

# 1. Intro to Data Science

**Data Science** е област која комбинира математика, статистика, програмирање и доменско знаење со цел извлекување на вредни информации од податоците.

Се користи за **анализа, моделирање и предвидување** и има широка примена (бизнис, медицина, финансии итн.).

## Револуцијата на податоците

Живееме во време кога количината на податоци експоненцијално расте. Оваа "податочна револуција" е овозможена од неколку фактори:

- Дигитализација на скоро сите аспекти на животот (социјални мрежи, сензори, IoT, паметни уреди)
- Намалување на трошоците за складирање и обработка на податоци
- Развој на моќни алгоритми и модели за анализа на огромни количини податоци

**Заклучок:** Податоците станаа клучен ресурс за носење подобри одлуки, откривање на нови трендови и автоматизација на процеси.

## Четирите парадигми на науката

Во историјата, науката е развиена преку четири главни парадигми:

- **Емпириска наука** – базирана на набљудувања и експерименти
- **Теоретска наука** – користи математички модели за да ги објасни природните феномени
- **Компјутерска наука** – симулации и моделирање со помош на компјутери
- **Data driven science** – користи големи количини податоци и машинско учење за автоматско извлекување на модели и заклучоци

**Заклучок:** Data Science овозможува нов начин на истражување каде што машините учат од податоци, без потреба од експлицитно програмирање.

Традиционалното програмирање vs Машинско учење

Традиционално програмирање	Машинско учење
Човекот пишува код	Машината учи од податоци
Влез (податоци) + Програма = Излез	Влез (податоци) + Излез = Програма (модел)
При нови податоци потребно е рачно ажурирање на кодот	Моделот автоматски се адаптира на нови податоци

## Машинско учење и длабоко учење (Deep Learning)

**Машинско учење (ML):** Алгоритми кои учат од податоци за да направат предвидувања.

**Длабоко учење (DL):** Подгранка на ML која користи невронски мрежи со повеќе слоеви за обработка на сложени податоци (слики, текст, звук).

## Разлики меѓу Data Science, Business Intelligence и Machine Learning

**Business Intelligence (BI)** – Анализира податоци за носење бизнис одлуки.

**Machine Learning (ML)** – Автоматско учење од податоци за предвидувања.

**Data Science (DS)** – Комбинира статистика, ML и анализа на податоци.

**Заклучок:** Data Science е поширок концепт кој вклучува ML, BI и Advanced Analytics.

Разлика помеѓу бази на податоци и Data Science

Бази на податоци	Data Science
Мали до средни податоци	Масивни податоци (Big Data)
Фокус на конзистентност и безбедност	Фокус на брзина и аналитика
Структурирани податоци (SQL)	Неструктуирани податоци (текст, слики, сензори итн.)

## Вештини за Data Scientist

Статистика и математика – веројатност, хипотези и модели.

Програмирање – Python, R, SQL, Spark.

Обработка на податоци – Работа со Big Data, ETL процеси.

Машинско учење – Supervised/unsupervised learning, deep learning.

Доменско знаење – Разбирање на индустријата (медицина, финансии, маркетинг).

**Заклучок:** Data Science е мешавина од технички вештини, аналитичко размислување и бизнис експертиза.

## Предизвици во Data Science

Најголемите потешкотии при работа со податоци:

**Проверка на претпоставките** – Дали податоците навистина ја одразуваат реалноста?

**Валидирање на моделите** – Дали моделот ќе функционира во иднина?

**Транзиција од прототип во продукција** – Чистење на податоци, интеграција со системи.

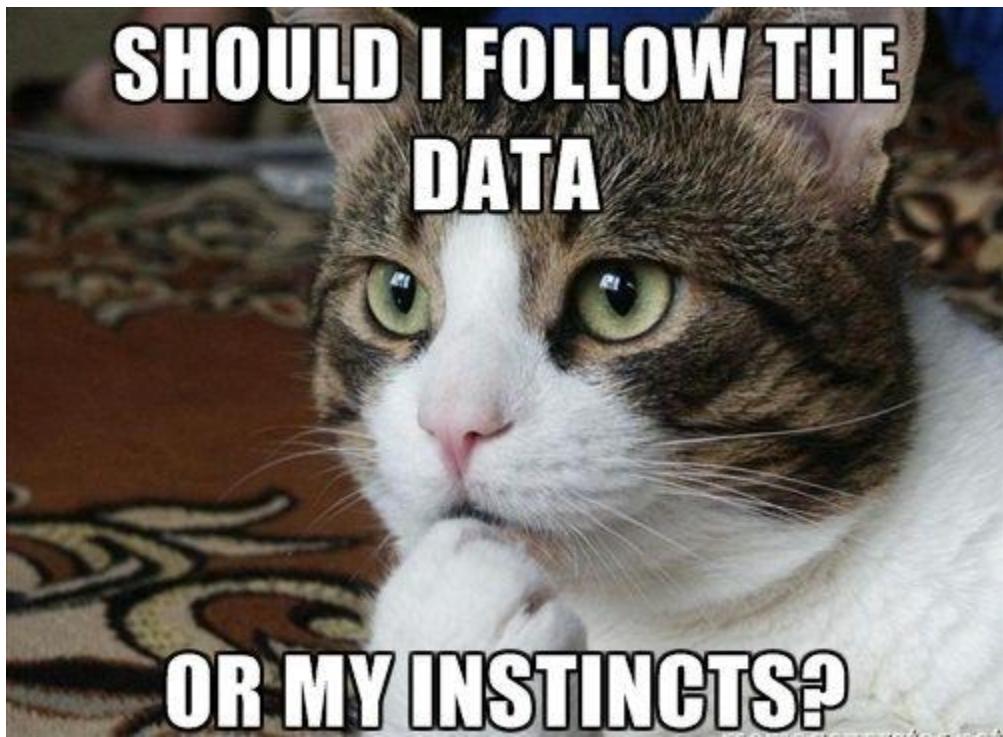
**Доверба во резултатите** – Како да убедите менаџментот дека моделот е точен?

Заклучок: Успехот во Data Science не зависи само од математиката, туку и од вештината за комуникација и имплементација на решенијата.

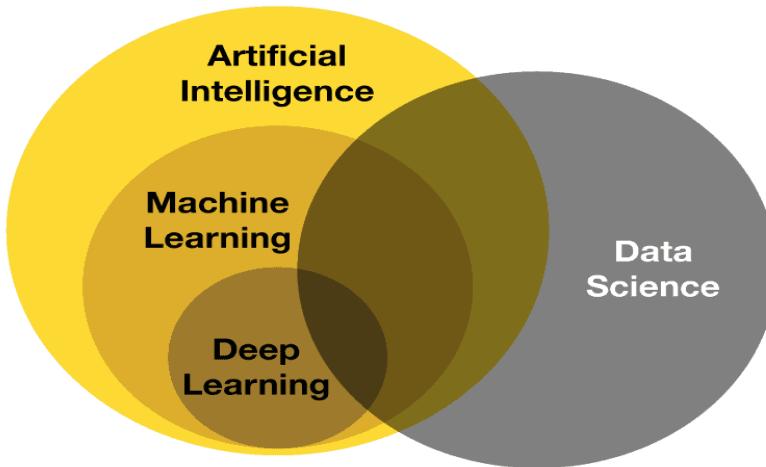
## Data Science во пракса

80% од времето на Data Scientist оди на подготовкa на податоците (чистење, трансформација). ☺

Заклучок: Data Science во реалноста бара многу експериментирање, а не само програмирање.



## 2. Data Science Process



**Вештачката интелигенција (AI)** е гранка што создава системи способни за извршување задачи што обично бараат човечка интелигенција, како што се учење, планирање и разбирање јазици.

Примери:

- Гласовни асистенти – Siri, Alexa, Google Assistant
- Препознавање лица – Користено во паметни телефони и безбедносни системи
- Чатботови и виртуелни асистенти – Поддршка за корисници преку AI системи (ChatGPT, Meta AI)
- Автономни возила – Tesla Autopilot и самоуправувачки автомобили
- Генеративна интелигенција – AI создава слики, текст или музика (DALL·E, GPT-4, DeepDream)

**Машинско учење (Machine Learning)** е гранка од вештачката интелигенција (AI) што користи алгоритми за да научи од податоци и да прави предвидувања без експлицитно програмирање.

Примери:

- Класификација на е-пошта (spam или не)
- Препораки на Netflix и Amazon

**Длабоко учење (Deep Learning)** е подгранка на машинското учење што користи невронски мрежи со повеќе слоеви за обработка на сложени податоци како слики, текст и звук.

Примери:

- Face recognition
- Автоматски превод на јазици

### Клучни задачи во Data Science процесот:

- Чистење и обработка на податоци
- Анализа и моделирање
- Визуелизација и презентација на резултатите

### Категории на податоци:

- Структурирани податоци
- Неструктурирани податоци
- Природен јазик
- Податоци генериирани од машина (Machine Generated)
- Граф базирани податоци
- Аудио, видео и слики
- Стриминг

### Главни чекори во Data Science Process

- 1. Поставување на целта – Дефинирање на проблемот**
- 2. Собирање податоци – Внатрешни или надворешни извори**

Типови на извори:

- Внатрешни: Бази, CRM системи, Excel датотеки (може да се чуваат како databases, data marts, data warehouses, data lakes)
- Надворешни: Open Data (Twitter API, Google Trends, Open data (пр. .org/.gov) )
- Генериирани од машини: Сензори, логови, IoT уреди

- 3. Подготовка на податоци – Отстранување грешки и некомпатибилности**

Вклучува: прочистување, трансформација и комбинирање на податоците.

2 типа на грешки:

- Грешки при интерпретација (пр. возраст поголема од 200год.)

- Неконзистентност (пр. обележување пол со Ж во една табела, а со "Женски" во друга табела)

Чести грешки: грешки при внес, непотребни празни места, невозможни вредности, недостасувачки вредности, outliers.

Аномалии и отстапки (Outliers) - неочекувани вредности што отскокнуваат од мнозинството.

Решенија:

- Отстранување на очигледни грешки
- Истражување дали отстапките имаат некаква важност

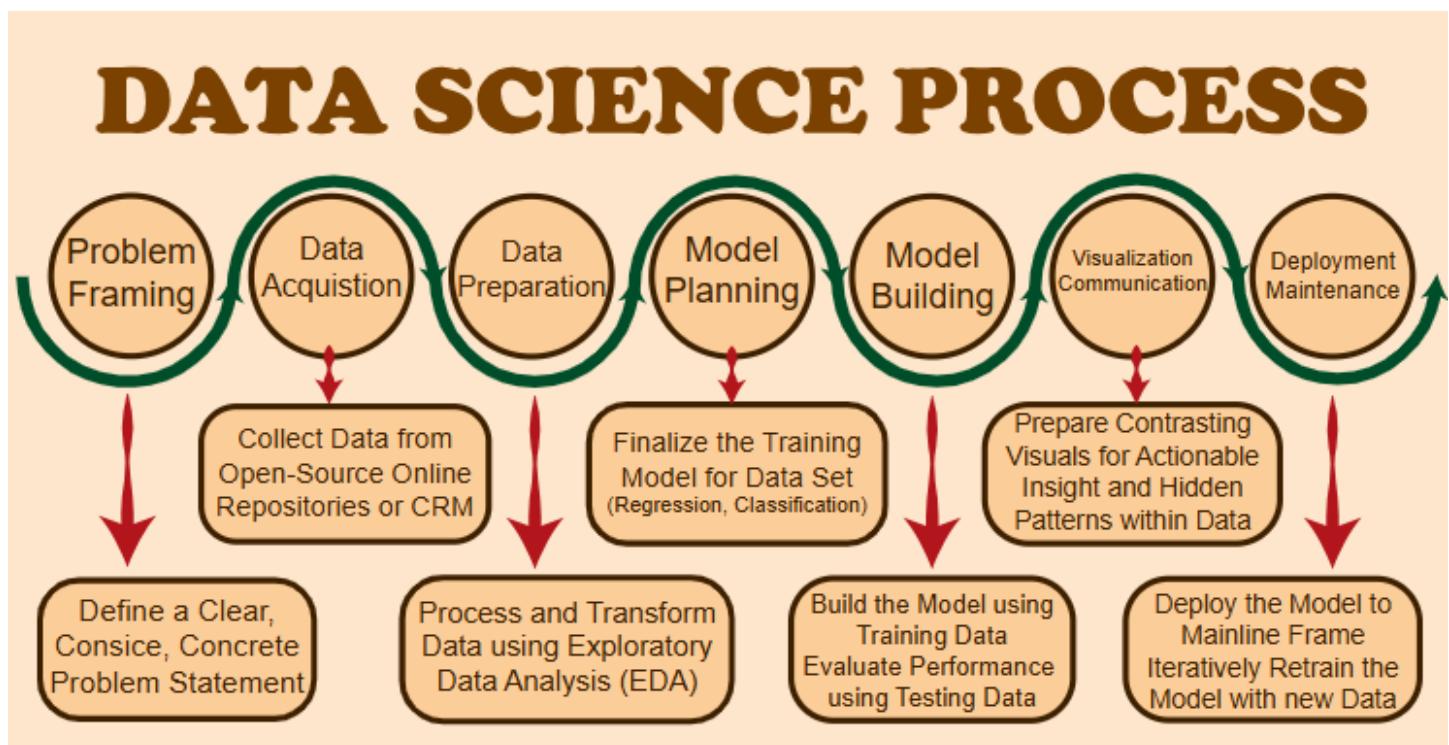
#### 4. Истражување на податоците – Графички приказ, статистики

Истражување на податоците, нивната распределба, корелација итн.

#### 5. Градење модели – избирање модели и променливи кои ќе се користат во моделот, извршување на моделот, евалуација и споредба на модели.

#### 6. Презентација и автоматизација

**Ова е итеративен процес, значи чекорите често се повторуваат!**



### **3. Разбирање на податоците**

“A **datum** is a single measurement of something on a scale that is understandable to both the recorder and the reader. Data are multiple such measurements.”

Заклучок: Сè може да е податок.

#### Извори на податоци и разновидност на податоци

Постојат податоци кои се користат за функционирање на бизнисите, следење на инвентар, интеракција со клиенти, финансиски трансакции итн. Од друга страна постојат податоци кои се јавно достапни, податоци од социјалните медиуми, слики, видеа и многу текстуални информации.

“Експлозијата“ на разновидноста на податоците се случива со развојот на IoT.

Карактеристики на податоците (3V)

**Volume** – Волумен – Колку податоци има?

**Variety** – Разновидност – Колку се различни типовите на податоци?

**Velocity** – Брзина – Со која брзина се генерираат нови податоци?

#### Начини на приирање податоци онлајн

**API (Application Programming Interface)** – користење на множество на функции развиено од компанија за пристап до нивните услуги. Често се плаќа за да се користи. Примери: Google Map API, Facebook API, Twitter API.

**RSS (Rich Site Summary)** – сумаризира често ажурирана онлајн содржина во стандарден формат. Бесплатно за читање доколку го има на сајтот. Пример за користење: кај сајтови поврзани со вести, блогови.

**Web scraping** – користење на софтвер, скрипти или рачно извлекување на податоци од она што е прикажано на страната или што се содржи во HTML – от. Најчесто тие податоци се ставаат во табели.

#### Dark data

Неискористени и скриени податоци кои имаат потенцијал да создадат нови вредности. Тоа се информации кои ги приираат организациите, ги процесираат

и чуваат додека им траат бизнис активностите, но не успеваат да ги искористат за други намени.

### Streaming data

Може да е во било која форма (структурирана/неструктурирана...). Има едно додатно свойство во однос на “обичните” податоци – податоците се појавуваат во системот **кога се случува некој настан**, наместо да се чуваат во складиште.

Процесите треба да се адаптираат на овие податоци.

Примери: “What’s trending” on Twitter, live sporting or music events, stock market.

### Lambda архитектура

Lambda архитектурата е начин на обработка на податоци што е наменет за работа со **огромни количини на податоци**, користејќи и **batch** (групна) и **stream** (во реално време) обработка.

- **Batch обработка** се користи за да се добие **детална и точна слика** од сите податоци што се собрани во некој период.
- **Stream обработка** се користи за да се добијат **брзи резултати** од податоците што пристигнуваат во реално време.

Оваа архитектура се обидува да постигне рамнотежа помеѓу:

- брзина (ниска латентност) – колку брзо добиваме резултати,
- голема обработка (throughput) – колку податоци може да се обработат,
- и отпорност на грешки (fault-tolerance) – да не се изгубат податоци при проблеми.

Со други зборови, Lambda архитектурата овозможува да имаш **и точни резултати од минатото, и брзи реакции на нови податоци**, така што ги комбинира најдобрите карактеристики од двата света.

### Дигитални Близнаци (Digital Twins)

- Дигитален близнак е дигитална копија на некој реален објект или систем.

- Од гледна точка на симулација, концептот на дигитални близнаци е **следниот чекор во развојот на моделирање, симулации и оптимизација**.

Тоа вклучува:

- Нови алгоритми и методи за симулации во реално време на различни физички објекти и процеси.
- Искористување на податоците собрани од дигиталните близнаци за подобрување на деловните и производните процеси, како и подобрување на секојдневниот живот.

Пример: Ако имаш фабричка машина, можеш да направиш нејзина дигитална верзија што во реално време покажува како работи, предвидува дефекти и помага да се донесуваат подобри одлуки.

### Езеро на податоци (Data Lake)

- Data Lake е **огромен складиштен систем за податоци**, базиран на евтини технологии, кој овозможува **собирање, обработка, чување и истражување на сирови податоци** (raw data).
- Може да се справи со големи количини на податоци што пристигнуваат брзо, без разлика дали се структурирани (како табели) или неструктурни (како слики, видеа, текстови).
- **Повеќето податоци во езерото немаат веднаш препознаена вредност**, но се чуваат за подоцна кога може да станат корисни.
- Податоците стануваат **достапни веднаш штом се внесат**, и се користи пристапот "**Schema on Read**", што значи дека **структурата на податоците се чита и дефинира дури кога ќе се користат**, не при внесување.
- За полесно управување со разнородните податоци, Data Lake треба да содржи метаподатоци (информации за податоците) или семантички модел, за да се разбере што точно се чува и како може да се користи.

На кратко, Data Lake е големо „езеро“ каде што се собираат различни видови на податоци, кои подоцна може да се анализираат и искористат кога ќе бидат потребни.

## Типови на податоци

1. Едноставни/атомични:

- Нумерички (integer, float)
- Boolean (binary/ T/F)
- Стингови

2. Датум и време

3. Листи

4. Dictionaries (key value pairs)

5. Квантитативни променливи

- дискретни (пр. број на студенти)
- непрекинати (пр. висина)

6. Категориски променливи

- номинални (не може да се подредат – нема смисла, пр. боја на очи)
- ординални (постои подредување/градација, пр. лошо/добро/мн. добро)

## Складирање на податоците

**Табеларни податоци** – податочно множество во дводимензионална табела каде секоја редица претставува запис, а секоја колона карактеристика/мерка/променлива/атрибут.

Број на атрибути – **димензионалност**.

**Структурирани податоци** – секој запис е во вид на речник (jsno, xml...)

**Полуструктурирани податоци** – не сите записи се претставени со ист сет на клучеви или некои податоци не ја користат клуч:вредност структурата.

## Чести проблеми со податоците

Вредности кои недостасуваат, Погрешни вредности, Неуреден формат на податоците, Неискористливи податоци – не може да одговорат одредено прашање.

## Истражување на податоците: Дескриптивни статистики

**Популација** – целото множество на објекти/настани кои се проучуваат.

**Примерок** – репрезентативно подмножество од објектите/настаните кои се проучуваат. Примерокот е потребен бидејќи не секогаш е можно да се работи со податоците од популацијата – поради големиот број податоци.

Пристрасност кај примероците:

**Selection bias** – некои објекти или записи е повеојатно да бидат избрани

**Volunteer/nonresponse bias** – објектите/записите кои не се лесно достапни не се претставени.

### Просек

Претставува “типична” вредност од примерокот или каде е центарот на дистрибуцијата на податоците.

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{1}{n} \sum_{i=1}^n x_i$$



### Медијана

Исто претставува “типична” вредност од примерокот или каде е центарот на дистрибуцијата на податоците.

$$\text{Median} = \begin{cases} x_{(n+1)/2} & \text{if } n \text{ is odd} \\ \frac{x_{n/2} + x_{(n+1)/2}}{2} & \text{if } n \text{ is even} \end{cases}$$

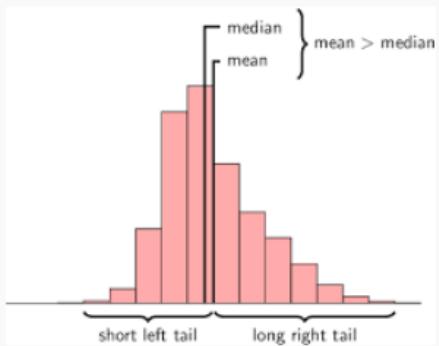
Example (already in order):

Ages: 17, 19, 21, 22, 23, 23, 23, 38

Median =  $(22+23)/2 = 22.5$

## Просекот е чувствителен на outliers.

The mean is sensitive to outliers:



The above distribution is called **right-skewed** since the mean is greater than the median. Note: **skewness** often “follows the longer tail”.

## Мода

Начин да се најде “најрепрезентативна“ вредност – вредност која се појавува најголем број пати.

## Ранг/опсег

Начин да се измери колку се “распрнати“ вредностите во примерокот. Покажува колкава е разликата помеѓу најекстремните вредности.

$$\text{Range} = \text{Maximum Value} - \text{Minimum Value}$$

## Варијанса

Мери колку во просек вредностите на примерокот отстапуваат од средната вредност.

$$s^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

## Стандардна девијација

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

## Нормални распределби, Средна вредност, Варијанса

- Средната вредност на едно множество од вредности е просекот, односно збирот на сите вредности поделен со нивниот број.
- Варијансата е мерка за тоа колку се „широко“ распределени вредностите околу средната вредност. Поточно, варијансата е просечната квадрирана оддалеченост на точките од средната вредност.
- Стандардната девијација е коренот од варијансата, и исто така мери колку вредностите отстапуваат од просекот.
- **Нормалната распределба (normal distribution)** е целосно дефинирана со нејзината средна вредност и варијанса.

## Користење на точни статистики и графови

**За континуирани променливи со нормална распределба (пример: висина, температура):**

- Прикажи:  
Број на примероци (N), просек (mean), стандардна девијација, минимум, максимум
- Користи графици:  
Хистограми, точкести графици (dot plots), box плотови, scatter плотови

**За континуирани променливи со асиметрична (skewed) распределба (пример: плата, време на чекање):**

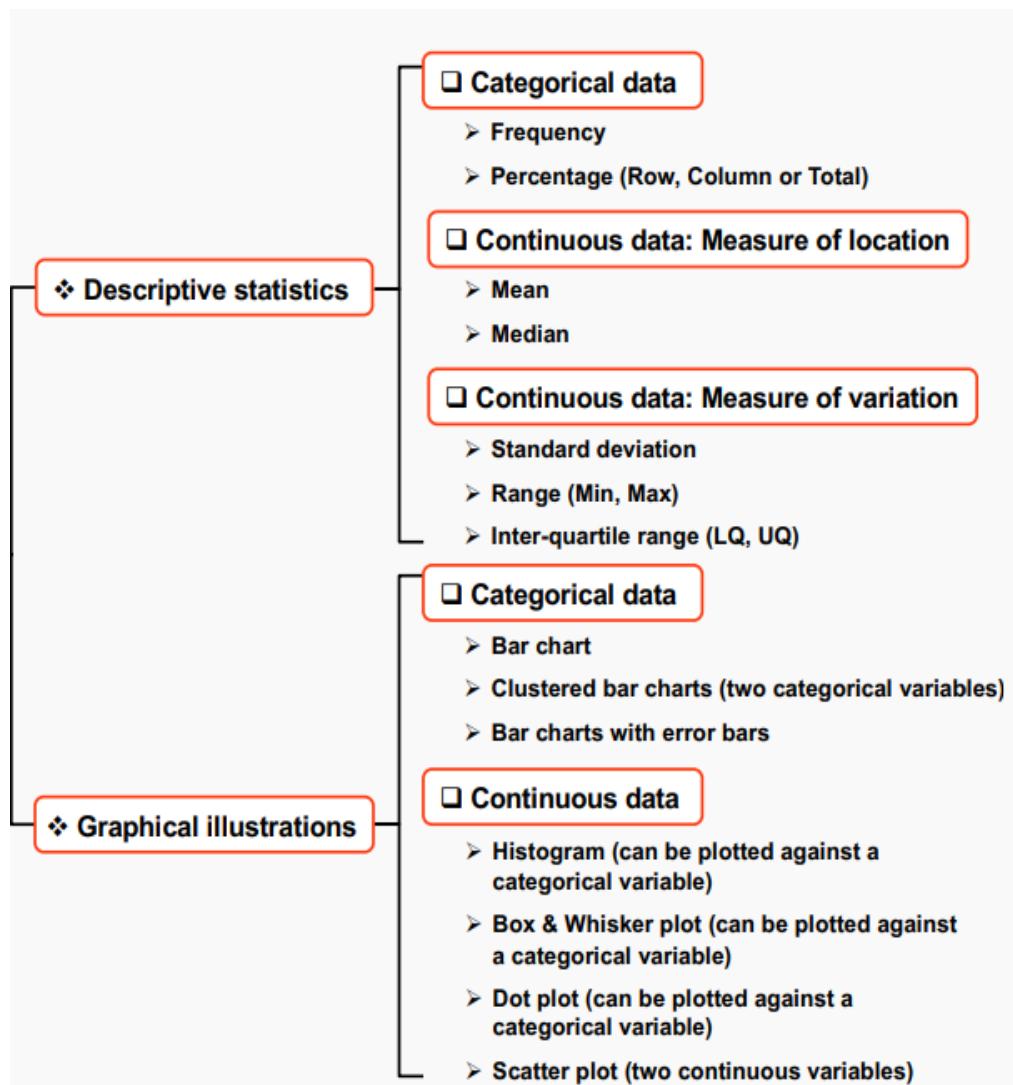
- Прикажи:  
Број на примероци (N), медијана, долен квартил, горен квартил, минимум, максимум, геометриска средина
- Користи графици:  
Хистограми, точкести графици, box плотови, scatter плотови

**За категориски променливи (пример: пол, боја, тип на производ):**

- Прикажи:  
Бројење на фреквенции, проценти

- Користи графици и табели:

Еднодимензионални и дводимензионални табели (one-way и two-way tables), бар графици



## Визуелизации

Визуелизациите ни помагаат да ги анализираме и истражиме податоците. Тие помагаат да:

- Се откријат скриени шаблони и трендови
- Се формулираат или тестираат хипотези
- Се комуницираат резултатите од моделирање
- Се одреди кој е следниот чекор во анализата или моделирањето

## Видови на визуелизации

Кога правиш визуелизација, прво прашај се: **Што сакаш да покажеш за твоите податоци?**

### **Дистрибуција (Distribution)**

Покажува **како се распределува една променлива** низ можните вредности.

Пример: Хистограм, box plot

### **Однос (Relationship)**

Покажува **каков е односот помеѓу две или повеќе променливи.**

Пример: Scatter plot, bubble chart

### **Состав (Composition)**

Покажува **како се дели податочниот сет на подгрупи или категории.**

Пример: Pie chart, stacked bar chart

### **Споредба (Comparison)**

Покажува **како се споредуваат вредности или трендови помеѓу различни променливи или групи.**

Пример: Линиски график (line chart), групирани барови

## Дополнително објаснување на графиците:

### **Histogram**

Се користи за прикажување на распределбата на една континуирана променлива. Ги групира вредностите во интервали (бинови) и покажува колку податоци спаѓаат во секој интервал. Корисно за откривање на обликот на распределбата (нормална, асиметрична итн.).

### **Pie chart**

Се користи за прикажување на состав (композиција), односно како целината е поделена на делови. Секој дел од „питата“ претставува **процент или удел на некоја категорија** во вкупниот број.

### **Scatter plot**

Се користи за прикажување на односот помеѓу две континуирани променливи. Секој податочен пар е точка на графикот. Овој график помага да се откријат **корелации, трендови или групирања.**

### **Stacked area graph**

Се користи за прикажување на составот на податоци со текот на времето, каде

што вкупната сума и нејзините делови се важни. Областа е поделена по категории, и се гледа како секоја категорија придонесува кон вкупната сума **низ времето**.

