Математичка гимназија

Матурски рад
из програмирања

Надгледано машинско учење

Ученик
Коста Чоловић, IV$_A$

Ментор
Коста Ђуришић

Београд, јун 2025.

Садржај

1 Увод

1.1 Шта је машинско учење

Машинско учење (енгл. *Machine Learning*) је област вештачке интелигенције која за циљ има конструисање статистичких алгоритама и математичких система који имају способност адаптације на нове ситуације и учења на основу искуства.

Машинско учење је пронашло примену у многим пољима, попут обраде природног језика (енгл. *Natural language processing*), рачунарског вида (енгл. *Computer vision*), препознавања говора, филтрирања електронске поште, пољопривреде и медицине.

1.2 Кратка историја машинског учења

Појам машинског учења први је увео Артур Семјуел (енгл. *Arthur Samuel*) 1959. године. Написао је програм за предвиђање вероватноће победе у игри Дама.

Психолог Доналд Хеб (*eng. Donald Hebb*) објавио је 1949. године теоретску нервну структуру која је поставила темељ модерних алгоритама машинског учења.

Током шездесетих година двадесетог века, компанија Рејтион (енгл. *Raytheon Company*) направила је Сајбертрон (енгл. *Cybertron*) - експерименталну „учећу машину", која је служила за анализу сонарних сигнала, електрокардиодијаграма и говорних образаца. Податке је памтила помоћу бушених картица и била тренирана од стране људског оператора.

Том М. Мичел (енгл. *Tom M. Mitchell*) даје формалну дефиницију алгоритма машинског учења: „Кажемо да рачунарски програм учи из искуства E у односу на неку класу задатака T и меру перформансе P ако се његова перформанса P при задацима из T повећава при искуству E ".

1.3 Типови машинског учења

1.3.1 Надгледано учење

Надгледано машинско учење (енгл. *Supervised Machine Learning*) подразумева вид учења где се рачунар обучава на датим улазним подацима (примерима) облика $\{x_i, y_i\}_{i=1}^{N}$, где је x_i D-димензиони вектор у ком свака димензија $j \in \{1, \ldots, D\}$ садржи по једну вредност, у ознаци $x^{(j)}$. На пример, ако би вектори x_i представљали особе, вредност $x^{(1)}$ би могла престављати пол, $x^{(2)}$ висину у cm, $x^{(3)}$ телесну масу у kg итд. Излазна вредност y_i може бити реалан број, члан коначног скупа класа $\{1, 2, \ldots, C\}$ или нека сложенија структура. Ови подаци се обрађују и циљ је формирање оптималне функције која би са високом тачношћу предвиђала излазне вредности (у ознаци $\hat{y}_i$) за нове, претходно невиђене улазне векторе. Ово је најчешће коришћени тип машинског учења у пракси и он ће бити фокус овог рада.

1.3.2 Ненадгледано учење

Код ненадгледаног машинског учења (енгл. *Unsupervised Machine Learning*), рачунару је дат скуп неозначених примера $\{x_i\}_{i=1}^{N}$, а циљ је креирање модела који дате векторе користи при решавању практичног проблема или их трансформише у нови облик (нпр. редукција димензионалности или детекција аномалија).

1.3.3 Полунадгледано учење

Код полунадгледаног машинског учења (енгл. *Semi-Supervised Machine Learning*), скуп улазних података се састоји од скупа означених, као и неозначених примера (број неозначених примера је углавном много већи од броја означених). Полунадгледано учење се од надгледаног разликује по томе што неозначени примери потенцијално могу довести до проналаска бољег модела.

1.3.4 Подржано учење

Подржано машинско учење (енгл. *Reinforcement Machine Learning*) је област машинског учења где рачунарски програм интерагује са динамичном средином у којој се налази. Програм у тој средини мора обавити одређени задатак (нпр. вожња аутомобила или играње одређене друштвене игре) током ког добија повратне информације тј. „награду" коју покушава да максимизује.

1.4 Математичке основе

Машинско учење се у великој мери ослања на математичке концепте како би алгоритми могли да уче из података и доносе одлуке. У овом одељку представљамо неке од кључних математичких алата који превазилазе средњошколско градиво, попут парцијалних извода, градијената и метода оптимизације, који чине основу већине алгоритама машинског учења.

1.4.1 Парцијални извод

У математичкој анализи, парцијални извод функције са више променљивих је њен извод по једној од тих променљивих (нпр. x), док се остале сматрају константним. Може се посматрати као стопа промене у x-смеру.

Дефиниција 1.1. Нека је $\mathbb{U}$ отворени подскуп скупа $\mathbb{R}^n$. *Парцијални извод* функције f у тачки $a = (a_1, .., a_n) \in \mathbb{U}$ по i-тој променљивој x_i се дефинише као

$$\frac{\partial}{\partial x_i} f(a) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_{i-1}, a_i + h, a_{i+1}, \ldots, a_n) - f(a_1, \ldots, a_i, \ldots, a_n)}{h}.$$

1.4.2 Градијент

Дефиниција 1.2. *Градијентом* функције f у тачки a називамо вектор парцијалних извода и записујемо:

$$\nabla f(a) = \left(\frac{\partial f}{\partial x_1}(a), \ldots, \frac{\partial f}{\partial x_n}(a) \right).$$

1.4.3 Параметарска процена

У контексту машинског учења, параметарска процена се односи на одређивање оптималних вредности параметара модела на основу скупа података. Циљ је пронаћи такве вредности параметара које омогућавају моделу да што боље предвиди или класификује податке.

Дефиниција 1.3. Нека је *модел* функција $f(x; \theta)$, где је x улазни податак, а θ вектор параметара које треба проценити. Параметарска процена се врши тако да се минимизује функција грешке $L(\theta)$ — тзв. *функција губитка* — која мери одступање између стварних и предвиђених вредности.
Најчешће коришћени приступи за параметарску процену су метода најмањих квадрата (енгл. *MSE*; користи се код регресионих модела) и максимална веродостојност (енгл. *MLE*; посебно значајна у пробабилистичким моделима).

1.4.4 Градијентни спуст

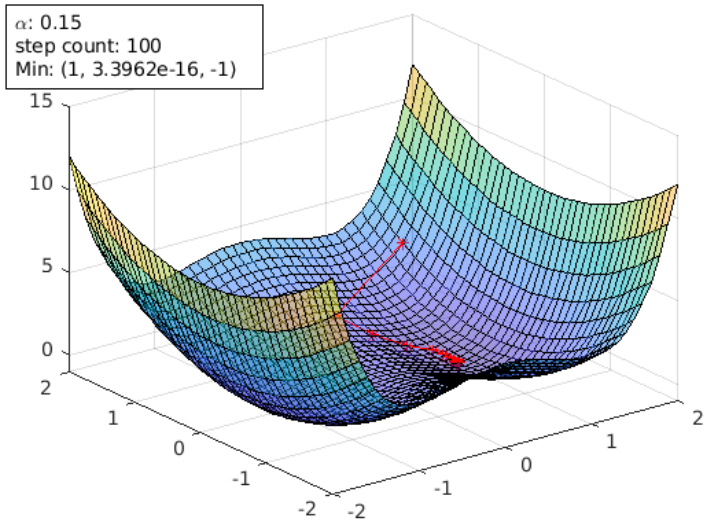
Градијентни спуст је нумеричка метода оптимизације која се користи за минимизацију функције губитка у машинском учењу. Суштина методе је итеративно померање у смеру негативног градијента функције губитка, како би се пронашао минимум те функције (слика 1.1).

Дефиниција 1.4. Нека је $L(\theta_i)$ функција губитка коју желимо да минимизујемо у односу на параметре θ_i. Ажурирање параметара се врши по правилу:

$$\theta_n = \theta_{n-1} - \eta \cdot \nabla_\theta L(\theta_{n-1}),$$

где је η коефицијент учења (енгл. *Learning Rate*), а $\nabla_\theta L$ градијент функције губитка.

У пракси постоје различите варијанте градијентног спуста, а најпознатије су *Batch* (користи цео скуп података за израчунавање градијента), *Stochastic* (користи само један податак по итерацији) и *Mini-Batch* градијентни спуст (користи мали део података по итерацији, што представља компромис између претходна два приступа).



Слика 1.1. Графички приказ алгоритма градијентог спуста.

1.5 Основни појмови

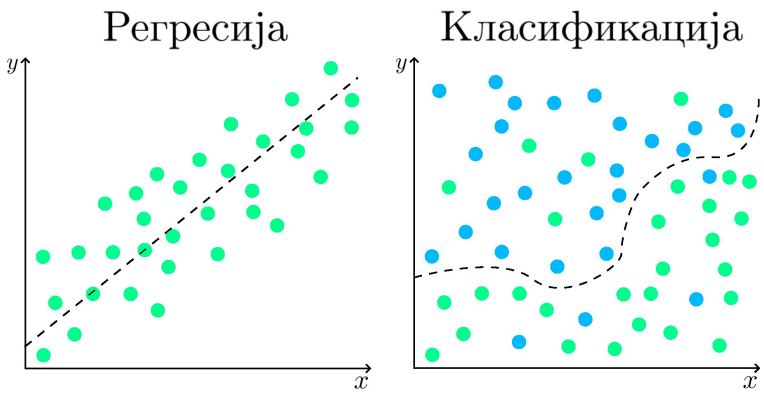
У овом одељку биће представљене кључне разлике између класификације и регресије, два основна типа надгледаног учења, као и разлика између учења заснованог на моделу и учења заснованог на инстанци. Такође, биће објашњени појмови плитког и дубоког учења, који се односе на сложеност и структуру алгоритама, посебно у контексту неуронских мрежа. Разумевање ових појмова кључно је за дубље разумевање метода и техника које ће бити обрађене у наредним поглављима.

1.5.1 Класификација и регресија

Класификација је проблем аутоматског означавања (енгл. *labelling*) неозначених података. Познат пример класификације је детекција спам порука, односно електронских писама. Класификациони алгоритми се тренирају на скупу означених података, са циљем да ће у будућности бити у могућности да припишу одговарајуће ознаке новим, неозначеним примерима. Примери такве ознаке су одређена вероватноћа (0 - 100%) и бинарна вредност (спам/није спам). Класификација може бити бинарна (1 или 0) и вишекласна (више од 2 могуће вредности).

Регресија је проблем предвиђања вредности (односно мете, енг. *target*), која је углавном реалан број, када је дат скуп улазних података. Познат пример је процена цене куће на основу њених карактеристика (површина, број соба, локација...).

Главна разлика између класификације и регресије јесте у томе што код класификације модел бира најпогоднију од понуђених ознака, док код регресије не постоји скуп понуђених вредности, већ модел сам предвиђа одговарајућу вредност (видети слику 1.2).



Слика 1.2. Графички приказ регресивног и класификационог модела.

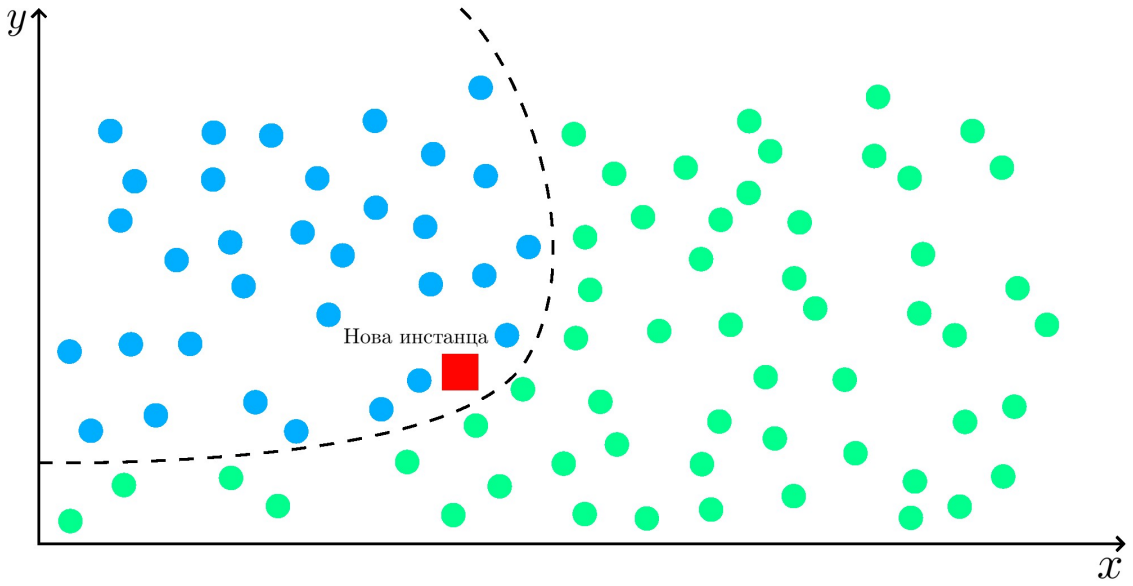
1.5.2 Учење засновано на моделу

Учење засновано на моделу (енгл. *Model-Based Learning*) подразумева креацију математичког модела који може предвидети исходе на основу улазних података (видети слику 1.3). Модел се тренира на великом скупу података, а потом се користи за предвиђања на основу нових података, у односу на које се може генерализовати. Модел се углавном креира помоћу статистичких алгоритама, попут линеарне регресије, логистичке регресије, стабла одлучивања и неуронских мрежа. О овим алгоритмима ће бити речи касније.

Предности овог типа учења су брзина (углавном бржи од учења заснованог на инстанци), тачнија предвиђања и лакша интерпретација података (лакше разумемо однос између улазних и излазних променљивих, што може помоћи при идентификацији најзначајнијих променљивих).

Са друге стране, неке од мана учења заснованог на моделу су потреба за већим скупом података и високостручним знањем.

Чест пример употребе учења заснованог на моделу у пракси је предвиђање цене некретнина на основу њених карактеристика. У овом случају, модел би могао користити линеарну регресију и био би трениран на скупу података који садржи цене различитих некретнина, а онда коришћен за предвиђање на основу нових података.



Слика 1.3. Графички приказ учења заснованог на моделу

1.5.3 Учење засновано на инстанци

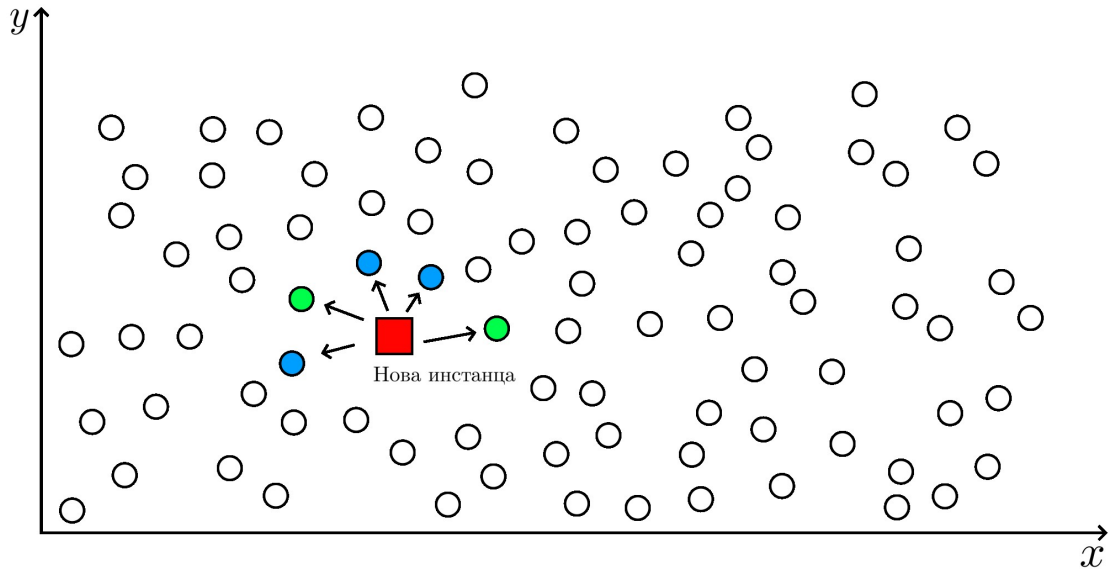
Учење засновано на инстанци (енгл. *Instance-Based Learning*) подразумева коришћење целог скупа података за предвиђања. Машина учи тако што чува све инстанце података, а онда их користи за предвиђања при новим подацима. Нови подаци се упоређују са раније виђеним инстанцама помоћу тзв. „мере сличности" (енгл. *Similarity Measure*), а најприближније поклапање се користи за предвиђање (слика 1.4).

