błąd TD wynosi:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}; \omega) - V(s_t; \omega) \tag{1.74}$$

Aktualizacja actor (Sutton and Barto, 2018):

$$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \log \pi(a_t|s_t; \theta) \tag{1.75}$$

Zastosowania uczenia ze wzmocnieniem. Uczenie ze wzmocnieniem znalazło szerokie zastosowanie w wielu dziedzinach (Sutton and Barto, 2018; Szepesvári, 2010):

- **Robotyka i automatyka** — kontrola robotów, manipulacja obiektami, nawigacja autonomiczna.
- **Gry** — osiągnięcie nadludzkiego poziomu w grach planszowych (Go, szachy) i grach wideo (Atari, StarCraft, Dota 2).
- **Optymalizacja procesów przemysłowych** — kontrola parametrów procesów wytwórczych, zarządzanie łańcuchem dostaw, optymalizacja zużycia energii.
- **Systemy rekomendacyjne** — personalizacja rekomendacji z uwzględnieniem długoterminowego zaangażowania użytkowników.
- **Finanse** — handel algorytmiczny, zarządzanie portfelem, optymalizacja strategii inwestycyjnych.
- **Healthcare** — personalizacja terapii, optymalizacja dawkowania leków, zarządzanie zasobami szpitalnymi.
- **Zarządzanie ruchem i logistyka** — optymalizacja sygnalizacji świetlnej, routing pojazdów autonomicznych.

W kontekście procesów wytwórczych, uczenie ze wzmocnieniem pozwala na dynamiczną optymalizację parametrów produkcji w czasie rzeczywistym, adaptację do zmieniających się warunków oraz minimalizację kosztów przy jednoczesnym utrzymaniu wysokiej jakości produktów (Sutton and Barto, 2018).

1.5 Uczenie głębokie (Deep Learning)

Uczenie głębokie to poddziedzina uczenia maszynowego oparta na sztucznych sieciach neuronowych z wieloma warstwami przetwarzającymi informację w sposób hierarchiczny. W przeciwieństwie do klasycznych algorytmów uczenia maszynowego, które wymagają ręcznego inżynierowania cech (feature engineering), głębokie sieci neuronowe automatycznie uczą się reprezentacji danych na wielu poziomach abstrakcji. Niższe warstwy uczą się prostych wzorców (np. krawędzie w obrazach), podczas gdy wyższe warstwy komponują te proste wzorce w bardziej złożone reprezentacje (np. części obiektów, całe obiekty). Ta hierarchiczna nauka reprezentacji pozwala głębokim sieciom osiągać wyniki przewyższające tradycyjne metody w wielu zadaniach, szczególnie w dziedzinie wizji komputerowej, przetwarzania języka naturalnego i rozpoznawania mowy (Goodfellow et al., 2016; LeCun et al., 2015).

Architektura głębokiej sieci neuronowej. Podstawowym elementem sieci neuronowej jest perceptron wielowarstwowy (Multi-Layer Perceptron, MLP), składający się z warstw neuronów połączonych wagami. Dla wejścia $\boldsymbol{x} \in \mathbb{R}^d$, warstwa ukryta oblicza (Goodfellow et al., 2016):

$$\boldsymbol{h} = \sigma(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \tag{1.76}$$

gdzie $W^{(1)}$ to macierz wag, $\boldsymbol{b}^{(1)}$ to wektor biasów, a $\sigma(\cdot)$ to nieliniowa funkcja aktywacji.

Dla sieci wielowarstwowej transformacja danych przez kolejne warstwy wyraża się jako (Goodfellow et al., 2016):

$$\boldsymbol{h}^{(l)} = \sigma^{(l)}(W^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{1.77}$$

gdzie l oznacza numer warstwy, $\boldsymbol{h}^{(0)} = \boldsymbol{x}$ to dane wejściowe, a $\boldsymbol{h}^{(L)}$ to wyjście sieci.

Warstwa wyjściowa dla klasyfikacji wieloklasowej często używa funkcji softmax (Goodfellow et al., 2016):

$$P(y = k|\boldsymbol{x}) = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)} \tag{1.78}$$

gdzie $\boldsymbol{z} = W^{(L)}\boldsymbol{h}^{(L-1)} + \boldsymbol{b}^{(L)}$ to logity, a K to liczba klas.

Funkcje aktywacji. Nieliniowe funkcje aktywacji są kluczowe dla zdolności sieci do modelowania złożonych zależności. Najczęściej stosowane funkcje aktywacji to (Goodfellow et al., 2016; LeCun et al., 2015):

ReLU (Rectified Linear Unit): Najpopularniejsza funkcja aktywacji w głębokich sieciach (Goodfellow et al., 2016):

$$\text{ReLU}(x) = \max(0, x) \tag{1.79}$$

ReLU jest prosta obliczeniowo, łagodzi problem zanikającego gradientu i empirycznie przyspiesza trenowanie.