### Multiple histograms

Се користат за споредба на распределби на различни групи или категории. Прикажуваат повеќе хистограми една до друга или преклопени, за да се спореди **како се разликуваат податоците помеѓу групите**.

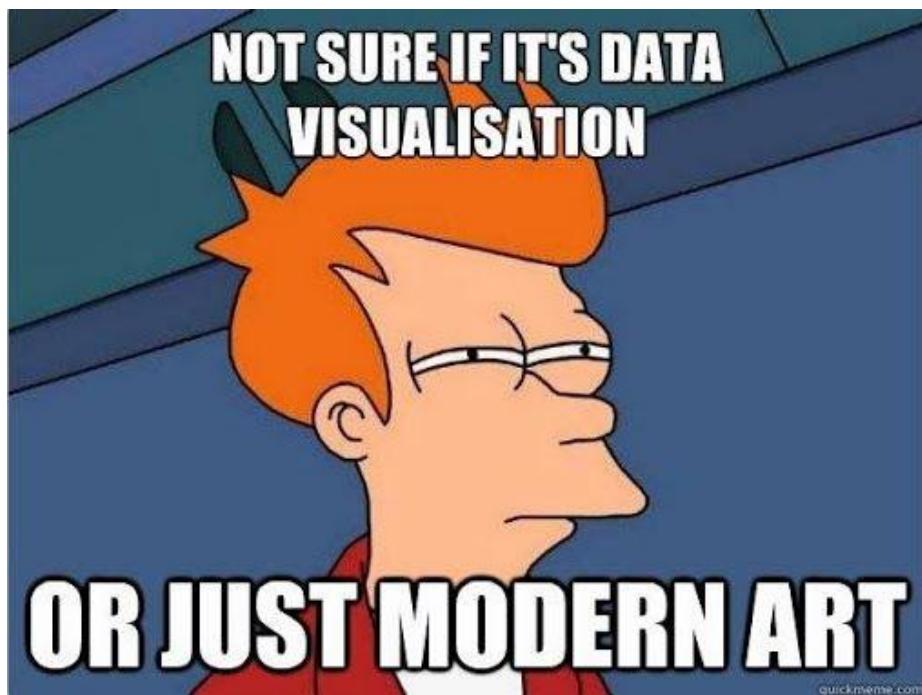
### Boxplot

Се користи за сумирање на распределбата на податоци преку пет броеви: минимум, долен квартил, медијана, горен квартил и максимум. Исто така ги покажува потенцијалните **отстапки** (outliers). Корисен е за споредба на распределби помеѓу различни групи.

### Проблеми со визуелизација

Комплексни податоци, голема димензионалност, категориски променливи.

Пример: при голема димензионалност тешко и бескорисно е да се направи scatter plot, би било полесно да се направат повеќе plots од помала димензионалност.



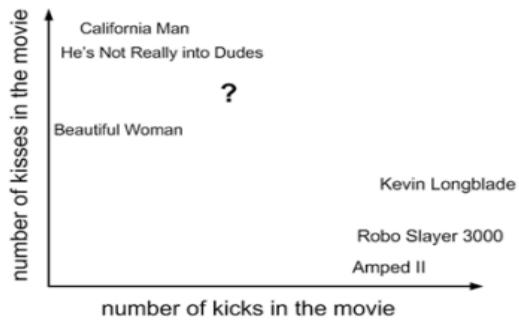
## 4. Подготовка на податоците

Прочистување, интегрирање и трансформирање податоци.

Овој чекор е важен за добар перформанс на моделите – **garbage-in equals garbage-out**.

### Едноставен начин на предвидување: КНН – К најблиски соседи

Movie title	# of kicks	# of kisses	Type of movie
California Man	3	104	Romance
He's Not Really into Dudes	2	100	Romance
Beautiful Woman	1	81	Romance
Kevin Longblade	101	10	Action
Robo Slayer 3000	99	5	Action
Amped II	98	2	Action
?	18	90	Unknown

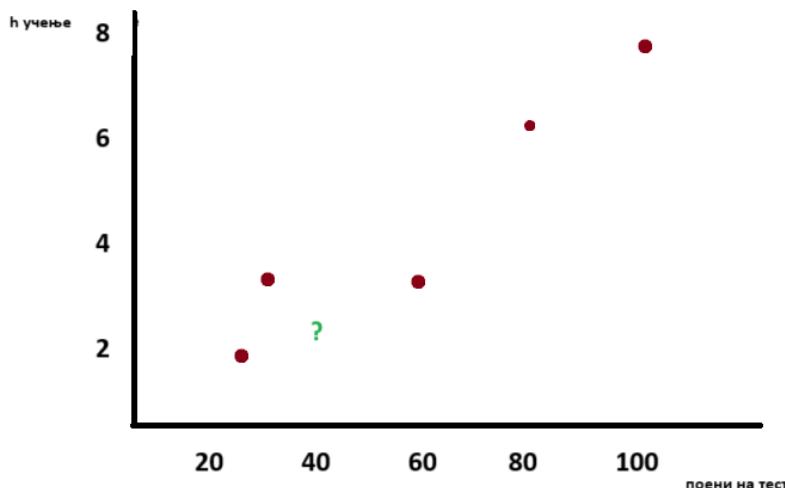


**КНН** е алгоритам кој прави предвидувања врз основа на сличност. Тој функционира така:

1. Се избира бројот на најблиски точки (K) што ќе ги разгледуваме.
2. Се пресметува колку е блиска новата точка до сите други (на пример со Евклидово растојание).
3. Се гледаат K-те најблиски точки и нивните класи.
4. Се предвидува класата на новата точка според мнозинството (или просекот ако е регресија).

\*\* Во примерот ? е филмот за кој треба да го најдеме жанрот, за K=3 негови соседи се “He's Not Really into Dudes”, “Beautiful Woman” и “California Man”. Сите 3 филмови се романси па и новиот филм може да се класифицира како романса.

Пример со регресија:



Часови учење	Поени на тест
2	28
3,7	35
3,8	60
6	80
8	100
?	40

$$K = 3$$

$$2 + 3,7 + 3,8 = 9,5 / 3 = 3,167 \text{ часа}$$

### Прочистување на податоците

Потребно е да се направи прочистување на податоците од грешки при внес, непотребни празни места, невозможни вредности, вредности кои недостасуваат и аутлаери.

Често се случуваат интерпретациски грешки – пр. возраст 150год. и грешки во конзистентноста – пр. М/Ж во една табела за пол, а машко/женско во друга табела.

Исто така треба да се внимава да се користат исти единици мерки.

### Вредности кои недостасуваат

Не значи дека навистина недостасуваат (пр. на прашање за годишен приход на населението полето ќе остане празно кај децата), но треба да се справиме со нив бидејќи голем број од ML техниките не можат да се справат со нив.

**Missingness Indicator Variable** – треба да се користи бидејќи рупата на индивидуи со недостасувачка вредност може систематски да се разликува од оние кај кои таа вредност е измерена. Ако ги третираме како да се исти, може да дојде до пристрасност при квантитативна анализа на релациите и до послаби резултати при предвидување.

На пример на прашање “Дали сте користеле опоиди?”, дел од луѓето може да не одговорат – Дали тоа значи дека користеле опоиди? Односно дали тоа што не одговориле ни носи некоја дополнителна информација. Дали треба да ги третираме исто како оние кои не користеле?

Всушност овој пристап креира трета група (колона)– “не одговорил“.

### Наједноставни начини за справување со вредности кои недостасуваат

Бришење на редици (записи) или колони (атрибути) во кои има многу вредности кои недостасуваат.

Пополнување на вредностите со просек или медијана (за квантитативни променливи) или со мода (за категориски променливи).

### Видови на вредности кои недостасуваат

1. **Missing Completely at Random (MCAR)** – веројатноста некоја вредност да недостасува е иста за сите записи. Како да правиме дупки во податочното множество. **Ова е “најдобар случај“ и најдобро е да се импутираат вредностите.**
2. **Missing at Random (MAR)** – веројатноста вредноста да недостасува зависи само од веќе достапни информации (од други променливи во податоците).
3. **Missing Not at Random (MNAR)** – веројатноста вредноста да недостасува зависи од информации кои не се снимени, а тие информации исто така би можеле да ги објаснат недостасувачките вредности.

### **Примери:**

#### **MCAR (Missing Completely At Random)**

При внесување на анкета, некои листови случајно се оштетиле и се изгубиле дел од податоците. Недостасувањето нема врска со никаква карактеристика на испитаникот.

#### **MAR (Missing At Random)**

Постарите испитаници почесто прескокнуваат прашања за користење на технологија, но нивната возраст е запишана. Недостасувањето зависи од други познати податоци (возраст).

## MNAR (Missing Not At Random)

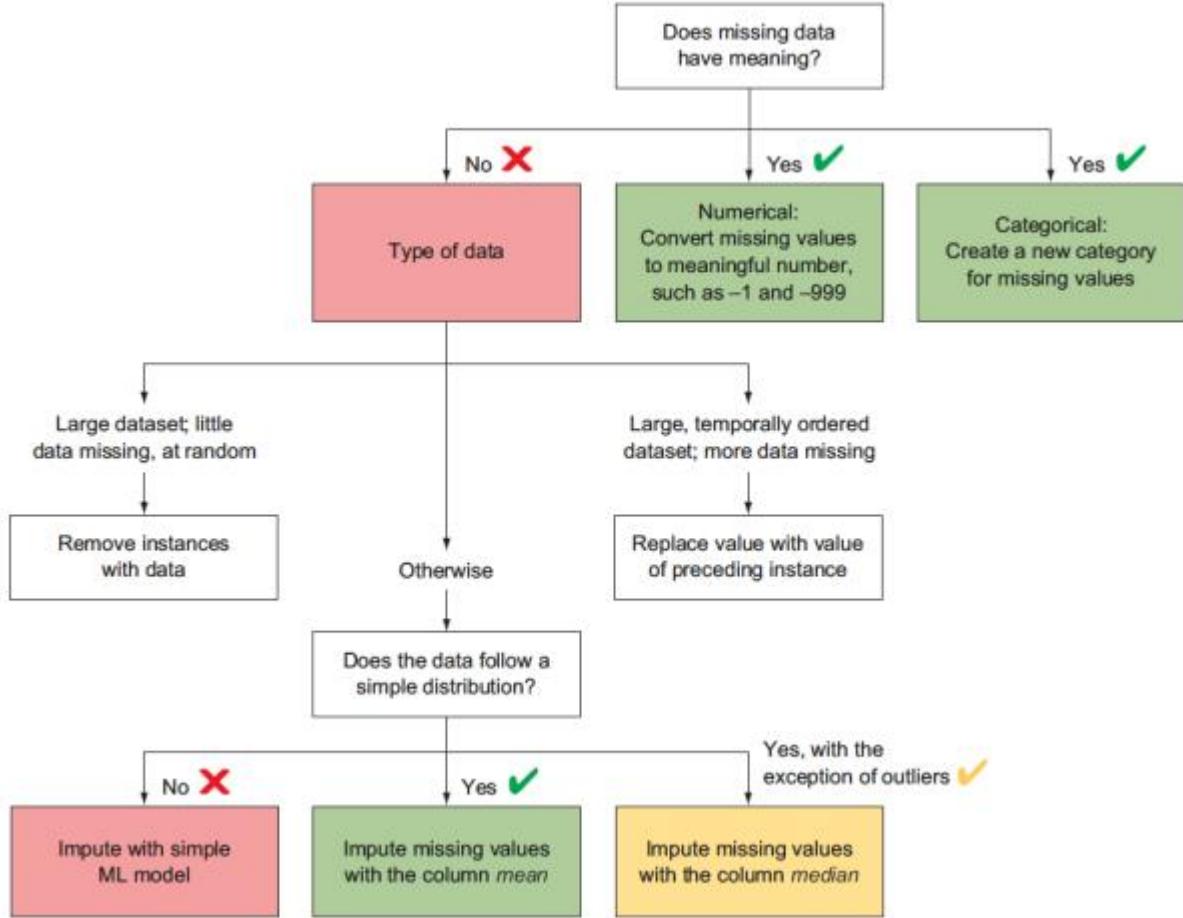
Луѓето со висока плата не сакаат да ја наведат во анкетата, а податокот за платата недостасува. Недостасувањето зависи од самата недостасувачка вредност (платата), која не ја знаеме.

### Методи на импутација

1. Импутација со просек/медијана за квантитативни вредности и со мода за категориски вредности.
2. Креирање нова променлива која е индикатор на недостаток и нејзино користење во модел за предвидување на одговорот.
3. Hot deck imputation – за секоја вредност која недостасува се избира случајна вредност од нејзината колона со која ќе се пополни.
4. Моделирање на импутирањето – пополнување со предвидени вредности ( $\hat{y}$ ).
5. Моделирање на импутирањето со неизвесност – пополнување со предвидени вредности + грешка ( $\hat{y} + \epsilon$ ).

## Dealing With Missing Values

Technique	Advantage	Disadvantage
Omit the values	Easy to perform	You lose the information from an observation
Set value to null	Easy to perform	Not every modeling technique and/or implementation can handle null values
Impute a static value such as 0 or the mean	Easy to perform You don't lose information from the other variables in the observation	Can lead to false estimations from a model
Impute a value from an estimated or theoretical distribution	Does not disturb the model as much	Harder to execute You make data assumptions
Modeling the value (nondependent)	Does not disturb the model too much	Can lead to too much confidence in the model Can artificially raise dependence among the variables Harder to execute You make data assumptions



## Аутлаери

Аутлаер е дел од примерокот кој отстапува од останатите податоци. Најлесно се воочуваат со box plot.

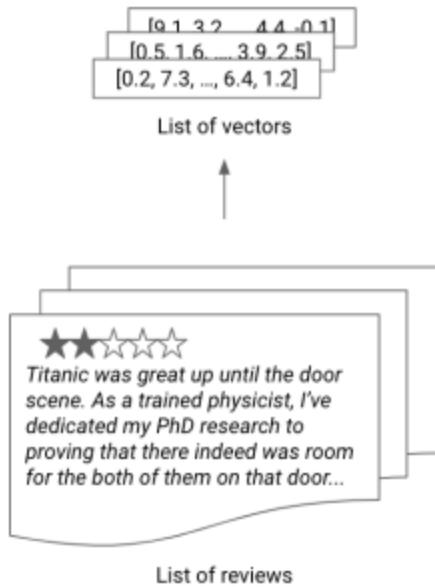
Аутлаерот не мора секогаш да значи дека вредноста е погрешна. На пример доколку се следат податоци од кредитни картички – аутлаерот би значел отстапување т.е можеби измама и во тој случај е важен.

## Справување со аутлаери

Отстранување, трансформација (пр. логаритам – 2,3,4,5,6,10,100 -> 0.3, 0.47, 0.6, 0.69, 1, 2 ), Truncation – скратување (пр. 2,3,4,5,6,10,100 -> 2,3,4,5,6,10,10).

## Векторизирање

Процес на претворање на сирови податоци во вектор кои ги претставува.



## Справување со категориски податоци

Категориските променливи можат да земат само ограничен и обично фиксен број на можни вредности.

На пример, ако податочниот сет содржи информации за корисници, честопати ќе има променливи како држава, пол, старосна група итн.

Овие променливи често се складирани како текстуални вредности кои ги опишуваат карактеристиките на набљудувањата. На пример, полот е описан како машки (M) или женски (F).

Многу модели во машинско учење се алгебарски.

Тоа значи дека нивниот влез **мора да биде нумерички**. За да ги користиме овие модели, категориите прво мора да се трансформираат во броеви, пред да се примени алгоритамот.

Иако некои ML библиотеки автоматски ги претвораат категоријалните податоци во бројки (преку вградени методи), многу други не го поддржуваат тоа и бараат рачна обработка.

## Енкодирање на категориски променливи

### **Label Encoding**

Секоја категорија добива уникатен број.

Пример: Male = 0, Female = 1

## One-Hot Encoding

За секоја категорија се креира посебна бинарна колона.

Пример: Gender\_Male = 1, Gender\_Female = 0

Возраст	Пол	Пол_машко	Пол_женско
25	М	1	0
28	Ж	0	1

## Binary Encoding

Категориите се претвораат прво во бројки (како кај label encoding), потоа тие бројки во бинарен формат.

Корисно кога има многу категории.

## Proxy Encoding

Се користи надворешна бројчана претстава која е поврзана со категоријата.

Пример: наместо држава како текст, се користи просечниот БДП на таа држава или слична метрика зависи која е целта на истражувањето.

## Векторизирање на текстуални податоци

Наједноставниот начин за векторизирање на текст е со вектор на бројач (**count vector**), што е еквивалент на one-hot encoding, но за зборови.

Се започнува со креирање на вокабулар – листа на уникатни зборови од целиот податочен сет.

На секој збор од вокабуларот му се доделува индекс (од 0 до големината на вокабуларот).

Потоа, секое реченица или пасус се претставува со листа долгa колку и вокабуларот.

За секоја реченица, бројот на секој индекс во листата претставува колку пати се појавува тој збор во дадената реченица.

Овој метод го игнорира редоследот на зборовите во реченицата, па затоа се нарекува „торба со зборови“ (**bag of words**).

		Input text										
Sentence 1		"Mary is hungry for apples."										
...		...										
Sentence 345		"John is happy he is not hungry for apples."										



Word index	MARY	IS	HUNGRY	HAPPY	FOR	...	APPLES	NOT	JOHN	HE	SAND
Sentence 1	1	1	1	0	1	...	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
Sentence 345	0	2	1	1	1	...	1	1	1	1	0

## Слики

Сликите веќае се векторизирани, бидејќи сликата не претставува ништо друго освен многудимензионалена низа од броеви.

Повеќето стандардни RGB слики со три канали се складираат како листа од броеви, чија должина е еднаква на:

висина × ширина × 3

(тројката доаѓа од црвен (red), зелен (green) и син (blue) канал).

**Пример: Слика од 100x200 пиксели има  $100 \times 200 \times 3 = 60,000$  броеви, каде секој број претставува интензитет на боја.**

## Стандардизација и нормализација на вредности на податоци

Некои алгоритми во машинско учење бараат нормализирање на податоците, што значи дека секој поединечен атрибут се трансформира така што ќе биде на иста нумеричка скала.

Овој чекор е многу важен кога работиме со параметри кои имаат различни мерни единици и размери.

На пример, некои техники (како Евклидово растојание) користат растојание меѓу вредности, па затоа сите параметри треба да бидат на иста скала за да може фер да се споредуваат.

**Две важни трансформации:**

### **1. Нормализација (Normalization)**

Сите нумерички променливи се трансформираат во опсег [0, 1].

Формула:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

### **2. Стандардизација (Standardization)**

Податоците се трансформираат да имаат средна вредност (mean) = 0 и стандардна девијација = 1.

Формула:

$$x_{new} = \frac{x - \mu}{\sigma}$$

## Намалување на бројот на променливи (Dimensionality Reduction)

Понекогаш имаме преголем број променливи (атрибути) и е потребно да ги намалиме, затоа што не додаваат нова или корисна информација за моделот.

**Прекумерниот број на променливи:**

- Го прави моделот потежок за обработка,
- Исто така, некои алгоритми не функционираат добро кога имаат премногу влезни податоци.

На пример, техниките базирани на Евклидова оддалеченост (како KNN) функционираат добро само до околу 10 променливи.

Постојат специјални методи во машинското учење кои овозможуваат намалување на бројот на променливи, но истовремено задржуваат што е можно повеќе од информацијата во податоците.

## Curse of Dimensionality

Проблемот е што кога се зголемува димензионалноста (брот на променливи), волуменот на просторот расте многу брзо, па достапните податоци стануваат разредени (**sparse**).

Оваа разреденост претставува проблем за методи што бараат статистичка значајност, бидејќи нема доволно информации во секој дел од просторот.

Квадратното растојание меѓу случајни точки и некоја избрана точка, со висока веројатност, ќе биде близку до просечното (или медијанското) квадратно растојание.

Со други зборови, разликите меѓу точките стануваат сè помалку значајни во повеќедимензионални простори.

Во такви случаи, се користат други мерки на сличност, како што е косинусна сличност (cosine similarity), кои се подобро прилагодени за високи димензии.

## Косинусна сличност (Cosine Similarity)

- Косинусната сличност претставува косинусот на аголот помеѓу два n-димензионални вектори во n-димензионален простор.
- Таа се пресметува како скаларен производ (dot product) на двата вектори, поделен со производот од нивните должини (или магнitudи):

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

каде што:

- $\mathbf{A} \cdot \mathbf{B}$  е **скаларниот производ** (dot product),
- $\|\mathbf{A}\|$  и  $\|\mathbf{B}\|$  се **должините (нормите)** на векторите  $\mathbf{A}$  и  $\mathbf{B}$ .

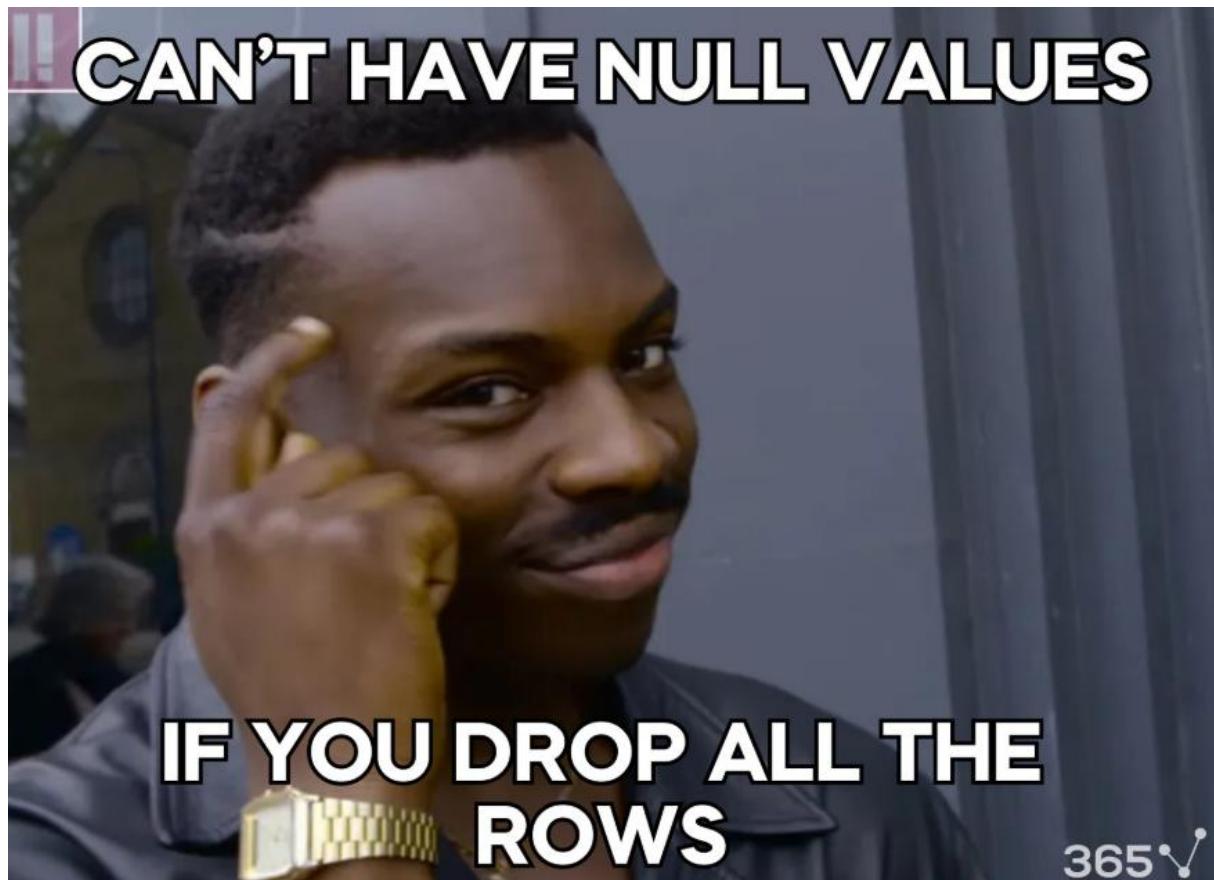
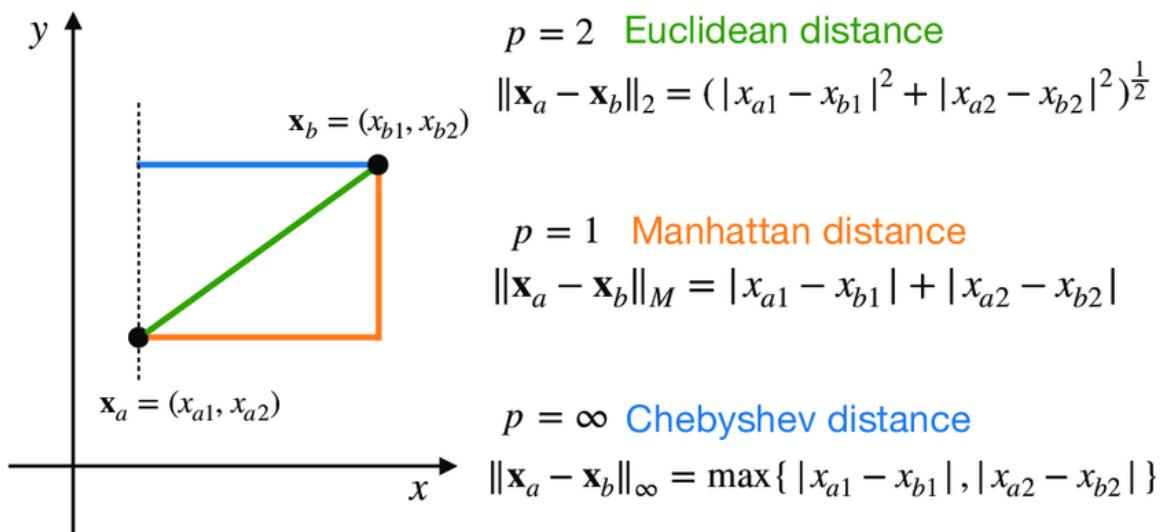
Што значи вредноста:

1 → векторите се идентични по насока (многу слични)

0 → векторите се ортогонални (немаат сличност)

-1 → векторите се спротивни по насока (целосна разлика)

\*\* Дополнително: Видови растојанија



# Machine Learning 1

## Што е машинско учење (Machine Learning)?

Машинското учење е гранка на вештачката интелигенција која се занимава со дизајн и развој на алгоритми што им овозможуваат на компјутерите да развиваат однесување базирано на емпириски податоци.

Бидејќи интелигенцијата бара знаење, потребно е компјутерите да стекнуваат знаење преку податоци.

### **Формална дефиниција:**

„Се вели дека компјутерска програма учи од искуство **E** со оглед на некоја класа на задачи **T** и мерка за перформанса **P**, ако нејзиниот перформанс на задачи од **T**, мерен преку **P**, **се подобрува со искуството E**.“

**Човечкото учење е сè уште многу пософистицирано** од дури и најнапредните ML алгоритми, но компјутерите имаат предност во тоа што можат **побрзо и пообемно да меморираат, повикуваат и обработуваат податоци**.

## Видови учење

### **Надгледувано учење (Supervised Learning)**

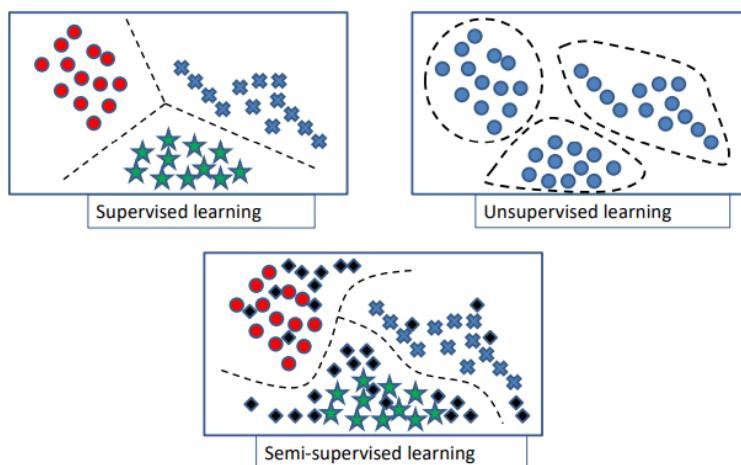
- Постојат лабели (labels)
- Тренинг-податоците ги вклучуваат посакуваните излези
- Цел: предвидување на иднината
- Учиме од познати минати примери за да предвидиме што ќе се случи понатаму

### **Ненадгледувано учење (Unsupervised Learning)**

- Нема лабели (labels)
- Тренинг-податоците не ги вклучуваат посакуваните излези
- Цел: да се разбере структурата на податоците
- Анализа и групирање (кластеризација), намалување на димензионалност, откривање скриени обрасци
- Учиме за минатото и структурата на податоците

## Учење со повторување (Reinforcement Learning)

- Учење преку награди и/или казни
- Се базира на секвенца на дејства
- Цел: да се научи која постапка носи најголема корист
- Пример: Обучување на агент (како робот или играч во игра) да донесува одлуки врз основа на повратна информација (reward)



## Што може да се прави со ML:

Класификација (која категорија), Регресија (колку), Кластерирање (која група), Аномалија (дали нешто е необично) и Учење со повторување (која акција).