При оваквој врсти учења се не креира модел, већ машина чува целокупан тренинг скуп података, који користи за предвиђања на основу нових података. Учење засновано на инстанци се често користи при препознавању образаца (енгл. *Pattern Recognition*), кластерској анализи (енгл. *Clustering*) и детекцији аномалија (енгл. *Anomaly Detection*).

Предности учења заснованог на инстанци укључују одсуство потребе за креацијом модела, могућност примене на мали скуп података (јер нема потребе за великим тренинг скупом) и високу флексибилност (машина чува све инстанце података и може их користити при предвиђању).

Неке од мана ове врсте учења су спорија предвиђања (углавном спорије од учења заснованог на моделу, јер машина мора да упореди нове податке са сваком инстанцом постојећих података да би извршила предвиђање), смањена тачност предвиђања (због одсуства модела) и ограничено разумевање података (није лако увидети везу између улазних и излазних података).

Познат пример примене оваквог учења је коришћење *k-NN* алгоритма (о коме ће бити речи касниије) при идентификацији цвећа.



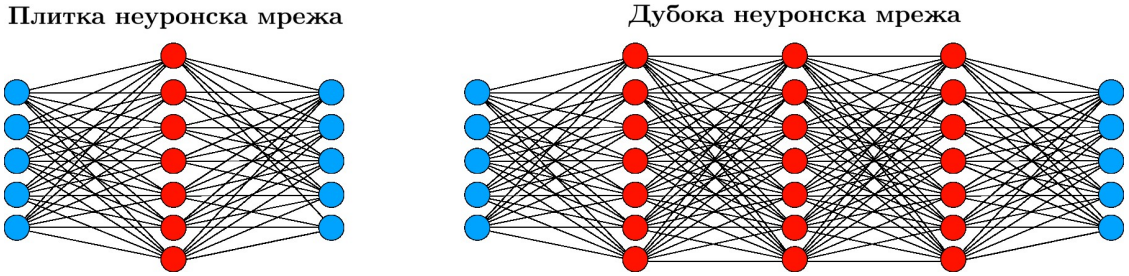
Слика 1.4. Графички приказ учења заснованог на инстанци

1.5.4 Дубоко и плитко учење

Дубоко учење (енгл. *Deep Learning*) је део машинског учења који се базира на коришћењу неуронских мрежа (видети слику 1.5) ради извршавања задатака попут регресије, класификације и *Feature Learning*-а. Модели дубоког учења састоје се од великог броја „слојева" (енгл. *Layers*), тј. „дубоки су" и углавном се користе на великим скуповима података, често без потребе за ручним подешавањем. Модел већину параметара „учи" од излаза слојева који му претходе. Многе репрезентације дубоког учења су налик интерпретацији обраде информација и шаблонима комуникације у биолошком нервном систему, одакле су неуронске мреже и добиле назив. Архитектуре ове врсте учења пронашле су примену у пољима попут рачунарског вида, обраде језика, препознавања звука и говора, филтрирања електронског садржаја, биоинформатике и фармакологије и превазишле резултате многих стручњака. Појам дубоког учења уведен је од стране Рине Дехтер (енгл. *Rina Dechter*) 1986. године.

Модели плитког учења (енгл. *Shallow Learning*) се, за разлику од дубоког, састоје од једног слоја. Већина алгоритама надгледаног машинског учења (већином они ограничене сложености), попут логистичке регресије, стабала одлучивања (енгл. *Decision Trees*) и модела потпорних вектора (енгл. *Support Vector Machine*), припадају овој групи (познати контрапример су вишеслојне неуронске мреже). Супротно дубоком учењу, параметри модела се "уче" директно из примера за тренирање и често захтевају ручно подешавање.

О дубоком и плитком учењу, као и о неуронским мрежама ће бити више речи касније.



Слика 1.5. Графички приказ модела плитке и дубоке неуронске мреже.

2 Линеарна регресија

Линеарна регресија се сматра једним од најпознатијих алгоритама у машинском учењу. Ово је статистички метод који служи за предвиђање излазних вредности y_i које линеарно зависе од улазних вредности x_i. Линеарна регресија може бити проста (предвиђање зависне променљиве на основу једне независне) или вишеструка (предвиђање зависне променљиве на основу више независних).

2.1 Проста линеарна регресија

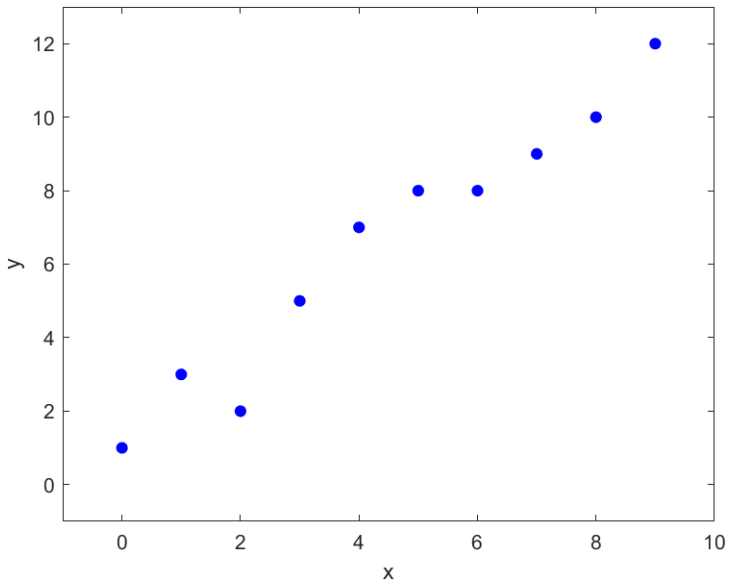
Проста линеарна регресија је тип линеарне регресије која за циљ има дескрипцију и квантификацију везе између једне независне (x) и једне зависне променљиве (y) помоћу линеарне функције. Улазни скуп садржи реалне вредности x_i, а излазни реалне вредности y_i. Посматрајмо следећи скуп уређених парова (x_i, y_i):

$$S = \{(0,1),(1,3),(2,2),(3,5),(4,7),(5,8),(6,8),(7,9),(8,10),(9,12)\}.$$

Наш циљ је проналажење модела, тј. једначине праве која ће најпрецизније предвиђати излазне вредности за сваки нови улаз (видети слику 2.1). Такву праву називамо регресивном правом. Једначина модела (регресивне праве) је

$$h(x_i) = \beta_1 x_i + \beta_0, \tag{2.1}$$

где је x_i i-ти улазни вектор, $h(x_i)$ предвиђена излазна вредност при i-тој опсервацији, а β_0 и β_1 регресивни коефицијенти који представљају пресек са y-осом и коефицијент правца регресивне праве респективно. Први корак при проналажењу модела јесте одређивање регресивних коефицијената β_0 и β_1. Ово ћемо урадити помоћу *OLS (Ordinary Least Squares)* принципа и statsmodels библиотеке у програмском језику *Python*.



Слика 2.1. Дијаграм расејања података из скупа S

Грешком остатка (ϵ_i) називамо разлику тачне и предвиђене излазне вредности у i-тој опсервацији. Из формуле 2.1 добијамо:

$$y_i = \beta_1 x_i + \beta_0 + \epsilon_i = h(x_i) + \epsilon_i \implies \epsilon_i = y_i - h(x_i).$$

Функцију y_i називамо моделом линеарне регресије.

Ради повећања прецизности модела, циљ нам је минимизација вредности грешке. Један од начина да се то постигне јесте проналазак вредности β_0 и β_1 за које је вредност следећег израза (*Squared Error* или *Cost Function*) минимална:

$$J(\beta_0, \beta_1) = \frac{1}{2n}\sum_{i=1}^{n}\epsilon_i^2 = \frac{1}{2n}\sum_{i=1}^{n}(y_i - h(x_i))^2 = \frac{1}{2n}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i))^2$$

Нађимо вредности за које је парцијални извод функције J по β_0 једнак нули:

$$\frac{\partial J}{\partial \beta_0} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - (\beta_0 + \beta_1 x_i)) = 0$$

$$\Leftrightarrow \; n\beta_0 + \beta_1\sum_{i=1}^{n}x_i = \sum_{i=1}^{n}y_i \Leftrightarrow \; \beta_0 = \frac{1}{n}\sum_{i=1}^{n}y_i - \frac{\beta_1}{n}\sum_{i=1}^{n}x_i$$

$$\Leftrightarrow \beta_0 = \overline{y} - \beta_1\overline{x}, \tag{2.2}$$

где су $\overline{x}$ и $\overline{y}$ средње вредности за x и y респективно.

Сада нађимо нулу парцијалног извода функције J по β_1:

$$\frac{\partial J}{\partial \beta_1} = -\frac{1}{n}\sum_{i=1}^{n} x_i(y_i - (\beta_0 + \beta_1 x_i)) = 0$$

$$\Leftrightarrow \beta_1 \sum_{i=1}^{n} x_i^2 + \beta_0 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

Из 2.2, заменом $\beta_0 = \overline{y} - \beta_1 \overline{x}$ у претходној једначини добијамо:

$$\begin{aligned}
&\beta_1 \sum_{i=1}^{n} x_i^2 - \beta_1 \overline{x} \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i - \overline{y} \sum_{i=1}^{n} x_i \\
\Leftrightarrow\ &\beta_1 \left(\sum_{i=1}^{n} x_i^2 - \overline{x} \sum_{i=1}^{n} x_i \right) = \sum_{i=1}^{n} x_i y_i - \overline{y} \sum_{i=1}^{n} x_i \\
\Leftrightarrow\ &\beta_1 \left(\sum_{i=1}^{n} x_i^2 - n\overline{x} \cdot \frac{1}{n} \sum_{i=1}^{n} x_i \right) = \sum_{i=1}^{n} x_i y_i - n\overline{y} \cdot \frac{1}{n} \sum_{i=1}^{n} x_i \\
\Leftrightarrow\ &\beta_1 \left(\sum_{i=1}^{n} x_i^2 - n\overline{x}^2 \right) = \sum_{i=1}^{n} x_i y_i - n\overline{x}\overline{y} \\
\Leftrightarrow\ &\beta_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\overline{x}\overline{y}}{\sum_{i=1}^{n} x_i^2 - n\overline{x}^2} \\
&= \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} x_i^2 - n\overline{x}^2} \\
&= \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} x_i^2 - 2n\overline{x}^2 + n\overline{x}^2} \\
&= \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} x_i^2 - 2\overline{x}\sum_{i=1}^{n} x_i + n\overline{x}^2} \\
&= \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i^2 - 2x_i\overline{x} + \overline{x}^2)} \\
&= \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x})^2} \\
&= \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x})^2}.
\end{aligned}$$

Тиме смо добили изразе за β_1 и β_0:

$$\boxed{\beta_1 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x})^2}} \qquad \boxed{\beta_0 = \overline{y} - \beta_1 \overline{x}} \tag{2.3}$$

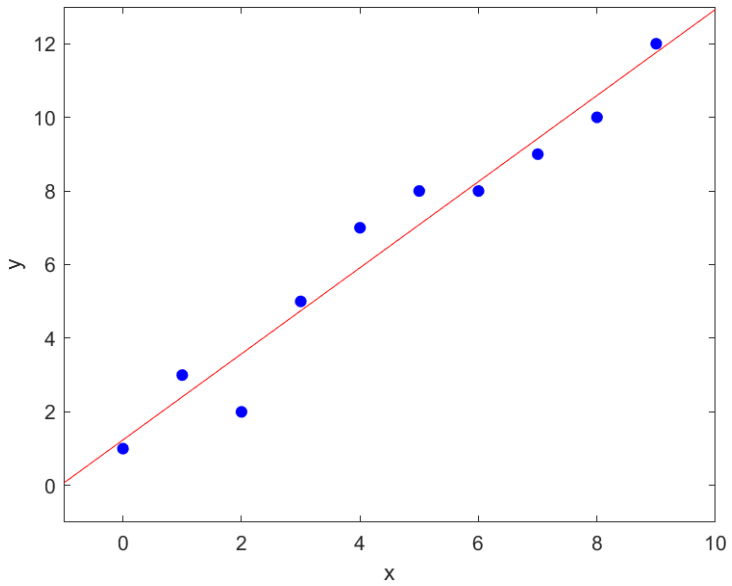
Ове вредности рачунар може израчунати за нас. Наводимо пример имплементације програма у програмском језику *Python* који рачуна изразе 2.3:

```
1   import numpy as np
2
3   def estimate_coef(x, y):
4       n = np.size(x)
5       m_x = np.mean(x)
6       m_y = np.mean(y)
7       b_1 = (np.sum(y * x) - n * m_y * m_x) / (np.sum(x * x) - n * m_x * m_x)
8       b_0 = m_y - b_1 * m_x
9       return (b_0, b_1)
10
11  x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
12  y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
13  b = estimate_coef(x, y)
14  print("Procenjeni koeficijenti:\nb_0 = {} \nb_1 = {}".format(b[0], b[1]))
```

Излаз програма нам даје приближне оптималне вредности параметара:

$\beta_0 = 1.2363636363636363$
$\beta_1 = 1.1696969696969697$

Можемо скицирати график функције $h(x) = \beta_1 x + \beta_0$ (слика 2.2).



Слика 2.2. Графички приказ модела линеарне регресије над скупом података S

2.2 Вишеструка линеарна регресија

Вишеструка линеарна регресија (енгл. *Multiple Linear Regression*) није ништа друго до екстензија просте линеарне регресије. Разматрамо n улазних вредности у форми p-димензионих вектора $\begin{bmatrix} x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix}$.

Излазне вредности представљамо у облику вектора y, где вредност y_i, $i \in \{1, 2, \dots, n\}$, одговара i - том улазном вектору.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Регресивна права за p улазних вредности је представљена једначином

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip},$$

где су β_i регресивни коефицијенти. Даље добијамо:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i = h(x_i) + \epsilon_i \implies \epsilon_i = y_i - h(x_i).$$

Краће се може записати

$$y = X\beta + \epsilon,$$

где су $X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix}$, $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$, $\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$.

Аналогно простој регресији, *squared error (cost)* функција је облика

$$S(\beta) = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \;\; y_i = X_i\beta.$$

Односно у матричном облику:

$$S(\beta) = (y - X\beta)^T (y - X\beta).$$

Као и раније, минимизујемо ову функцију тако што тражимо нулу њеног извода:

$$S(\beta) = (y - X\beta)^T (y - X\beta) = y^T y - 2\beta^T X^T y + \beta^T X^T X\beta$$

$$\frac{\partial S}{\partial \beta} = -2X^T y + 2X^T X\beta = 0 \implies X^T X\beta = X^T y$$

Добијамо решење:

$$\boxed{\hat{\beta} = (X^T X)^{-1} X^T y} \tag{2.4}$$

Након израчунавања вредности $\hat{\beta}$, предвиђене излазне вредности добијамо користећи формулу

$$\hat{y} = X\hat{\beta}.$$

Имплементирамо модел у *Python*-у, користећи скуп података о ценама кућа у Бостону са веб-странице *http://lib.stat.cmu.edu/datasets/boston*:

```
1   import numpy as np
2   import pandas as pd
3   from sklearn.model_selection import train_test_split
4
5   #Učitavanje podataka:
6   data_url = "http://lib.stat.cmu.edu/datasets/boston"
7   raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
8
9   #Formiranje matrica:
10  X = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
11  y = raw_df.values[1::2, 2]
12
13  # Delimo podatke na skup za testiranje i skup za treniranje (40% za testiranje, 60% za treniranje):
14  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
15
16  def estimate_coef(X, y):
17      beta = np.linalg.inv(X.T @ X) @ X.T @ y
18      return beta
19
20  coefficients = estimate_coef(X_train, y_train)
```

Податке са веб-странице смо поделили на два дела: део за тренирање и део за тестирање, у односу 3:2. Помоћу улазних и излазних података које учитавамо са веб-странице формирамо матрице X и y. Матрицу β рачунамо користећи функцију $estimate\_coef(X, y)$ и улазне податке за тренирање, према 2.4.

