

1. Вовед во машинско учење

Што е машинско учење (ML)?

- Гранка на вештачката интелигенција која се фокусира на развој на алгоритми што овозможуваат на компјутерите да ги подобрят своите перформанси врз основа на искуство.
- Наместо рачно програмирање правила, ML ги користи **податоците** за автоматско учење на шаблони и носење одлуки.

💡 Формална дефиниција:

"Програмата учи од искуство (E) во однос на класата на задачи (T) и мерката за перформанси (P), ако нејзините перформанси на T , како што е измерено со P , се подобруваат со E ."

Традиционално програмирање vs. Машинско учење

- Во традиционалното програмирање, човекот дефинира **правила** и го пишува кодот што обработува податоци и произведува излез.
- Во машинското учење, алгоритмот **сам ги учи правилата** од податоците.

2. Типови на машинско учење

(1) Надгледувано учење (Supervised Learning)

- Дадени се етикетирани податоци** (пример: слики од кучиња и мачки со етикети "куче" и "мачка").
- Моделот учи од овие примери за да предвидува во иднина.
- Поделено е на:
 - Класификација** (Classification) → предвидува категорија (пример: болен или здрав пациент).
 - Регресија** (Regression) → предвидува континуирана вредност (пример: цена на куќа според големина).

(2) Ненадгледувано учење (Unsupervised Learning)

- Нема етикети** во податоците, моделот треба **сам да открие структури**.
- Примери:

- **Кластеринг (Clustering)** → групирање на клиенти според купувачко однесување.
- **Димензионалност намалување** (PCA) → компресирање податоци со минимизирање на загуба.

(3) Учење со засилување (Reinforcement Learning)

- Моделот учи преку награди и казни.
- Најчесто се користи во **роботика, игри, финансии**.

📌 **Пример:** Обучување на компјутерски агент во шах → добива позитивни поени при добра одлука и негативни при грешка.

3. Клучни компоненти на ML алгоритмите

1. **Претставување (Representation)** – Како моделот ја структуира информацијата (пример: **неуронски мрежки, одлучувачки дрвја**).
 2. **Евалуација (Evaluation)** – Како мериме успешност (пример: **точност, прецизност, рекол**).
 3. **Оптимизација (Optimization)** – Како го подобруваме моделот (пример: **градиентен спуст**).
-

4. Примена на ML за носење одлуки

Пример: Одобрување на заем (Loan Approval)

- **Традиционален пристап:** Офицер ги проверува документите и носи одлука рачно.
- **ML пристап:** Модел учи од податоците и автоматски одлучува базирано на **кредитна историја, возраст, приходи**.

→ **Применет алгоритам: Логистичка регресија (Logistic Regression)**

- Моделот пресметува **веројатност** за враќање на заемот врз основа на **влезни карактеристики** (пример: кредитна линија, ниво на образование).
-

5. Предвидување со ML

(1) Линеарни vs. нелинеарни модели

- **Линеарни модели** (Linear Regression) → Кога постои **едноставна** врска меѓу предвидливите и зависните променливи.
- **Нелинеарни модели** (Neural Networks, Decision Trees) → Кога врската е **комплексна** и не може лесно да се претстави со права линија.

📌 **Пример:**

- **Линеарна регресија** → Цена на куќа базирано на квадратни метри.
 - **Нелинеарен модел** → Препознавање лица во слика.
-

6. Оценување на модели (Model Evaluation)

1. **Тренинг-тест поделба (Train-Test Split)** → Делиме податоци на **тренинг (80%)** и **тест (20%)** за да провериме како моделот генерализира.
 2. **Метрики за оценување:**
 - **Средна квадратична грешка (MSE, RMSE)** → Колку точни се предвидувањата.
 - **R² коефициент (R-squared)** → Колкав дел од варијансата во податоците моделот ја објаснува.
 - **Прецизност, рекол, F1-мерка** (за класификација).
-

7. Регресија и оптимизација

(1) Линеарна регресија

- Формула: $Y = \beta_0 + \beta_1 X + \varepsilon$
- Оценка на параметрите:
 - **Метод на најмали квадрати (OLS)**
 - **Градиентен спуст (Gradient Descent)** → Итеративно барање на најдобри вредности.

(2) Градиентен спуст

- Започнува со случајни вредности.

- Во секој чекор се движи во **најбрзиот пад** на грешката.
 - Контролирано со **learning rate (λ)**.
-

8. Класификација

Кога зависната променлива е **категориска**, користиме **класификациски алгоритми**.

(1) Логистичка регресија

- Место директна вредност, моделот предвидува **веројатност**.
- Формула: $p(X) = e^{\beta_0 + \beta_1 X}$ $p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$
- Го користи **Maximum Likelihood Estimation (MLE)** за наоѓање на параметри.