## Секој ML алгоритам има 3 компоненти:

Репрезентација, Евалуација и Оптимизација.

### Репрезентација

Во машинското учење, **начинот на кој се претставуваат податоците и моделите** игра клучна улога во тоа како алгоритмот ќе учи. Некои од најчестите форми на претставување се:

### **Дрва на одлука (Decision Trees)**

Јасна, разбиралива структура заснована на прашања и гранки.

### **Сетови на правила / Логички програми (Sets of Rules / Logic Programs)**

Ако-тогаш (if – then) правила за донесување одлуки.

## **Инстанци (Instances)**

Претставување преку примери; се користи кај алгоритми како KNN.

## **Графички модели (Bayes / Markov мрежи)**

Претставуваат веројатносни врски помеѓу варијабли.

## **Невронски мрежи (Neural Networks)**

Инспирирани од човечкиот мозок, се користат во deep learning.

## **Машини со поддржувачки вектори (Support Vector Machines - SVM)**

Геометриски пристап за класификација и регресија.

## **Ансамбл модели (Model Ensembles)**

Комбинирање на повеќе модели за подобра прецизност (на пр. Random Forest, Boosting).

И други...

## **Евалуација**

Во машинското учење, евалуацијата се однесува на начинот на кој ја мериме **успешноста на моделот**. Се користат различни метрики во зависност од типот на задачата (класификација, регресија и сл.).

Некои од најчестите мерки за евалуација се:

### **Точност (Accuracy)**

### **Прецизност и одзив (Precision and Recall)**

### **Квадратична грешка (Squared Error)**

### **Веројатност (Likelihood)**

### **Постериорна веројатност (Posterior Probability)**

### **Цена / Корисност (Cost / Utility)**

### **Маргина (Margin)**

### **Ентропија (Entropy)**

### **Кулбек-Лајблер дивергенција (Kullback-Leibler Divergence, K-L divergence)**

И други...

## Оптимизација

Во машинското учење, оптимизацијата е процес на пронаоѓање на најдобрите вредности за параметрите на моделот со цел да се минимизира (или максимизира) некоја функција на грешка или целна функција.

### **Видови оптимизација:**

Комбинаторна оптимизација (Combinatorial optimization)

Конвексна оптимизација (Convex optimization)

Оптимизација со ограничувања (Constrained optimization)

## Користење на податоци за донесување одлуки

### **Традиционален пристап:**

Човек рачно ги анализира апликациите за кредит, користејќи правила и интуиција.

### **Пристав со машинско учење:**

Машината учи оптимални одлуки директно од податоците, без потреба од однапред дефинирани правила.

Овој пристап:

- Е поточен,
- Се подобрува со повеќе податоци,
- Ги користи карактеристиките (features) како кредитен рејтинг, пол, занимање и сл.

### **Пример: Логистичка регресија за одобрување на заем**

- Еден од наједноставните модели за класификација.
- Ги користи влезните карактеристики (на пр. кредитен лимит, образование, возраст) за да предвиди веројатност за враќање на заем.
- Коефициентите во моделот се учат автоматски од тренинг податоци.

**Клучна поента:** Машинското учење автоматизира и подобрува процеси на донесување одлуки, особено кога има многу податоци.

## Линеарни vs. нелинеарни ML алгоритми

Кога врската меѓу влезот и излезот е сложена, едноставни модели како логистичка регресија не се доволни. Тогаш користиме покомплексни модели.

### Предвидување на променлива

Сакаме да предвидиме **една променлива** со помош на **други**. Примери: Прегледи на YouTube видео според должина, датум, претходни прегледи, Оценки за филм на Netflix според претходни оцени и демографија.

The **Advertising data set** consists of the sales of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper. Everything is given in units of \$1000.

TV	radio	newspaper	sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

**Одговорна (response) променлива (Y):** тоа е она што го предвидуваме – излез.  
Пр: продажба

**Објаснувачки променливи (предиктори) (X):** тоа се податоци што ги користиме за да го предвидиме Y

Пр: буџет за ТВ, радио, весник

### Вистински модел vs. статистички модел

- Се претпоставува дека постои **непозната функција f(X)** што го поврзува Y со X.  
Формула:  $Y=f(X)+\varepsilon$

каде  $\varepsilon$  е **случаен шум или грешка**, што не може да се објасни преку X.

- Статистички модел е **приближување на f(X)**.

## Што прави моделот?

- Целта на моделот е да **пронајде врска меѓу X и Y**.
- Едноставна идеја: ако имаш вредности на X и неколку Y вредности, **можеш да пресметаш просек на Y** за да предвидиш.

Пример: Ако за некоја вредност на X имаш 3 вредности на Y: 10, 12, 13 тогаш предвидениот Y би бил  $(10+12+13)/3 = 11.67$

## Предвидување vs. проценка

- **Inference (проценка)**: важна е функцијата  $f$ , сакаме да ја разбереме.
- **Prediction (предвидување)**: не нè интересира точното  $f$ , туку **точноста на предвидениот излез**.

$$\hat{y}_i = \hat{f}(x_{i,1}, \dots, x_{i,p})$$

Пр: Дали ќе се случи? – не е важно зошто, туку дали можеш да го погодиш.

Пример: Може да предвидиме Y како просек на Y вредностите на K соседите. Може да испробаме повеќе модели со различно K и да видиме кој дава најдобар резултат. Гледаме кој модел дава помала грешка со пресметување на MSE или RMSE.

## Евалуација на грешка (Error Evaluation)

Ги делиме податоците на **тренинг сет и тест сет**.

Моделот се тренира на дел од податоците (тренинг), а се тестира на друг дел (тест) за да видиме **колку добро предвидува непознати вредности**.

## Мерење на грешка

Користиме функција на грешка (loss function).

Најчеста е **Mean Squared Error (MSE)**:

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

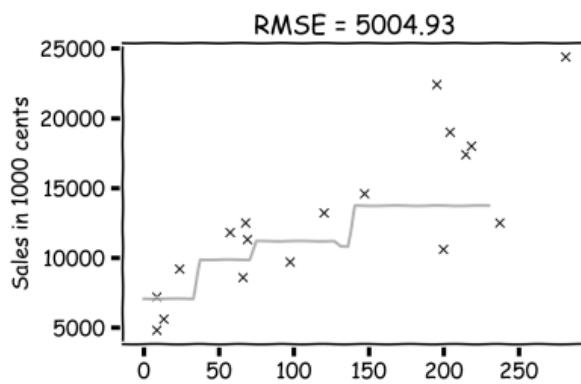
y<sub>i</sub> – вистинска вредност  
y<sup>h</sup><sub>i</sub> – предвидена вредност

$y_i - \hat{y}_i$  се нарекува **резидуал** и ја мери грешката за и-тото предвидување.

Исто така се користи и е **Root of the Mean of the Squared Errors (RMSE)**:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

## Model fitness



Се прашуваме: „Дали RMSE = 5.0 е доволно добро?“

Ако продажбите се мери во долари – да.

Ако се мери во центи – RMSE ќе изгледа како 5000 → но тоа не значи дека моделот е полош.

Затоа, важно е **да го споредиме RMSE со нешто значајно**, како што е просекот.

## R<sup>2</sup> (коефициент на определеност)

Покажува колку добро моделот ја објаснува варијабилноста на податоците.

$$R^2 = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_i (\bar{y} - y_i)^2}$$

If our model is as good as the mean value,  $\bar{y}$ , then  $R^2 = 0$

If our model is perfect then  $R^2 = 1$

$R^2$  can be negative if the model is worse than the average. This can happen when we evaluate the model in the test set.

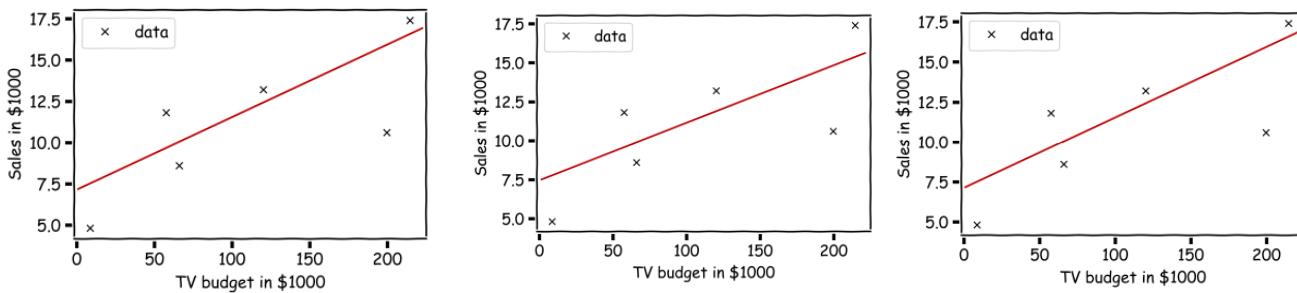
## Линеарни модели

Наместо да пресметуваме просек како во KNN, **претпоставуваме дека постои линеарна врска:**

$$Y = \beta_1 X + \beta_0 + \epsilon$$

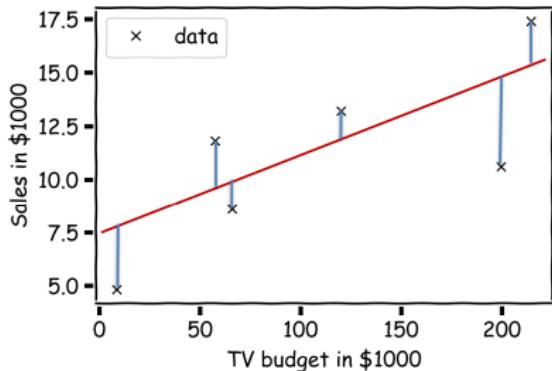
- $\beta_1$ : ефект на  $X$  врз  $Y$  (slope)
- $\beta_0$ : пресек на правата (intercept)

## Која права е најдобра?



**Одговор:** Оваа која има најмала вкупна грешка.

Се пресметуваат резидуалите:



## Како се избираат $\beta_0$ и $\beta_1$ ?

Се користи **функција на загуба (loss function)** – тута:

$$L(\beta_0, \beta_1) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n [y_i - (\beta_1 X + \beta_0)]^2.$$

Целта е да се пронајдат вредности на  $\beta_0$  и  $\beta_1$  што ќе ја минимизираат оваа грешка.

### „Brute force“ пристап

Пробување многу можни вредности за  $\beta_0$  и  $\beta_1$ , пресемтување на грешката за секоја, и **избирање на оние вредности со најмала загуба**.

### Точна формула за коефициентите

Без да пробуваме сите можности, можеме математички да ги изведеме најдобрите вредности:

$$\hat{\beta}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Оваа линија е **регресиона линија**, и го претставува најдоброто линеарно приближување на податоците.

$$\hat{Y} = \hat{\beta}_1 X + \hat{\beta}_0$$

### Проценка на регресионите коефициенти: Градиентски спуст (Gradient Descent)

Еден пофлексибilen метод е:

1. Почнуваш од некоја случајна почетна точка.
2. Определуваш во која насока да се движиш за да го намалиш загубувањето (лево или десно).
3. Ја пресметуваш наклоноста (slope) на функцијата во таа точка и:
  - чекориш надесно ако наклоноста е негативна,
  - чекориш налево ако наклоноста е позитивна.
4. Се враќаш на чекор #1.

- Знаеме дека треба да се движиме во **спротивна насока од дериватот** (наклоноста), и треба да направиме чекор **пропорционален на вредноста на дериватот**.
- Правење чекор значи:

$$w^{t+1} = w^t + \alpha * d$$

(ова е општа форма)

- Насока спротивна на дериватот и пропорционална на него значи:

$$w^{t+1} = w^t - \alpha * (\partial L / \partial w)$$

( $L$  е функцијата за загуба, а  $\partial L / \partial w$  е дериватот во однос на  $w$ )

**Превод во постандардна нотација:**

$$w(t+1) = w(t) - \alpha * (\partial L / \partial w)$$

- $\alpha$  (алфа) е **learning rate** – брзина на учење, т.е. колкав чекор правиме секојпат.
- $w = [w_0, w_1]$  – овде векторот  $w$  ги претставува регресионите коефициенти (на пример, пресеци и наклон во линеарна регресија).

**Краток преглед:**

- **Gradient Descent** е алгоритам за оптимизација од прв ред, што значи дека користи први изводи (деривати).
- Тоа е **итеративен метод**, се повторува додека не се достигне минимум.
- Функцијата за загуба **L се намалува** во насоката на **негативниот дериват**.
- Брзината на учење (**learning rate**) е контролирана од **големината на  $\alpha$** .

**Објаснување со пример:**

Замисли дека стоиш на некој рид (график на функција) и сакаш да стигнеш во најниската точка (минимумот).

1. Гледаш дали наклонот на теренот е нагоре или надолу.
2. Ако одиш надолу, се движиш во таа насока.
3. Ако теренот е многу стрмно надолу (голем дериват), правиш поголем чекор.
4. Ако е рамно (дериват близку до нула), значи си близку до минимум.

Така, со секој чекор се приближуваш кон точката каде што е загубата најмала — тоа се **најдобрите можни регресиони коефициенти**.

## Интерпретација на предикторите

Пример:

$$\hat{y} = 7.5 + 0.04 \times TV$$

Што значи 7.5?

Што значи 0.04?

Ако ја зголемиме инвестицијата во ТВ рекламирање за 1000 долари, колку би очекувале да порасне продажбата?

А што ако моделот е:

$$\hat{y} = 7.5 + 1.01 \times TV?$$

Во оваа равенка на регресијата,  $\hat{y}$  е предвидената вредност на продажбата. 7.5 е почетната вредност на продажбата кога инвестицијата во ТВ рекламирање е нула.

0.04 е коефициентот за променливата  $TV$  и покажува колку ќе се зголеми продажбата за секое зголемување на ТВ буџетот за една единица.

Ако единицата за  $TV$  е изразена во илјадници долари, тогаш:

Ако ја зголемиме инвестицијата за 1000 долари, продажбата ќе порасне за  $0.04 \times 1 = 0.04$  единици.

Ако пак користиме моделот  $\hat{y} = 7.5 + 1.01 \times TV$ , тогаш за секои 1000 долари дополнително, продажбата би се зголемила за 1.01 единици.

## **Заклучок:**

Коефициентите во регресијата ни покажуваат колкаво влијание има секој предиктор врз зависната променлива. Но дали ќе донесеме одлуки врз основа на тие вредности зависи од тоа колку им веруваме, што зависи од квалитетот на податоците, големината на примерокот и статистичката значајност на резултатите.

## Интерпретација на $\epsilon$ (грешката) во нашите набљудувања

Терминот  $\epsilon$  го интерпретираме како шум што се појавува поради случајни варијации во природни системи и неточности во мерењето со научни инструменти.

Ако ја знаевме точната форма на  $f(x)$ , на пример  $f(x) = \beta_0 + \beta_1 x$ , и ако немаше  $\epsilon$ , тогаш проценката на  $\beta_0$  и  $\beta_1$  би била совршено точна.

Но, постојат три причини поради кои не можеме целосно да им веруваме на вредностите на  $\beta_0$  и  $\beta_1$ :

1.  $\epsilon$  секогаш постои
2. не ја знаеме точната форма на  $f(x)$
3. имаме ограничена големина на примерок

### Интервали на доверба за проценките на предикторите:

Поради грешката  $\epsilon$ , секојпат кога ќе го измериме одговорот  $Y$  за иста вредност на  $X$ , ќе добиеме различна вредност на  $Y$ .

На пример, ако имаме три мерења за неколку различни вредности на  $X$ , ќе добиеме три различни резултати на  $Y$ , иако  $X$  е ист.

Овој концепт води до потребата од интервали на доверба, кои ни кажуваат во кој опсег најверојатно се наоѓа вистинската вредност на некој параметар, како  $\beta_0$  или  $\beta_1$ , со дадено ниво на сигурност, на пример 95 проценти.

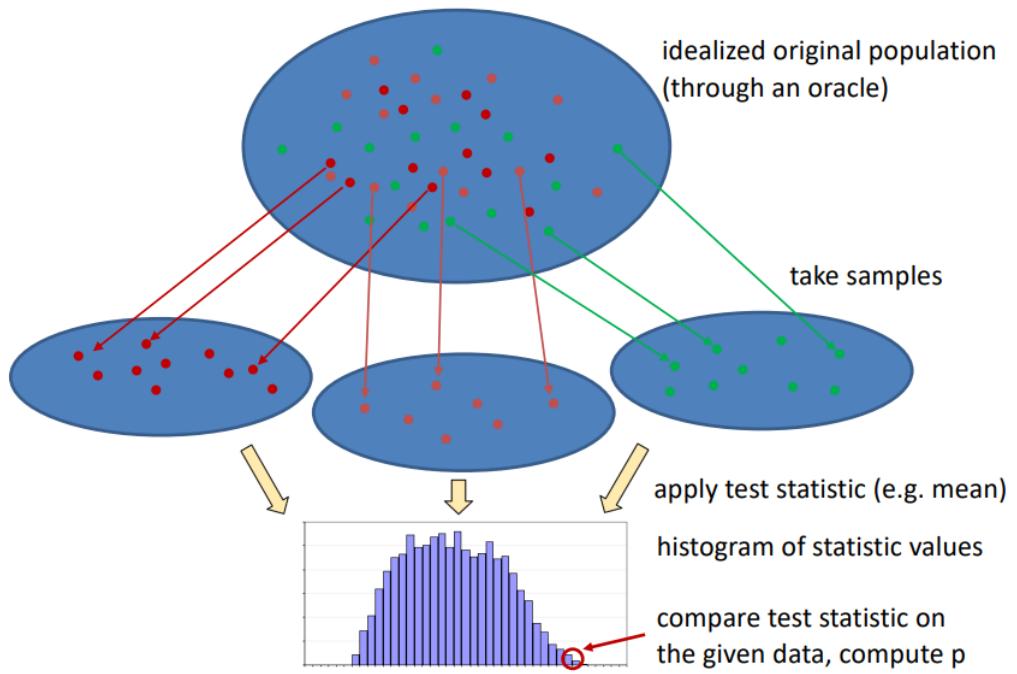
### Bootstrapping за процена на грешка при семплирање

Bootstrapping е практика каде што се проценуваат својства на некој статистички показател (естиматор) со тоа што тие својства се мерат преку повторено земање примероци од веќе набљудуваните податоци.

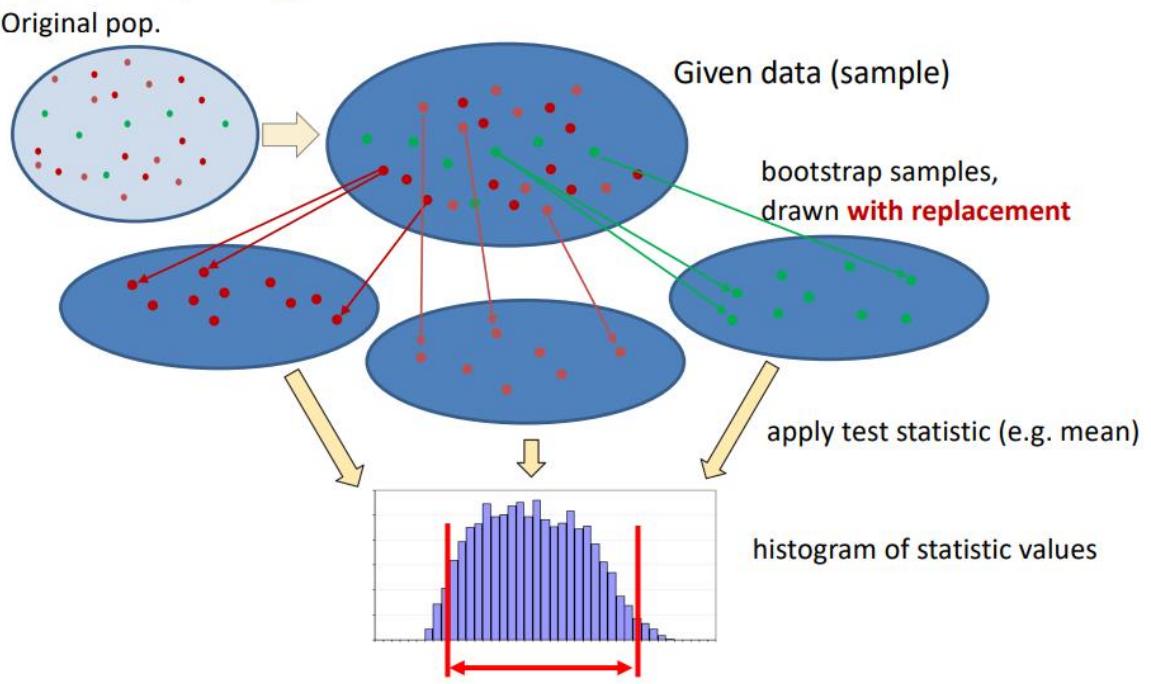
На пример, можеме да ги пресметаме  $\beta_0$  и  $\beta_1$  повеќе пати со тоа што случајно земаме примероци (со враќање) од нашата постоечка база на податоци. Потоа ја користиме варијансата на тие повеќекратни пресметки за да ја добиеме приближната вистинска варијанса на  $\beta_0$  и  $\beta_1$ .

Ова ни овозможува да добиеме претстава за нестабилноста или сигурноста на нашите проценки, без потреба од нови податоци или строги теоретски претпоставки.

## Idealized Sampling



## Bootstrap Sampling



## Интервали на доверба за проценките на предикторите (продолжение):

- Земаме повеќе примероци (на пример 100 пати) и секојпат ја пресметуваме проценката за  $\beta_0$  и  $\beta_1$ .
- Гледаме како се менуваат тие проценки од примерок до примерок.
- Варијансата (расфраноста) на овие вредности се нарекува **стандардна грешка** на  $\beta_0$  и  $\beta_1$ , и се бележи како  $SE(\beta_0)$  и  $SE(\beta_1)$ .
- Со помош на овие стандардни грешки можеме да пресметаме **интервали на доверба**.

**Што е интервал на доверба?** Тоа е опсег на вредности за кои веруваме дека со  $n\%$  сигурност (на пример 95%) ја содржат вистинската вредност на параметарот  $\beta_1$  (или  $\beta_0$ ).

На пример: Ако интервалот на доверба за  $\beta_1$  е [0.3, 0.7], тогаш со 95% сигурност веруваме дека вистинската вредност на  $\beta_1$  е некаде помеѓу 0.3 и 0.7.

### **Цел:**

Да ја измериме **несигурноста** во нашите проценки и да видиме колку можеме да им веруваме.

## Стандардни грешки на проценките (Standard Errors)

- **Стандардната грешка (SE)** ни кажува колку е нестабилна една проценка, на пример  $\beta_0$  или  $\beta_1$ .
- Колку е помала стандардната грешка, толку сме посигурни дека нашата проценка е близку до вистинската вредност.

## **Како се пресметуваат стандардни грешки?**

### **Метод 1: Bootstrapping**

- Правиме многу примероци од податоците (со повторување).
- За секој примерок пресметуваме  $\beta_0$  и  $\beta_1$ .
- Го гледаме распснувањето (варијансата) на тие вредности:
  - $SE(\beta_0)$  = варијанса на  $\beta_0$
  - $SE(\beta_1)$  = варијанса на  $\beta_1$

## Метод 2: Аналитички (со формули)

Ако знаеме варијансата на шумот ( $\varepsilon$ ), можеме директно да ги пресметаме SE без бутстреп:

За проценетата права:

$$y_i = \hat{f}(x_i) + \varepsilon_i$$

Тогаш формулите за стандардни грешки се:

**SE( $\beta_1$ )** (на наклонот):

$$SE(b_1) = \sqrt{\frac{\sigma^2}{\sum(x_i - \bar{x})^2}}$$

**SE( $\beta_0$ )** (на пресекот):

$$SE(b_0) = \sqrt{\sigma^2 \left( \frac{1}{n} + \frac{\bar{x}^2}{\sum(x_i - \bar{x})^2} \right)}$$

### Како влијае квалитетот на податоците?

- Повеќе податоци ( $n \uparrow$ )  $\rightarrow SE \downarrow \rightarrow$  поточни проценки
- Податоците да се поразновидни (поголема  $\sum(x_i - \bar{x})^2$ )  $\rightarrow SE \downarrow$
- Помал шум/грешка ( $\varepsilon \downarrow$ )  $\rightarrow SE \downarrow$

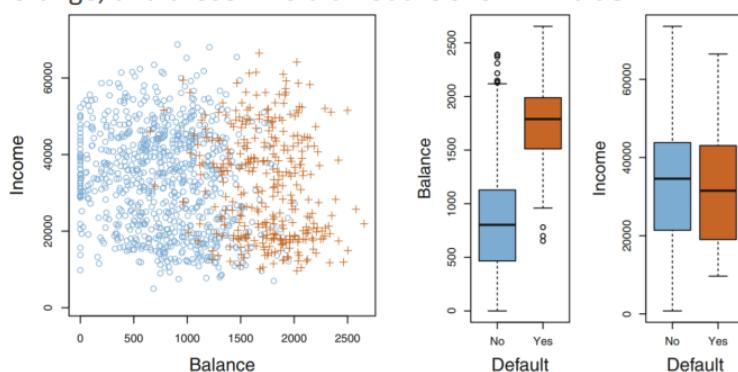
**Заклучок:** Стандардните грешки ни овозможуваат да изградиме **интервали на доверба** и да процениме колку можеме да им веруваме на нашите предиктори. Колку се помали SE, толку е помала несигурноста во моделот.

## Класификација

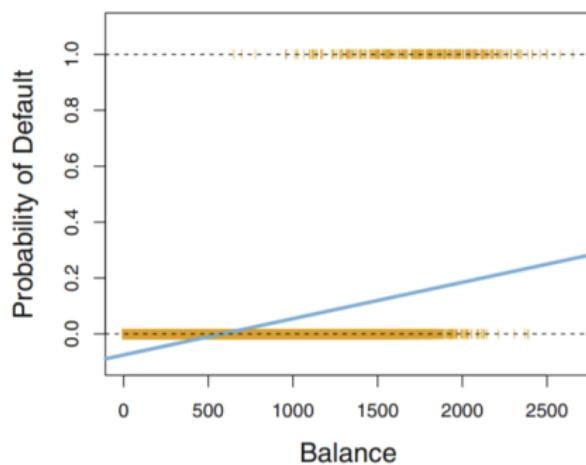
До сега, методите што ги разгледавме беа насочени кон моделирање и предвидување на квантитативна одговорна (response) променлива (на пр. број на такси превози, број на изнајмени велосипеди итн). Линеарната регресија добро функционира во вакви ситуации. Кога одговорната (response) променлива е категоријална (односно, припаѓа на категории или класи), тогаш проблемот веќе не се нарекува регресија, туку **класификација**. Целта е да се обидеме да го класифицираме секој случај (набљудување) во одредена категорија, позната и како класа или кластер, која е дефинирана од променливата  $Y$ , врз основа на сет од предвидувачки променливи  $X$ .

Класификацијата е различна од регресијата бидејќи наместо да предвидуваш бројчена вредност, ти се обидуваш да предвидиш категорија, како што е "да" или "не", "позитивно" или "негативно", "тип А" или "тип Б". **Ова е многу корисно во ситуации кога треба да се донесат одлуки базирани на групи или типови, а не на конкретни бројки.**

The individuals who defaulted on their credit card payments are shown in orange, and those who did not are shown in blue.