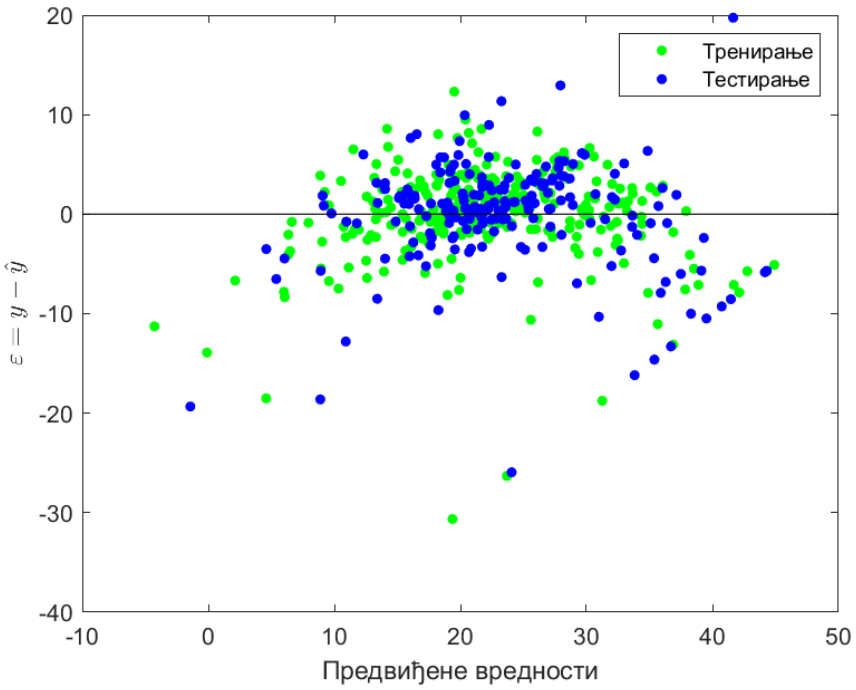
Можемо увести појам коефицијента детерминације (енгл. *Variance Score*), у ознаци R^2, који рачунамо по формули

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y_i})^2}.$$

Он показује који део варијансе (дисперзије) зависне променљиве модел објашњава. У нашем случају $R^2 \approx 0.691$, што значи да наш модел објашњава око 69% дисперзије зависне променљиве на основу независних.
Програм је израчунао следеће приближне вредности коефицијената:

$$\beta = \begin{bmatrix} -0.0577075382 \\ 0.0668993014 \\ 0.0164344779 \\ 0.0231982477 \\ -3.86764806 \\ 5.72587584 \\ -0.00558060867 \\ -0.968248259 \\ 0.167973590 \\ -0.0105074574 \\ -0.301011561 \\ 0.0148466879 \\ -0.411698518 \end{bmatrix}.$$

Можемо скицирати график зависности грешке $\epsilon_i = y_i - \hat{y}_i$ од предвиђене вредности $\hat{y}_i$ (видети слику 2.3).

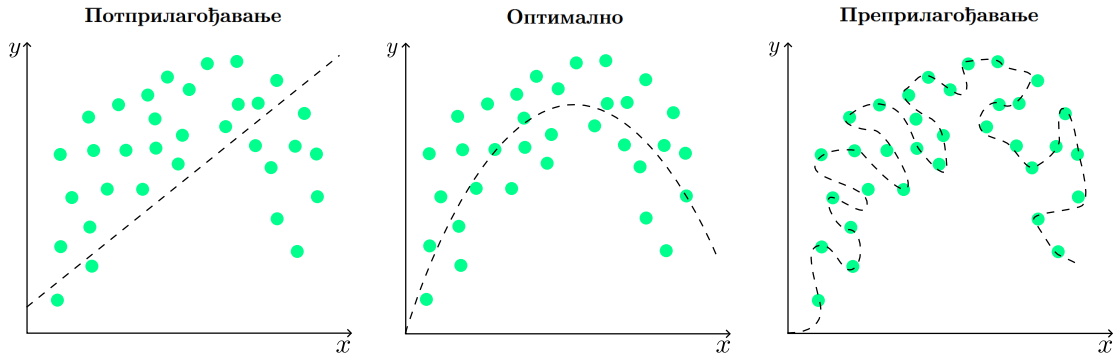


Слика 2.3. Графички приказ грешке модела вишеструке линеарне регресије

3 Полиномна регресија

Полиномна регресија (енгл. *Polynomial Regression*) је облик регресије где је однос између независне променљиве x и зависне променљиве y моделован у форми полинома n-тог степена. Ово омогућава постојање нелинеарне зависности између x и условне очекиване вредности променљиве y (у ознаци $E(y|x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n$). Налик линеарној регресији, циљ је минимизација грешке, али код полиномне регресије то радимо користећи полиномну функцију. Коришћење чланова вишег степена повећава флексибилност модела, који самим тим боље истиче обрасце у скупу података.

Одабир степена полинома подразумева компромис између пристрасности (тенденције модела да константно предвиђа исту вредност, без обзира на праву вредност зависне променљиве) и варијансе (тенденције модела да предвиђа различите ствари за исту улазну вредност, у зависности од конкретних коришћених тренинг података). Полином вишег степена може умањити пристрасност, али такође може повећати варијансу и довести до тзв. "Преприлагођавања" модела (енгл. *Overfitting*; модел даје одличне резултате на тренинг скупу, док на тест скупу прави значајно веће грешке; видети слику 3.1). Полином нижег степена, са друге стране, може смањити варијансу и повећати пристрасност модела. Постоји мноштво метода за одабир степена полинома (*Cross-Validation*), критеријуми попут AIC и BIC...).



Слика 3.1. Потприлагођавање, оптимални модел и преприлагођавање

Слично вишеструкој линеарној регресији, улазни подаци су облика матрице

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & x_2^3 & \dots & x_2^d \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \dots & x_n^d \end{bmatrix}.$$

Излаз је у облику полиномне функције $\hat{y}$, степена d, са параметрима $\theta_0, \theta_1, \dots, \theta_d$:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d,$$

Грешка коју настојимо да минимизујемо је познатог облика:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

Вектор θ добијамо решавањем следеће једначине:

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} = (X^T X)^{-1} X^T y.$$

Имплементирајмо *Python* код на десеточланом скупу података о радним позицијама и одговарајућим годишњим платама у доларима (видети табелу 3.1):

Позиција	Ниво	Годишња плата у $
Business Analyst	1	45 000
Junior Consultant	2	50 000
Senior Consultant	3	60 000
Manager	4	60 000
Country Manager	5	110 000
Region Manager	6	150 000
Partner	7	200 000
Senior Partner	8	300 000
C-level	9	500 000
CEO	10	1 000 000

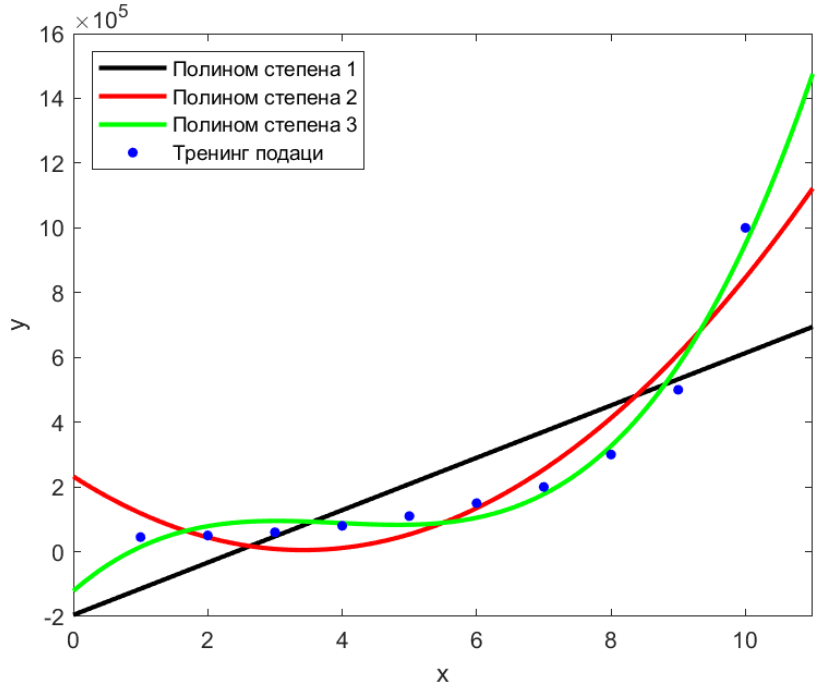
Табела 3.1: Улазни подаци за наш модел полиномне регресије.

```python
import numpy as np

nivoi = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
plate = np.array([45000, 50000, 60000, 80000, 110000,
                  150000, 200000, 300000, 500000, 1000000])

def kreiraj_polinomni_ulaz(x, degree):
    n = len(x)
    X_poly = np.ones((n, degree + 1))
    for d in range(1, degree + 1):
        X_poly[:, d] = x ** d
    return X_poly

def izracunaj_koeficijente(X, y):
    return np.linalg.inv(X.T @ X) @ X.T @ y

def predvidi(X, koeficijenti):
    return X @ koeficijenti

stepeni = range(1,101)
for stepen in stepeni:
    X_poly = kreiraj_polinomni_ulaz(nivoi, stepen)
    koeficijenti = izracunaj_koeficijente(X_poly, plate)
    predvidjene_plate = predvidi(X_poly, koeficijenti)

    R2 = np.corrcoef(plate, predvidjene_plate)[0, 1]**2
    print("Stepen", stepen)
    #print(koeficijenti)
    print(R2)
    print("\n")
```

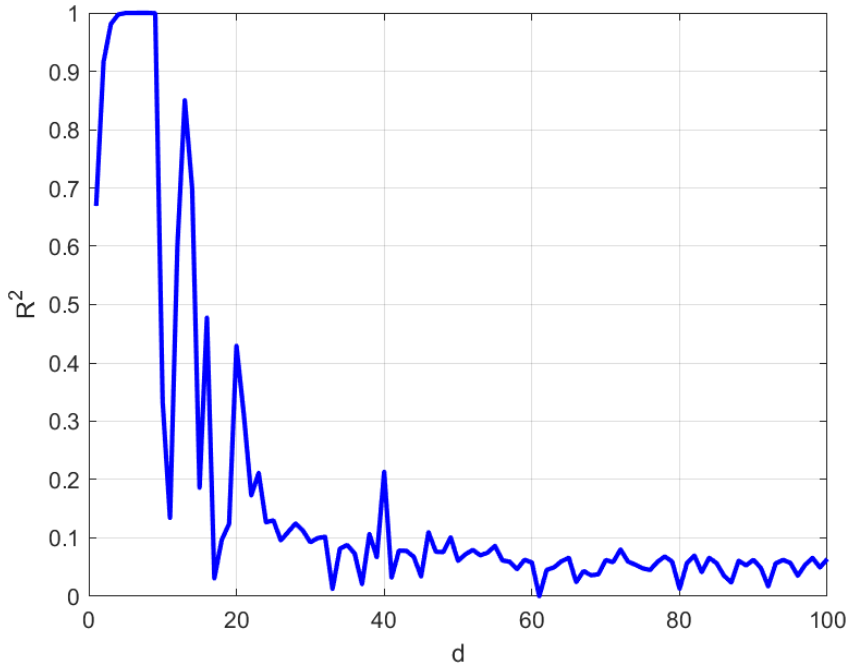
Програм даје следећи излаз за полиноме степена 1, 2 и 3 (слика 3.2) и одговарајуће коефицијенте детерминације редом:

$$f_1(x) = 80878.78787879x - 195333.33333333, \qquad R_1^2 \approx 0.66904$$
$$f_2(x) = 19431.81818182x^2 - 132871.21212121x + 232166.66666666, \qquad R_2^2 \approx 0.91621$$
$$f_3(x) = 4120.0466x^3 - 48548.9510x^2 + 180664.3357x - 121333.333, \qquad R_3^2 \approx 0.98121$$

Вредности коефицијената за полиноме степена већег од 3 су представљене на слици 3.3. Ови резултати показују како повећање степена полинома у почетку значајно побољшава модел, али након одређене тачке долази до деградације перформанси услед преприлагођавања и нумеричких проблема, што потврђује важност пажљивог одабира степена модела.



Слика 3.2. Полиномна регресија степена 1, 2 и 3



Слика 3.3. График зависности коефицијента детерминације од степена полинома.

4 Логистичка регресија

За почетак напоменимо да логистичка регресија није врста регресије, већ класификациони алгоритам. Назив потиче из математичке статистике (математичка формулација логистичке регресије слична је формулацији линеарне регресије). Логистичку регресију можемо посматрати као екстензију линеарне регресије која се користи за класификационе проблеме. Објаснићемо случај бинарне класификације, која се природно може проширити до вишекласне класификације.

Користићемо сигмоидну функцију (σ), облика слова S, чији је домен скуп реалних бројева, а кодомен интервал $(0, 1)$. На пример, ако имамо две класе, класу 0 и класу 1 и ако је $\sigma(x) > 0.5$ (гранична вредност) за неки улаз x, тада кажемо да x припада класи 1 (у супротном припада класи 0).

Улазни и излазни подаци су представљени у следећој форми:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix}, \quad Y = \begin{cases} 0, & \text{за } K_0 \\ 1, & \text{за } K_1 \end{cases}$$

Применимо мултилинеарну функцију на улазне вредности X:

$$z = \sum_{i=1}^{n} \omega_i x_i + b = w \cdot X + b$$

Вредност x_i представља i-ту опсервацију из X, $\omega = \begin{bmatrix} \omega_1 & \omega_2 & \dots & \omega_m \end{bmatrix}^T$ је коефицијент тежина, а b је терм пристрасности, односно пресек.

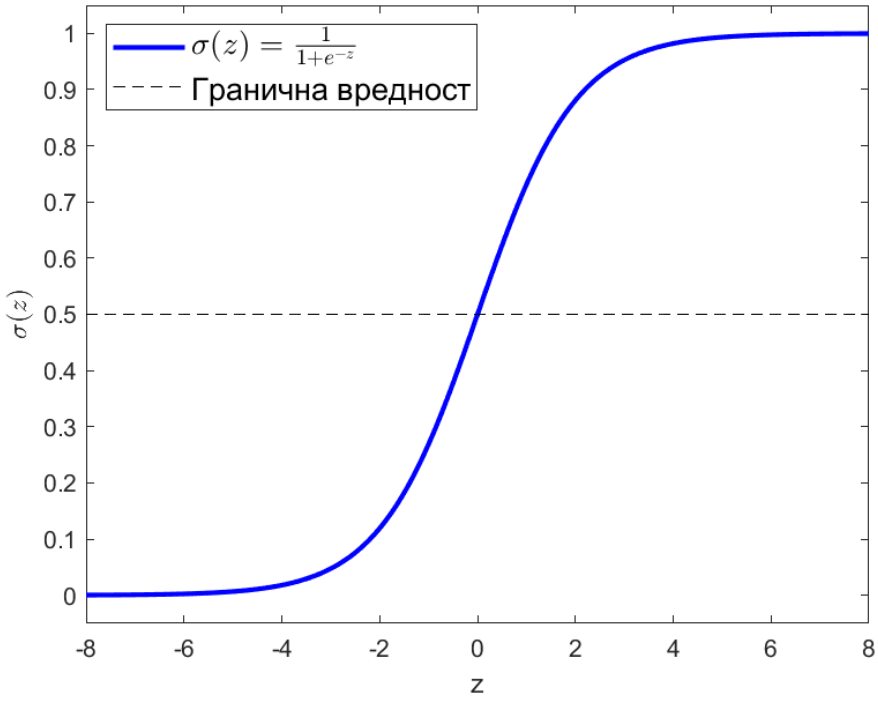
Сигмоидна функција је дефинисана на следећи начин (видети слику 4.1):

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \left(\sigma(z) \in (0, 1), \lim_{z \to +\infty} \sigma(z) = 1, \lim_{z \to -\infty} \sigma(z) = 0 \right).$$

Улаз је реалан број z, а излаз функције одговара вероватноћи између 0 и 1 (предвиђено z). Дакле, можемо рећи да сигмоидна функција „конвертује" непрекидне улазне податке у вероватноћу.

Вероватноће припадања класама се рачунају на следећи начин:

$$P(y=1) = \sigma(z) \text{ (класа 1) и } P(y=0) = 1 - \sigma(z) \text{ (класа 0)}$$



Слика 4.1. Графички приказ сигмоидне функције.