Leaky ReLU: Wariant ReLU pozwalający na małe ujemne wartości (Goodfellow et al., 2016):

$$\text{LeakyReLU}(x) = \max(\alpha x, x) \tag{1.80}$$

gdzie typowo $\alpha = 0.01$, co zapobiega „umieraniu"neuronów.

Sigmoid: Klasyczna funkcja aktywacji mapująca wartości do przedziału (0,1) (Goodfellow et al., 2016):

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.81}$$

Obecnie rzadziej używana w warstwach ukrytych ze względu na problem zanikającego gradientu.

Tanh: Funkcja tangensa hiperbolicznego mapująca do przedziału (-1,1) (Goodfellow et al., 2016):

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.82}$$

1.5.1 Algorytm wstecznej propagacji (Backpropagation)

Backpropagation to algorytm służący do efektywnego obliczania gradientów funkcji straty względem wszystkich wag w sieci neuronowej, umożliwiający ich optymalizację metodami gradientowymi. Algorytm wykorzystuje regułę łańcuchową do propagowania błędu od warstwy wyjściowej do warstw wejściowych, obliczając gradienty w sposób sekwencyjny i efektywny obliczeniowo (Goodfellow et al., 2016).

Funkcja straty. Dla klasyfikacji wieloklasowej typowo stosuje się entropię krzyżową (cross-entropy) (Goodfellow et al., 2016):

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log(\hat{y}_{ik}) \tag{1.83}$$

gdzie y_{ik} to prawdziwa etykieta (one-hot encoding), $\hat{y}_{ik}$ to predykcja sieci, a θ reprezentuje wszystkie parametry sieci.

Propagacja wsteczna. Dla warstwy wyjściowej gradient błędu wynosi (Goodfellow et al., 2016):

$$\delta^{(L)} = \frac{\partial L}{\partial \boldsymbol{z}^{(L)}} \tag{1.84}$$

Dla warstw ukrytych gradient propaguje się wstecznie (Goodfellow et al., 2016):

$$\delta^{(l)} = \left((W^{(l+1)})^\top \delta^{(l+1)} \right) \odot \sigma'^{(l)}(\boldsymbol{z}^{(l)}) \tag{1.85}$$

gdzie $\odot$ oznacza iloczyn Hadamarda (element-wise), a $\sigma'^{(l)}$ to pochodna funkcji aktywacji.

Gradienty względem wag i biasów (Goodfellow et al., 2016):

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)}(\boldsymbol{h}^{(l-1)})^\top, \quad \frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)} \tag{1.86}$$

1.5.2 Metody optymalizacji

Trenowanie głębokich sieci neuronowych wymaga efektywnych algorytmów optymalizacji, które są rozszerzeniami klasycznego gradientu prostego (Gradient Descent).

Stochastic Gradient Descent (SGD). SGD aktualizuje parametry na podstawie gradientu obliczonego na małym batchu danych zamiast całego zbioru treningowego (Goodfellow et al., 2016):

$$\theta \leftarrow \theta - \eta \nabla_\theta L_{\text{batch}}(\theta) \tag{1.87}$$

gdzie η to współczynnik uczenia (learning rate), a L_{batch} to średnia strata na batchu.

Momentum. Momentum przyspiesza SGD w kierunkach z konsekwentnym gradientem i tłumi oscylacje (Goodfellow et al., 2016):

$$\boldsymbol{v}_t = \beta \boldsymbol{v}_{t-1} + \eta \nabla_\theta L(\theta) \tag{1.88}$$

$$\theta \leftarrow \theta - \boldsymbol{v}_t \tag{1.89}$$

gdzie $\beta \in [0, 1)$ (typowo 0.9) kontroluje wpływ poprzednich gradientów.

Adam (Adaptive Moment Estimation). Adam to obecnie najpopularniejszy optymalizator, łączący momentum z adaptacyjnymi współczynnikami uczenia dla każdego parametru (Goodfellow et al., 2016):

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \nabla_\theta L(\theta) \tag{1.90}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)(\nabla_\theta L(\theta))^2 \tag{1.91}$$

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t}, \quad \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t} \tag{1.92}$$

$$\theta \leftarrow \theta - \eta \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \tag{1.93}$$

gdzie typowo $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$.

1.5.3 Regularyzacja w sieciach głębokich

Głębokie sieci neuronowe są podatne na przeuczenie ze względu na dużą liczbę parametrów. Stosuje się różne techniki regularyzacji aby poprawić generalizację.