Тука не може да се користи линеарна регресија бидејќи би добиле нешто вакво:



## Затоа користиме логистичка регресија.

Замисли дека сакаш да предвидиш нешто што има само два можни исхода — на пример, дали некој ќе плати навреме или не (да/не), дали е болен или здрав, дали ќе купи производ или не.

Обичната регресија (линеарна регресија) предвидува бројки — на пример, колку пари ќе потрошиш, колку километри ќе возиш. Но во нашата ситуација треба да предвидиме нешто што не е бројка, туку избор помеѓу две категории.

Логистичката регресија решава тоа така што наместо да предвидува директно „да“ или „не“, таа предвидува колку е веројатно да се случи „да“. Тоа значи дека ни дава број помеѓу 0 и 1, каде 0 значи „веројатноста е многу мала“ и 1 значи „веројатноста е многу голема“. На пример, 0.7 значи 70% шанса дека ќе се случи „да“.

Потоа, ако веројатноста е поголема од некој праг (на пример 0.5 или 50%), тогаш предвидуваме дека одговорот е „да“, а ако е помала — „не“.

## Како работи?

Логистичката регресија користи една посебна математичка формула (логистичка функција) која ги зема податоците (на пр. колку личноста има пари, колку долгови) и ги претвора во веројатност од 0 до 1. Тоа е како „претворач“ што го зема резултатот од едноставна формула и го става во границите од 0 до 1.

Таа формула не само што ја дава веројатноста, туку ни овозможува да ја сфатиме важноста на секој фактор (на пр. колку годишниот приход влијае врз тоа дали ќе плати или не).

## Правење предвидувања

Откако ќе го решиме логистичкиот модел, добиваме вредности за коефициентите:

$$\beta_0 = -10.6513 \text{ и } \beta_1 = 0.0055$$

Овие бројки се користат во формулата на логистичката регресија за да се пресмета веројатноста за неплаќање (default) според билансот (balance) на кредитната картичка.

Формулата изгледа така:

$$p(X) = 1 / (1 + e^{-(\beta_0 + \beta_1 \cdot X)})$$

## Пример 1:

За личност со биланс од \$1,000, ја вметнуваме вредноста во формулата:

$$\begin{aligned} p(1000) &= 1 / (1 + e^{-(-10.6513 + 0.0055 \times 1000)}) \\ &= 1 / (1 + e^{-(-10.6513 + 5.5)}) \\ &= 1 / (1 + e^{-(-5.1513)}) \\ &= 1 / (1 + e^{5.1513}) \\ &\approx 1 / (1 + 172.79) \\ &\approx 0.0057 \rightarrow \text{под 1% шанса за неплаќање} \end{aligned}$$

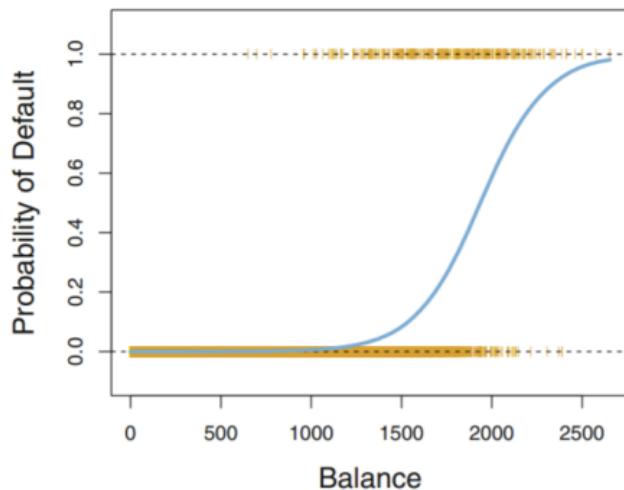
## Пример 2:

За личност со биланс од \$2,000:

$$\begin{aligned} p(2000) &= 1 / (1 + e^{-(-10.6513 + 0.0055 \times 2000)}) \\ &= 1 / (1 + e^{-(0.3487)}) \\ &\approx 1 / (1 + 0.7055) \\ &\approx 0.586 \rightarrow \text{58.6% шанса за неплаќање} \end{aligned}$$

## Заклучок:

Како што се зголемува билансот на кредитната картичка, расте и веројатноста дека личноста нема да ја исплати навреме. Логистичката регресија ни овозможува да ја пресметаме оваа веројатност за секој поединец, врз основа на нивните податоци.



## Multiple логистичка регресија

Сега ќе разгледаме ситуација каде што сакаме да предвидиме бинарен исход (на пример: „да“ или „не“, „болен“ или „здрав“) користејќи **повеќе предиктори (влијателни фактори)**.

### Што значи ова?

Во обичната логистичка регресија имавме **само еден предиктор**, на пример само билансот на кредитна картичка (balance).

Во **multiple логистичка регресија**, можеме да користиме **повеќе информации** за една личност, како на пример:

- Годишен приход (income)
- Возраст (age)
- Биланс на кредитна картичка (balance)
- Број на месечни трансакции
- И сл.

### Како изгледа формулата?

Формулата за веројатноста станува:

$$p(X) = 1 / (1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \dots + \beta_n X_n)})$$

Тука:

- $p(X)$  е веројатноста за „да“ (на пр. default = Yes)
- $X_1, X_2, \dots, X_n$  се предикторите (влезните фактори)
- $\beta_0, \beta_1, \dots, \beta_n$  се тежините/коефициентите што моделот ги учи од податоците

### Пример:

Ако сакаме да предвидиме дали некој ќе направи неплаќање (default), можеме да користиме:

- $X_1 = \text{билианс}$  на кредитна картичка
- $X_2 = \text{годишен приход}$

Тогаш моделот може да изгледа вака:

$$\log(p / (1 - p)) = \beta_0 + \beta_1 \times \text{баланс} + \beta_2 \times \text{приход}$$

Со тоа, моделот ја комбинира информацијата од двата фактори за да даде по-точна проценка.

### **Заклучок:**

**Multiple логистичка регресија** е логично продолжение на едноставната логистичка регресија. Наместо да предвидува на основа на само еден фактор, таа користи повеќе фактори истовремено за да даде попрецизна веројатност за одреден исход.

Мерки за класификација и дијагностичко тестирање, кои се користат за да се процени колку добро работи еден модел

#### **1. Accuracy – Точност**

Колкав процент од сите предвидувања се точни.

Accuracy = (Точно позитивни (TP) + Точно негативни (TN)) / Вкупно случаи

Пример: Од 100 луѓе, моделот правилно предвидел 90. Accuracy = 90%.

#### **2. Precision – Прецизност**

Од сите случаи за кои моделот кажал "да", колку навистина се "да".

Precision = Точно позитивни (TP) / (Точно позитивни (TP) + Лажно позитивни (FP))

Пример: Ако моделот предвиди 10 луѓе дека нема да платат, но само 6 од нив навистина не платиле → Precision = 6 / 10 = 60%.

#### **3. Recall – Одзив**

Од сите вистински "да" случаи, колку успеавме да фатиме.

Recall = Точно позитивни (TP) / (Точно позитивни (TP) + Лажно негативни (FN))

Пример: Имало 20 луѓе што не платиле, моделот фатил 15 → Recall = 15 / 20 = 75%.

#### **4. Sensitivity (Чувствителност)**

Истото што и Recall. Кажува колку добро ги препознаваме позитивните случаи.

#### **5. Specificity (Специфичност)**

Колку добро ги препознаваме негативните случаи.

Specificity = Точно негативни (TN) / (Точно негативни (TN) + Лажно позитивни (FP))

Пример: Од 80 луѓе што навистина ќе платат, моделот правилно препознал 70 → Specificity =  $70 / 80 = 87.5\%$ .

### Confusion Matrix (Матрица на конфузија)

Тоа е табела која визуелно прикажува колку добро работи еден класификациски алгоритам, т.е. како ги предвидува различните класи.

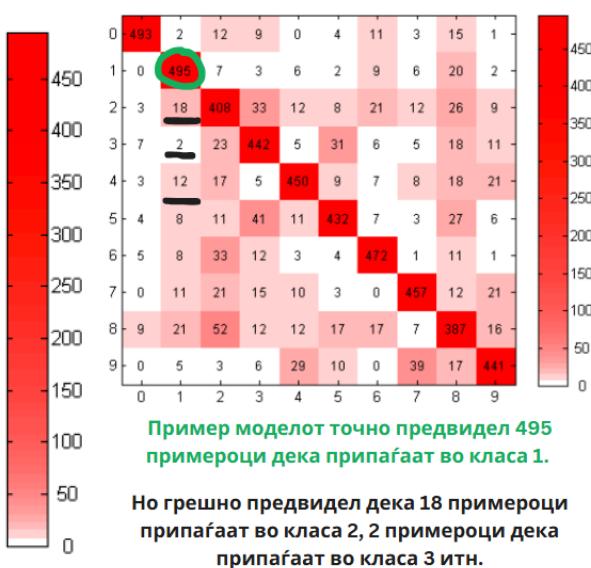
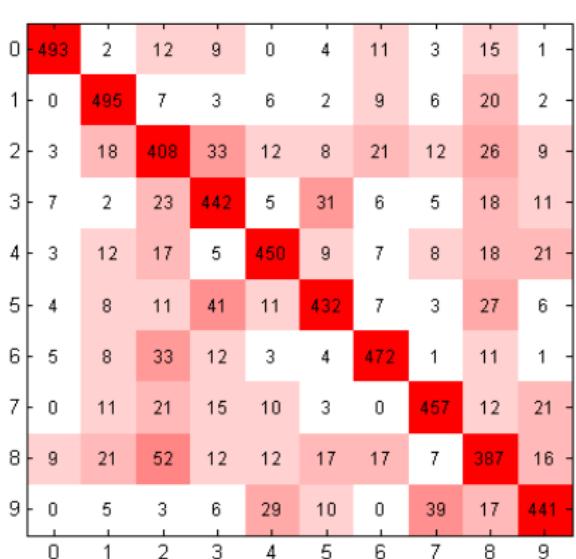
Се нарекува и **матрица на грешки**, бидејќи ни покажува каде моделот **прави точни предвидувања**, а каде **греши**.

Пример за бинарна класификација:

	Вистински: Да (Positive)	Вистински: Не (Negative)
Предвидено: Да	True Positive (TP)	False Positive (FP)
Предвидено: Не	False Negative (FN)	True Negative (TN)

Confusion matrix ни овозможува да ги пресметаме:

- **Accuracy**
- **Precision**
- **Recall / Sensitivity**
- **Specificity**
- **F1 Score** (баланс помеѓу precision и recall)



I know you are more confused than that matrix but we gotta go on. ☺

## ROC Plot (Receiver Operating Characteristic)

ROC кривата е график што се користи за оценување на перформансите на еден класификациски модел, особено користен кај бинарна класификација.

Оски на ROC графикот:

- Y-оска (Vertical):

$$\text{True Positive Rate (TPR)} = \text{NTP} / (\text{NTP} + \text{NFN})$$

Ова е исто што и Recall или Sensitivity

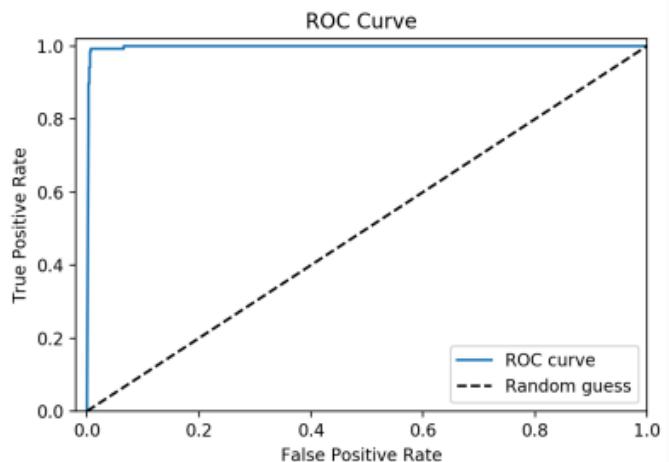
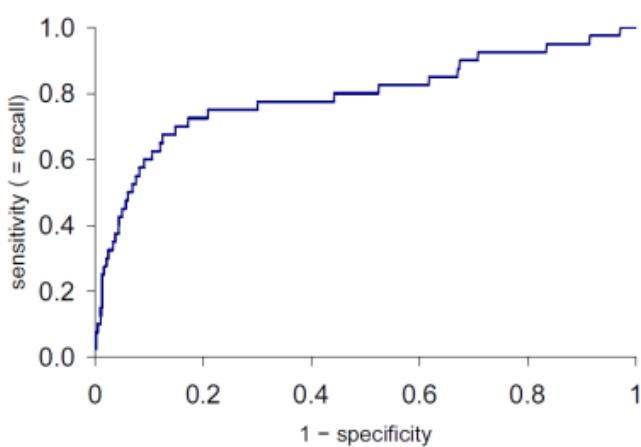
→ Покажува колкав процент од вистинските позитивни случаи сме ги погодиле.

- X-оска (Horizontal):

$$\text{False Positive Rate (FPR)} = \text{NFP} / (\text{NFP} + \text{NTN})$$

Ова е  $1 - \text{Specificity}$

→ Покажува колкав процент од негативните случаи сме ги предвиделе погрешно како позитивни.



Што прикажува ROC кривата?

ROC кривата прикажува како се менува односот помеѓу TPR и FPR додека го менуваме прагот (threshold) на моделот — на пр., од  $p > 0.9$  до  $p > 0.1$ .

### Идеален модел:

$\text{TPR} = 1$  и  $\text{FPR} = 0$

Тоа значи: ги фаќа сите позитивни случаи, без ниту една грешка кај негативните.

На графикот ова е **горниот лев агол**.

### Лош модел (случаен):

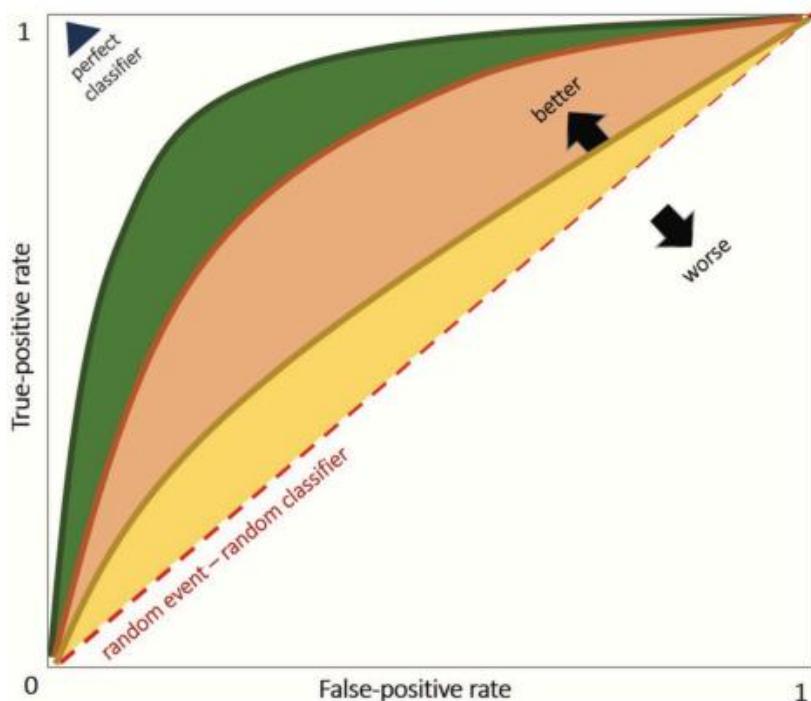
Дијагонална линија (од  $(0,0)$  до  $(1,1)$ ).

$\text{TPR} = \text{FPR} \rightarrow$  моделот претпоставува на случајна основа.

### Добар модел:

Кривата е **над дијагоналата**, близку до горниот лев агол.

Колку подалеку е кривата од дијагоналата нагоре и лево  $\rightarrow$  толку подобар е моделот.



### AUC (Area Under Curve):

Мерка за „површината под ROC кривата“. Се движи од 0 до 1. Треба да биде меѓу 0.5 и 1.

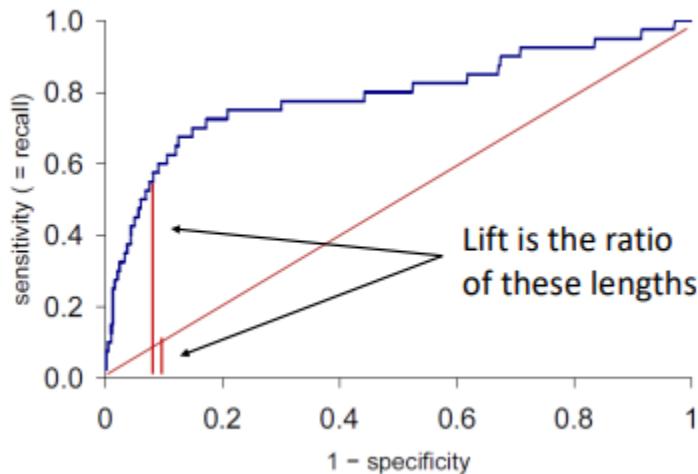
**$\text{AUC} \approx 1 \rightarrow$  одличен модел**

**$\text{AUC} \approx 0.5 \rightarrow$  случајно претпоставување**

**$\text{AUC} < 0.5 \rightarrow$  полошо од случајно**

## Lift Plot (Лифт график)

Lift графикот е алатка слична на ROC, но се користи за подобро да се оцени **корисноста на моделот во пракса**, особено во ситуации кога сакаме да ги идентификуваме **највредните корисници или случаи** (на пример: кои најверојатно ќе купат, ќе кликнат).



**Lift ни кажува:** Колку подобро се однесува моделот во споредба со случајно претпоставување?

**Како се пресметува Lift?**

**Lift = Precision на моделот / Precision при случаен избор**

Ако моделот има  $\text{Lift} = 4 \rightarrow$  Тоа значи дека е 4 пати подобар од случајно претпоставување.

Ако  $\text{Lift} = 1 \rightarrow$  Моделот нема подобра точност од случаен избор.

Колку повисок е Lift во првите неколку проценти од податоците, толку подобро.

**Пример:**

Ако имаш 10.000 корисници и од нив 500 ќе направат купување (5%), случаен модел би погодил 5%.

Но твојот модел кај првите 10% (1.000 корисници) погодува 150 купувачи  $\rightarrow$  прецизност = 15%

$\text{Lift} = 15\% / 5\% = 3 \rightarrow$  3 пати подобар од случајно.

# Machine Learning 2

## Полиноминална регресија

### Што е Полиноминална Регресија?

Полиноминалната регресија е проширување на линеарната регресија каде што се додаваат степени на независната променлива ( $x^2$ ,  $x^3$ , ...,  $x^M$ ) за да се добие нелинеарна врска.

Формула:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_M x^M + \varepsilon$$

Иако изгледа нелинеарно, **технички е линеарен модел** бидејќи е линеарен во параметрите ( $\beta$ ).

### За што се користи?

Кога податоците не можат добро да се моделираат со права линија.

Се користи за **криволинеарно "curve fitting"** – приближување на податоци со криви линии.

Примери: Раствор на население со тек на време, Висина на проектил во зависност од времето, Комплексни трендови во продажба или температури.

### Како изгледа во практика?

- $x, x^2, x^3\dots$  се третираат како одделни предиктори (features).
- Се гради матрица на податоци со сите степени на  $x$  до степен  $M$ .
- Се користи истата техника како кај линеарна регресија за наоѓање на параметрите  $\beta$ .

### Клучен концепт: Curve Fitting

Со зголемување на степенот  $M$ , моделот станува се пофлексибilen.

**Ockham's razor:** користи го **наједноставниот модел што добро ги објаснува податоците**.

**Пример:** Ако податоците се лесно објасниви со  $x^2$ , нема потреба од  $x^5$ .

## Проблем: Overfitting (претерано прилагодување)

Ако се избере премногу висок степен на полином (на пример  $x^{10}$ ), моделот ќе го следи шумот во податоците и ќе има лоша генерализација.

**Симптоми:** Ниска грешка на training set, Висока грешка на test set.

Целта е да се најде **идеалната сложеност** на моделот.

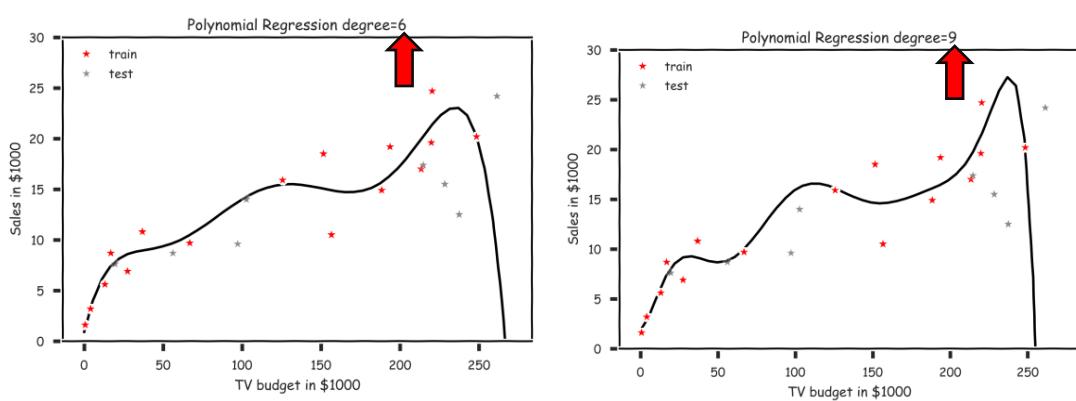
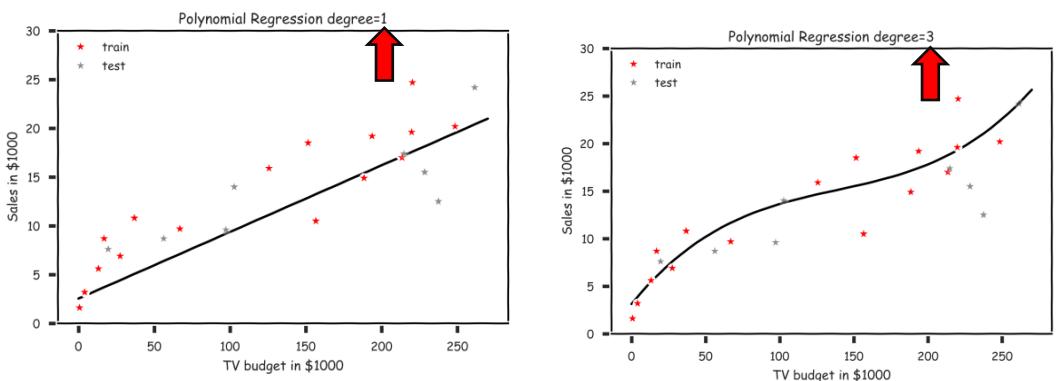
## Решение: Model Selection и Cross-validation

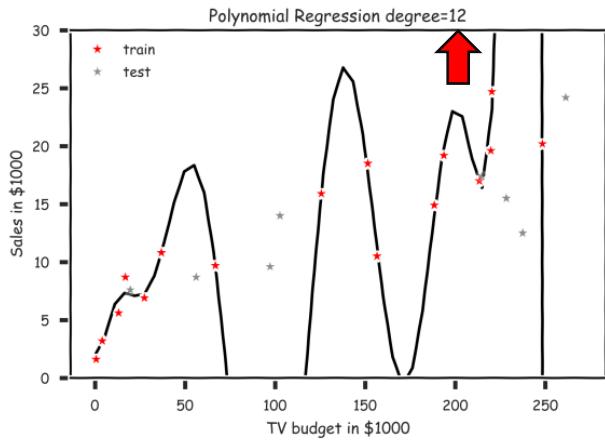
Избор на степенот M се прави преку **Cross-validation**:

- Поделба на податоците во **тренинг и валидациски сетови**
- Измери просечната грешка за различни M
- Избери M со најниска просечна грешка

## Bias-Variance Tradeoff

- **Мал степен (low M)** → висок **bias** (моделот е премногу прост).
- **Голем степен (high M)** → висока **variance** (моделот е нестабилен).
- Цел: **Рамнотежа** помеѓу bias и variance.

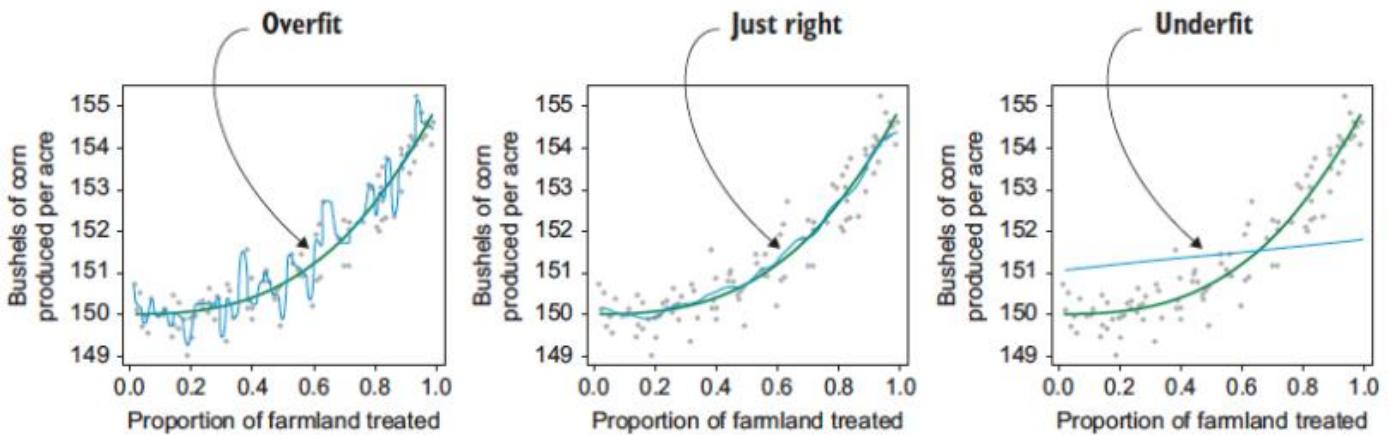




## Overfitting

Overfitting е кога моделот премногу добро **ги учи податоците од тренинг сетот**, вклучувајќи и случајни шумови или нестандартни детали кои не се генерални правила. Како резултат, **моделот има многу мала грешка на тренинг податоците, но многу слабо работи на нови, невидени податоци.** Тоа значи дека моделот не ја фаќа вистинската структура или правило, туку специфичките на примерите што ги научил.

Overfitting **најчесто се случува кога моделот е премногу сложен за количината или квалитетот на податоците**, на пример, кога користиме полином со премногу висок степен. За да се избегне, користиме техники како cross-validation, ограничување на комплексноста на моделот, или regularization.



## Избор на модел (Model Selection)

Изборот на модел претставува користење на одреден метод за да се одреди колку сложен треба да биде моделот. Тоа може да значи, на пример, да се изберат само некои од променливите (predictors) или да се избере соодветен степен на полиномот кај полиноминална регресија.

Главната причина зошто правиме избор на модел е за да **избегнеме overfitting**, односно моделот да не биде премногу прилагоден на тренирачките податоци.

Overfitting може да се случи кога:

- има премногу променливи во моделот
- просторот на карактеристики (feature space) е премногу голем
- степенот на полиномот е премногу висок

## Валидација (Validation)

Кога правиме модел, обично ги делиме податоците на два дела:

- **тренинг сет (обука)** – за да го научиме моделот
- **валидационен сет** – за да провериме колку добро моделот работи

На пример: ако имаш 100 податоци, можеш да искористиш 80 за тренирање и 20 за валидација. Ако моделот добро предвидува на тие 20, веројатно ќе работи добро и на други нови податоци.