Изведимо једначину логистичке регресије. За почетак дефинишемо појам шансе као однос повољних и неповољних исхода (ово се разликује од вероватноће која представља однос повољних и свих могућих исхода):

$$\frac{p(x)}{1-p(x)} = e^z$$

Трансформишемо израз:

$$\begin{aligned}
&z = \omega \cdot X + b \\
&\implies \frac{p(x)}{1-p(x)} = e^{\omega \cdot X + b} \\
&\Leftrightarrow p(x) = e^{\omega \cdot X + b} \cdot (1 - p(x)) = e^{\omega \cdot X + b} - e^{\omega \cdot X + b} \cdot p(x) \\
&\Leftrightarrow p(x)(1 + e^{\omega \cdot X + b}) = e^{\omega \cdot X + b} \\
&\Leftrightarrow p(x) = \frac{e^{\omega \cdot X + b}}{1 + e^{\omega \cdot X + b}} = \frac{1}{1 + e^{-(\omega \cdot X + b)}}
\end{aligned}$$

Коначно, добијамо финални израз једначине логистичке регресије:

$$\boxed{p(X; b, \omega) = \frac{1}{1 + e^{-(\omega \cdot X + b)}}} \tag{4.1}$$

Предвиђене вероватноће ће бити:

$$p(X; b, \omega) = p(x) \quad \text{за} \quad y = 1$$

$$1 - p(X; b, \omega) = 1 - p(x) \quad \text{за} \quad y = 0$$

Уводимо функцију веродостојности (енгл. *Likelihood function*):

$$L(b, \omega) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1 - y_i}$$

Примењујемо природни логаритам са обе стране:

$$\begin{aligned}
\ln\left(L(b, \omega)\right) &= \sum_{i=1}^{n} \left[y_i \ln\Big(p(x_i)\Big) + (1 - y_i) \ln\Big(1 - p(x_i)\Big) \right] \\
&= \sum_{i=1}^{n} \left[y_i \ln\Big(p(x_i)\Big) + \ln\Big(1 - p(x_i)\Big) - y_i \ln\Big(1 - p(x_i)\Big) \right] \\
&= \sum_{i=1}^{n} \left[\ln\Big(1 - p(x_i)\Big) \right] + \sum_{i=1}^{n} \left[y_i \ln\Big(\frac{p(x_i)}{1 - p(x_i)}\Big) \right] \\
&= \sum_{i=1}^{n} \left[-\ln\Big(1 + e^{\omega \cdot x_i + b}\Big) \right] + \sum_{i=1}^{n} \left[y_i (\omega \cdot x_i + b) \right]
\end{aligned}$$

Израчунајмо парцијални извод функције $\ln(L(b, \omega))$ по свакој тежини ω_j.

$$\begin{aligned}
\frac{\partial \ln(L(b, \omega))}{\partial \omega_j} &= \sum_{i=1}^{n} \frac{\partial}{\partial \omega_j} \left[y_i (\omega x_i + b) - \ln\Big(1 + e^{\omega x_i + b}\Big) \right] \\
&= \sum_{i=1}^{n} \left[x_{ij} \Big(y_i - p(x_i; b, \omega) \Big) \right]
\end{aligned}$$

Помоћу ових парцијалних извода даље тривијално добијамо градијент функције. Ово радимо да бисмо, слично као раније, пронашли максимум функције.

4.1 Бинарна логистичка регресија

Имплементирајмо бинарну логистичку регресију у *Python*-у. Распоређујемо податке у две категорије, $y = 0$ и $y = 1$, користећи уграђену библиотеку *sklearn*. Користимо 30-димензиони скуп података *breast_ cancer*.

Скуп података учитавамо и делимо на део за тестирање и део за тренирање. Затим користимо уграђену функцију *LogisticRegression*, која ради на принципу који смо објаснили. Након тога користимо тест продатке за израчунавање тачности имплементираног модела.

```python
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = load_breast_cancer(return_X_y=True)

# Delimo podatke na skup za treniranje i skup za testiranje
X_train, X_test,y_train, y_test = train_test_split(X, y,test_size=0.20,random_state=23)

# Regresija
clf = LogisticRegression(random_state=0)
clf.fit(X_train, y_train)

# Predviđanje
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Tačnost modela logističke regresije (u %):", acc*100)
```

Тачност модела: 96.49122807017544%

4.2 Вишекласна (мултиномна) логистичка регресија

Код мултиномне логистичке регресије (енгл. *Multinomial Logistic Regression*), као што сам назив каже, могу постојати 3 или више класа које немају никакав квантитативни значај (нпр. „болест А", „болест Б", „болест Ц"). У овом случају користимо тзв. *Softmax* функцију. Софтмакс функција за K класа је представљена формулом:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}.$$

У овој формули, K представља број елемената вектора z, док су i и j итератори који пролазе кроз цео вектор.
Тада ће вероватноћа за класу c бити:

$$P(Y = c|\overrightarrow{X} = x) = \frac{e^{\omega_c \cdot x + b_c}}{\sum_{k=1}^{K} e^{\omega_k \cdot x + b_k}}$$

Имплементирајмо мултиномну регресију у *Python*-у. Слично бинарној логистичкој регресији, користимо уграђену библиотеку *sklearn*, која садржи 64-димензионе податке сврстане у 10 класа. Поново делимо податке на скупове за тестирање и тренирање и користимо *LogisticRegression* функцију. Затим вршимо предвиђање на тест скупу и рачунамо тачност модела.

```
from sklearn.model_selection import train_test_split
from sklearn import datasets, linear_model, metrics

digits = datasets.load_digits()
X = digits.data
y = digits.target

# Delimo podatke na skup za treniranje i skup za testiranje
X_train, X_test,y_train, y_test = train_test_split(X, y,test_size=0.4,random_state=1)

# Pravimo objekat logističke regresije
reg = linear_model.LogisticRegression()
reg.fit(X_train, y_train)

# Vršimo predviđanje na test skupu
y_pred = reg.predict(X_test)

print("Tačnost modela (u %):",metrics.accuracy_score(y_test, y_pred)*100)
```

Тачност модела: 96.52294853963839%

4.3 Евалуација модела

Након што је модел трениран, неопходно је извршити његову евалуацију како би се проценила његова способност да генерализује на нове, невиђене податке. У контексту класификације, а посебно код бинарне логистичке регресије, правилна евалуација је кључна за доношење закључака о поузданости модела.

Сама тачност (енгл. *Accuracy*) често није довољна да у потпуности проценимо успешност класификационог модела, нарочито када постоји неуравнотеженост класа (нпр. много више примера класе 0 него класе 1). У тим случајевима, додатне метрике попут одзива, прецизности и F_1-скора пружају бољи увид у перформансе.

Прецизност (енгл. *Precision*) је проценат тачно класификованих позитивних примера међу онима које је модел означио као позитивне.

Одзив (енгл. *Recall*) је проценат тачно класификованих позитивних примера међу свим стварно позитивним примерима.

F_1-скор је хармонијска средина прецизности и одзива; користи се када је потребан баланс између ова два.

Такође можемо приказати матрицу конфузије (енгл. *Confusion Matrix*) која нам показује број тачних и погрешних класификација по класама (слика 4.2).

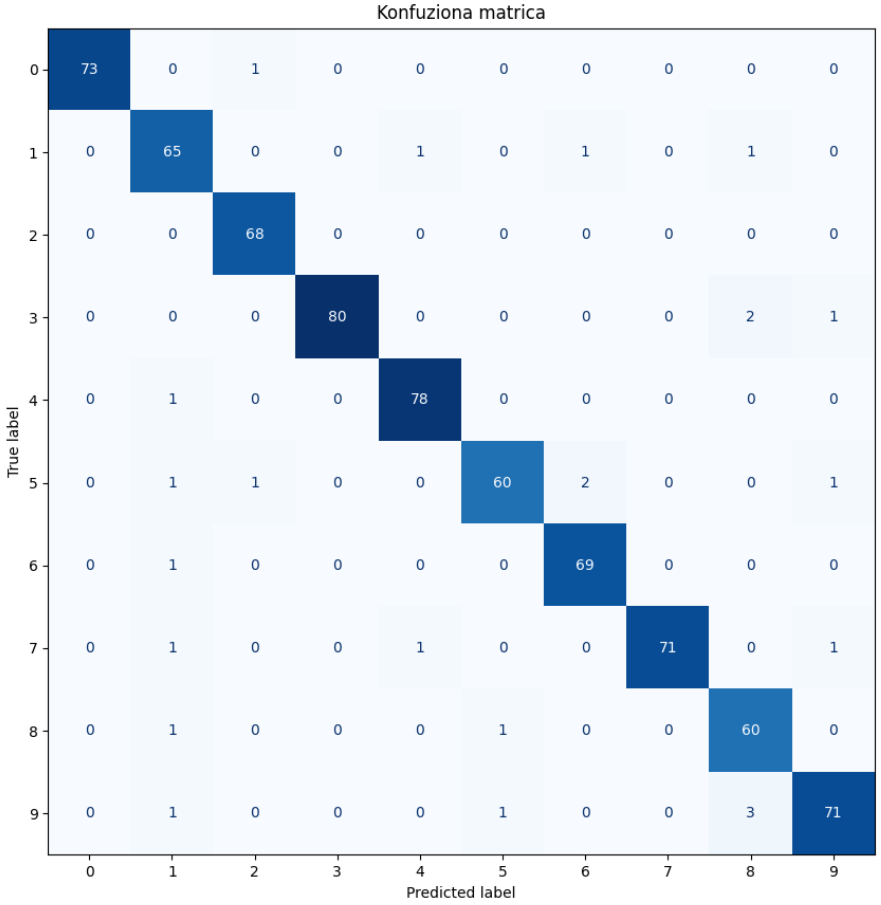
Овде су приказане следеће вредности: тачно позитивни (TP), тачно негативни (TN), лажно позитивни (FP) и лажно негативни (FN).

На основу ових вредности можемо израчунати:

$$\text{Прецизност} = \frac{TP}{TP + FP}$$

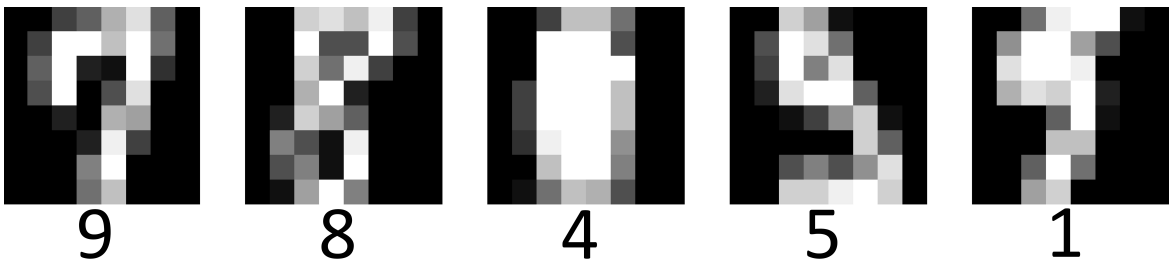
$$\text{Одзив} = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2 \cdot \text{Прецизност} \cdot \text{Одзив}}{\text{Прецизност} + \text{Одзив}}$$



Слика 4.2. Матрица конфузије за бинарну логистичку регресију.

Неки од примера за које је алгоритам погрешно предвидео вредности су представљени на слици 4.3.



Слика 4.3. Неки од примера са погрешно предвиђеним вредностима.

5 Стабла одлучивања

Стабла одлучивања (енгл. *Decision Trees*) користе се за откривање образаца информација у скупу података. Често се користе у класификацији и регресији, а широку примену имају у медицинској дијагностици, препознавању говора, финансијским предикцијама и анализи ризика.

Након што је конструисано, стабло се може искористити за предвиђање излаза нових података на основу скупа на ком је тренирано.

Свако стабло се може посматрати као структура података за складиштење искустава. На пример, када играмо неку игру први пут, не знамо која стратегија је најефикаснија, па доносимо одлуке на основу претходног искуства са другим играма. Сваког следећег пута када играмо исту игру, увиђамо све више образаца, на основу којих подешавамо тј. прилагођавамо стратегију, која постаје све ефикаснија.

Приликом конструисања стабла одлучивања, алгоритам мора да одлучи који атрибут да користи као основу за гранање у сваком чвору. Идеја је да се подаци у сваком кораку поделе тако да гране постану што „чистије“, односно да у свакој грани преовладава једна класа.

За то се користе различите метрике за мерење нечистоће скупа података, од којих су најпознатије ентропија (код ID3 алгоритма), информациони добитак (енгл. Information Gain) и *Gini* индекс (код CART алгоритма).

Ентропија (енгл. *Entropy*) представља меру несигурности, односно неуређености скупа података. У контексту стабала одлучивања, користи се за квантификацију хомогености скупа — мања ентропија значи да је скуп података „чистији“, тј. да већина примера припада истој класи. Математички се дефинише на следећи начин:

$$E(S) = -\sum_{i=1}^{n} p_i \log_2(p_i),$$

где је p_i вероватноћа да пример из скупа S припада класи i. Ако су сви приме-

ри из исте класе, ентропија је једнака нули (нема несигурности). У супротном - ако су примери равномерно распоређени по класама, вредност ентропије достиже максимум.

Информациони добитак (енгл. *Information Gain*) мери колико се ентропија смањује када се скуп података подели на основу одређеног атрибута. Другим речима, он квантификује колико информација добијамо одабиром одређеног атрибута за гранање. Рачуна се као разлика између ентропије целог скупа и пондерисане суме ентропија подскупова након поделе:

$$IG(S, A) = E(S) - \sum_{v \in \mathbb{V}} \frac{|S_v|}{|S|} E(S_v),$$

где је $\mathbb{V}$ скуп свих могућих вредности атрибута A, а S_v подскуп података у којем атрибут A има вредност v. Атрибут са највећим информационим добитком се бира за следеће гранање у стаблу.

Gini индекс је алтернативна мера „нечистоће“ скупа података, која мери вероватноћу да би случајно одабрани пример био погрешно класификован ако би класификација била извршена насумично, у складу са расподелом класа у том скупу. Дефинише се као:

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2,$$

где је p_i вероватноћа да пример припада класи i. Нижа вредност *Gini* индекса указује на већу хомогеност скупа. За разлику од ентропије, *Gini* индекс нешто брже опада и често води до сличних, али не увек идентичних стабала.

У следећем примеру користимо податке из табеле 5.1, како бисмо одредили место рођења одређене особе.

Име	Пол	Место рођења
Петар	мушко	Београд
Милица	женско	Нови Сад
Јована	женско	Нови Сад
Софија	женско	Нови Сад
Марко	мушко	Београд

Табела 5.1: Табеларни приказ скупа података који користимо.

Можемо искористити грамзиви алгоритам да бисмо на основу података из табеле креирали стабло. Проналазимо све јединствене парове атрибут-вредност и пребројавамо појављивања сваког од њих (табела 5.2).

Атрибут и вредност	број
име = Петар	1
име = Милица	1
име = Јована	1
име = Софија	1
име = Марко	1
пол = мушко	2
пол = женско	3

Табела 5.2

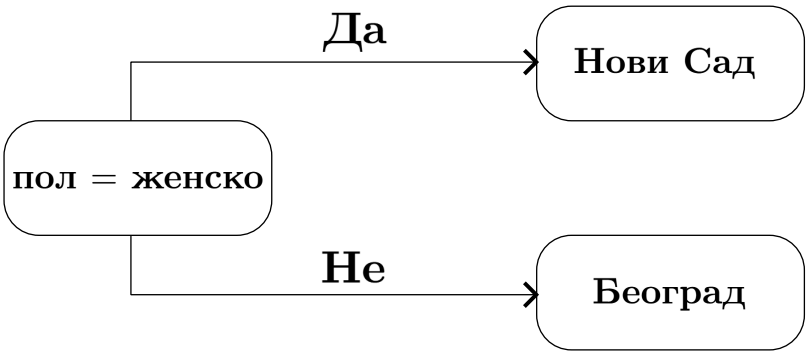
Сада делимо скуп података на два подскупа, пол = женско и пол ≠ женско (видети табелу 5.3).