Dropout. Dropout to technika regularyzacji polegająca na losowym wyłączaniu neuronów podczas trenowania z prawdopodobieństwem p (typowo 0.5). Podczas forward pass każdy neuron jest zachowywany z prawdopodobieństwem $1-p$ (Goodfellow et al., 2016):

$$\boldsymbol{h}^{(l)} = \boldsymbol{r}^{(l)} \odot \sigma(W^{(l)}\boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{1.94}$$

gdzie $\boldsymbol{r}^{(l)}$ to wektor binarny z elementami losowanymi z rozkładu Bernoulliego $\text{Bernoulli}(1-p)$.

Podczas testowania wszystkie neurony są aktywne, a ich wyjścia są skalowane przez $(1-p)$ aby zachować oczekiwaną wartość aktywacji.

Batch Normalization. Batch Normalization normalizuje aktywacje każdej warstwy, co stabilizuje i przyspiesza trenowanie. Dla mini-batcha aktywacji $\{x_1, \ldots, x_m\}$, normalizacja wynosi (Goodfellow et al., 2016):

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{1.95}$$

gdzie $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$ to średnia batcha, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ to wariancja batcha.

Następnie stosuje się przekształcenie liniowe z nauczalnymi parametrami (Goodfellow et al., 2016):

$$y_i = \gamma\hat{x}_i + \beta \tag{1.96}$$

gdzie γ i β są parametrami uczonymi przez backpropagation.

L2 Regularization (Weight Decay). Dodanie kary L2 do funkcji straty zapobiega zbyt dużym wartościom wag (Goodfellow et al., 2016):

$$L_{\text{total}} = L(\theta) + \lambda \sum_{l} \|W^{(l)}\|_F^2 \tag{1.97}$$

gdzie $\|\cdot\|_F$ to norma Frobeniusa, a λ kontroluje siłę regularyzacji.

1.5.4 Konwolucyjne sieci neuronowe (CNN)

Konwolucyjne sieci neuronowe to architektura zaprojektowana specjalnie do przetwarzania danych o strukturze kratowej, takich jak obrazy. CNN wykorzystują operację splotu (convolution) zamiast pełnego połączenia między neuronami, co drastycznie redukuje liczbę parametrów i pozwala na uczenie się lokalnych wzorców niezależnie od ich położenia w obrazie (translational invariance) (LeCun et al., 2015; Goodfellow et al., 2016).

Warstwa konwolucyjna. Operacja splotu 2D dla obrazu wejściowego X i filtra (kernela) W wynosi (Goodfellow et al., 2016):

$$Y_{ij} = (X * W)_{ij} = \sum_{m}\sum_{n} X_{i+m,j+n} W_{mn} + b \tag{1.98}$$

gdzie Y to mapa cech (feature map), a b to bias.

Dla wielokanałowego wejścia (np. RGB) z C_{in} kanałami i C_{out} filtrami (Goodfellow et al., 2016):

$$Y_{ij}^{(k)} = \sum_{c=1}^{C_{\text{in}}} (X^{(c)} * W^{(k,c)})_{ij} + b^{(k)} \tag{1.99}$$

Pooling. Warstwy poolingu redukują wymiary przestrzenne map cech, zwiększając niezmienniczość na małe translacje. Max pooling wybiera maksymalną wartość z okna (Goodfellow et al., 2016):

$$Y_{ij} = \max_{(m,n)\in\mathcal{R}_{ij}} X_{mn} \tag{1.100}$$

gdzie $\mathcal{R}_{ij}$ to region poolingu dla pozycji (i, j).

Average pooling oblicza średnią (Goodfellow et al., 2016):

$$Y_{ij} = \frac{1}{|\mathcal{R}_{ij}|} \sum_{(m,n)\in\mathcal{R}_{ij}} X_{mn} \tag{1.101}$$

Typowa architektura CNN składa się z naprzemiennych warstw konwolucyjnych (z ReLU) i poolingu, zakończonych w pełni połączonymi warstwami do klasyfikacji (LeCun et al., 2015).

1.5.5 Rekurencyjne sieci neuronowe (RNN)

Rekurencyjne sieci neuronowe to architektura zaprojektowana do przetwarzania sekwencji danych o zmiennej długości, takich jak tekst, mowa czy szeregi czasowe. RNN utrzymują ukryty stan $\boldsymbol{h}_t$, który jest aktualizowany w każdym kroku czasowym na podstawie bieżącego wejścia $\boldsymbol{x}_t$ i poprzedniego stanu $\boldsymbol{h}_{t-1}$, co pozwala sieci na „zapamiętywanie"informacji z przeszłości (Goodfellow et al., 2016).