### **ВАЛИДАЦИСКИ СЕТ ≠ ТЕСТ СЕТ**

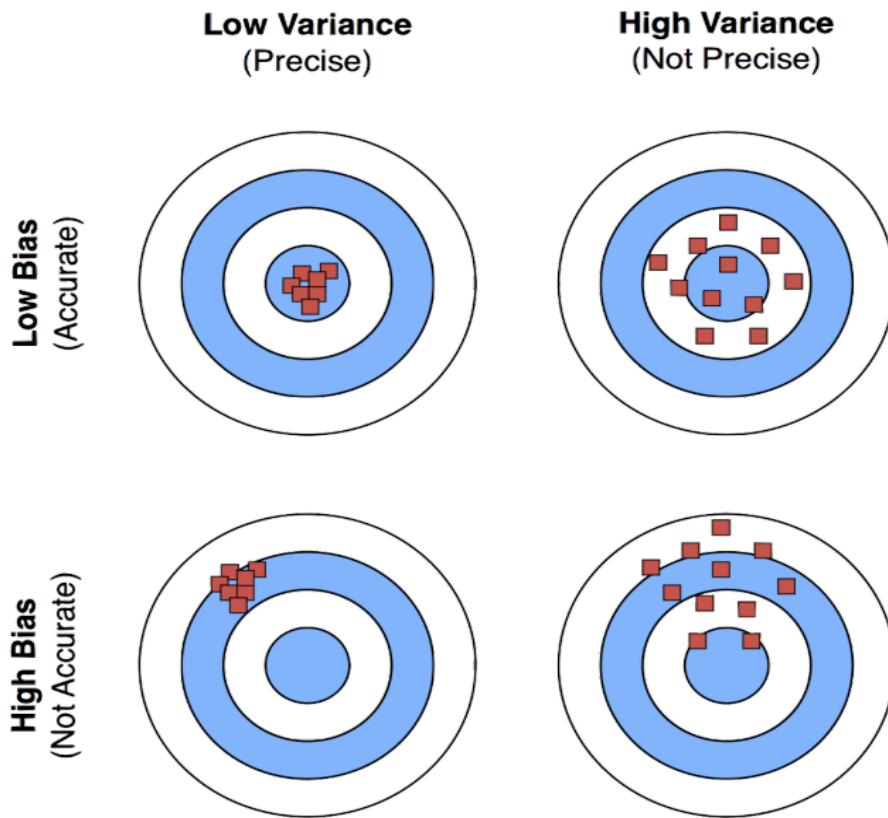
	<b>Кога се користи</b>	<b>Дали влијае на моделот?</b>
<b>Валидациски сет</b>	<b>За избор и подесување</b>	<b>Да</b>
<b>Тест сет</b>	<b>За финална проверка</b>	<b>Не</b>

## Крос валидација (Cross-validation)

Ова е подобра верзија на валидација. Наместо да делиш само еднаш, ги делиш податоците повеќе пати и во делови.

Најпозната е **K-fold cross-validation**:

- Ги делиш податоците на **K еднакви делови** (на пример 5).
- 4 дела ги користиш за тренирање, а 1 дел за тестирање.
- Ова се повторува 5 пати, така што секој дел ќе биде тест барем еднаш.
- На крајот, се зема просекот од сите резултати.



### 1. Low Bias, Low Variance (горе лево)

- Точност: Висока
- Прецизност: Висока
- Сите погодоци се **блиску до центарот и блиску еден до друг**.
- **Идеален модел** – добро ги учи податоците и предвидува стабилно на нови примери.

## 2. Low Bias, High Variance (горе десно)

- **Точност:** Во просек добра
- **Прецизност:** Слаба
- Погодоците се околу центарот, но **растурени**.
- Моделот **учел добро**, но е **нестабилен** – на различни податоци дава различни резултати.

Проблем: **Overfitting** – премногу добро учи деталите, но не генерализира добро

## 3. High Bias, Low Variance (долу лево)

- **Точност:** Лоша
- **Прецизност:** Висока
- Сите погодоци се близки еден до друг, но **далеку од центарот**.
- Моделот **не ја научил вистинската шема** – препрост е за проблемот.

Проблем: **Underfitting** – не учи доволно, затоа не функционира добро

## 4. High Bias, High Variance (долу десно)

- **Точност:** Лоша
- **Прецизност:** Лоша
- Погодоците се и **растурени и далеку од центарот**.
- Моделот е и **препрост и нестабилен**.

Најлоша ситуација – не учи добро и не е доследен

## Регуларизација

Регуларизација е техника што се користи во машинско учење и статистика за да се спречи overfitting на моделот.

Кога тренираме модел, сакаме да ја минимизираме **функцијата на загуба** (loss function) -  $L(\theta)$ , која мери колку лошо моделот ги предвидува излезите. Регуларизацијата ја модифицира оваа функција така што додава дополнителен термин **кој го казнува моделот ако има несакани својства**.

**Модифицираната функција на загуба изгледа така:**

$$L_{\text{reg}}(\theta) = L(\theta) + \lambda R(\theta)$$

$L(\theta)$  – оригиналната функција на загуба (на пример, средна квадратна грешка)

$R(\theta)$  – **регуларизациски термин**, кој дефинира што точно сакаме да казнуваме (на пример, големи вредности на параметрите)

$\lambda$  – скалар што ја контролира **тежината** на регуларизацијата (колку важен е тој термин)

Со минимизирање на  $L_{\text{reg}}(\theta)$ , не само што се обидуваме да го намалиме бројот на грешки, туку и да добиеме **попрост модел**, кој:

- не се потпира премногу на некои од влезните карактеристики
- има помали параметри (во случај на L2 регуларизација)
- користи помал број на параметри (во случај на L1 регуларизација)

**Најчести типови на регуларизација:**

1. **L2 регуларизација (Ridge)**: ја казнува големината на параметрите.

$$R(\theta) = \|\theta\|_2^2 = \sum \theta_i^2$$

2. **L1 регуларизација (Lasso)**: тера некои параметри да станат 0 → **sparse**.

$$R(\theta) = \|\theta\|_1 = \sum |\theta_i|$$

## Избор на $\lambda$

И кај Ridge и кај LASSO регресија, гледаме дека колку е поголема вредноста на параметарот за регуларизација  $\lambda$ , толку посилно казнуваме големи вредности на параметрите  $\beta$ .

Ако  $\lambda$  е многу близку до нула, добиваме обична MSE (средна квадратна грешка), односно Ridge и LASSO регресијата се сведуваат на обична линеарна регресија.

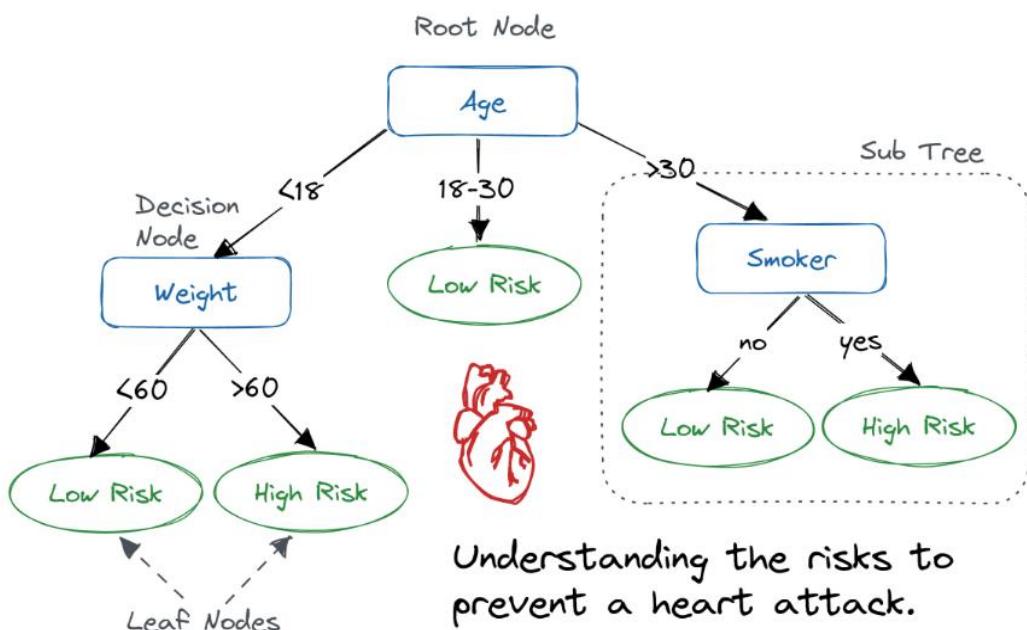
Ако  $\lambda$  е доволно голема, терминот за MSE во регуларизираната функција на загуба станува беззначаен, и регуларизацискиот термин ќе ги принуди Ridge и LASSO решенијата да бидат многу близку до нула.

За да избегнеме произволен избор на  $\lambda$ , треба да го избереме со крос-валидација.

**Заклучок:** Регуларизацијата ни помага да добиеме **поуниверзален модел**, кој подобро функционира на нови податоци, преку контрола на сложеноста на моделот преку дополнителен казнувачки термин во функцијата на загуба.

## Decision trees – Дрва на одлуки

**Дрво на одлука** е модел во машинско учење кој се користи за донесување одлуки. Тоа функционира како дијаграм што поставува прашања и врз основа на одговорите, ве води до крајна одлука или предвидување.



## Како изгледа дрво на одлука во машинско учење?

Во машинско учење, дрвото:

- На почетокот (коренот) поставува **најважното прашање** (на пр. „Возраст“).
- Секоја гранка претставува одговор („да“ или „не“).
- Се дели податочниот сет на помали делови според карактеристиките (features).
- На крајот на дрвото (листовите), добиваме **класа** (на пример: "висок ризик" или "низок ризик").

Дрвата на одлуки се користат за:

- **Класификација:** дали некој ќе купи производ, дали е болен или не итн.
- **Регресија:** предвидување на бројчени вредности, на пример, цена на куќа.

Предности	Недостатоци
Лесни за разбирање и објаснување	Склоност кон overfitting
Не бараат многу претходна обработка на податоци	Малите промени во податоците можат да го сменат дрвото
Работат и со категоријални и со бројчени податоци	Можат да бидат нестабилни
Генерираат јасни правила за одлука	Често се помалку точни од други модели (без енсембли)
Брзо тренирање и предвидување	Тешко се справуваат со многу сложени релации

## За какви податоци работи?

- **Бројчени** (може да се споредуваат со прагови).
- **Категоријални** (потребна е енкодирање или специјално справување).

## Критериуми за делење

1. **Classification Error:** мера на неточност по поделбата.
2. **Gini Index:** мера на „чистота“ на групите.
3. **Entropy (Information Gain):** мера на неизвесност во податоците.

## Услови за застанување

- Сите примери во гранката се од иста класа.
- Премалку податоци за нова поделба.
- Максимална длабочина на дрвото.
- Премала добивка од поделбата (e.g. Gini или Entropy).

## Pruning (сечење на дрвото)

- Наместо да го ограничиме растот однапред, **дрвото се гради до крај**, па потоа се **сече делови** што не придонесуваат за точноста.
- Се користи **cost-complexity**:  $C(T) = \text{Error}(T) + \alpha \cdot |T|$

## Bias vs Variance

- Мали дрва → висок **bias**, ниска точност.
- Големи дрва → висок **variance**, overfitting.
- Цел: најди баланс со **крос-валидација**.

## Параметри во sklearn

- `max_depth`, `min_samples_split`, `criterion`, `min_impurity_decrease`, итн.  
- се користат за да се **контролира растот и прецизноста** на дрвото.

# Machine Learning 3

## Ensemble models

Енсембл моделите комбинираат повеќе модели за да се добие посилен модел.

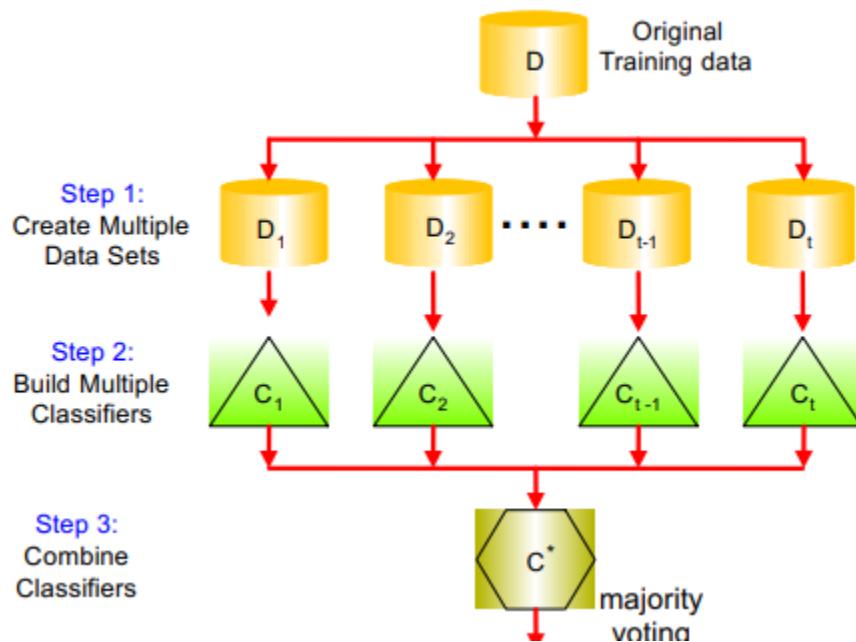
- Bagging, Random Forest, Boosting (XGBoost)

## Проблем со дрвата на одлука

Лесно и брзо се толкуваат и тренираат, но при покомплексни граници/функции ни е потребно големо дрво – тоа има висока варијанса – доведува до overfitting.

## Bagging

Метод за градење еден модел преку тренирање и агрегирање на повеќе модели.



Еден начин да се намали големата варијанса на резултатите од некој експеримент е да се повтори експериментот повеќепати и потоа да се пресмета просекот на добиените резултати.

**Bootstrap** – генерираме повеќе примероци од тренинг податоците, користејќи техника на „bootstrapping“. На секој примерок тренираме целосно **decision tree** дрво.

**Агрегирање** – за даден влез пресметуваме просек на излезите од сите модели за тој влез.

**Кај класификација**, се враќа класата што е најчесто предвидена од повеќето модели (гласање по мнозинство).

**Кај регресија**, се враќа просечната вредност од сите модели.

**Овој метод се нарекува Bagging (скратено од Bootstrap Aggregating), и е воведен од Breiman во 1996 година.**

### Предности на bagging:

- **Експресивност** – со користење на целосни дрва секој модел може да се справи со комплексни функции и граници на одлука.
- **Ниска варијанса** – правење просек на предвидувањата на сите модели ја намалува варијансата на финалното (крајното) предвидување, под претпоставка дека сме избрале доволно голем број на дрва.

Сепак може да се случи overfitting ако дрвата се многу големи или доколку се не доволно длабоки може да се случи underfitting.

Главниот недостаток на bagging е тоа што просечениот модел **не е лесен за интерпретација** – односно, не може лесно да се следи „логиката“ зад некој излез преку серија одлуки засновани на вредностите на предикторите.

### Out-of-Bag Error (OOB Error)

Со енсембл методите добиваме нова метрика за проценка на предиктивната ефикасност на моделот, наречена **out-of-bag (OOB) грешка**.

Дадено е тренинг множество и енсембл од модели, каде секој модел е трениран на „bootstrap“ примерок – ја пресметуваме **OOB грешката** на просечниот модел на следниот начин:

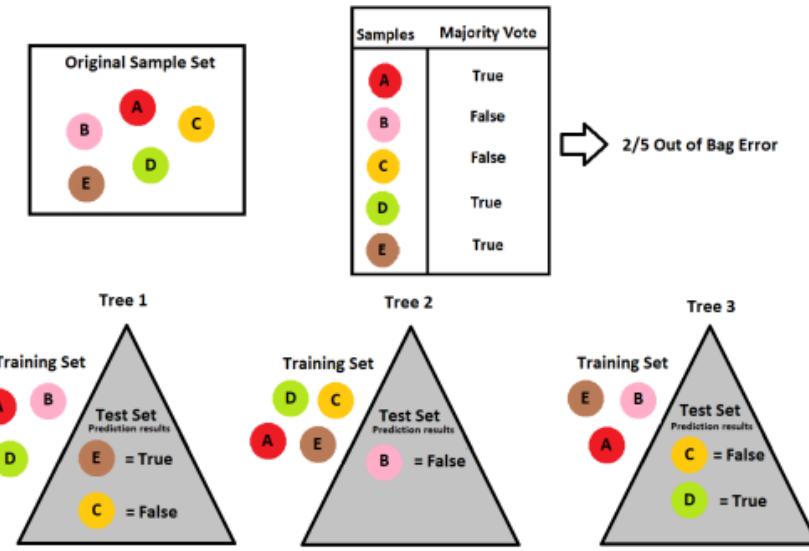
1. За секоја точка од тренинг множеството, пресметуваме просечна предвидена вредност од сите модели чиј **bootstrap примерок не ја содржи таа точка**. Потоа ја пресметуваме грешката (или квадрираната грешка) на таа просечна предикција.  
Ова се нарекува **точка-по-точка OOB грешка** (point-wise out-of-bag error).
2. Просекот од сите точка-по-точка OOB грешки за целото тренинг множество го дава финалниот OOB резултат.

Classification

$$Error_{OOB} = \frac{1}{n} \sum_i^n e_i = \frac{1}{n} \sum_i^n \mathbb{I}(\hat{y}_{i,pw} \neq y_i)$$

Regression

$$Error_{OOB} = \frac{1}{n} \sum_i^n e_i = \frac{1}{n} \sum_i^n (y_i - \hat{y}_{i,pw})^2$$



Во пракса, дрвата во енсемблот кај bagging често се **високо корелиирани**.

Замисли дека во тренинг множеството има еден **многу силен предиктор**, наречен  $x_1$ , меѓу неколку умерено важни предиктори. Поради алчниот (greedy) алгоритам за учење, **повеќето модели во енсемблот ќе го изберат токму  $x_1$  за првите делења**.

Каков е резултатот? Секое дрво станува **многу слично на другите**, односно тие се **идентично распределени**, па и очекуваниот излез на просечниот модел е **приближно ист како оној на секое поединечно дрво**.

## Random Forest

Random Forest е модифицирана форма на bagging која создава енсембл од **независни decision trees**.

За да ги декорелираме дрвата, го правиме следново:

1. Секое дрво го тренираме на **свој bootstrap примерок** од целото тренинг множество (исто како каде bagging).
2. За секое дрво, **при секое делење, случајно се избира подмножество од  $J'$  предиктори** (од сите можни предиктори).

Од тие  $J'$  предиктори се избира **најдобриот предиктор и соодветниот праг (threshold)** за поделба.

## Подесување (tuning) на Random forest

RF моделите имаат неколку хиперпараметри кои треба да се подесат:

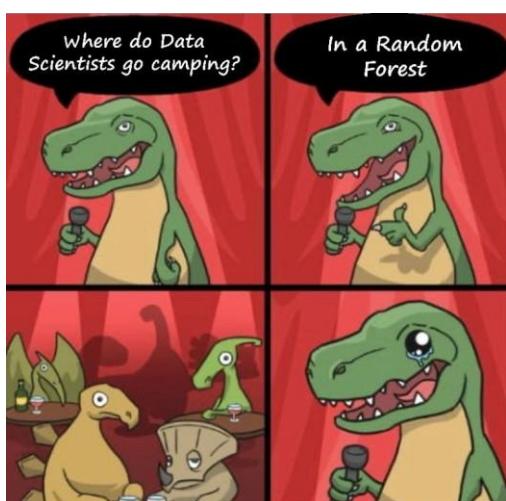
- Бројот на предиктори кои случајно се избираат при секоја поделба
- Вкупниот број на дрва во енсемблот
- Минималната големина на leaf nodes

Постојат **стандардни вредности** за секој од хиперпараметрите на random forest, препорачани од искусни практичари.

Но генерално, **овие параметри треба да се подесуваат користејќи ја Out-of-Bag (OOB) грешката**, со што тие стануваат  **зависни од податоците и конкретниот проблем**.

Пример – број на предиктори што се избираат при секое делење:

- За класификација:  $\sqrt{N}$  (каде што N е вкупниот број на предиктори)
- За регресија: N / 3
- Користејќи ја OOB грешката, тренирањето и крос валидацијата можат да се направат во еден процес – тренирањето се прекинува кога OOB грешката се стабилизира.



**Кога има многу атрибути (предиктори), но само неколку од нив се важни:**  
Random Forest може да не работи добро, затоа што при секое делење од дрвото, има мала шанса да се избере некој од важните предиктори, па многу од дрвјата ќе бидат слаби.

**Зголемување на бројот на дрвја (на пример од 100 на 500):**  
Обично не води до overfitting (претерано учење на податоците).  
Може само да го подобри моделот или да остане ист.

### Зошто е тоа така?

Кога имаме повеќе дрвја, **се намалува случајноста** (variance) и моделот станува постабилен, барем колку што е стабилно едно дрво само.

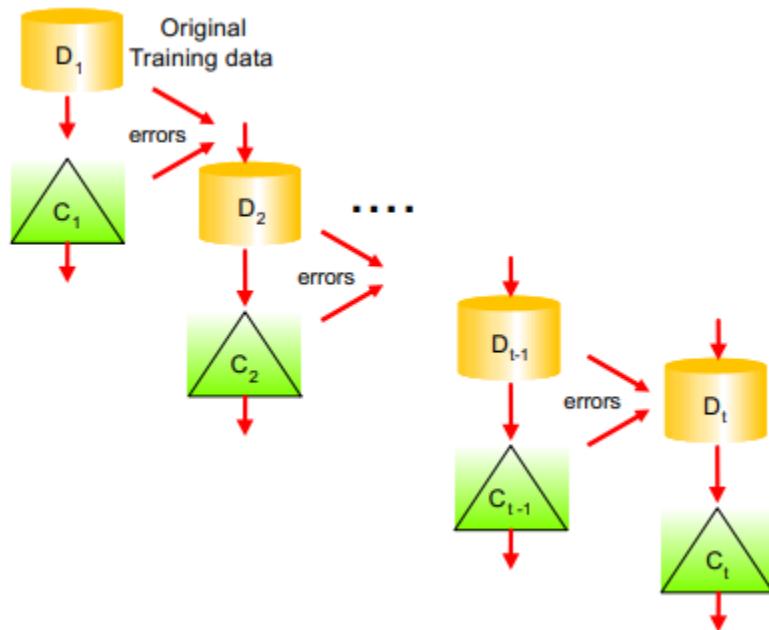
### Но ако ставиме премногу дрвја (илјадници):

Дрвјата може да почнат да личат едно на друго (се корелираат)

Тоа може повторно да ја зголеми варијансата и да не даде дополнителна добивка.

## Boosting

**Boosting** е енсембл техника која комбинира **повеќе слаби модели** (обично мали decision trees), тренирани **секвенцијално**, така што секој нов модел се обидува да ги поправи грешките на претходните, со цел да се добие **сilen и точен предвидувачки модел**.



## Два главни типа на Boosting:

### 1. Адаптивен Boosting (Adaptive Boosting):

- Им дава поголема тежина на примерите кои претходниот слаб модел ги предвидел погрешно.
- Се фокусира на примерите што биле тешки за претходниот модел.
- Пример: AdaBoost

### 2. Градиентски Boosting (Gradient Boosting):

- Секој нов модел се тренира врз остатокот (residual), односно на грешките што ги направил досегашниот модел.
- Овој пристап е сличен на градиентски спуст, бидејќи постепено го подобрува моделот.
- Пример: XGBoost

(Резидуали се разликата помеѓу вистинските (набљудуваните) вредности и предвидените вредности од моделот)

## Gradient Boosting

Главната идеја кај boosting е дека можеме да земеме повеќе едноставни модели  $\{T_h\}$  и да ги комбинираме (собереме) во еден сложен модел.

Секој модел  $T_h$  поединечно може да не е добар, но нивната линеарна комбинација може да биде многу изразена и флексибилна.

Прашање: Кои модели да ги вклучиме во енсемблот? Кои да бидат тежините (кофициентите) во комбинацијата?

## Алгоритам за Gradient Boosting

Gradient Boosting е метод за постепено градење сложен регресиски модел, преку додавање на едноставни модели. Секој нов модел што го даваме ја поправа слабоста на претходните.

Чекори:

1. Тренирај прв едноставен модел  $T^1$  врз тренинг податоците  $\{x_1, y_1\}, \dots, \{x_n, y_n\}$   
Постави  $T \leftarrow T^1$   
Пресметај остатоци (residuals):  $r_i = y_i - T(x_i)$
2. Тренирај нов едноставен модел  $T^2$  врз остатоците  $(r_1, \dots, r_n)$

3. Ажурирај го моделот:

$$T \leftarrow T + \lambda \cdot T^2$$

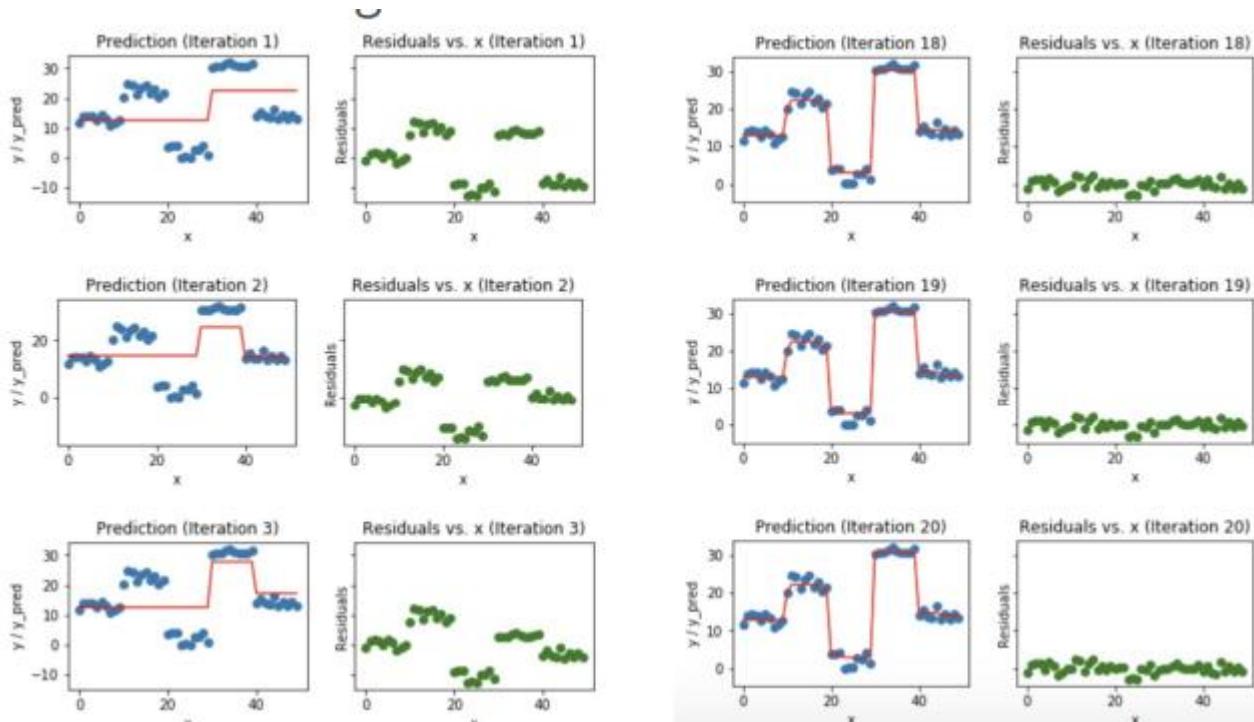
( $\lambda$  = learning rate = колку „тежи“ новиот модел)

4. Пресметај нови остатоци:

$$r_i \leftarrow r_i - \lambda \cdot T^2(x_i)$$

5. Повтори ги чекорите 2–4 додека не се исполнит условот за запирање (на пример: број на итерации или мали остатоци)

**Learning rate –  $\lambda$**  контролира колку новиот модел влијае врз конечниот резултат. Мал  $\lambda$  значи поспоро, но повесно учење.



## Што е стапка на учење (learning rate)?

Стапката на учење е бројка која кажува колку големи чекори прави алгоритамот кога учи (кога ги менува предвидувањата на моделот).

### Зошто е важна?

Ако чекорите се многу мали (најмалата стапка на учење), моделот многу бавно се учи и ќе му треба многу време (многу итерации) да стигне до добар резултат.

Ако чекорите се многу големи (прекумерно голема стапка), моделот може да „прескокнува“ околу добриот резултат и никогаш да не се стабилизира.

## Како да ја избереш стапката на учење?

Ако е константна, обично пробуваш различни вредности (пример: 0.1, 0.01, 0.001) и гледаш која дава најдобри резултати. Ова го правиш со помош на „кросвалидација“ (тестирање на моделот на различни делови од податоците).

Понекогаш е подобро стапката на учење да се менува во текот на учењето:

- Кога си далеку од добриот резултат (оптимум), може да правиш поголеми чекори (поголема стапка).
- Кога си близу до добриот резултат, треба да правиш помали чекори (помала стапка) за да не го прескокнеш.