Име	Пол	Место рођења
Милица	женско	Нови Сад
Јована	женско	Нови Сад
Софија	женско	Нови Сад

Име	Пол	Место рођења
Петар	мушко	Београд
Марко	мушко	Београд

Табела 5.3: Име = женско (лево) и Име ≠ женско (десно).

Уколико сви чланови подскупа враћају исту вредност (као место рођења у нашем случају), тада тај подскуп сматрамо завршеним. У супротном, понављамо процес, избацјујући све атрибут-вредност парове чија се вредност подудара у целом скупу. Овај процес се назива рекурзивно партиционисање (енгл. *Recursive partitioning*). У нашем случају, добија се само једно правило: пол = женско (видети слику 5.1).



Слика 5.1. Графички приказ добијеног стабла одлучивања.

Сада можемо имплементирати ово стабло у *Python*-у.

```
from collections import Counter
podaci = [
    ['Ime', 'Pol', 'MestoRodjenja'],
    ['Petar', 'musko', 'Beograd'],
    ['Milica', 'zensko', 'Novi Sad'],
    ['Jovana', 'zensko', 'Novi Sad'],
    ['Sofija', 'zensko', 'Novi Sad'],
    ['Marko', 'musko', 'Beograd']
]
rezultatOznaka = 'MestoRodjenja'    #klasifikacija prema mestu rođenja
rezultatIndeks = podaci[0].index(rezultatOznaka)
atrIndex = [i for i, label in enumerate(podaci[0]) if i != rezultatIndeks]
cvorovi = []
brojPoslednjegCvora = 0
workQueue = [(-1, brojPoslednjegCvora, set(range(1, len(podaci))))]

while workQueue:
    roditeljCvorId, cvorId, podaciRedIndeks = workQueue.pop()
    jedinstveniIshodi = {podaci[i][rezultatIndeks] for i in podaciRedIndeks}
    if len(jedinstveniIshodi) == 1:
        cvorovi.append((cvorId, jedinstveniIshodi.pop()))
        continue
    atrVrednostRezultati = [] # Pronalazak najboljeg atributa za podelu
    for atrIndeks in atrIndex:
        for redIndeks in podaciRedIndeks:
            red = podaci[redIndeks]
            vrednost = red[atrIndeks]
            atrVrednostRezultati.append((atrIndeks, vrednost))
    najcesci = Counter(atrVrednostRezultati).most_common(1)
```

```
30      if not najcesci:
31          continue  # Nema validnog deljenja
32      (atrIndeks, atrVrednost), _ = najcesci[0]
33      isti = {redIndeks for redIndeks in podaciRedIndeks
34              if podaci[redIndeks][atrIndeks]==atrVrednost}
35      razliciti = podaciRedIndeks - isti # Delimo podatke
36      brojPoslednjegCvora += 1  # Dodavanje novih čvorova
37      istiId = brojPoslednjegCvora
38      workQueue.append((cvorId, istiId, isti))
39      brojPoslednjegCvora += 1
40      razlicitiId = brojPoslednjegCvora
41      workQueue.append((cvorId, razlicitiId, razliciti))
42      cvorovi.append((cvorId, atrIndeks, atrVrednost, istiId, razlicitiId))
43
44  cvorovi.sort(key=lambda n: n[0])
45
46  def is_leaf(cvor):
47      return len(cvor) == 2
48
49  for cvor in cvorovi:
50      if is_leaf(cvor):
51          print(f'{cvor[0]}: {cvor[1]}')
52      else:
53          cvorId, atrIndeks, atrVrednost, cvorIdIsti, cvorIdRazliciti = cvor
54          print(f'{cvorId}: {podaci[0][atrIndeks]} = {atrVrednost},
55                  Da->{cvorIdIsti}, Ne->{cvorIdRazliciti}')
```

Излаз програма:
0: Pol = zensko, Da->1, Ne->2
1: Novi Sad
2: Beograd

Ово се подудара са нашом шемом представљеној на слици 5.1. Сада је још остало да искористимо модел за предвиђање места рођења нове особе. То ради следећи код:

```
testPodaci = ["Marija", "zensko"]

trenutniCvor = cvorovi[0]
while True:
    if is_leaf(trenutniCvor):
        print("predvidjanje: {}".format(trenutniCvor[1]))
        break
nodeId, atrIndeks, atrVrednost, cvorIdIsti, cvorIdRazliciti = trenutniCvor
trenutniCvor = cvorovi[cvorIdIsti if
    testPodaci[atrIndeks] == atrVrednost else cvorIdRazliciti]
```

Излаз програма:
0: Pol = zensko, Da->1, Ne->2
1: Novi Sad
2: Beograd
predvidjanje: Novi Sad

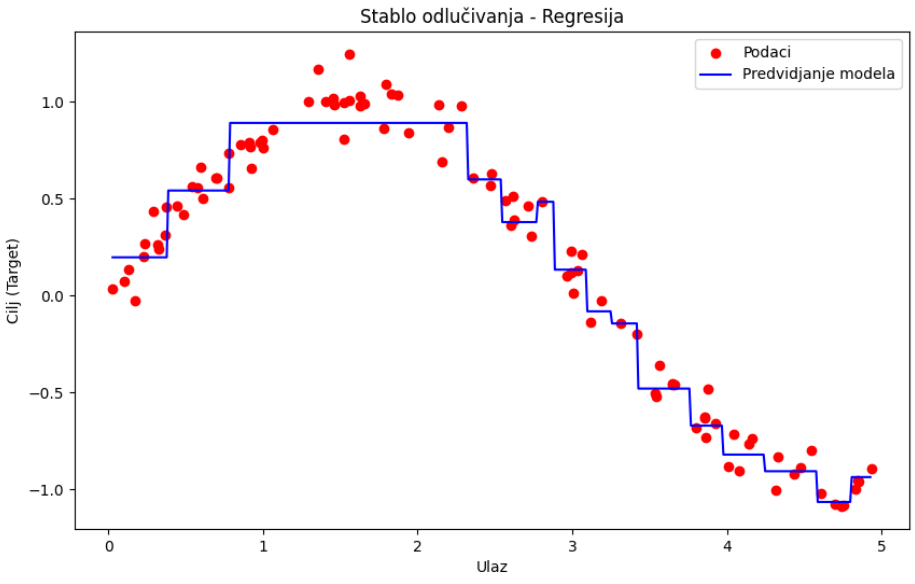
Веће моделе стабла одлучивања можемо лако имплементирати помоћу sklearn библиотеке, коју увозимо на следећи начин:

```
from sklearn.tree import DecisionTreeRegressor, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Лако се може креирати велик синтетички скуп података помоћу np.random.seed() функције, а потом поделити на део за тестирање и део за тренирање, помоћу train_test_split() функције, као и раније. Имплементирамо код (излаз је представљен на слици 5.2. Програм као излаз такође даје шематски приказ стабла):

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from sklearn.tree import DecisionTreeRegressor, export_text
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_squared_error
6
7  np.random.seed(42)
8  X = np.sort(5 * np.random.rand(100, 1), axis=0)
9  y = np.sin(X).ravel() + np.random.normal(0, 0.1, X.shape[0])
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
regressor = DecisionTreeRegressor(max_depth=4, random_state=42)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"MSE: {mse:.4f}")
X_grid = np.arange(min(X), max(X), 0.01)[:, np.newaxis]
y_grid_pred = regressor.predict(X_grid)
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='red', label='Podaci')
plt.plot(X_grid, y_grid_pred, color='blue', label='Predvidjanje modela')
plt.title("Stablo odlučivanja - Regresija")
plt.xlabel("Ulaz")
plt.ylabel("Cilj (Target)")
plt.legend()
plt.show()
from sklearn.tree import plot_tree

plt.figure(figsize=(20, 10))
plot_tree(regressor,feature_names=["Ulaz"],filled=True,rounded=True,fontsize=10)
plt.title("Struktura Stabla Odlučivanja")
plt.show()
```



Слика 5.2. Графички приказ модела стабла одлучивања.

6 Метода потпорних вектора

6.1 Линеарни *SVM* модел

Рецимо да желимо да направимо модел за детекцију спам имејлова. Наш скуп података за тренирање модела би састојао од великог броја имејлова (рецимо 10,000) обележених ознаком „spam" или „nije_spam". Тако форматиране улазне податке можемо представити помоћу уређених парова (x_i, y_i), где y_i узима вредност 0 или 1, а x_i је N-димензиони вектор облика

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iN} \end{bmatrix},$$

где компонента x_i представља једну реч у српском језику (нпр. можемо имати листу од 30.000 речи, где x_{i1} одговара речи „а", x_{i2} „аба", ..., x_{i30000} „шчепати") и узима једну вредност из скупа $\{0, 1\}$ (компонента x_{ij} узима вредност 1 ако се реч j појављује у имејлу i, односно 0 ако се не појављује.). Ова метода је позната под називом „вреће речи" (енгл. *Bag of Words*).

За реализацију овог задатка можемо користити алгоритам методе потпорних вектора (енгл. *Support Vector Machine*, или скраћено - *SVM*). Овај алгоритам захтева да позитивне (1) компоненте имају вредност $+1$, а негативне (0) вредност -1.

Алгоритам представља векторе као тачке у $30,000$-димензионом простору и конструише $30,000$-димензиону хиперраван која раздваја позитивне од негативних тачака (слика 6.1). Граница која раздваја примере из различитих класа се назива граница одлуке (енгл. *Decision Boundary*). Једначина хиперравни је представљена следећим изразом:

$$\omega_1 x_{i1} + \omega_2 x_{i2} + \cdots + \omega_N x_{iN} - b = 0 \text{ (краће записано } \omega^T x_i - b = 0).$$

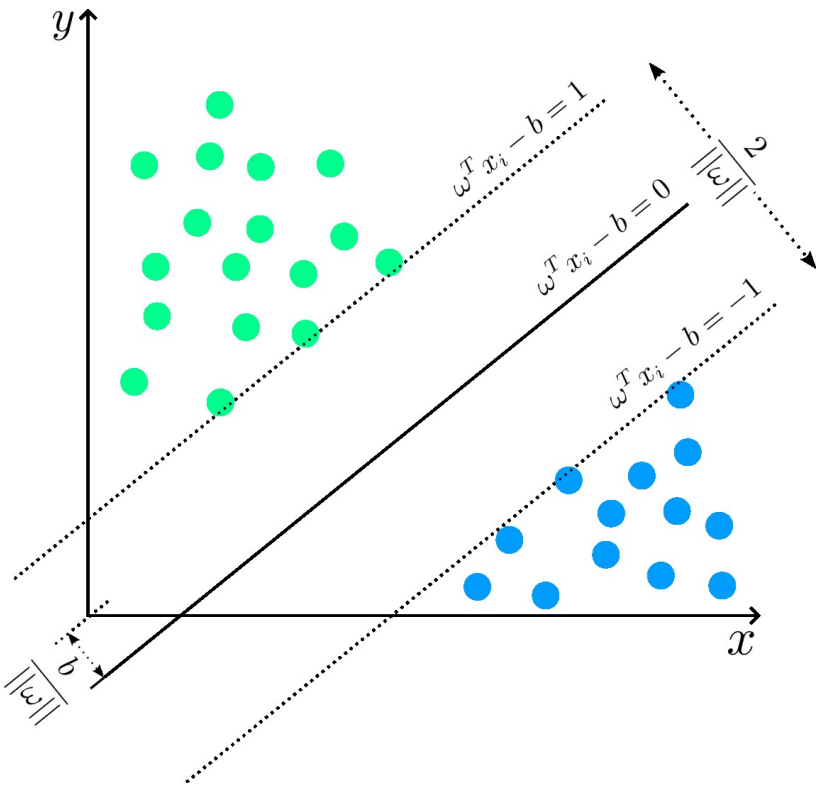
Излазна вредност за неки улазни x_i има вредност

$$y = sgn(\omega^T x_i - b) = \begin{cases} +1, \ \omega^T x_i - b > 0 \\ -1, \ \omega^T x_i - b < 0 \end{cases} .$$

Као и код претходних алгоритама, циљ *SVM* алгоритма је проналазак оптималних вредности параметара ω и b решавањем оптимизационог проблема. Постављамо следећа ограничења:

$$\omega^T x_i - b \geq +1 \text{ ако је } y_i = +1, \text{односно } \omega^T x_i - b \leq -1 \text{ ако је } y_i = -1$$

$$\Leftrightarrow y_i(\omega^T x_i - b) \geq 1,\ i \in \{1, \ldots, N\}$$



Слика 6.1. Графички приказ SVM модела.

Такође желимо да максимизујемо дистанцу између два међусобно најближа члана сваке класе, како бисмо максимално повећали прецизност модела у будућности. Ова дистанца се назива маргина (енгл. *Margin*). То радимо минимизовањем еуклидске норме (интензитета) вектора ω, која се рачуна на следећи начин:

$$||\omega|| = \sqrt{\omega_1^2 + \omega_2^2 + \cdots + \omega_N^2} = \sqrt{\sum_{i=1}^{N} \omega_i^2}\ .$$

Ово радимо како бисмо максимизовали дистанцу између паралелних хиперравни представљених једначинама $\omega^T x_i - b = 1$ и $\omega^T x_i - b = -1$ (видети слику 6.1). Изведимо формулу ове дистанце. Рецимо да имамо две хиперравни, π_1 и π_2, представљене следећим једначинама:

$$\pi_1 : \omega^T x_i = b_1, \;\; \pi_2 : \omega^T x_i = b_2, \;\; x_i, \omega \in \mathbb{R}^n; \; b_1, b_2 \in \mathbb{R}$$

Без умањења општости, рецимо да постоје $X_1, X_2 \in \mathbb{R}^n$ тд. $X_1 \in \pi_1, \; X_2 \in \pi_2$.

$$\implies \exists c_1, c_2 \in \mathbb{R} \text{ тд. } X_1 = c_1\omega, \; X_2 = c_2\omega$$

$$\implies \omega^T(c_1\omega) = b_1 \; \wedge \; \omega^T(c_2\omega) = b_2$$

$$\implies c_1 = \frac{b_1}{\omega^T\omega} = \frac{b_1}{||\omega||^2} \; \wedge \; c_2 = \frac{b_2}{\omega^T\omega} = \frac{b_2}{||\omega||^2}$$

$$\implies d(\pi_1, \pi_2) = ||X_1 - X_2|| = \left|\left| \frac{b_1}{||\omega||^2}\omega - \frac{b_2}{||\omega||^2}\omega \right|\right| = \frac{||(b_1 - b_2)\omega||}{||\omega||^2} = \frac{|b_1 - b_2|}{||\omega||}$$

Тиме смо добили израз за дистанцу:

$$\boxed{d(\pi_1, \pi_2) = \frac{|b_1 - b_2|}{||\omega||} = \frac{2}{||\omega||}} \tag{6.1}$$

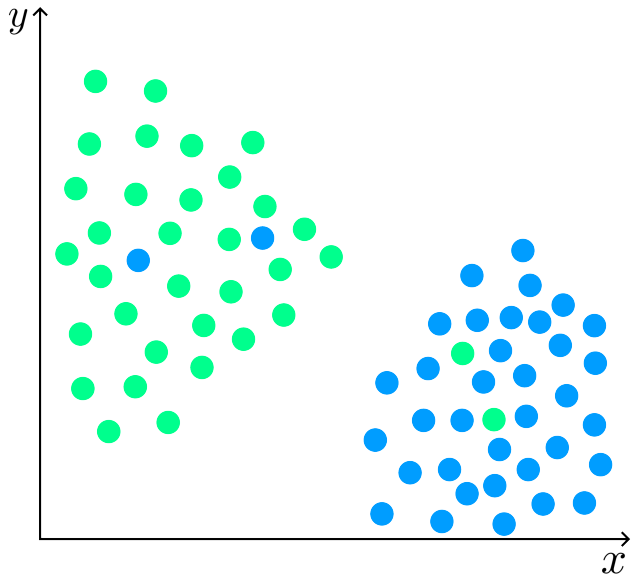
Вредност израза 6.1 достиже максимум када вредност израза $||\omega||$ достигне минимум. Дакле, закључујемо да је маргина максимална за минималну вредност еуклидске норме вектора ω.

Ова верзија алгоритма се назива линеарни модел (енгл. *Linear Model*), јер је граница одлуке представљена линеарно (помоћу хиперравни), што није увек могуће. О нелинеарним *SVM* моделима ћемо дискутовати у одељку 4.5.3.