Podstawowe RNN. Aktualizacja stanu ukrytego w podstawowym RNN (Goodfellow et al., 2016):

$$\boldsymbol{h}_t = \tanh(W_{hh}\boldsymbol{h}_{t-1} + W_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h) \tag{1.102}$$

Wyjście w kroku t (Goodfellow et al., 2016):

$$\boldsymbol{y}_t = W_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y \tag{1.103}$$

Podstawowe RNN cierpią na problem zanikającego/eksplodującego gradientu podczas uczenia długich sekwencji, co ogranicza ich zdolność do uczenia się długoterminowych zależności.

LSTM (Long Short-Term Memory). LSTM to zaawansowana architektura RNN rozwiązująca problem długoterminowych zależności poprzez wprowadzenie mechanizmu bramek (gates). Komórka LSTM składa się z stanu komórki $\boldsymbol{c}_t$ i trzech bramek: forget gate, input gate i output gate (Goodfellow et al., 2016).

Bramka zapominania (forget gate) (Goodfellow et al., 2016):

$$\boldsymbol{f}_t = \sigma(W_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \tag{1.104}$$

Bramka wejściowa (input gate) (Goodfellow et al., 2016):

$$\boldsymbol{i}_t = \sigma(W_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \tag{1.105}$$

Kandydat nowego stanu komórki (Goodfellow et al., 2016):

$$\tilde{\boldsymbol{c}}_t = \tanh(W_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c) \tag{1.106}$$

Aktualizacja stanu komórki (Goodfellow et al., 2016):

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{1.107}$$

Bramka wyjściowa (output gate) (Goodfellow et al., 2016):

$$\boldsymbol{o}_t = \sigma(W_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \tag{1.108}$$

Stan ukryty (Goodfellow et al., 2016):

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{1.109}$$

LSTM skutecznie modeluje zależności na dystansie dziesiątek lub setek kroków czasowych.

Zastosowania uczenia głębokiego. Uczenie głębokie zrewolucjonizowało wiele dziedzin i znalazło szerokie zastosowania (Goodfellow et al., 2016; LeCun et al., 2015):

- **Wizja komputerowa** — klasyfikacja obrazów, detekcja obiektów, segmentacja semantyczna, rozpoznawanie twarzy.
- **Przetwarzanie języka naturalnego** — tłumaczenie maszynowe, analiza sentymentu, chatboty, generowanie tekstu.
- **Rozpoznawanie mowy** — transkrypcja audio, synteza mowy, asystenci głosowi.
- **Systemy rekomendacyjne** — personalizacja treści, przewidywanie preferencji użytkowników.
- **Medycyna** — diagnostyka obrazowa (RTG, MRI, CT), przewidywanie chorób, odkrywanie leków.
- **Autonomiczne pojazdy** — percepcja środowiska, planowanie trasy, kontrola pojazdu.
- **Przemysł i procesy wytwórcze** — kontrola jakości wizualna (inspekcja defektów), predykcyjne utrzymanie ruchu oparte na danych sensorycznych, optymalizacja parametrów procesów w czasie rzeczywistym.
- **Finanse** — wykrywanie oszustw, prognozowanie rynków, automatyczny trading.

W kontekście procesów wytwórczych, głębokie sieci konwolucyjne (CNN) umożliwiają automatyczną inspekcję jakości produktów na liniach produkcyjnych, wykrywając defekty z dokładnością przewyższającą inspekcję ludzką. Rekurencyjne sieci neuronowe (RNN/LSTM) są wykorzystywane do analizy szeregów czasowych z czujników w celu predykcji awarii maszyn i optymalizacji harmonogramów konserwacji (LeCun et al., 2015; Goodfellow et al., 2016).

Bibliografia

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631, 2019. doi: 10.1145/3292500.3330701.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, Philadelphia, PA, 2007. Society for Industrial and Applied Mathematics.

David Birkes and Yadolah Dodge. *Alternative Methods of Regression*. Wiley, New York, 1993.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi: 10.1109/TIT.1967.1053964.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2001.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press, 1996.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Andrew Gelman and Jennifer Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, Cambridge, 2006.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016. URL http://www.deeplearningbook.org.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2nd edition, 2009.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, New York, 1st edition, 2013.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. doi: 10.1038/nature14539.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, 2008.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, MA, 2012.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 485–492, 2016. doi: 10.1145/2908812.2908918.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 3rd edition, 2007.

J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi: 10.1007/BF00116251.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 25, 2012.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Xiaojin Zhu and Andrew B. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005. doi: 10.1111/j.1467-9868.2005.00503.x.