### Пример со планинар

Замисли дека си планинар и се спушташ по планина до најниската точка (оптимум).

Ако чекориш премногу малку (мала стапка), ќе ти треба многу време да стигнеш долу.

Ако чекориш премногу големи чекори (голема стапка), може да се лизнеш и да се вратеш нагоре, или да прескокнеш целата низина и да паднеш на друга страна.

На почеток, кога си на врвот, можеш да чекориш поголеми чекори за побрзо да слезеш. Кога си близку до дното, чекориш помали чекори за да не пропаднеш во некоја дупка.

**Заклучок:** Стапката на учење е брзината со која моделот учи — треба да биде избалансирана за да учи доволно брзо, но и стабилно.

## XGBoost

XGBoost е многу ефикасна имплементација на Gradient Boosting со одлуки (Decision Trees) и има неколку интересни карактеристики:

**Регуларизација:** Може да користи L1 или L2 регуларизација за да се спречи overfitting на моделот.

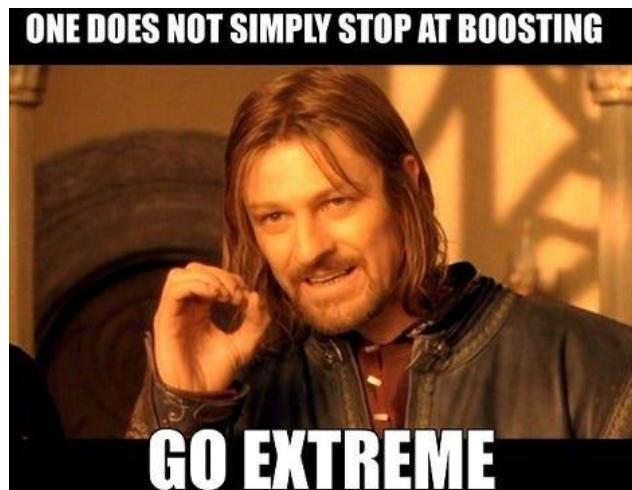
**Обработка на податоци со празни (sparse) вредности:** Вклучува алгоритам кој го разбира и ефективно обработува податоците со многу празни (недостасувачки) вредности.

**Weighted quantile sketch:** Користи посебен алгоритам за да се справи со тежините на податоците, што е важно кога некои податоци се поважни од други.

**Паралелно учење:** Користи повеќе јадра на процесорот (CPU) преку блок структура во дизајнот, што го забрзува тренирањето.

**Cache awareness:** Секој процес (thread) има свои внатрешни бафери за складирање на статистики, што го подобрува перформансот.

**Out-of-core computing:** Може да работи со огромни сетови на податоци кои не се вклопуваат во RAM меморијата, користејќи дисковен простор за оптимизирано учење.



## Наивен Баесов класификатор

**Наивниот Баесов класификатор** е многу брз и ефикасен алгоритам за класификација базиран на Баесовата теорема. Се нарекува „наивен“ бидејќи претпоставува дека сите карактеристики (features) се независни едни од други, што не е секогаш точно – но често работи многу добро во пракса.



### Баесова теорема (формула):

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

- $P(C|X)$ : веројатноста класата C да е точна дадено X (твоите податоци)
- $P(X|C)$ : веројатноста да ги добиеме податоците X ако знаеме дека се од класата C
- $P(C)$ : веројатноста за класата C (предзнаење)
- $P(X)$ : вкупна веројатност за X (ова може да се игнорира за споредба)

## Пример: Дали пораката е Spam или Not Spam?

Порака	Класа
“Бесплатни пари”	Spam
“Состанок денес”	Not Spam
“Освоивте награда”	Spam
“Проектна презентација”	Not Spam

**Нова порака: „Бесплатна награда“**

Сакаме да ја класифицираме: **Spam** или **Not Spam**?

**Фреквенции на зборови:**

Збор	Во Spam	Во Not Spam
бесплатна	1	0
награда	1	0

**Пресметка со Laplace smoothing:**

- Број на зборови во Spam = 4
- Број на зборови во Not Spam = 4
- Вкупно различни зборови (речник) = 6
- $P(\text{spam})=0.5$ ,  $P(\text{not\_spam})=0.5$

(Често користиме Laplace smoothing – додаваме +1 на сите броеви за да избегнеме деление со нула)

**За Spam:**

$$P(\text{бесплатна}|\text{spam}) = \frac{1+1}{4+6} = \frac{2}{10} = 0.2$$

$$P(\text{награда}|\text{spam}) = \frac{1+1}{4+6} = 0.2$$

$$P(\text{spam}|\text{порака}) = 0.5 \cdot 0.2 \cdot 0.2 = 0.02$$

**За Not Spam:**

$$P(\text{бесплатна}|\text{not\_spam}) = \frac{0+1}{4+6} = \frac{1}{10} = 0.1$$

$$P(\text{награда}|\text{not\_spam}) = \frac{0+1}{4+6} = 0.1$$

$$P(\text{not\_spam}|\text{порака}) = 0.5 \cdot 0.1 \cdot 0.1 = 0.005$$

## Заклучок :

$$P(\text{Spam} \mid \text{порака}) = 0.02$$

$$P(\text{Not Spam} \mid \text{порака}) = 0.005$$

=> Класификација:  SPAM

**Наивниот Баесов класификатор работи и со нумерички вредности.**

Кога имаме **нумерички атрибути**, наместо да броиме појави (како кај текст), се претпоставува дека тие атрибути следат некоја веројатносна распределба — најчесто **Гаусова (нормална) распределба**.

Оваа варијанта се вика **Gaussian Naïve Bayes**.

# Artificial Neural Networks and Deep Learning

**Невронските мрежи** се алгоритми во машинското учење инспирирани од начинот на кој функционира човечкиот мозок. Тие се составени од **вештачки неврони** организирани во **слоеви** (layers) – влезен, скриени и излезен слој. Нивната задача е да научат шеми од податоци, на пример:

- да препознаат лице на слика,
- да класифицираат е-пошта како спам или не,
- да предвидат вредност на берзата, итн.

Пример:

Ако на мрежата ѝ покажеш многу слики од мачки и кучиња, таа учи да ги разликува преку анализа на features од сликите.

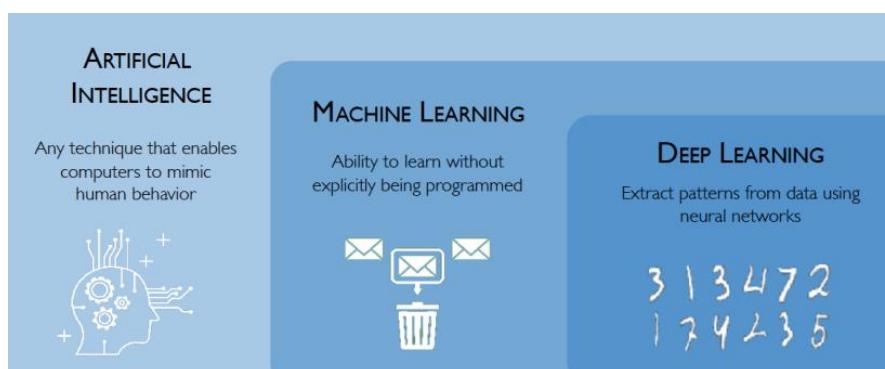
**Длабоко учење е подобласт на невронските мрежи** – значи, сите модели од длабоко учење се невронски мрежи, но не сите невронски мрежи се "длабоки".

Длабоко учење се однесува на **невронски мрежи со многу скриени слоеви** – затоа се вика "длабоко". Овие мрежи се многу моќни и можат:

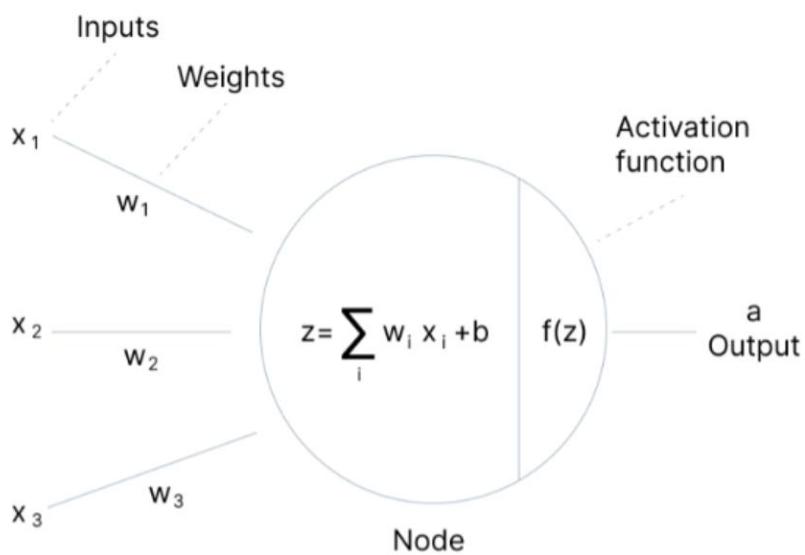
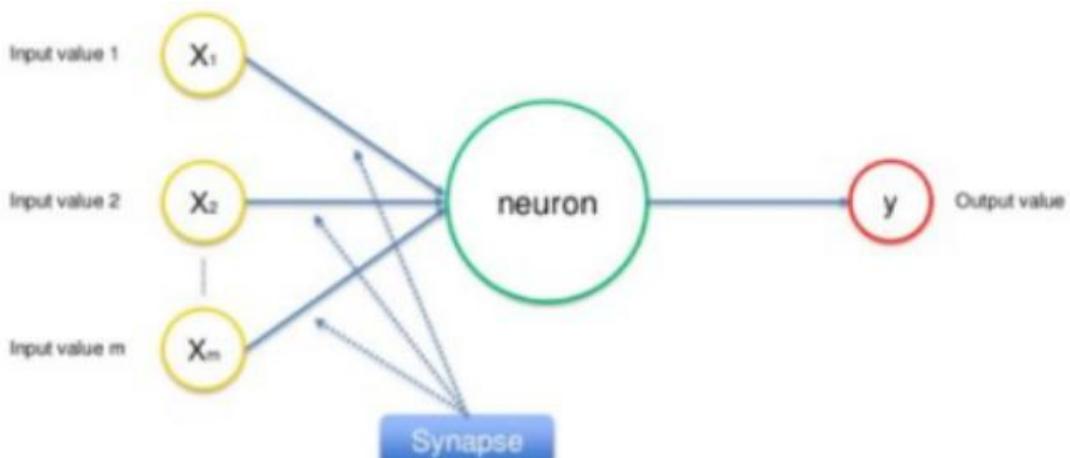
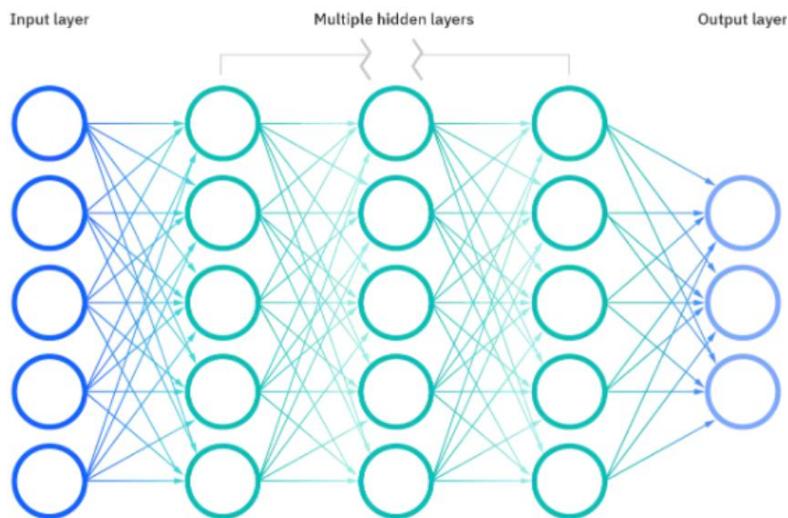
- да препознаваат сложени шеми,
- да обработуваат слики, видео и аудио,
- да преведуваат текстови, генерираат говор, итн.

◆ Примери на длабоко учење:

- **Convolutional Neural Networks (CNNs)** – за обработка на слики,
- **Recurrent Neural Networks (RNNs)** – за анализа на временски податоци (на пр. говор, текст),
- **Transformers (како ChatGPT)** – за природен јазик и генерација на текст.



## Анатомија на невронските мрежи



Секој јазол, или неврон, е поврзан со друг и има соодветна тежина и праг.

**Невроните** се основната единица на една невронска мрежа. Кога невронот ќе се активира, тој ја пресметува својата состојба така што ги собира сите влезни вредности помножени со нивните соодветни тежини. Но, невроните секогаш имаат уште еден дополнителен влез – **пристрасноста (bias)**.

Сите неврони во една мрежа се поделени во три групи:

- **Влезни неврони** – примаат информации од надворешниот свет
- **Скриени неврони** – ја обработуваат таа информација
- **Излезни неврони** – даваат некаков заклучок или резултат

**Активиската функција** одлучува дали еден неврон треба да се активира или не. Тоа значи дека таа одредува дали влезот на невронот е важен за предвидување, користејќи едноставни математички операции.

Улогата на активиската функција е да изведе излезна вредност врз основа на збир на влезни вредности што се внесуваат во еден јазол (или слој).

**Најчести активиски функции:** Sigmoid, Tanh, ReLU, leaky ReLU, Generalized ReLU, MaxOut, Softplus, swish.

**Sigmoid** – ја претвора вредноста во број меѓу 0 и 1. Добра за веројатности, но може да "заспие" (споро учење при екстремни вредности).

**Tanh** – слична на Sigmoid, но вредностите се меѓу -1 и 1. Почесто се користи бидејќи е посиметрична.

**ReLU (Rectified Linear Unit)** – враќа 0 за негативни вредности, а истата вредност за позитивни. Брза и многу популарна.

**Leaky ReLU** – слична на ReLU, но дозволува мала негативна вредност наместо 0 (за да не "умира" невронот).

**Generalized ReLU** – проширување на ReLU со прилагодливи параметри.

**MaxOut** – избира најголема вредност од неколку влезови. Моќна, но потешка за пресметка.

**Softplus** – мазна верзија на ReLU, никогаш не дава точно 0.

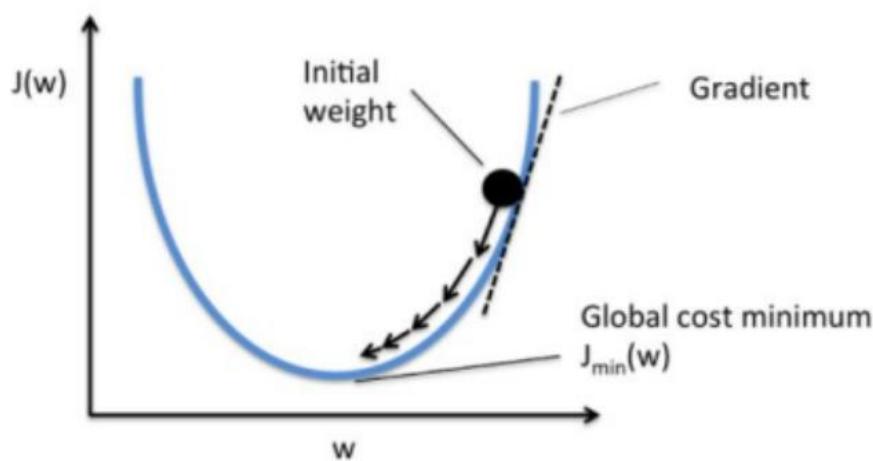
**Swish** – комбинација од влезот и sigmoid функција, често работи подобро од ReLU во сложени модели (го користи и Google).

## Gradient descent

Градиентскиот спуст е начин машината да учи од грешките и постепено да се подобрува.

- Машината прави предвидување → гледа колку погрешила (грешка).
- Градиентскиот спуст ѝ кажува **во која насока и колку да ги смени параметрите** за следниот пат помалку да греши.
- Ова се повторува многу пати додека **грешката не се намали што е можно повеќе**.

**Цел:** Да се најдат такви вредности (параметри) со кои моделот прави **најточни можни предвидувања**.



**Оската  $w$**  претставува тежините (weights) – параметрите што ги менуваме во моделот.

**Оската  $J(w)$**  е функцијата на трошок (cost function) – таа покажува колкава е грешката на моделот за дадена тежина  $w$ .

**Сината крива** е графикот на грешката во зависност од тежината. Целта е да стигнеме до **најниската точка** на таа крива – таму е **најмала грешка**, т.е. **оптимални параметри**.

**Црната точка е почетната тежина** – од таму почнува учењето.

**Црната стрелка (Gradient)** покажува насоката на градиентот (каде расте грешката).

**Црните кратки стрелки надолу** покажуваат **насоката во која се движиме** – спротивна од градиентот – за да ја намалиме грешката.

**Global cost minimum** е најдолната точка – таму сакаме да стигнеме.

**Перцепtronот е наједноставниот неврон во невронска мрежа. Тој:**

1. Прима влезови (inputs) – на пример: бројки од некои податоци.
2. Ги множи со тежини (weights) – секој влез има важност.
3. Ги собира (сума).
4. Додава пристрасност (bias).
5. Применува активациска функција – одлучува дали "ќе се активира" или не.
6. Враќа резултат (излез/output).

Овој процес се нарекува:

### **Forward Propagation (Пропагација нанапред)**

Ова е првиот чекор:

Податоците влегуваат, минуваат низ мрежата и добиваме излез. Секој неврон пресметува резултат и го праќа на следниот слој. Така, информацијата се движи нанапред низ мрежата – од влезот до излезот.

### **Backpropagation (Повратна пропагација)**

Ова е вториот чекор, што се случува по forward propagation:

Се мери грешката (разликата меѓу вистински и предвиден резултат). Се оди наназад низ мрежата и се пресметува како да се сменат тежините за да биде моделот по-точен.

Тежините се ажурираат со помош на градиентски спуст, така што мрежата учи од грешките.

**Loss function** – ја мери разликата т.е грешката меѓу предвидувањата од моделот и вистинските вредности.

Output Type	Output Distribution	Output layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary Cross Entropy
Discrete	Multinoulli	Softmax	Cross Entropy
Continuous	Gaussian	Linear	MSE (Mean Squared Error)
Continuous	Arbitrary	-	-

**Невронските мрежи работат подобро и побрзо ако се нормализираат податоците (средна вредност 0 и стд.девијација 1)!**

**Batch Normalization** е техника која се користи во невронски мрежи за да се забрза и стабилизира учењето.

Ги нормализира влезовите на секој слој, односно ги доведува до сличен опсег (со слична средна вредност и варијанса). Се применува за секој мини-беч (мал дел од податоците што се користи за една итерација во тренингот).

## **Regularization (Регуларизација)**

Регуларизацијата е техника што ја користиме во процесот на учење на моделите (за да го направиме моделот **подобро да се прилагодува на нови, невидени податоци** — односно, да има помала грешка при генерализација).

Кога моделот е премногу сложен, тој може да научи „премногу добро“ т.е да се прилагоди на податоците од тренинг сетот, вклучувајќи и шум или грешки во нив. Ова се нарекува **overfitting** (прекумерно прилагодување). Регуларизацијата помага да се спречи ова прекумерно прилагодување.

Регуларизацијата **додава некоја промена или ограничување на алгоритамот**, која го прави моделот да биде „поизмерен“ и да не се фокусира прецизно на сите детали од тренинг податоците. На пример, тоа може да значи дека ќе се „наградува“ да има помали тежини во мрежата, или да се казнува премногу сложен модел.

**Заклучок:** Регуларизацијата ја намалува грешката при тестирање или при внес на нови податоци, спречува overfitting и ја регулира сложеноста на моделот.

### **Видови регуларизација:**

**L1 регуларизација:** Прави некои тежини да се точно нула, па моделот користи помалку карактеристики.

**L2 регуларизација:** Ги прави тежините помали, но не ги брише, за да се избегне претренирање.

**Dropout:** Случајно „гаси“ дел од невроните за време на тренинг, па мрежата учи поотпорно.

**Early Stopping:** Се запира тренингот порано, кога моделот почнува да се влошува на нови податоци, за да не се претренира.

# Advanced neural networks

Главни видови на невронски мрежи:

## 1. Feedforward Neural Networks (FFNN)

- Наједноставни, сигналот оди само од влез кон излез (без враќање назад).
- Се користат за класични задачи како класификација и регресија.

## 2. Convolutional Neural Networks (CNN)

- Специјализирани за обработка на слики и видеа.
- Користат филтри (конволуции) за да извлечат важни карактеристики.

## 3. Recurrent Neural Networks (RNN)

- Можат да обработуваат секвенции и временски серии.
- Имаат меморија и ја паметат претходната информација.

## 4. Long Short-Term Memory (LSTM)

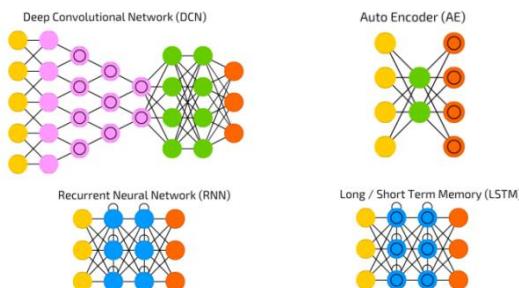
- Вид на RNN, но подобро ја чува и контролира долгорочната меморија.
- Често се користи за јазик, превод, и аудио.

## 5. Generative Adversarial Networks (GANs)

- Се состојат од две мрежи што се „борат“ — една создава (генерира) нови податоци, а другата проверува дали се вистински.
- Се користат за генерирање слики, видеа и други податоци.

## 6. Autoencoders

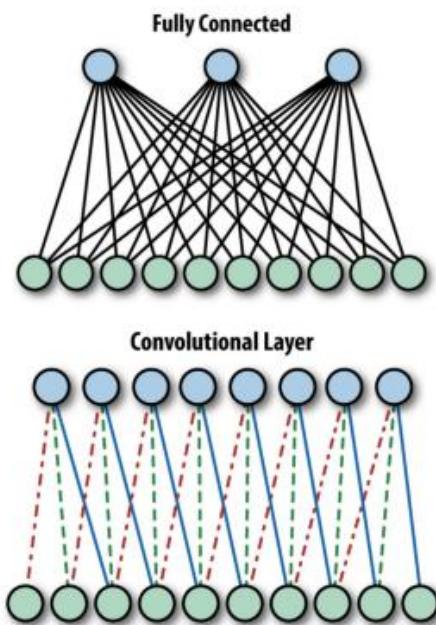
- Мрежи кои учат да ја реконструираат влезната информација.
- Се користат за намалување на димензионалност и откривање аномалии.



Во целосно поврзаниот слој (**fully connected layer**), секој неврон е поврзан со сите неврони од претходниот слој.

Во конволуцискиот слој (**convolutional layer**), секој неврон е поврзан само со ограничен број неврони во локален дел од претходниот слој.

Дополнително, во конволуцискиот слој, сите неврони ги користат истите тежини за овие врски, што е прикажано со истиот стил на линиите.



## CNN

Инпутот може да има голема димензија (бр. на атрибути), па користењето на целосно поврзана НМ би користела голем број на параметри.

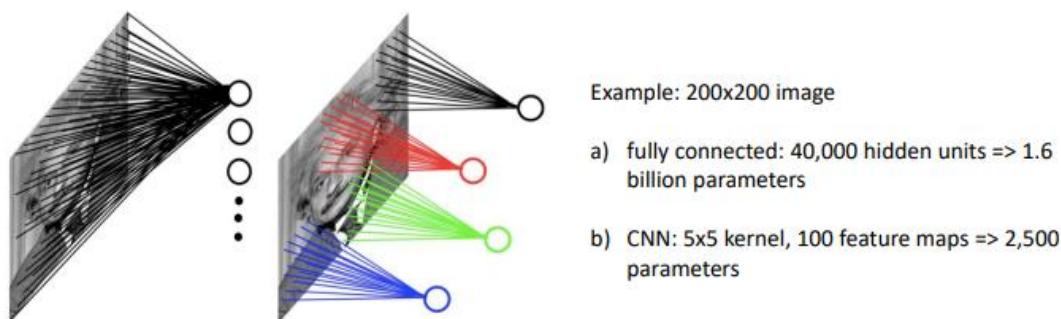
Инспирирано од експериментите на Hubel и Wiesel (1962), **CNN се специјален тип на НМ** каде што скриените слоеви се поврзани само со рецептивното поле. Бројот на параметри кај CNN е многу помал.

**Рецептивното поле** е мал дел од влезните податоци (на пример, дел од сликата) на кој е „насочен“ еден неврон во конволуциска мрежа.

Наместо невронот да гледа во целата слика одеднаш, тој гледа само мала локална област (на пример, 5x5 пиксели).

Тоа поле му овозможува на невронот да „извлече“ локални карактеристики (како линии, рабови или текстури) од таа мала област.

Потоа, повеќе неврони со поместени рецептивни полиња заедно го покриваат целото поле на сликата.



## Три фази на конволуциски слој

### 1. Convolution stage (Конволуција)

Се применува филтер (kernel) на влезните податоци, кој „се пролизгува“ преку сликата или претходниот слој и извлекува локални карактеристики (на пример, рабови, текстури).

Резултатот е нова „карактеристична мапа“ (feature map).

### 2. Nonlinearity (Нелинеарност)

На излезот од конволуцијата се применува нелинеарна функција, како што е **ReLU** (Rectified Linear Unit) или **tanh**.

Ова ја прави мрежата способна да учи сложени и нелинеарни модели, затоа што без ова, конволуциите би биле само линеарни операции.

### 3. Pooling (Пулинг)

Ова е чекор за намалување на големината на излезот и за собирање важни информации. Најчести се:

**Max pooling** – го избира најголемиот број од мала област (на пример 2x2 пиксели).

**Average pooling** – пресметува просек од мала област.

Со пулинг се добива резиме на локалните информации и се намалува бројот на параметри, што ја прави мрежата поефикасна.

## Пример 1: CNN за слика

Класификација на слика (пр: препознавање дали е куче или маче)

**Влез:** Слика со големина 100x100 пиксели и 3 бои (RGB)

**Како работи:**

Конволуциски слој користи филтри (на пример, 5x5) кои „ги гледаат“ малите делови од сликата и извлекуваат карактеристики како рабови, бои, текстури.

Повторени слоеви од конволуција и пулинг ја намалуваат големината, но ја зголемуваат комплексноста на карактеристиките.

На крајот, целосно поврзан слој ги класифицира сликите во категории (куче, маче).

## Пример 2: CNN за текст

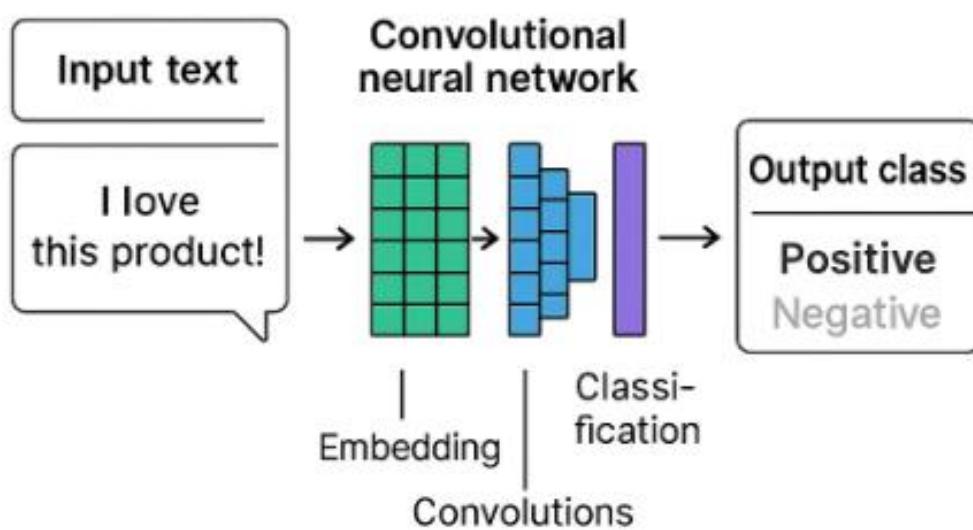
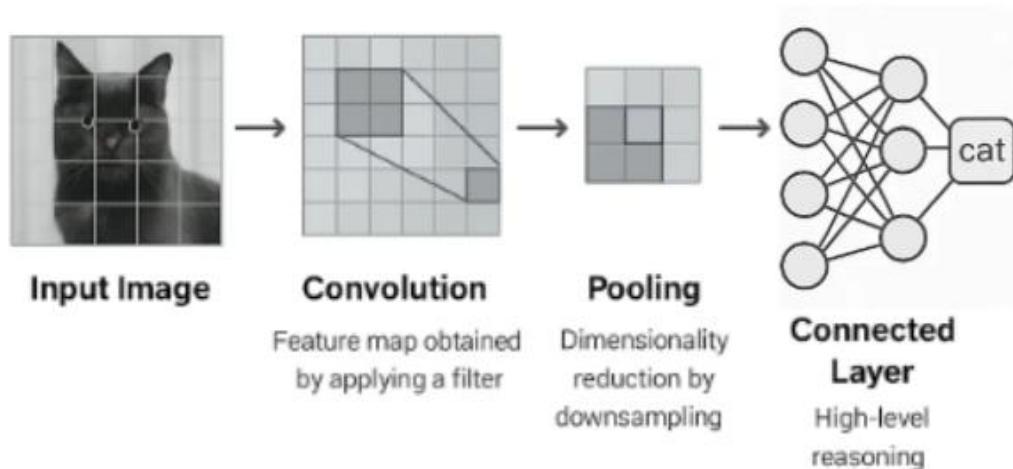
Класификација на текст (пр: дали е порака позитивна или негативна)

**Влез:** Ред од зборови, претставени како низа од бројки (векторска репрезентација, на пример word embeddings)

**Како работи:** Конволуциски филтри поминуваат преку мал број збора (на пример, 3 последователни збора) за да фатат локални шаблони или важни комбинации (на пример, „многу добар“ или „не ми се допаѓа“).