6.2 Шум у скупу података *SVM* модела

Шта радити када је у скупу података присутан „шум" (слика 6.2)? У датом примеру је очигледно да би се подаци лако могли одвојити правом линијом на две групе када не би било шума.
Како бисмо решили овакве проблеме, уводимо тзв. функцију губитка шарке (енгл. *Hinge Loss Function*), која је једнака нули ако су испуњени услови за SVM модел које смо раније навели, тј. ако је $\omega^T x_i$ на правој страни границе одлуке:

$$L(y) = \max(0, \; 1 - y_i(\omega^T x_i - b)).$$



Слика 6.2. Графички приказ шумовитог скупа података.

За податке на погрешној страни границе одлуке, вредност функције је пропорционална њиховој удаљености од границе одлуке.

Сада желимо да минимизујемо следећи израз:

$$C||\omega||^2 + \frac{1}{N}\sum_{i=1}^{N}\max(0,\ 1 - y_i(\omega^T x_i - b)), \tag{6.2}$$

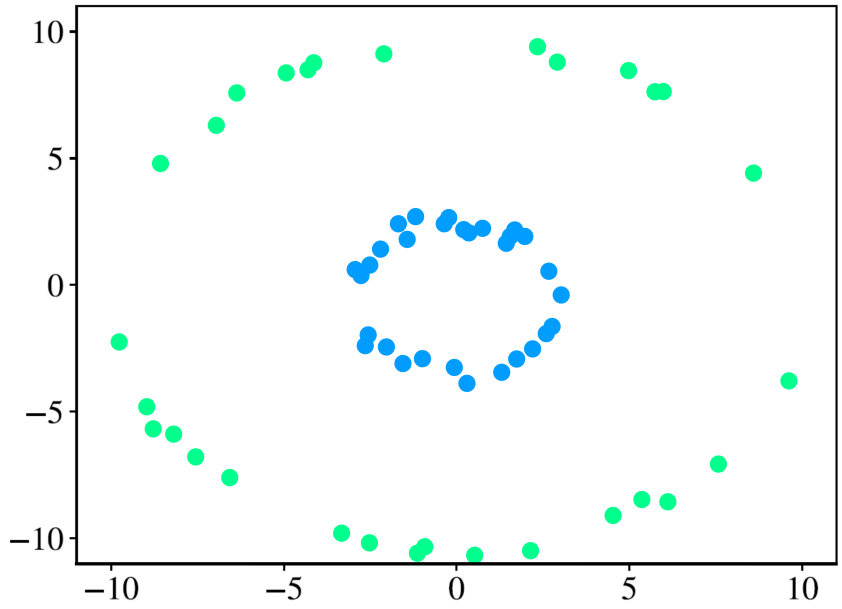
где је C хиперпараметар који одређује компромис између увећања границе одлуке и обезбеђивања да свако x_i лежи на правој страни границе одлуке. Вредност овог параметра се углавном одређује експериментално.

Корисно је напоменути да постоје две врсте SVM модела, *soft-margin* (користе Hinge Loss функцију) и *hard-margin* SVM модели (не користе Hinge Loss функцију).

Из израза 4.7 се може видети да за велике вредности параметра C, други сабирак постаје занемарљив, што доводи до тога да модел проналази највишу границу, не узимајући у обзир погрешно класификоване податке. Са друге стране, мање вредности параметра C доводе то тога да минимизовање броја грешака постане приоритет алгоритма (науштрб величине границе одлуке).

6.3 Нелинеарни *SVM* модели

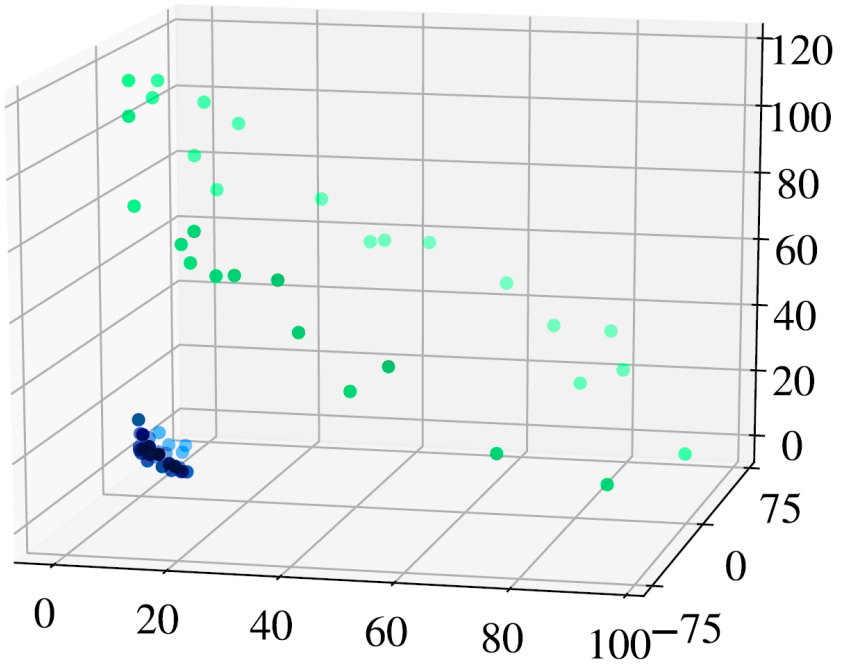
Такође се може десити да се подаци у свом простору не могу одвојити помоћу хиперравни (слика 6.3). Овакав проблем можемо превазићи трансформацијом почетног простора у простор веће димензионалности, што омогућава да подаци постану линеарно одвојиви у овом новом простору. Ова метода се назива кернелски трик (енгл. *The Kernel Trick*).



Слика 6.3. Графички приказ нелинеарно раздвојивог скупа података.

Ефекат примене кернелског трика приказан је на слици 6.4. Као што се може уочити, могуће је претворити линеарно неразвојив скуп података у линеарно раздвојив скуп више димензоналности (у овом случају трансформишемо $2D$ скуп у $3D$ простор), при чему користимо специфичну трансформацију $\phi : x \mapsto \phi(x)$, где је $\phi(x)$ вектор више димензионалности од вектора x.

Рецимо, у нашем примеру са слике 6.3, $x = \begin{bmatrix} p \\ q \end{bmatrix}$ и $\phi(x) = \begin{bmatrix} p^2 \\ \sqrt{2}pq \\ q^2 \end{bmatrix}$.



Слика 6.4. Примена кернелског трика за постизање линеарног раздвајања.

Да бисмо разумели како кернели функционишу, потребно је прво објаснити метод проналаска оптималних вредности параметара ω и b. За решавање проблема 6.1, углавном се користи тзв. метода Лагранжових мултипликатора (енгл. *Lagrange Multipliers*). Уместо решавања оригиналног проблема из 6.1, згодно је решити еквивалентан проблем, формулисан на следећи начин:

$$\max_{\alpha_1,\ldots,\alpha_N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \left[\sum_{k=1}^{N} y_i \alpha_i (x_i x_k) y_k \alpha_k \right] \text{ тд. } \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ и } \alpha_i \geq 0,\ i \in \{1, \ldots, N\},$$

где се чланови α_i називају Лагранжови мултипликатори. Овако постављен проблем се ефикасно може решити помоћу програмских алгоритама.

На овај начин можемо избећи трансормацију x_i у $\phi(x_i)$ и x_j у $\phi(x_j)$, тако што, служећи се кернелским триком, проналазимо скаларни производ вектора и квадрирамо га, чиме добијамо идентичан резултат. На пример, у раније дискутованом примеру, $\left(\begin{bmatrix} p_1 \\ q_1 \end{bmatrix} \cdot \begin{bmatrix} p_2 \\ q_2 \end{bmatrix} \right)^2 = p_1^2 p_2^2 + 2 p_1 p_2 q_1 q_2 + q_1^2 q_2^2$. Ово је био пример употребе квадратног кернела $k(x_i, x_j) = (x_i, x_j)^2$. Постоји мноштво кернелских функција, од којих је најпознатији *RBF* кернел:

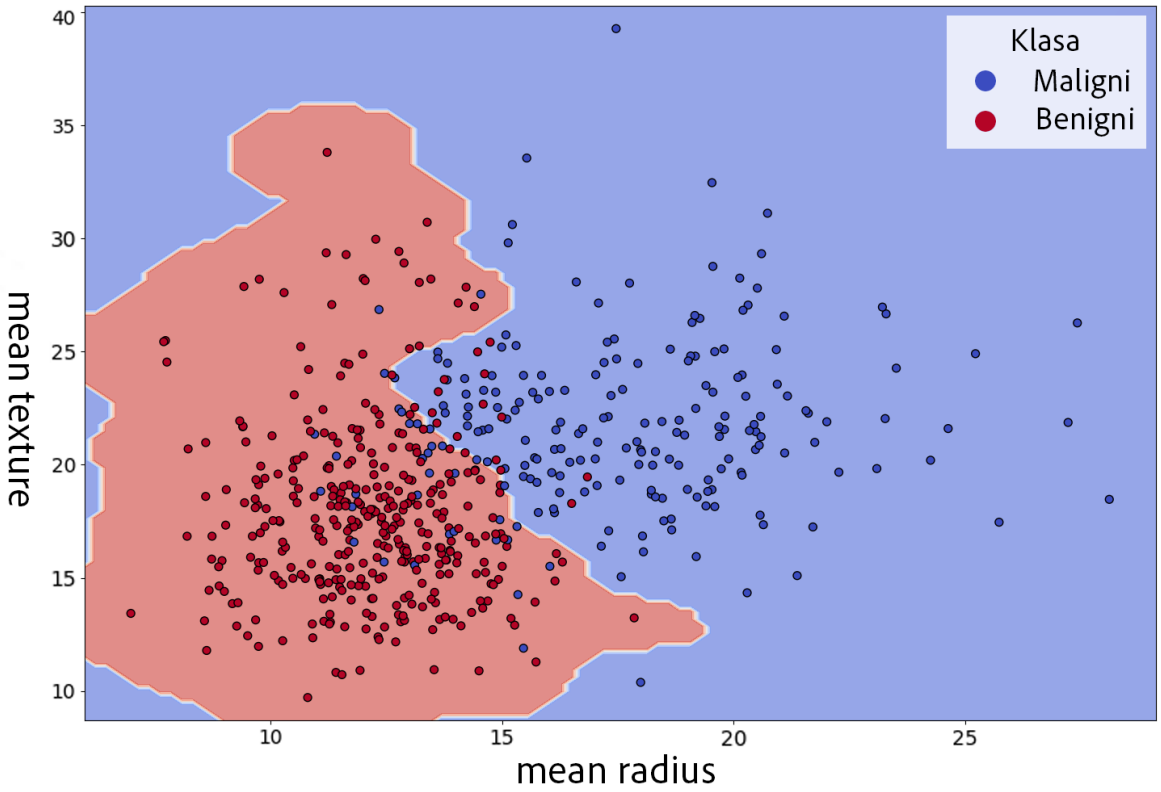
$$k(x, x') = \exp\left(- \frac{||x - x'||^2}{2\sigma^2} \right),$$

где је $||x - x'||$ еуклидска дистанца између два вектора.

6.4 Имплементација *SVM* модела у *Python*-у

Имплементирамо SVM модел на breast_cancer скупу података. Графички приказ модела је представљен на слици 6.5.

```python
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Učitavanje skupa podataka
cancer = load_breast_cancer()
X = cancer.data[:, :2]  # Samo prve dve osobine
y = cancer.target

# Pravljenje i treniranje modela
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
svm.fit(X, y)

# Prikaz Granice odluke
disp = DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    cmap=plt.cm.coolwarm,
    alpha=0.6,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1],
)

# Dijagram rasejanja podataka
scatter = plt.scatter(
    X[:, 0], X[:, 1],
    c=y,
    cmap=plt.cm.coolwarm,
    edgecolor='k',
    s=50
)

# Dodavanje legende
handles, labels = scatter.legend_elements()
plt.legend(handles, ["Maligni", "Benigni"], title="Klasa")

plt.title("SVM Decision Boundary na Breast Cancer Podacima")
plt.show()
```



Слика 6.5. Графички приказ SVM границе одлуке на breast_cancer подацима.

7 Неуронске мреже

Неуронска мрежа (енгл. *Neural Network*) је угнеждена математичка функција. На пример, трослојна неуронска мрежа је функција облика

$$f_{NN}(x) = f_3(f_2(f_1(x))),$$

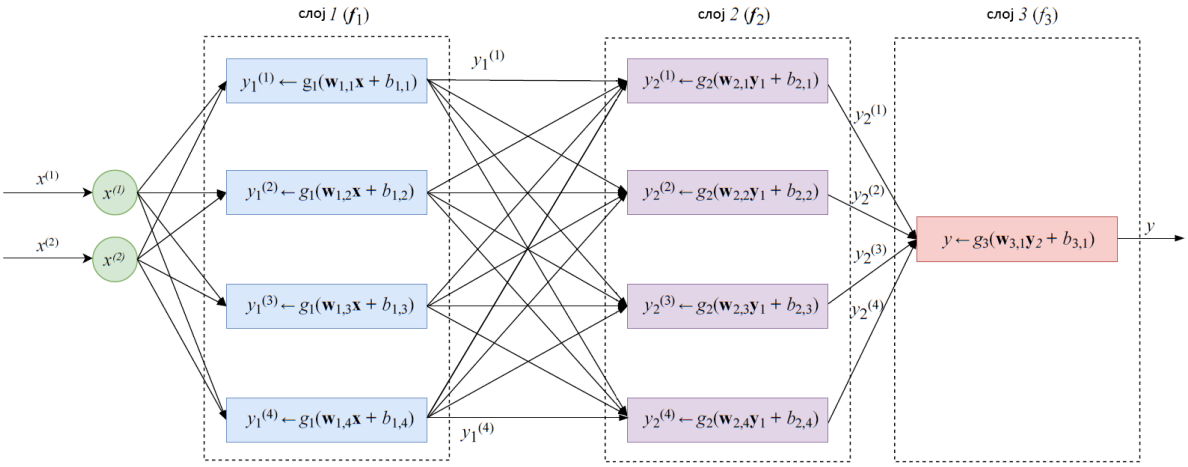
где су f_2 и f_3 векторске функције облика $f_l(z) = g_l(W_l z + b_l)$. $l \in \{1, \ldots, N\}$ је индекс слоја (енгл. *Layer index*). Функција g_l се назива активациона функција и представља фиксирану, углавном нелинеарну, функцију која се бира пре почетка тренирања модела. Параметри (матрица W_l и вектор b_l) за сваки слој су добијени алгоритмом градијентног спуста.

Позабавимо се сада вишеслојним перцептроном (енгл. *Multilayer Perceptron - MLP*), једном од познатих архитектура неуронских мрежа. Ради једноставности размотримо трослојни MLP модел (слика 7.1), који као улаз узима дводимензиони вектор, а даје скалар као излаз. Излаз сваке јединице, тј. неурона (круг или правоугаоник на слици), је резултат математичке операције (уписане унутар правоугаоника).

Прво, сви улазни подаци једне јединице групишу се у један улазни вектор. Затим, слично моделу линеарне регресије, јединица примењује линеарну трансформацију на вектор. Коначно, јединица примењује активациону функцију g на резултат линеарне трансформације, што даје излазну вредност у облику реалног броја.

На слици 7.1, активациона функција g има један индекс - l, који представља индекс слоја ком јединица припада. Углавном, све јединице једног слоја користе исти индекс (али то није неопходно). Такође, сваки слој може имати различит број јединица. Свака јединица има параметре $w_{l,u}$ и $b_{l,u}$, где је u индекс јединице, а l индекс слоја. Вектор y_{l-1} је у свакој јединици дефинисан

као $y_{l-1} = \begin{bmatrix} y_{l-1}^{(1)} \\ y_{l-1}^{(2)} \\ y_{l-1}^{(3)} \\ y_{l-1}^{(4)} \end{bmatrix}$, а вектор x је у првом слоју дефинисан као $x = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(D)} \end{bmatrix}$.



Слика 7.1. Графички приказ вишеслојног перцептрона.

Као што се може видети на слици 7.1, у вишеслојном перцептрону сваки излаз једног слоја повезан је са сваким улазом наредног слоја. Ово представља потпуно повезану архитектуру (енгл. *Fully-connected architecture*). Неуронска мрежа може садржати потпуно повезане слојеве (енгл. *Fully-connected layers*), чије јединице као улаз примају излазе сваке јединице претходног слоја.

7.1 *Feedforward* архитектура неуронских мрежа

Вишеслојни перцептрон је један од примера архитектуре неуронских мрежа под називом *Feedforward* мреже (неуронске мреже ширења унапред). Овакву мрежу карактерише једносмеран ток информација међу њеним слојевима. Информације се крећу искључиво унапред - од улазних, преко скривених и до излазних неурона, без било каквих петљи или циклуса (за разлику од рекурентних мрежа, у којима се информације могу кретати у два смера).

Уколико желимо да решимо регресивни или класификациони проблем, налик онима у претходним поглављима, последњи слој (десно на слици 7.1) мреже углавном садржи један неурон. Уколико је активациона функција g_N последњег слоја линеарна, онда мрежа има улогу регресионог модела. Слично, уколико је g_N логистичка функција, мрежа се понаша као бинарни класификатор.

Функција $g_{l,u}$ може бити произвољна, са условом да мора бити диференцијабилна на целом или макар већини свог домена. Дифернцијабилност је неопходна због градијентног спуста, који користимо ради проналаска оптималних вредности параметара ($w_{l,u}$ и $b_{l,u}$). Главна сврха нелинеарних компоненти функције f_{NN} је омогућавање неуронској мрежи да апроксимира нелинеарне функције. У супротном, функција f_{NN} би била линеарна, без обзира на број слојева (јер је $W_l z + b_l$ линеарна, а слагањем две линеарне функције добија се такође линеарна функција).

Логистичка (сигмоидна) функција, хиперболични тангенс и исправљачка (ректификациона) функција су чест избор за активациону функцију:

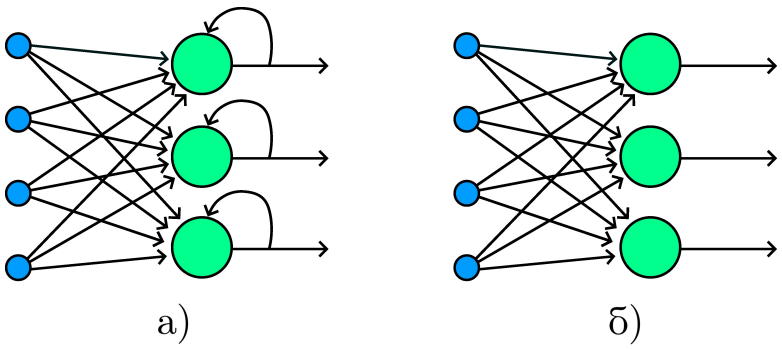
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases} \quad .$$

7.2 Дубоко учење

Појам дубоког учења се односи на тренирање неуронских мрежа које садрже више од два унутрашња, односно скривена, слоја. Осим раније поменутих *Feedforward* неуронских мрежа, постоје и рекурентне неуронске мреже (*RNN*), које су цикличне структуре (слика 7.2).



Слика 7.2. Графички приказ а) рекурентне и б) Feedforward неуронске мреже.

7.2.1 Бекпропагација

Алгоритам бекпропагације (енгл. *Backpropagation*; пропагација уназад) омогућава ефикасно израчунавање парцијалних извода грешке по тежинама мреже, користећи формулу за извод композитне функције. Тежине се затим ажурирају помоћу градијентног спуста. Објаснимо за почетак алгоритам на вишеслојном перцептрону са једним скривеним слојем.

Претпоставимо следећу структуру мреже:

$$x \in \mathbb{R}^n,\ h \in \mathbb{R}^m,\ \hat{y} \in \mathbb{R},\ y \in \mathbb{R},\ L = \frac{(y-\hat{y})^2}{2},$$

$$W^{(1)} \in \mathbb{R}^{m\times n},\ b^{(1)} \in \mathbb{R}^m,\ W^{(2)} \in \mathbb{R}^{1\times m},\ b^{(2)} \in \mathbb{R}\ ,$$

где је x улазни слој, h скривени слој са активационом функцијом σ, $\hat{y}$ излазни неурон са активационом функцијом f, y циљна (тачна) излазна вредност, L функција грешке, $W^{(1)}$ и $b^{(1)}$ тежине и помераји за улаз $\rightarrow$ скривени слој респективно, а $W^{(2)}$ и $b^{(2)}$ тежине и померај за скривени слој $\rightarrow$ излаз респективно.

- Корак 1: Пропагација унапред
  Скривени слој:

  $$z^{(1)} = W^{(1)}x + b^{(1)} \text{ (примена линеарне трансформације)}$$

  $$h = \sigma(z^{(1)}) \text{ (нелинеарна активација)}$$

  Излазни слој:

  $$z^{(2)} = W^{(2)}h + b^{(2)}$$

  $$\hat{y} = f(z^{(2)})$$

  Функција грешке (нпр. MSE):

  $$L = \frac{(y-\hat{y})^2}{2}$$

- Корак 2: Бекпропагација (рачунање градијената)
  Циљ нам је израчунавање градијената $\frac{\partial L}{\partial W^{(2)}}$, $\frac{\partial L}{\partial b^{(2)}}$, $\frac{\partial L}{\partial W^{(1)}}$, $\frac{\partial L}{\partial b^{(1)}}$
  Излазни слој:

  $$\delta^{(2)} = \frac{\partial L}{\partial z^{(2)}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{(2)}} = (\hat{y}-y)\cdot f'(z^{(2)}) \text{ (грешка на излазу)}$$

  $$\frac{\partial L}{\partial W^{(2)}} = \delta^{(2)} \cdot h^T \text{ (градијент излазне тежине)}$$

  $$\frac{\partial L}{\partial b^{(2)}} = \delta^{(2)} \text{ (градијент помераја)}$$

- Корак 3: Ажурирање параметара
  Користимо стохастички градијентни спуст са кораком учења η. Дакле, параметри се ажурирају на следећи начин:

$$W^{(2)} \leftarrow W^{(2)} - \eta \cdot \frac{\partial L}{\partial W^{(2)}}$$
$$b^{(2)} \leftarrow b^{(2)} - \eta \cdot \frac{\partial L}{\partial b^{(2)}}$$
$$W^{(1)} \leftarrow W^{(1)} - \eta \cdot \frac{\partial L}{\partial W^{(1)}}$$
$$b^{(1)} \leftarrow b^{(1)} - \eta \cdot \frac{\partial L}{\partial b^{(1)}}$$

Показали смо метод функционисања алгоритма бекпропагације на мрежи са једним дубоким слојем. Код дубљих мрежа, алгоритам се користи рекурзивно - грешка се „шири"уназад слој по слој. Дакле, за сваки слој l важи:

$$\boxed{\delta^{(l)} = \left(W^{(l+1)}\right)^T \cdot \delta^{(l+1)} \cdot \sigma'(z^{(l)})} \tag{7.1}$$

7.2.2 Изазови дубоког учења

У прошлости, како је број слојева растао, постајало је све теже тренирати моделе. Главне проблеме при градијентном спусту представљали су експлодирајући и нестајући градијент. Експлодирајући градијент је представљао значајно мањи проблем од нестајућег и лакше се решавао.

Шта је заправо проблем нестајућег градијента? За ажурирање параметара у неуронској мрежи користи се алгоритам бекпропагације. Међутим, у неким случајевима се може десити да је градијент превише мали, што доводи до тога да неки параметри неће моћи да промене вредност (што се може видети у изразу 7.1). У најгорем случају, овај проблем може довести до потпуног заустављања тренинга неуронске мреже.

За неке од популарних активационих функција, попут оних наведених раније, важи $\frac{df}{dx} \in (0, 1)$. Множење n бројева у интервалу $(0, 1)$ доводи до тога да се градијент првих (левих) слојева смањује експоненцијално како n расте (тј. леви слојеви се јако споро тренирају). *ReLU* функција има значајно мање проблема са овиме. LTSM мреже и неке технике (попут *Skip Connections*) дозвољавају тренирање јако дубоких мрежа. Битно је напоменути да се у пракси многи данашњи пословни проблеми могу решити помоћу мрежа са 2 до 3 скривена слоја.

7.2.3 Кратка историја дубоког учења

Током 1920-их, Ленц (нем. Wilhelm Lenz) и Изинг (нем. Ernst Ising) развијају Изингов модел, који се данас може тумачити као рани облик рекурентних неуронских мрежа. Шуничи Амари (јап. Shun'ichi Amari) унапређује овај модел 1972. године, дајући му адаптивна својства. Совјетски математичар Ивакненко (укр. Олексíй Григорович Івахненко) објављује први функционални алгоритам дубоког учења, GMDH, још 1965. године. Први вишеслојни перцептрон трениран стохастичким градијентним спустом објављен је 1967, а Кунихико Фукушима 1969. уводи *ReLU* функцију и развија неокогницију, рану верзију конволуционих неуронских мрежа.

Бекпропагација, заснована на правилу извода сложене функције које је још 1673. године формулисао Лајбниц, добија савремен облик кроз рад Линаинмае (фин. Seppo Linnainmaa) 1970. године. Прву примену на неуронске мреже даје Пол Вербос (енгл. Paul Werbos) 1982, а алгоритам постаје широко познат и коришћен након популаризације 1986. године.

Током 1980-их и 1990-их развијени су кључни темељи дубоког учења. CNN мреже су први пут успешно коришћене за препознавање писма и медицинских слика, а LeNet-5 из 1998. постаје важна примена у банкама. RNN мреже су развијане за обраду секвенци, али су имале проблем нестајућих градијената, који је решен увођењем LSTM архитектуре 1995. године. Шмидхубер (нем. Jürgen Schmidhuber) је још 1991. развио концепте претренираних хијерархија и адверсаријалних мрежа, које су касније инспирисале GAN-ове. Иако у то време неуронске мреже нису надмашиле традиционалне моделе у говорној обради, крајем 1990-их почињу прве успешне индустријске примене дубоког учења, посебно у препознавању говорника.

У периоду 1990–2010, неуронске мреже су изгубиле популарност због високих рачунарских захтева и недостатка разумевања биолошке основе, па су се уместо њих користили једноставнији модели са ручно израђеним особинама, попут Габорових филтера и методе потпорних вектора. Прекретница се догодила средином 2000-их: LSTM је постао конкурентан традиционалним системима за препознавање говора, а касније и победио на такмичењу у препознавању рукописа (2009). Истовремено, Хинтон и сарадници развили су дубоке вероватносне мреже (енгл. deep belief networks), које су показале да је могуће учити сложене дистрибуције података, али су биле споре.

У индустрији, CNN је још раније био примењен за обраду чекова у САД, али тек око 2010. године долази до већих улагања у моделе за препознавање

говора. Кључно је било откриће да обична бекпропагација са великим количинама података може надмашити тадашње генеративне моделе (GMM-HMM), што је покренуло примену дубоких неуронских мрежа у комерцијалним системима за препознавање говора.

Револуција у дубоком учењу започела је напретком у рачунарском виду заснованом на конволуционим неуронским мрежама. Иако су CNN-ови и раније постојали, значајан помак догодио се тек уз брже графичке процесоре – први значајан успех забележен је 2011. године са мрежом DanNet, која је по први пут надмашила човека у задацима визуелног препознавања. Године 2012, AlexNet је победио на такмичењу ImageNet, чиме је дубоко учење постало доминантно у области слике. У наредним годинама уследили су модели попут VGG, Inception и ResNet, који су омогућили веома дубоке мреже (20+ слојева). CNN-ови су се касније комбиновали с LSTM-овима за описивање слика и друге комплексне задатке. У генеративном моделовању, GAN-ови (2014) су омогућили стварање висококвалитетних слика, што је кулминирало успехом StyleGAN-a (2018) и касније дифузионих модела (2022) попут DALL-E 2 и Stable Diffusion. Паралелно, LSTM модели су побољшали аутоматско препознавање говора, па је Гоогле 2015. пријавио умањење грешака од чак 49% .

Дубоко учење је данас основа бројних технологија у визуелном препознавању и препознавању говора, а пионири овог поља награђени су Тјуринговом наградом 2018. године.

7.3 Конволуционе неуронске мреже

Оптимизација неуронских мрежа са великим бројем слојева је прилично тежак задатак, јер број параметара драстично расте са повећањем броја слојева. Конволуциона неуронска мрежа (енгл. *Convolutional Neural Network* - скраћено CNN) је посебна врста FFNN мрежа која значајно смањује број параметара у великим мрежама уз минималну редукцију квалитета модела. CNN мреже се користе при процесирању слике и звука, где превазилазе већину осталих модела. Објаснићемо метод функционисања конволуционе неуронске мреже на примеру класификације слика.

Суседни пиксели на слици су углавном сличне вредности (изузетак су они на ивицама објеката), па ако истренирамо неуронску мрежу да препознаје ивице и регионе пиксела сличне боје, моћи ће да „препозна" објекат представљен на слици. Ово ћемо постићи користећи тзв. технику клизећег прозора (енгл. *Moving Window*). Тренираћемо више мањих регресивних модела одједном, сваки на по једном квадратном исечку слике.

Један такав мали регресивни модел биће налик оном на слици 7.1 (осим што би имао само први слој). За детекцију образаца, он мора „научити" параметре $p \times p$ филтер матрице F, где је p величина исечка слике. Претпоставимо (ради једноставности) да је улазна слика црно-бела и $p = 3$. Ако 1 представља црне, а 0 беле пикселе, нека улазна матрица P би могла изгледати овако:

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

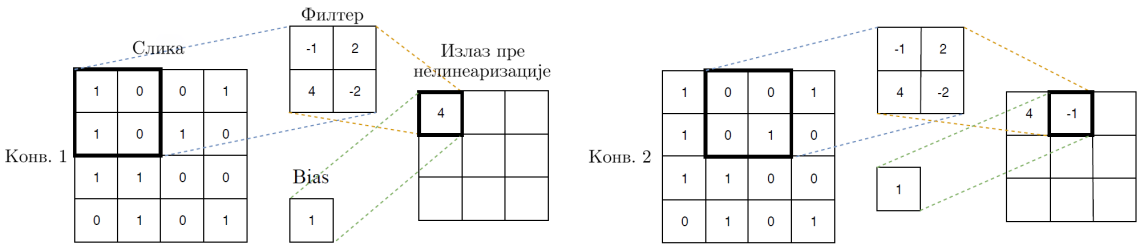
Оваква матрица би представљала образац у облику крста. Матрица модела који би служио за детектовање оваквих (и само оваквих) образаца би на позицијама где се налазе јединице имала позитивне бројеве, док би на позицијама са нулом имала бројеве блиске нули. Нека филтер матрица изгледа овако:

$$F = \begin{bmatrix} 0 & 2 & 3 \\ 2 & 4 & 1 \\ 0 & 3 & 0 \end{bmatrix}.$$

Нека је сада вектор v дефинисан на следећи начин:

$$v = P \cdot F.$$

Што је матрица F сличнија матрици P, то је вредност $S = \sum_{i=1}^{n} v_i$ (сума вредности свих компонената вектора v) већа. Ова операција - сумирање вредности компонената скаларног производа улазне матрице исечка и филтер матрице, назива се конволуција. Да је матрица P имала другачији облик, вредност S била би мања. Такође, за сваки филтер F постоји и вредност пристрасности (енгл. *Bias*) b, која се сабира са резултатом конволуције пре нелинеаризације.