Пулинг слоевите ги собираат најважните карактеристики од целиот текст.

На крајот, спој за класификација дава резултат (позитивен/негативен текст).



## Autoencoders

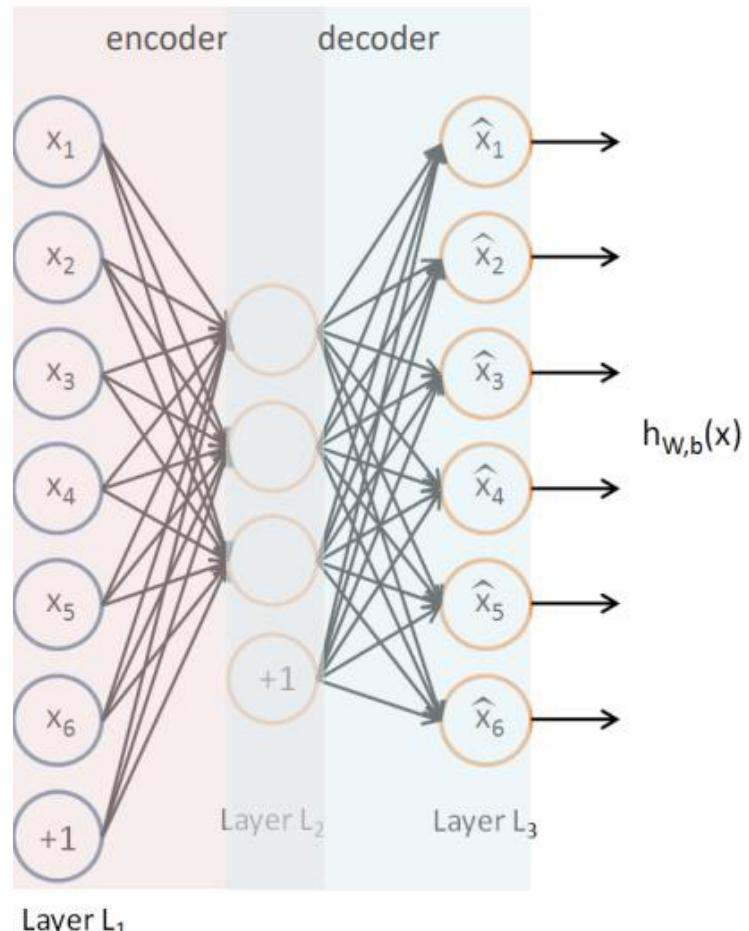
Автоенкодер е тип на **feedforward невронска мрежа** која учи да го „препознае“ својот влез и да го репродуцира истиот на излезот. Со други зборови, целта ѝ е да ги копира влезните податоци на излезот:

$$y^{(i)} = x^{(i)}$$

Мрежата е составена од два дела:

**Енкодер (Encoder)** — го зема влезот и го компресира (собира) во помала, скриена претстава (hidden layer). Ова е делот од влезот до скриениот слој.

**Декодер (Decoder)** — го користи компресираниот код од скриениот слој за да ја реконструира оригиналната информација на излезот (делот од скриениот слој до излезот).



## Deep Autoencoders

Длабок автоенкодер е автоенкодер кој има **повеќе скриени слоеви** во делот на енкодерот и во делот на декодерот. Тоа значи дека информацијата поминува низ повеќе слоеви додека се компресира и додека се реконструира.

### Gradient vanishing problem (Проблем со „изчезнување“ на градиентот)

Кога мрежата има многу слоеви, при процесот на учење (особено при враќање назад на грешките, backpropagation), **градиентите** (кои помагаат да се прилагодат тежините) стануваат многу мали.

Поради тоа, тежините во првите слоеви многу тешко се менуваат, па мрежата учи многу бавно или воопшто не учи добро.

Ова е **еден од главните предизвици при учење на многуслојни (длабоки) невронски мрежи**.

## На невронските мрежи им треба меморија!

Кога работиме со секвенци (редови од податоци), како текст или говор, многу е важно мрежата да памети претходни информации за да разбере контекстот.

На пример, во реченица, значењето на зборот зависи од зборовите пред него.

### **Sequence modelling (Моделирање на секвенции)**

Се користи за задачи каде редоследот на податоците е важен, како текст, говор, или времески серии.

#### **Named Entity Recognition (NER) — Препознавање именувани ентитети**

Задачата е да се најдат и означат специфични зборови или фрази во текстот кои се име на луѓе, места, организации, и слично.

На пример, во реченицата:

„Apple“ може да биде име на компанија,

„London“ е локација,

„John“ е име на човек.

#### **Како работи со невронска мрежа:**

**Влез:** реченица претставена со низа броеви (на пример, векторски репрезентации на зборови).

Мрежата ја процесира секвенцата и со меморија (на пример, со RNN или LSTM) ја памети претходната информација.

**Излез:** за секој збор, мрежата предвидува дали тој е име на човек, локација, организација или нешто друго.

### Recurrent Neural Network (RNN)

Во RNN, **излезот од скриениот слој (hidden layer)** не се заборава, туку се **чува во меморија** и се користи повторно како дел од влезот за следниот чекор.

Така, мрежата има меморија за претходните информации, што ѝ овозможува да работи со **секвенци и редоследи**.

#### **Како функционира:**

За секој елемент  $X_i$  од влезната низа (на пример, збор во реченица), мрежата генерира излез  $Y_i$ , но тој излез зависи не само од тековниот влез  $X_i$ , туку и од сите претходни влезови  $x_1, x_2, \dots, x_{i-1}$ . Истата мрежа се користи повторно и повторно за секој елемент во низата (постојано ги обновува своите скриени состојби).

## Long Short-Term Memory (LSTM)

**LSTM** е посебен тип на невронска мрежа, подобрена верзија на класичниот RNN, која има специјални „ќелии“ (cells) со механизми за контролирање на информацијата.

Овие ќелии содржат **gates** – “врати” кои решаваат кога да:

- примат нова информација (input gate)
- заборават (избришат) стара информација (forget gate)
- испратат информација напред како излез (output gate)

### Зошто е важен LSTM?

Класичните RNN имаат проблем со „изчезнување на градиентот“ (gradient vanishing) при учење на долги низи, што го отежнува паметењето на информации кои се далеку во минатото.

LSTM ги решава овие проблеми така што може да **памети важни информации подолго време** и да ги заборави неважните.

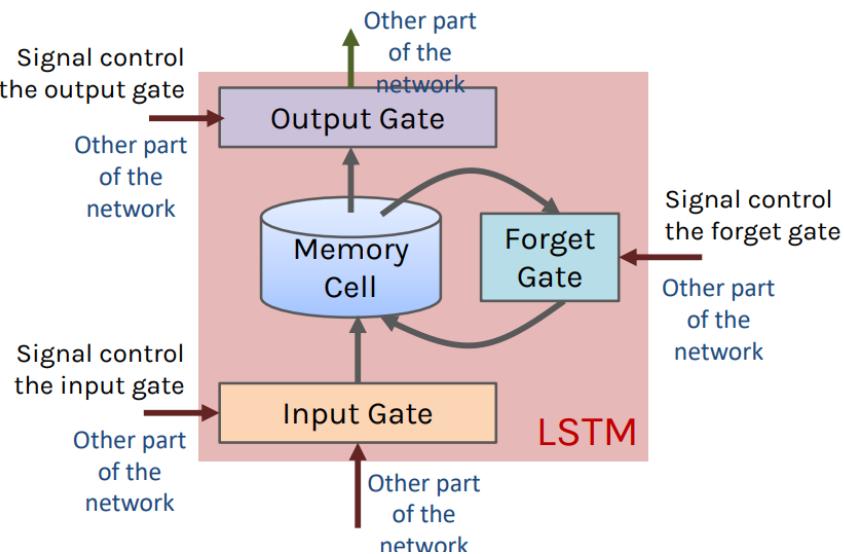
### Како работи?

Секој LSTM ќелиски модул има 4 главни влезови (вклучувајќи и претходната состојба) и 1 излез.

Gates се базираат на **сигмоидна функција (f)** што одлучува дали ќе ги „отвориме“ или „затвориме“ вратите (вредности од 0 до 1).

Преку овие врати, мрежата учи кога да зачува, кога да избрише, а кога да пренесе информација.

Special Neuron: 4 inputs & 1 output



# NLP – Natural Language Processing

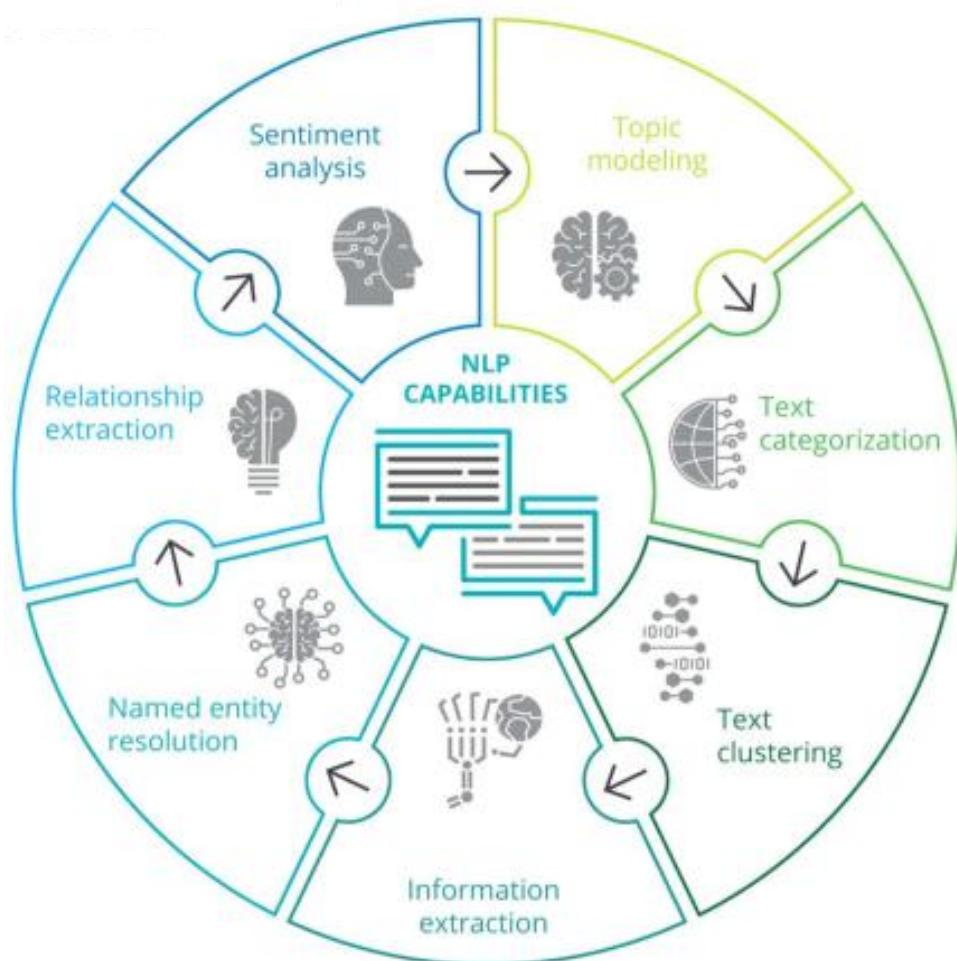
NLP е поле во КН и ВИ кое се фокусира **на интеракцијата помеѓу човечкиот јазик и компјутерот**. Целта е да се овозможи компјутерите да го разбираат, обработуваат и генерираат природниот јазик – да се имитира човечко разбирање.

**Примери за NLP:** Spell check, Autocomplete, Speech to text, Spam filter, Related keywords on search engines, Siri, Alexa, Google Assistant...

**Како бизнисите може да го користат NLP:**

- Подобрување на корисничко искуство со autocomplete, spell check, autocorrect
- Chatbots – автоматизирана поддршка
- Анализирање на feedback од купувачите

Седум клучни техники на NLP:



## **1. Sentiment Analysis (Анализа на сентимент)**

Процес на одредување на емоционалниот тон на текстот – дали е позитивен, негативен или неутрален.

**Пример:** Рецензии на филмови, коментари на социјални мрежи.

## **2. Topic Modeling (Моделирање на теми)**

Автоматско откривање на скриени теми во збирка на текстови, без потреба од рачно обележување.

**Пример:** Групирање на вести според теми како политика, спорт, технологија.

## **3. Text Categorization (Категоризација на текст)**

Доделување на текстови во претходно дефинирани категории.

**Пример:** Е-пошта во спам или неспам, новост во бизнис, забава и слично.

## **4. Text Clustering (Кластерирања на текст)**

Групирање на слични текстови заедно, без однапред дефинирани категории.

**Пример:** Групирање на документи со слична содржина без да се знае темата однапред.

## **5. Information Extraction (Извлекување информации)**

Автоматско вадење на структуриран податок од неструктурен текст како факти, датуми, локации, имиња.

**Пример:** Од новинарска статија да се извлече: Име: Петар, Локација: Скопје, Дата: 5 јуни 2025.

## **6. Named Entity Recognition (NER) – Препознавање на именувани ентитети**

Идентификување и класификација на посебни имиња во текстот, како лица, компании, места, датуми.

**Пример:** Во реченицата „Apple отвори канцеларија во Берлин“ – Apple е организација, Берлин е локација.

## **7. Relationship Extraction (Извлекување на односи)**

Наоѓање на релации меѓу ентитети во текстот.

**Пример:** Од реченицата „Илон Маск е основач на Tesla“ се извлекува релација: Илон Маск - основач - Tesla.

### Предизвици кај NLP:

- Зборот/реченицата може да имаат повеќе значења
- Тешко препознавање на сарказам и иронија
- Потребни се специфични модели за различни јазици
- Голема димензионалност на зборови

## Основни поими кај NLP:

**Синтакса** – граматичка структура на реченицата.

**Семантика** – значењето на зборовите/фразите/речениците.

**Part-of-Speech (POS)** – идентификување на граматичката категорија на зборот (именка, придавка, глагол...)

**Bag-of-Words (BoW)** – метод со кој се претставува колку пати се појавува одреден збор во реченица/текст (или бинарно – дали се појавува), игнорирајќи го редоследот.

**N-gram** – секвенца од N зборови/знаци во текстот. (најчесто користено 2-5 )

Unigram – секој збор одделно.

Bigram – парови од 2 последователни зборови

Trigram – тројки од 3 последователни збора итн...

## Bag of words Featurization

Создава нумерички репрезентации на текстуални податоци, при што секој збор е мапиран во фреквенцијата на неговата појава. Најчест начин на репрезентација на текст кај машинско учење.

**Едноставен пример:** 1. Кучето лае.

2. Мачката седи.

Вокабулар = [кучето, лае, мачката, седи]

Фреквенции: 1. [1, 1, 0, 0]

2. [0, 0, 1, 1]

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

## Еден од предизвиците е големината на репрезентацијата.

Бидејќи зборовите следат дистрибуција по законот на степен (power-law), големината на речникот расте речиси линеарно со големината на корпусот. Бидејќи ретките зборови не се појавуваат доволно често за да го информираат моделот, вообичаено е да се отфрлат невообичаените зборови што се појавуваат, на пример, помалку од 5 пати.

## N – grams

Каде BoW се губи редоследот на зборовите и се намалува значењето на реченицата. Две реченици може да имаат различно значење, а ист BoW вектор.

### Карakterистики на N-grams

Во машинското учење со текстуални податоци, честопати е корисно да се користи комбинација од повеќе n-грам карактеристики – на пример: униграми (единечни зборови), биграми (парови од последователни зборови), и триграми (тројки од зборови).

Униграмите се појавуваат почесто и помагаат при откривање на послаби, пошироки влијанија врз текстот. Од друга страна, биграмите и триграммите можат да „фатат“ посилни, но поконкретни значења.

На пример, изразот „**the white house**“ има значење што се разликува од збирот на значењата на поединечните зборови „**the**“, „**white**“, „**house**“, или дури и од биграмите „**the white**“ и „**white house**“ земени одделно.

Кога n-grams се комбинираат со техниката BoW, тие значително ја подобруваат точноста на моделите за класификација и други задачи со текст. Типично, ваквите комбинации ги даваат следниве резултати по точност:

**3-грамови < 2-грамови < 1-грамови < 1+2-грамови < 1+2+3-грамови**

Тоа значи дека најдобри резултати се добиваат кога се користат сите три (униграми, биграми и триграми) заедно. Подобрувањето на точноста најчесто е неколку проценти – доволно значајно за реални апликации.

### Големина на N-grams

Користењето на n-грамови носи предизвици во однос на големината на збирот на карактеристики (feature set). Ако првичната големина на речникот е  $|V|$ , тогаш бројот на можни комбинации за:

- биграми е  $|V|^2$
- триграми е  $|V|^3$

Среќна околност е што природниот јазик има фреквенциска структура според **power-law** – што значи дека најголемиот број од n-грамовите што се појавуваат се прилично чести.

Затоа, речник што ги содржи најчестите n-грамови ќе опфати голем дел од сите што ќе се сретнат во текстот.

### Пример за големини на речници со n-грамови:

- Речник со униграми: 40.000 зборови
- Речник со биграми: 100.000 комбинации
- Речник со триграми: 300.000 комбинации

Со овие големини, можно е да се опфати **повеќе од 80%** од карактеристиките што се појавуваат во текстуалните податоци.

### N-grams како јазични модели

N-grams можат да се користат за градење статистички модели на текст. Кога се користат на овој начин, тие се нарекуваат **n-gram јазични модели (n-gram language models)**.

Овие модели доделуваат **веројатност на секој n-грам**, така што **збирот на веројатностите за сите можни n-грамови со иста големина (n)** е еднаков на 1.

Со ваков модел, може да се пресмета **вкупната веројатност или веројатноста на појавување на одредена реченица**, врз основа на веројатностите на нејзините поединечни n-грамови.

### Skip-grams

Можеме да го анализираме значењето на одреден збор со тоа што ќе ги разгледаме контекстите во кои се појавува. Контекстот е збирот на зборови што се појавуваат во близина на тој збор, односно на одредено растојание: ... -3, -2, -1, +1, +2, +3, ... во секоја реченица каде што се среќава зборот.

**Skip-gram** е група од **неконсеквентни зборови** (со зададено растојание), кои се појавуваат во некоја реченица. Наместо да гледаме само зборови веднаш до дадениот збор, skip-gram овозможува да се вклучат и зборови кои се „прескокнати“ со одреден број позиции.

На овој начин, може да се изгради **BoSG – „bag of skip-grams“** репрезентација за секој збор, базирана на табела со skip-grams. Оваа техника овозможува побогато и пофлексибилно претставување на контекстот околу зборовите.

## PoS (Part of Speech) – Дел на говорот

**PoS** се однесува на **граматичката категорија** на зборовите во реченицата – на пример:

- именки (nouns)
- глаголи (verbs)
- придавки (adjectives)
- прилози (adverbs)
- заменки (pronouns)
- и други.

Овие категории ни помагаат да разбереме **улогата и функцијата** на зборот во контекст на реченицата.

## PoS Tagging – Означување на делови на говорот

**PoS tagging** претставува **процес на автоматско доделување на граматичка категорија** (т.е. дел на говорот) на секој збор во даден текст.

Пример:

„The dog runs fast.“

PoS ознаки:

- **The** – детерминатор (DT)
- **dog** – именка (NN)
- **runs** – глагол (VBZ)
- **fast** – прилог (RB)

Овој процес е важен во обработката на природен јазик (NLP) бидејќи помага во:

- разбирање на синтаксата и структурата на речениците,
- анализа на значењето,
- подобрување на машинското преведување, одговарање на прашања, и многу други задачи.

## Named Entity Recognition (NER)

**Named Entity Recognition (NER)** или **препознавање на именувани ентитети** е важна техника во обработката на природен јазик (NLP) која служи за **препознавање и класификација на клучни информации во текстот**.

NER автоматски го пронаоѓа и класифицира текстот во **одредени категории**, како на пример:

- **Имиња на лица** (пр. *Elon Musk*)
- **Имиња на организации** (пр. *Google, United Nations*)
- **Локации** (пр. *Skopje, Europe*)
- **Датуми и времиња** (пр. *8 June 2025, yesterday*)
- **Парични износи** (пр. *\$1000*)
- **Производи, настани, бренди, итн.** ( зависи од моделот)

**Пример:**

Текст: "Barack Obama was born in Hawaii in 1961."

NER ќе го препознае:

- **Barack Obama** → Person
- **Hawaii** → Location
- **1961** → Date

Зошто е важен NER?

**Го структуира неструктуриниот текст.**

Се користи во **пребарување на информации, чат-ботови, машинско преведување, анализи на документи, препознавање на субјекти во новинарски статии, итн.**

## Grammars (Граматики)

Во компјутерската лингвистика и NLP, **граматиката** е формален систем на правила што дефинира **како може да се состави валидна реченица од еден јазик**.

Најчесто се користи **синтаксичка граматика**, која опишува:

кои структури се дозволени (на пр. подмет + прирок),

како зборовите и фразите можат да се комбинираат.

Пример од англиска граматика (во Backus–Naur форма – BNF):

$S \rightarrow NP\ VP$

$NP \rightarrow Det\ N$

$VP \rightarrow V\ NP$

$Det \rightarrow "the" \mid "a"$

$N \rightarrow "cat" \mid "dog"$

$V \rightarrow "chased" \mid "saw"$

Ова значи дека реченицата ( $S$ ) се состои од именска фраза ( $NP$ ) и глаголска фраза ( $VP$ ), итн.

## **Recursion in Grammars (Рекурзија во граматики)**

**Рекурзијата** е важен и моќен концепт во граматиките. Таа значи дека **правило може да се повикува самото себе**, што овозможува создавање на **неограничено долгии сложени реченици**.

Пример:

$NP \rightarrow NP\ PP$

$PP \rightarrow P\ NP$

Овде, именската фраза ( $NP$ ) може да содржи предлошка фраза ( $PP$ ), која пак содржи нова именска фраза ( $NP$ ).

Со ова, можеме да добиеме реченици како:

"The cat on the mat in the room under the stairs slept."

Ова покажува **рекурзивна структура** – фраза во фраза во фраза – што е многу природно за човечкиот јазик.

Зошто е ова важно во NLP?

- За парсирање на реченици (разбирање на структурата).
- Во машинско преведување и гласовно разбирање.
- За генерирање на природен јазик (на пр. во chatbots или генеративни системи).

## Word2Vec

**Word2Vec** е популарен алгоритам во обработката на природен јазик (NLP) кој се користи за **претставување на зборови како вектори од броеви**. Оваа техника го претвора текстот во **бројчена форма**, така што машините можат полесно да го обработат и „разберат“ значењето на зборовите.

Word2Vec претставува секој збор како **вектор во повеќедимензионален простор**, така што:

- зборови со **слично значење** ќе бидат **блиску** еден до друг во векторскиот простор,
- а зборови со различно значење ќе бидат подалеку.

Пример:

„king“ – „man“ + „woman“ ≈ „queen“

Word2Vec користи **невронска мрежа со една скриена (hidden) слој** и се тренира на голем текстуален корпус. Постојат две главни архитектури:

1. **CBOW (Continuous Bag of Words):**

→ предвидува го средниот збор, знаејќи ги зборовите од околината.

2. **Skip-gram:**

→ зема еден збор и предвидува ги зборовите што се околу него (во контекстот).

Обезбедува **семантички смислени вектори** (односно, „разумно“ разбирање на значењето на зборови),

Може да се користи за **класификација на текст, кластерирање, машинско преведување, препознавање на слични зборови**, итн.

## Ограничувања на Word2Vec

Иако Word2Vec добро ги пресметува сличностите меѓу зборови, има ограничувања:

- Најголемиот проблем е што **секој збор добива само една векторска репрезентација**, без разлика во каков контекст се појавува.
- На пример, зборот „**bank**“ може да значи:
  - речен брег (*river bank*)
  - инвестициска банка (*investment bank*)

Но Word2Vec ќе создаде **единствен просечен вектор** за зборот, кој **не го претставува добро ниту едно од значењата**.

## Решенија и напредок

Поради оваа слабост, беа развиени **покомплексни техники со контекстуални вгнездувања на зборови** (*Contextual Word Embeddings*), како:

- **ELMo (Embeddings from Language Models)** – кој го зема во предвид контекстот на зборот во реченицата.

Подоцна се појавија **Transformers**, кои донесоа голем напредок:

- **BERT** (Bidirectional Encoder Representations from Transformers)
- **GPT** (Generative Pre-trained Transformer)

Овие модели создаваат различна репрезентација на зборот во зависност од неговиот контекст, што овозможува подлабоко и попрецизно разбирање на јазикот.

## Transfer learning (Преносно учење)

Во традиционалното супервизирано учење, треба да имаме **многу означени податоци** за да изградиме добар модел. Но во реалниот свет, тоа често не е возможно.

**Преносното учење** решава овој проблем така што:

- **искористува знаење стекнато од друга слична задача или домен,**
- и го **пренесува** за да се подобри моделот за новата задача, каде што има малку или никакви означени податоци.

## Предности на Transfer Learning

- Помага при решавање на **комплексни задачи** со ограничени ресурси.
- Работи и кога има **многу малку означени податоци**.
- Овозможува **пренос на знаење** од еден модел/домен на друг.
- Се смета како чекор кон **вештачка општа интелигенција (AGI)**.
- Создава **поиздржливи модели** што можат да решаваат повеќе задачи.

## Тренинг на трансформери

Трансформерите (како BERT, GPT и сл.) обично поминуваат низ два чекора:

1. **Предтренирање (pretraining)** – на големи, необележани текстуални податоци.
2. **Фино прилагодување (fine-tuning)** – на мали, специфични обележани податоци за конкретна задача.

Задачи што се користат во овие чекори:

- предвидување на следна реченица
- одговарање на прашања
- анализа на сентимент
- разбирање на текст (reading comprehension)
- парафразирање

## Примена на трансформери

Трансформерите се најчесто користени во NLP.