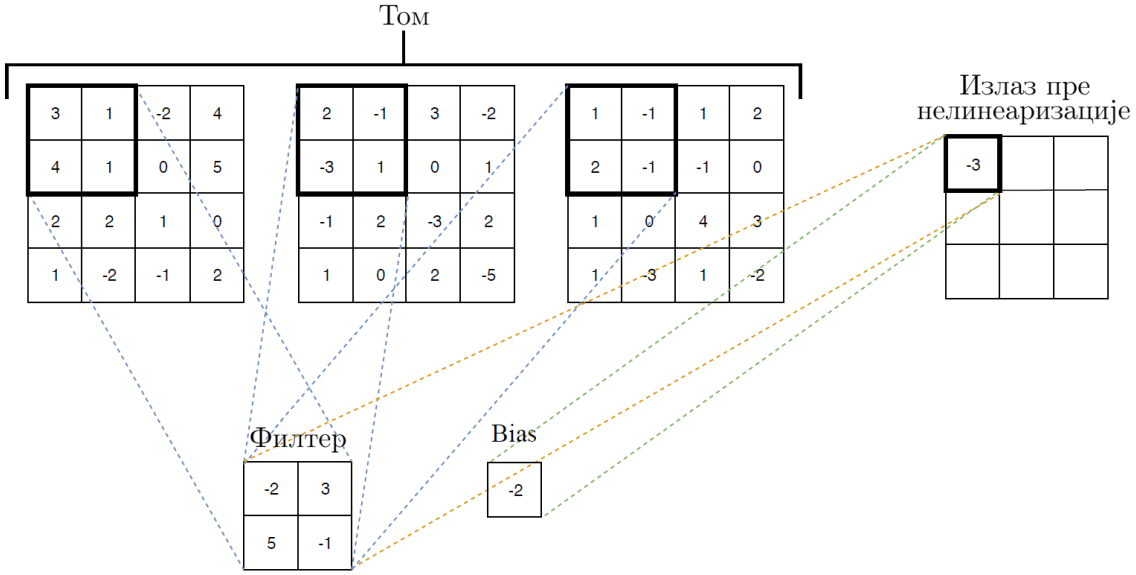


Слика 7.3. Графички приказ прва два корака конволуције на улазној слици.

Један слој CNN мреже састоји се од више конволуционих филтера са својим вредностима пристрасности. Сваки филтер првог слоја „клизи" преко улазне слике (лево → десно и горе → доле) и конволуција се извршава при свакој итерацији (слика 7.3).

Вредности унутар филтерске матрице, као и вредност параметра пристрасности одређују се помоћу градијентног спуста са бекпропагацијом (минимизује се *cost* функција). Нелинеаризација се примењује на суму конволуције и параметра пристрасности. Углавном се у свим скривеним слојевима користи *ReLU* активациона функција. Активациона функција излазног слоја зависи од врсте задатка за који се мрежа користи. Ако слој l има N_l филтера, излаз конволуцијског слоја l састоји се од N_l матрица (по једна за сваки филтер).

Уколико CNN има два конволуциона слоја (l и $l+1$) један за другим, тада ће слој $l+1$ третирати излаз слоја l као скуп од N_l матрица слике. Такав скуп се назива „том" (енгл. *Volume*). Сваки филтер слоја $l+1$ извршава конволуцију над целим томом. Конволуција исечка тома представља суму конволуција одговарајућих исечака индивидуалних матрица од којих је том сачињен (слика 7.4).



Слика 7.4. Графички приказ конволуције тома сачињеног од три матрице.

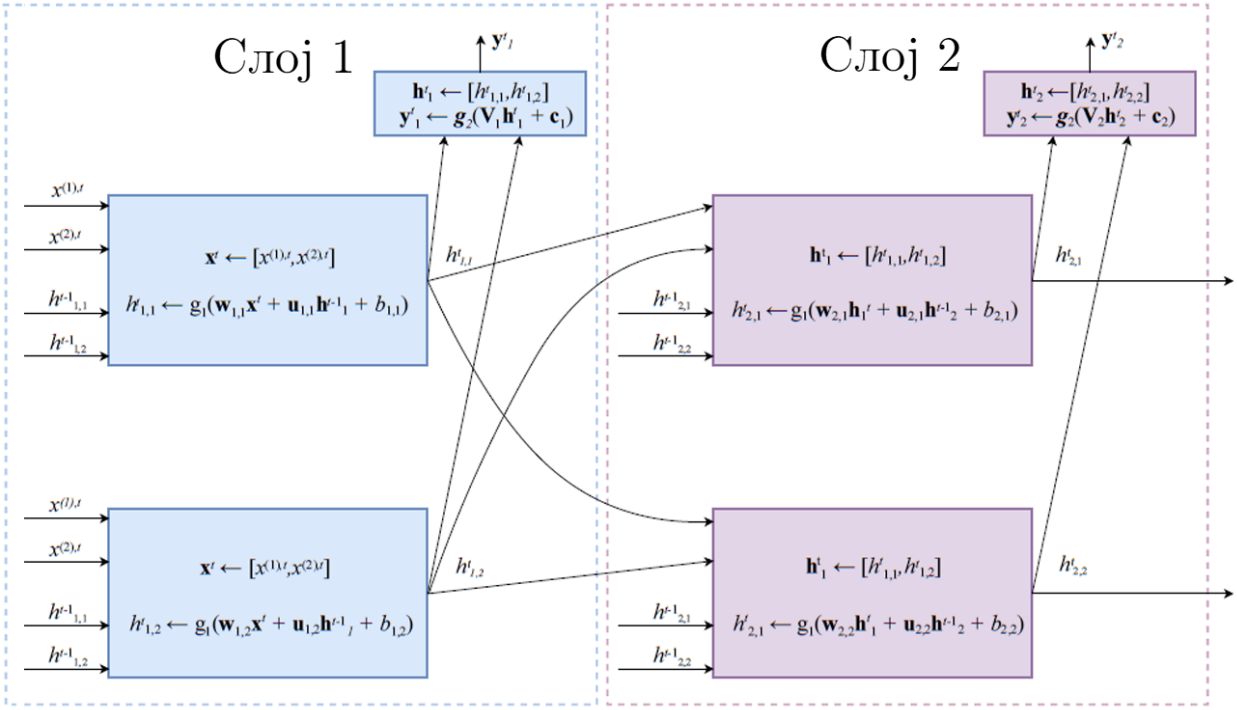
У рачунарском виду (енгл. *Computer Vision*), CNN-ови често добијају томове као улаз, јер се слика углавном представља помоћу три канала (црвена, зелена и плава боја), од којих је сваки у форми монохроматске слике.

7.4 Рекурентне неуронске мреже

Рекурентне неуронске мреже (енгл. *Recurrent Neural Networks* - скраћено RNN) користе се за обележавање, класификацију и генерисање секвенци. Секвенца је матрица чији сваки ред представља улазни вектор (при чему је

редослед редова битан). RNN-ови се често користе за текстуалну и аудио обраду, које представљају секвенце карактера/звука.

Као што је поменуто раније, рекурентне мреже, за разлику од *feedforward* мрежа, садрже петље. Свака јединица u рекурентог слоја l има стање $h_{l,u} \in \mathbb{R}$, које се може замислити као меморија јединице u. Свака јединица u у мрежи, у сваком слоју l, прима два улаза: вектор излаза претходног слоја $l-1$ и вектор стања истог слоја l из претходног „временског корака" (слика 7.5).



Слика 7.5. Графички приказ прва два слоја рекурентне неуронске мреже.

Сваки тренинг пример представљен је у облику матрице вектора X облика $X = \begin{bmatrix} x_1 & x_2 & \dots & x_{N_x} \end{bmatrix}$, где N_x представља дужину улазне секвенце. Уколико је X нпр. текстуална реченица, онда улазни вектор x^t, $t \in \{1, \dots, N_x\}$, представља реч у реченици X на позицији t.

Као што је представљено на слици 7.5, неуронска мрежа „чита" један по један улазни вектор у сваком временском кораку (индекс t представља временски корак). Током сваког временског корака t у свакој јединици u сваког слоја l, рачуна се линеарна комбинација улазног вектора са вектором стања $h_{l,u}^{t-1}$ овог истог вектора из претходног временског корака $t-1$. Линеарна

комбинација два вектора рачуна се користећи два параметарска вектора, $w_{l,u}$ и $u_{l,u}$ и параметар $b_{l,u}$. Вредност $h^t_{l,u}$ се затим добија применом активационе функције g_1 (чест избор је хиперболични тангенс) на резултат линеарне комбинације. Излазни вектор y^t_l се углавном рачуна за цео слој l одједном, помоћу активационе функције g_2 (чест избор је $Softmax$ функција), која узима вектор као улаз и враћа нови вектор исте димензионалности. Функција g_2 се примењује на на линеарну комбинацију вектора стања $h^t_{l,u}$, који се израчунава помоћу параметарске матрице V_l и параметарског вектора $c_{l,u}$. Ове вредности се добијају помоћу градијентног спуста са бекпропагацијом.

За тренирање RNN мрежа користи се посебна верзија бекпропагације - тзв. „бекпропагација кроз време" (енгл. *Backpropagation through Time*). И хиперболички тангенс и $Softmax$ функција имају проблем нестајућег градијента. Чак и ако наш RNN има само један или два рекурентна слоја, због секвенцијалне природе улаза, бекпропагација мора да „размотава" мрежу кроз време. Ово у пракси значи: што је дужа улазна секвенца, размотана мрежа је дубља.

Рекурентне мреже имају још један значајан проблем. Како дужина улазне секвенце расте, улазни вектори са почетка секвенце имају тенденцију да „буду заборављени", јер на стање сваке јединице значајно утичу скорије прочитани улазни вектори. Ово доводи до тога да узрочно-последични однос између удаљених речи у дугачким реченицама може бити изгубљен.

Најефективније рекурентне мреже коришћне у пракси су тзв. *gated* RNN-ови. Овде спадају *Long Short-Term Memory (LSTM)* мреже и мреже базиране на GRU-у (*Gated Recurrent Unit*). Корисна ствар код коришћења *gated* јединица у RNN-овима је то што те мреже могу чувати информације у својим јединицама ради поновног коришћења у будућности. Тренирана мрежа може „читати" улазну секвенцу вектора и у неком раном тренутку t одлучити да сачува специфичне информације у вези са улазним векторима. Ове информације модел може касније искористити ради процесирања улазних вектора ближих крају улазне секвенце. На пример, уколико улазни текст почиње речју „он", модел може сачувати информацију о полу, како би исправно интерпретирао реч „њега" касније у реченици.

Одлуке у вези са чувањем информација су имплементиране помоћу тзв. гејтова (енгл. „*gates*"). Најједноставнија архитектура *gated* јединица је *Minimal Gated GRU*, која се састоји од меморијске ћелије и гејта за заборављање. Наводимо пример GRU јединице првог слоја рекурентне мреже (јединица u у слоју l узима два улаза: вектор вредности меморијских ћелија свих јединица у истом слоју из претходног временског корака h_l^{t-1} и улазни вектор x^t):

$$\tilde{h}_{l,u}^t \leftarrow g_1(w_{l,u}x^t + u_{l,u}h_l^{t-1} + b_{l,u}),$$

$$\Gamma_{l,u}^t \leftarrow g_2(m_{l,u}x^t + o_{l,u}h^{t-1} + a_{l,u}),$$

$$h_{l,u}^t \leftarrow \Gamma_{l,u}^t \tilde{h}_l^t + (1 - \Gamma_{l,u}^t)h_l^{t-1},$$

$$h_l^t \leftarrow [\ h_{l,1}^t,\ \ldots\ ,\ h_{l,N_l}^t\ ]$$

$$y_l^t \leftarrow g_3(V_l h_l^t + c_{l,u}),$$

где је g_1 активациона функција (хиперболични тангенс), док је g_2 гејт функција и имплементира се као сигмоидна функција, која узима вредности у интервалу $(0, 1)$. Уколико гејт $\Gamma_{l,u}$ има вредност блиску 0, тада меморијска ћелија задржава вредност из претходног временског корака h_l^{t-1}. Са друге стране, уколико гејт $\Gamma_{l,u}$ има вредност блиску 1, тада меморијска ћелија добија нову вредност $\tilde{h}_{l,u}^t$. Функција g_3 је углавном $Softmax$ функција.

Gated јединица узима улаз и чува га током неког времена. Ово је еквивалентно примењивању идентитетске функције ($f(x) = x$) на улаз. Како је извод идентитетске функције константан, када се мрежа са *gated* јединица тренира помоћу бекпропагације, градијент неће нестати.

8 Закључак

У овом раду представили смо основне концепте надгледаног машинског учења, његове математичке темеље и неке од најчешће коришћених алгоритама, укључујући линеарну и логистичку регресију, као и стабла одлучивања. Анализирали смо како ови модели функционишу, како се тренирају, евалуирају и примењују у реалним ситуацијама.

Приказали смо и математичку позадину која стоји иза ових метода, као и имплементације у програмском језику Пытхон, уз конкретне примере. Посебан акценат стављен је на интерпретацију резултата и евалуацију модела, што је кључни корак за њихову примену у пракси.

8.1 Остали алгоритми надгледаног машинског учења

Поред обрађених алгоритама, у пракси се често користе и многи други модели надгледаног учења, као што су:

- k најближих суседа ($k - NN$): једноставан алгоритам који класификује пример на основу већине гласова његових најближих суседа у скупу података.
- Наивни Бајесов класификатор (*Naive Bayes*): пробабилистички модел заснован на Бајесовој теореми и претпоставци о независности улазних особина.
- *Random Forest*: ансамбл метода која комбинује више стабала одлучивања како би се побољшала тачност и стабилност модела.
- *Gradient Boosting* (нпр. *XGBoost, LightGBM*): метода која итеративно гради снажне предикторе комбиновањем слабијих модела.

Ови алгоритми се све више користе због своје ефикасности, скалабилности и робусности у раду са великим и комплексним скуповима података.

8.2 Будућност вештачке интелигенције

Вештачка интелигенција, а самим тим и машинско учење, непрекидно напредује и постаје све важнији део савременог друштва. У будућности можемо очекивати:

- Ширу примену у свакодневном животу: од паметних уређаја, персонализованих препорука, до аутономних возила и медицинске дијагностике.
- Развој објашњиве В.И. (*Explainable AI*): све више се ради на томе да модели буду транспарентнији и да њихова предвиђања могу да се објасне и разумеју.
- Етику и регулацију: са порастом моћи В.И, све је важније разматрати њене етичке последице и увести адекватне законске оквире.
- Учење без надзора и учење појачањем: све више пажње се посвећује методама које не захтевају ручно означене податке, већ уче из саме структуре података или интеракције са окружењем.
- Интеграцију са другим пољима: В.И. се све више интегрише са биологијом, хемијом, физиком и друштвеним наукама, чиме доприноси револуцији у многим областима науке и технологије.

Машинско учење остаје један од најдинамичнијих и најперспективнијих праваца у савременој науци, а његово разумевање и примена постаће незаобилазна вештина у готово свим гранама људске делатности.

Литература

[1] A. Samuel, „Some Studies in Machine Learning Using the Game of Checkers”, IBM Journal of Research and Development, 1959.

[2] R. Kohavi, F. Provost, „Glossary of terms", 1998.

[3] P. M. Milner, „The Mind and Donald O. Hebb", Scientific American, 1993.

[4] „ Science: The Goof Button", Tajm magazin, 1961.

[5] T. M. Mitchell, „Machine Learning", 1997.

[6] A. Burkov, "The Hundred-page Machine Learning Book", 2019.

[7] https://www.ibm.com/think/topics/semi-supervised-learning, datum pristupa: 21.12.2024.

[8] https://www.theclassificationsociety.org, datum pristupa: 24.12.2024.

[9] N. Matloff, „Statistical Regression and Classification: From Linear Models to Machine Learning", 2017.

[10] J. Michael, „Model-Based Machine Learning", 2023.

[11] I. Goodfellow, Y. Bengio, A. Courville, „Deep Learning", 2016.

[12] A. Vehtari, A. Gelman, J. Hill, „Regression and Other Stories", 2020.

[13] J. M. Hilbe, „Practical Guide to Logistic Regression", 2016.

[14] C. Sheppard, „Tree-based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting", 2017.

[15] P. Janičić, M. Nikolić, „Veštačka inteligencija", Beograd, 2021.

[16] S. P. Boyd, L. Vandenberghe „Convex Optimization", 2004.

[17] https://www.wikipedia.org

[18] I. Goodfellow, Y. Bengio, A. Courville „Deep Learning (Adaptive Computation and Machine Learning series)", 2016.

[19] I. Griva, S. G. Nash, A. Sofer „Linear and Nonlinear Optimization", 2009.

[20] B. Schölkopf, A. J. Smola „Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond", 2001.

[21] B. Schölkopf, A. J. Smola „Dive Into Deep Learning", 2023.

[22] S. G. Brush „History of the Lenz-Ising Model", 1967.

[23] R. Venkatesan, B. Li „Convolutional Neural Networks in Visual Computing: A Concise ", 2017.

[24] D. Mandić, J. Chambers „Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability", 2001.

[25] https://www.ibm.com/think/insights/artificial-intelligence-future,