Популарни модели:

- GPT-2, GPT-3
- BERT
- XLNet
- RoBERTa

Овие модели имаат примена во:

- машински превод (на пр. английски ↔ македонски),
- сумирање на документи,
- генерирање на текст,
- Named Entity Recognition (NER),
- анализа на биолошки секвенци (на пр. ДНК анализа)

# Natural Language Processing and Transformers – 2

**Јазичен модел (Language Model)** претставува модел кој го опишува јазикот што го користи одреден ентитет или контекст, како на пример:

- Поединечен говорник или автор
- Одреден жанр (на пр. правен, медицински, социјални мрежи)
- Одредена област или домен (на пр. академско пишување, неформални разговори)

**Формална дефиниција:** Јазичниот модел пресметува веројатност за било која секвенца од зборови.

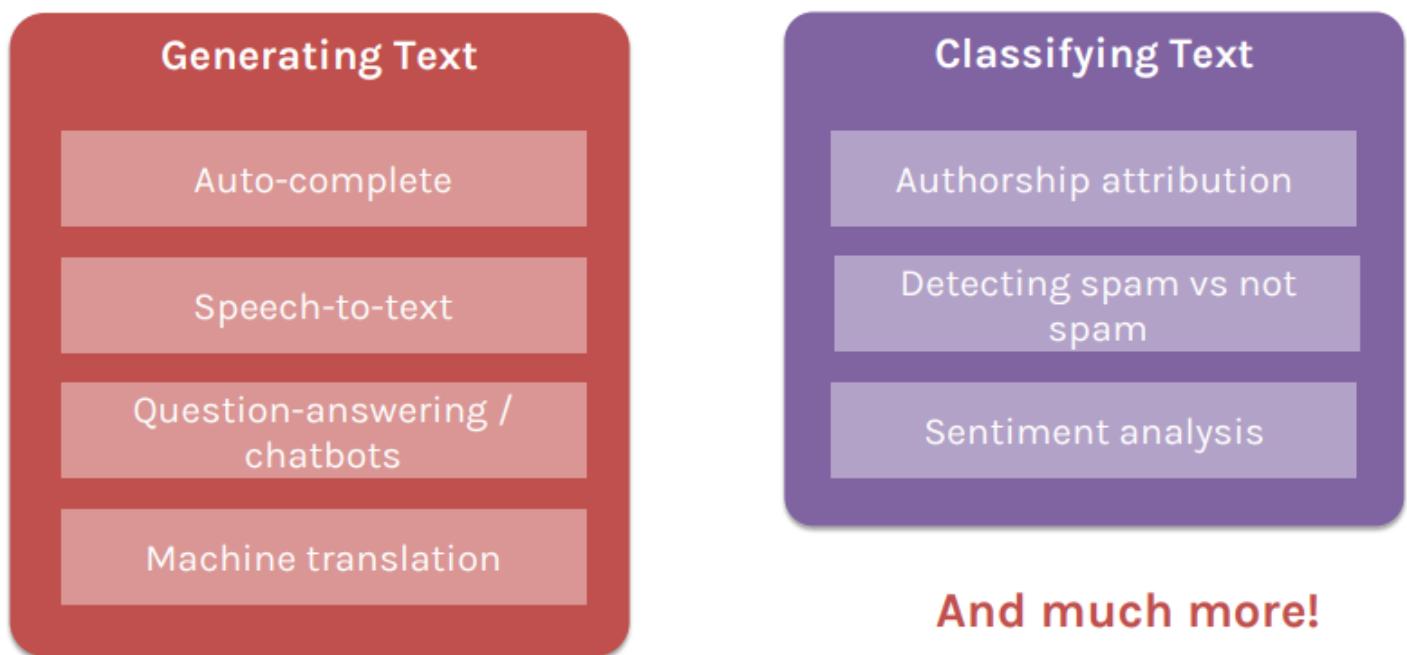
На пример, нека  $X$  = „Елени задоцни на час“.

Тогаш, веројатноста што ја проценува јазичниот модел е:

$$P(X)=P(\text{"Елени задоцни на час"})$$

Ова значи: **Колкава е веројатноста оваа реченица да се појави во јазикот на дадениот ентитет или домен.**

**Примери:** Генерирање текст (autocomplete, google translate), Класификација на лажни вести...



## Language Modelling: Биграми (Bigrams)

Како можеме да изградиме јазичен модел?

Еден начин е со **биграмски модел**, кој е **алтернативен пристап** на едноставниот униграм модел. Наместо да ги гледаме зборовите поединечно, **набљудуваме парови од последователни зборови**.

### **Биграмски модел (Bigram Model)**

Во биграмскиот модел, **веројатноста на целата реченица** се пресметува како производ од **веројатностите на секој збор врз основа на претходниот збор**.

**Пример:**

Нека реченицата е: „**Елени задоцни на час**“

Биграмскиот модел пресметува:

$$P(\text{"Елени задоцни на час"}) = P(\text{"Елени"}) \cdot P(\text{"задоцни"} | \text{"Елени"}) \cdot P(\text{"на"} | \text{"задоцни"}) \\ \cdot P(\text{"час"} | \text{"на"})$$

Значи, моделот гледа **секој збор во контекст на претходниот збор**.

### **Како функционира?**

1. **Ги броиме сите парови зборови** (биграми) во даден корпус (збирка на текстови).
2. Се пресметува условна веројатност:

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

каде  $w_i$  е тековниот збор, а  $w_{i-1}$  претходниот.

#### **Предности:**

Полесен и побрз од комплексни модели.

Ја фаќа зависноста помеѓу зборови (за разлика од униграм моделот кој ги смета зборовите независни).

#### **Недостатоци:**

Не зема подлабок контекст (само еден претходен збор).

Проблем со **нулта веројатност** ако парот зборови никогаш не се појавил  
→ затоа се користат техники како **smoothing**.

## Language modeling: RNN

**Рекурентната невронска мрежа** е тип на невронска мрежа специјално дизајнирана за **обработка на секвенци од податоци**, како што се реченици.

### **Структура на RNN:**

**Влезен слој (Input layer):** Ги прима зборовите еден по еден како вектори (на пр.  $x_1, x_2, x_3, \dots$ ).

**Скриен слој (Hidden layer):** Задржува „меморија“ за претходниот контекст преку повратна врска (рекуренција). Се ажурира со секој нов збор.

**Излезен слој (Output layer):** Генерира веројатност за следниот збор ( $\hat{y}_1, \hat{y}_2, \dots$ )

### **СИЛНИ СТРАНИ на RNN:**

**Може да обработува секвенци од произволна должина,** не само фиксна големина (како Feedforward Neural Networks).

**Задржува контекст** преку скриената состојба – т.е., „меморија“ за тоа што дошло претходно.

**Истите тежини се користат за секој влез,** што значи редоследот на зборовите се задржува, а бројот на параметри е помал.

### **ПРОБЛЕМИ со RNN:**

**Споро тренирање** – се користи техника наречена *Backpropagation Through Time (BPTT)*, која е посложена.

**Градиентите лесно исчезнуваат или експлодираат,** особено кога секвенците се долги (познато како *vanishing/exploding gradients*).

**Тешко задржување на долгочен контекст** – RNN имаат ограничена способност да „се сеќаваат“ на зборови кои се многу далеку во минатото.

## Language modeling: LSTM

**LSTM** е посебен тип на **рекурентна невронска мрежа (RNN)**, дизајниран да се справи со **долгорочни зависности во текстот** — нешто со што „обичните“ (vanilla) RNN имаат проблем.

**Проблем кај обичните RNN:** Во класичен RNN, скриената состојба (hidden state) се ажурира со секој нов збор — што значи дека старите информации многу лесно се „бришат“ и градиентите исчезнуваат при тренирање.

LSTM додава специјален дел наречен "мемориска ќелија" (memory cell), која трајно складира информации низ времето и може да:

- задржи важни информации на долг рок
- заборави непотребни информации
- додаде нови информации кога е потребно

LSTM има неколку „врати“ (gates) – тие се мали невронски мрежи кои одлучуваат што да задржат, што да избришат и што да додадат:

1. **Forget Gate** – одлучува што да се „заборави“ од меморијата
2. **Input Gate** – одлучува што ново да се додаде
3. **Output Gate** – одлучува што да се испрати како излез

#### Предности:

Речиси секогаш има подобри резултати од обичните (vanilla) RNN-и

→ LSTM подобро ја „разбира“ структурата на реченицата и контекстот.

#### Многу добро „фаќа“ долгорочни зависности

→ На пример: знае дека зборовите „ако“ и „тогаш“ се поврзани иако се далеку еден од друг во реченицата.

#### Недостатоци:

Има повеќе тежини за учење (parameters)

→ Поради комплексната структура со повеќе „врати“, LSTM има **повеќе параметри** од обичен RNN.

Бара повеќе податоци за тренирање

→ Ако имаш малку податоци, обичен RNN може да даде подобри резултати (поради помалата комплексност).

Сè уште може да страда од исчезнување или експлозија на градиентите

→ Иако е многу подобар од RNN, не е целосно имун на овие проблеми при долги секвенци.

## Bi-directional RNN / LSTM

**Идеја:** Мрежата ги чита податоците и од двете насоки

- Една RNN/LSTM оди од лево кон десно (нормално)
- Друга RNN/LSTM оди од десно кон лево (назад)

На крај, двете информации се спојуваат и даваат поцелосен контекст.

### **Предности:**

**Подобро разбирање на контекстот (и од минатото и од иднината)**

**Поголема точност во NLP задачи, како:**

- Тагирање на делови на говорот (POS tagging)
- Named Entity Recognition (NER)
- Машински превод
- Анализа на сентимент

### **Недостатоци:**

**Поспоро тренирање** (две мрежи наместо една)

**Поголема мемориска потрошувачка**

Не е погоден за **стриминг податоци во реално време**, бидејќи мора да ги знае и идните зборови

**Пример:** „Марко не дојде на училиште затоа што беше болен.“

Ако само гледаме од лево кон десно, до зборот „затоа што“ не знаеме дали причината е добра или лоша.

Но ако **ги гледаме и зборовите после „затоа што“**, добиваме подобро разбирање.

## ELMo (Embeddings from Language Models)

### **Општа идеја на ELMo:**

Целта е да се добијат **богати, контекстуализирани претставувања (embeddings)** на зборовите.

Наместо секој збор да има **фиксен вектор** (како кај Word2Vec), ELMo му дава **различен вектор на зборот во зависност од контекстот**.

## ELMo: Stacked Bi-directional LSTMs

### Како работи?

Користи **двонасочни LSTM мрежи** → ги зема и претходните и следните зборови во предвид.

Користи **повеќе слоеви (stacked)** → секој слој учи сè покомплексни и поапстрактни значења.

На крајот ги **комбинира сите скриени состојби (hidden layers)** од различните слоеви.

Овие комбинирани претставувања се **оптимизираат за конкретна задача**, како:

- Класификација на сентимент
- Named Entity Recognition
- Превод и сл.

**Пример:** Зборот "bank" ќе има различна векторска претстава во:

- "She sat by the **bank** of the river."
- "He went to the **bank** to deposit money."

Благодарение на контекстот, ELMo може да разбере разликата.

## Sequence-to-Sequence (Seq2Seq) модели

### Зошто ни треба Seq2Seq?

Кога преведуваме реченица збор по збор, често добиваме лоши преводи, бидејќи:

- Не го гледаме целиот контекст.
- Не ја земаме предвид структурата и дужината на целата реченица.

### Целта на Seq2Seq моделите:

Да се справат со:

- Превод на реченици од една јазична структура во друга
- Резимирање, автоматски дијалози, генерирање реченици, итн.

## Како работи Seq2Seq?

Моделот се состои од два дела:

### 1. Енкодер RNN (Encoder)

Го чита целото влезно речење (на пример, на английски). Ја претвора целата секвенца во една контекстуална векторска претстава (се нарекува **context vector** или **hidden state**). Пример: „I am hungry“ се претвора во context\_vector

### 2. Декодер RNN (Decoder)

Почнува од context\_vector и генерира излезно речење (на пример, на француски), збор по збор. Генерира зборови сè додека не стигне до посебен симбол <EOS> (end of sequence). При тестирање, секој излезен збор ( $\hat{y}_i$ ) станува влез за следниот чекор.

### Тренирање (Training):

Се користи ист метод како кај RNN: **Backpropagation Through Time (BPTT)**. Се мери грешката на декодерот (на пример, колку далеку е преводот од вистинскиот).

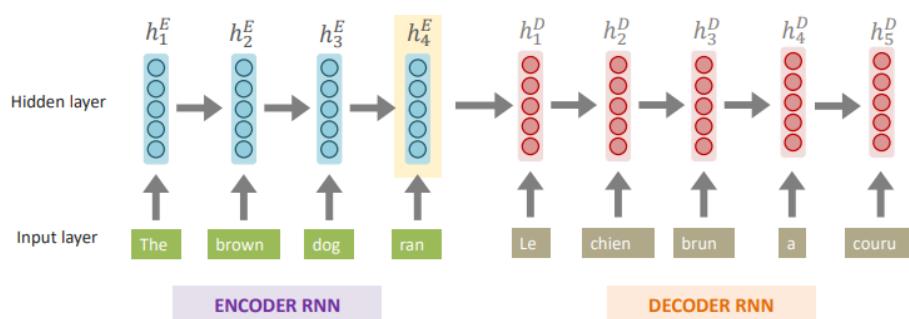
Се ажурираат тежините на двете мрежи – од крајот на декодерот до почетокот на енкодерот.

### Предности:

- Може да работи со секвенци од различна должина (на пример, 5 зборови влез  $\rightarrow$  7 зборови излез)
- Може да фати долгорочни зависности меѓу зборовите
- Флексибilen е за превод, дијалог, резиме, chatbot, итн.

### Ограничување:

Се потпира на една единствена скриена состојба како „меморија“ – што не е секогаш доволно (ова ќе го надмине attention механизмот кај модерните модели како Transformer/BERT).



## Attention механизам во Seq2Seq модели

Што ако декодерот може **селективно да се фокусира** на делови од влезната секвенца при секој чекор?

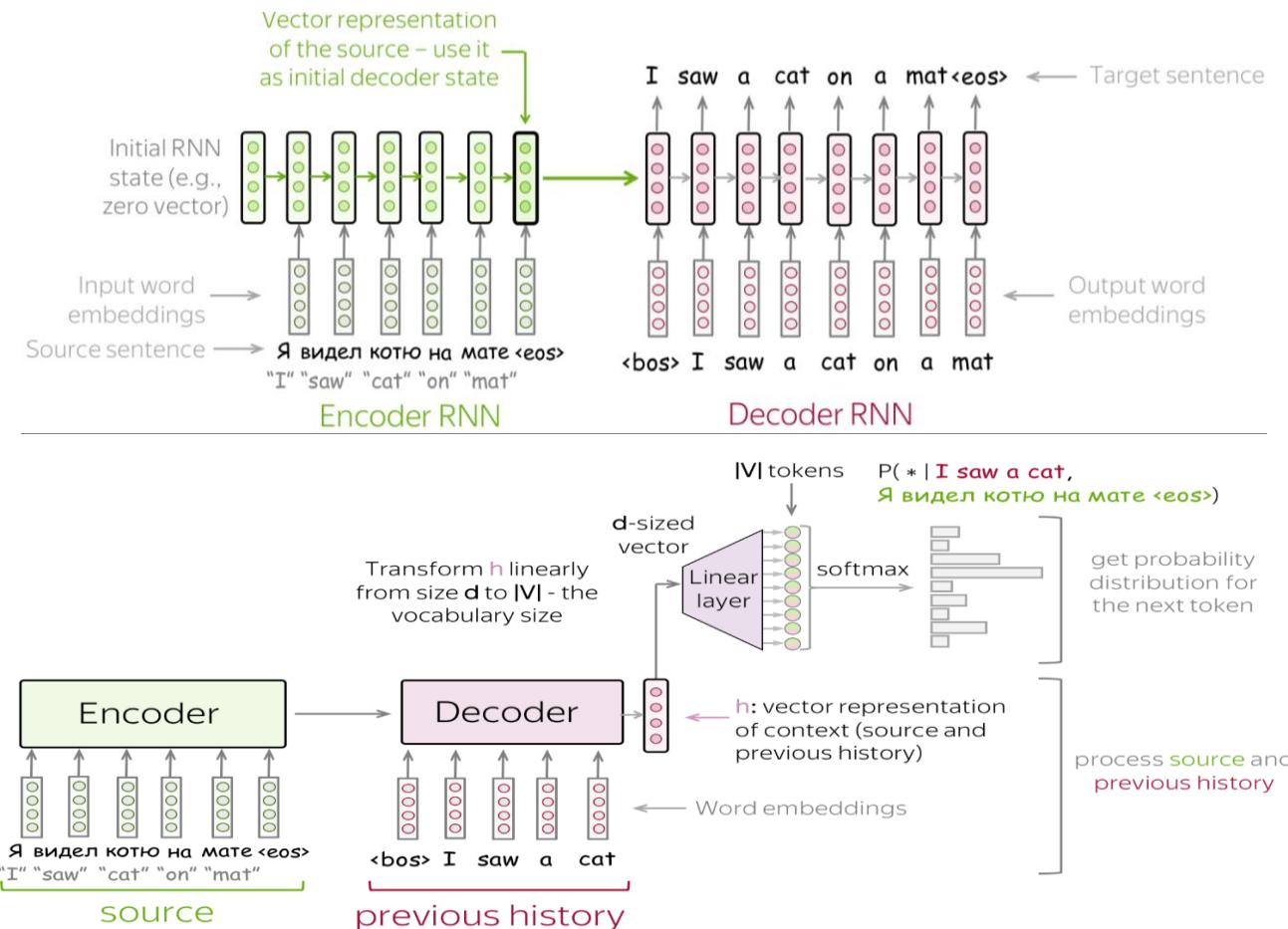
**Attention** овозможува декодерот да **доделува различни тежини на сите скриени состојби** од енкодерот.

Овој механизам создава динамична, тежинска комбинација од енкодер излезите, со што го насочува декодерот **да ја нагласи најрелевантната информација**.

Со тоа што се фокусира на вистинскиот контекст во вистинско време, attention помага да се:

- Обработуваат долги секвенци поефикасно
- Подобри точноста кај задачи како превод и резимирање
- Разјасни нејаснотии преку искористување на целиот влезен контекст

**Attention значително ги подобрува резултатите на Seq2Seq моделите**



Модел	Контекст	Справување со долги секвенци	Меморија/Состојба	Тип на излез	Предности	Недостатоци
Биграми	1 збор (претходен збор)	Лошо	Нема	Статистичка веројатност	Едноставен, брз за тренирање	Игнорира долгочен контекст, не разбира значење
RNN	Цела претходна секвенца	Ограничено	Скриена состојба $h$	По еден збор во секвенца	Може да обработува секвенци од променлива должина	Проблеми со исчезнувачки/експлодирачки градиенти
LSTM	Цела претходна секвенца	Добро	Скриена состојба $h$ , меморија $c$	По еден збор во секвенца	Подобро справување со долгочни зависимости	Повеќе параметри, бара повеќе податоци и време за тренирање
ELMo	Цела реченица (бидирекционално)	Одлично	Stack од bi-LSTM + тежинска комбинација	Векторска претстава на зборови	Богати, контекстуализирани вгнездувања	Голем, не е генеративен, скап за тренирање
Seq2Seq	Цела влезна секвенца	Добро	Енкодер-декодер, фиксен контекстен вектор	Цела излезна секвенца (разл. должина )	Поддржува влез/излез од различна должина, корисен за превод, дијалози и сл.	Целиот контекст е компримиран во еден вектор (освен ако нема attention)
Attention	Цела влезна секвенца (динамички и фокус)	Многу добро	Варијабилна тежинска комбинација од состојби	Подобрено Seq2Seq излез	Подобрување на долгочен контекст, визуализација на вниманието	Се користи како додаток; самостојно не е модел
Transformer	Цела секвенца, паралелно	Одлично	Self-attention наместо рекурзија	Цела секвенца истовремено (маскирана)	Брз, држи глобален контекст, state-of-the-art во NLP	Бара многу ресурси за тренирање, комплексен

## **Што е трансформер?**

Трансформер е вид на модел за обработка на податоци, најчесто користен за јазик (текст), кој го подобрува работењето на претходните модели како што се RNN и LSTM.

## **Зошто е важен трансформерот?**

Проблемите кај RNN и слични модели се:

- Тие тешко се справуваат со „долги врски“ во текстот (на пример, ако нешто е кажано на почетокот, да се поврзе со нешто на крајот од текстот).
- Тренингот трае долго затоа што секој дел од влезот се обработува последователно (еден по еден).
- Тешко е да се искористи моќта на современите компјутери кои сакаат да работат паралелно.

Трансформерот решава овие проблеми.

## **Како работи трансформерот?**

### **Влез:**

- Влезот е текст (на пример, реченица).
- Текстот се претвора во броеви (векторски претстави на зборови), наречени ембединг (embedding).

### **Клучна идеја: „Attention“ (внимание)**

- Трансформерот користи нешто што се вика „self-attention“ или самовнимание.
- Ова значи дека моделот гледа на секој збор во реченицата и го споредува со сите други зборови за да разбере колку тие се поврзани или важни едни за други.
- На пример, во реченицата „Кучето што лаеше беше големо“, зборот „кучето“ е поврзан со „лаеше“ и „големо“. Моделот ќе обрне внимание на овие врски.

### **Паралелна обработка:**

- За разлика од RNN кој обработува збор по збор, трансформерот гледа цела реченица одеднаш и ги обработува сите зборови паралелно.
- Ова го прави тренирањето многу побрзо.

## Архитектура

Трансформерот има два дела:

### 1. Encoder (Енкодер)

- Го зема влезниот текст и го претвора во внатрешна претстава (скриени вектори) што ги содржи сите важни информации и врски меѓу зборовите.
- Секој слој во енкодерот ги користи self-attention механизмите и мали вештини на учење (feed-forward мрежи) за подобро разбирање.

### 2. Decoder (Декодер)

- Го користи резултатот од енкодерот за да креира излез (на пример, превод на друг јазик или продолжување на текст).
- Декодерот исто така користи attention, но има и механизам за „последователно“ креирање на зборови (генерира збор по збор).

## Инпут и аутпут пример

Ако сакаме да го преведеме „Јас сакам јаболко“ на англиски:

- **Инпут:** „Јас сакам јаболко“ (кодерот го обработува целиот текст одеднаш)
- **Внатрешна претстава:** трансформерот разбира дека „Јас“ е субјект, „сакам“ е глагол, „јаболко“ е објект.
- **Аутпут:** „I want an apple“ (декодерот го генерира излезот збор по збор, користејќи ја информацијата од кодерот).

## Зошто трансформерите се толку моќни?

- Ги наоѓаат најважните врски во текстот без разлика на должината.
- Можат да се тренираат побрзо затоа што користат паралелна обработка.
- Се основа на вниманието кое е многу флексибилно и ја прави машината да „разбира“ подобро контекстот.
- Поради овие причини, денес најmodерните јазични модели како GPT, BERT, T5 се базираат на трансформери.

## GPT-2 и BERT

Трансформерот има два дела:

- **Encoder** (енкодер) кој ја обработува влезната информација (на пример текст)
- **Decoder** (декодер) кој создава излез (на пример превод или генерирање на текст) користејќи ја информацијата од кодерот.

Ако немаме влезен текст, туку **само сакаме да генерираме „следен збор“** еден по еден (на пример автоматско пишување на реченица), можеме да го отстраниме енкодерот и да користиме само декодерот за тоа. Оваа архитектура ја користи GPT (Generative Pre-trained Transformer).

Ако сакаме само **да обучиме јазичен модел кој ќе ни помогне во други задачи** (на пример препознавање на значење во текст), ни не треба декодерот, туку само енкодерот. Оваа верзија е BERT (Bidirectional Encoder Representations from Transformers).

Трансформер = кодер + декодер

GPT = само декодер (генерира текст последователно)

BERT = само енкодер (разбирање на текст)

## Unsupervised learning

**Unsupervised Learning** е област од машинското учење каде немаме обележани излезни податоци. Овде целта е да се најдат скриени шеми, структури или кластери во податоците. Алгоритмите од овој тип се користат кога сакаме да го анализираме начинот на кој податоците природно се групираат или да намалиме димензионалност, без претходно да знаеме како треба да изгледаат резултатите.

**Кластеринг** е основна техника во unsupervised learning. Тоа претставува процес на групирање на слични објекти така што објектите во истата група (кластер) се што е можно послични еден со друг, а различни од објектите во другите групи. Примери за примена на кластеринг се групирање на документи, слики, биолошки примероци, или кориснички однесувања.

**Постојат неколку главни типови на кластеринг:** partitioning (како што е K-means), hierarchical (агломеративен и дивизивен), и density-based (како DBSCAN). Секој од нив има различна примена и се однесува поинаку во присуство на шум, неправилни форми или кластери со различна густина.

**Во partitioning кластерингот, како K-means,** однапред одредуваме колку кластери сакаме да добиеме. Секој кластер има центар, наречен центроид, и секоја точка се доделува на најблискиот центроид. Процесот се повторува: се пресметуваат новите центроиди според новите групи, сè додека кластерите не престанат да се менуваат. Овој алгоритам е едноставен, но може да биде чувствителен на иницијалните вредности и не работи добро со кластери од различна големина, форма или густина.

**За да се подобри ова, се користи K-means++** методот за подобра иницијализација на центроидите. Наместо случаен избор, се избираат точки кои се најдалеку од веќе избраните, со што се подобрува стабилноста и точноста на алгоритмот.

**Hierarchical clustering** гради хиерархија од кластери и обично се претставува со dendrogram (дрвовидна структура). Најчест метод е агломеративниот (bottom-up), каде секоја точка започнува како посебен кластер, а потоа се спојуваат најблиските парови. Различни методи постојат за мерење на „блискост“ меѓу кластери: single-link (најблиска точка), complete-link (најдалечна точка), average-link (просек на сите растојанија), и Ward's метод (зголемување на вкупна грешка).

Овој тип на кластеринг има предност што не мора однапред да се знае бројот на кластери, но има и недостатоци како висока временска и просторна сложеност и осетливост на шумови.

**Density-based clustering (DBSCAN)** дефинира кластер како област со висока густина. Секој кластер се состои од core точки (со доволно блиски соседи), border точки (во близина на core, но не и самите core), и noise точки (точки што не припаѓаат на ниту еден кластер). Алгоритмот работи без да бара однапред број на кластери и е многу добар со неправилни форми и outliers. Но, има проблеми при различна густина или висока димензионалност на податоците.

**За да се оцени квалитетот на кластерите, се користат различни метрики.** На пример, **SSE** (Sum of Squared Errors) мери колку точките се близку до центарот на нивниот кластер. **Silhouette Score** мери сличност на точка со својот кластер во однос на другите. Понекогаш се користат и визуелизации или корелација на similarity матрици.

**Autoencoders** се невронски мрежи кои учат да го реплицираат влезот како излез. Тие го компресираат влезот во мала латентна претстава (bottleneck) и потоа ја реконструираат оригиналната слика или податок. Се користат за репрезентативско учење, детекција на аномалии и генеративно моделирање. Основниот autoencoder има encoder, bottleneck и decoder. Подобри варијанти вклучуваат deep autoencoders, convolutional autoencoders (за слики), overcomplete и variational autoencoders (VAE), кои користат статистички модели за подобра генерација.

Autoencoders можат да се обучуваат на податоци без ознаки и се добар пример за self-supervised learning, бидејќи целта е да се научи претставување од самиот податок. Latent просторот што се добива може да се користи за кластеринг или други задачи.

**Во практика, unsupervised learning се користи кога имаме многу податоци, но малку знаеме за нивната структура.** Техниките како кластеринг и автоенкодери овозможуваат разбирање, визуелизација и дури и генерација на нови податоци.

### Клучни поими:

- **Unsupervised learning** – тип на машинско учење каде што нема излезни ознаки; моделот сам наоѓа шеми во податоците.
- **Clustering** – процес на групирање слични податоци во кластери.
- **K-means** – алгоритам за кластеринг со однапред зададен број на кластери; се базира на растојание до центроиди.
- **K-means++** – подобрување на K-means преку паметен избор на почетни центроиди.

- **Hierarchical clustering** – градење хиерархија на кластери, прикажани преку дендрограм.
- **Agglomerative clustering** – тип на hierarchical кластеринг што започнува со поединечни точки и ги спојува.
- **DBSCAN** – алгоритам базиран на густина, што не бара однапред зададен број на кластери и открива шумови.
- **Core, Border, Noise points** – категории во DBSCAN.
- **SSE (Sum of Squared Errors)** – метрика за мерење на компактноста на кластери.
- **Silhouette Score** – метрика за проценка на квалитетот на кластеринг.
- **Autoencoder** – невронска мрежа што учи да го репродуцира влезот и создава скриена претстава (latent space).
- **Latent space** – компактна, скриена репрезентација на влезните податоци.
- **Variational Autoencoder (VAE)** – верзија на автоенкодер што моделира латентниот простор како веројатносна распределба.