(2) Оценка на класификација

- **Матрица на конфузија (Confusion Matrix)** → Проверува точност на предвидувањата.
 - **ROC крива и AUC (Area Under Curve)** → Колку добро моделот разликува класи.
-

9. Заклучок

- ML е **моќен пристап** за анализа на податоци и носење одлуки.
- Клучните чекори се: **подготовка на податоци, избор на модел, оценување и оптимизација**.
- Постојат различни **алгоритми** за различни **типови проблеми** (класификација, регресија, кластеринг).

Ова е детална анализа на лекцијата "**Machine Learning 2**", базирана на PDF-от што го испрати. Ќе ги покријам сите главни точки, вклучувајќи ги **дефинициите, важните концепти и деталите од сликите**.

1. Вовед во лекцијата

Лекцијата обработува две главни теми:

1. Полиноминална регресија (Polynomial Regression)

- Проблемот со **overfitting** (преприлагодување).
- Методи за **регуларизација** (Regularization) за намалување на комплексноста на моделот.

2. Одлучувачки дрвја (Decision Trees)

- Како тие работат.
 - Критериуми за поделба на податоците.
 - Методи за **гранење и кастрење на дрвјата** (pruning).
-

2. Полиноминална регресија (Polynomial Regression)

Полиноминалната регресија е **нелинеарен модел** кој ја проширува линеарната регресија со додавање на повеќе членови од истата променлива, но со различни степени.

📌 Формула:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_M x_M + \epsilon$$
$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_M x_M + \epsilon$$

Каде што:

- M е степенот на полиномот.
 - $\beta_0, \beta_1, \beta_2, \dots, \beta_M$ – параметри кои ги учиме од податоците.
 - ϵ – грешката во моделот.
- ◆ **Предност:** Може да фати **нелинеарни** трендови во податоците.
- ◆ **Недостаток:** Ако степенот M е **многу висок**, може да дојде до **overfitting**.
-

3. Преприлагодување (Overfitting)

📌 **Overfitting** (Преприлагодување) – кога моделот **предобро ги учи** податоците од тренинг сетот, но не може да **генерализира** на нови податоци.

Како изгледа overfitting?

- График:
 - Линеарен модел (подпрост) – Не фаќа доволно информации од податоците.
 - Соодветен модел – Добро ја објаснува врската.
 - Overfitted модел – Прекумерно се прилагодува на точките од тренинг сетот.

📌 Решение за overfitting:

1. Крос-валидација (Cross-Validation) – Поделба на податоците на тренинг и тест сет.
 2. Регуларизација (Regularization) – Додавање на казнени термини за сложените модели.
-

4. Крос-валидација (Cross-Validation)

(1) K-Fold Cross Validation

- Податоците се делат на **K** еднакви делови.
- **K-1** делови се користат за тренирање, а последниот дел за тестирање.
- Ова се повторува **K пати** и се зема просекот од сите тестирања.

📌 Предности:

- Го намалува ризикот од **overfitting**.
 - Дава **пореална** проценка на моделот.
-

5. Регуларизација (Regularization)

Регуларизацијата е метод за намалување на сложеноста на моделите и спречување на overfitting.

(1) LASSO регресија (L1 Regularization)

📌 Додава казна за **апсолутната вредност** на параметрите.

$$LLASSO = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^J |\beta_j|$$

$$L_{\text{Ridge}} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^J \beta_j^2$$

- Овој метод може **некои параметри да ги направи 0**, што води до **селекција на најважните променливи**.

(2) Ridge регресија (L2 Regularization)

-  Додава казна за **квадратот** на параметрите.

$$L_{\text{Ridge}} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^J \beta_j^2$$

- Сите параметри се **намалуваат**, но никогаш не стануваат **нула**.

Избор на λ (регуларизациски параметар)

- Ако λ е **премал** → Регресијата е скоро иста како класична линеарна регресија.
 - Ако λ е **преголем** → Моделот е премногу поедноставен.
 - **Најдобар избор на λ** → Со крос-валидација.
-

6. Одлучувачки дрвја (Decision Trees)

-  **Одлучувачко дрво (Decision Tree)** – модел кој **дели податоци хиерархиски**, користејќи **IF-THEN правила**.

Како работи?

1. Започнува со **целата група на податоци**.
2. **Бира најдобар критериум за поделба** (пример: "Дали цената на куќата е > 200,000?").
3. Податоците се **делат во две гранки**.
4. Процесот се повторува додека не се достигне **стоп услов**.

(1) Поделба на податоците (Splitting Criteria)

- ◆ **Грешка при класификација (Classification Error)** → Мери процент на неточно предвидени податоци.
- ◆ **Gini индекс** → Колку е „нечиста“ групата по поделбата.

- ◆ **Информациска ентропија (Information Entropy)** → Мери колку информација се губи при поделбата.

❖ Пример:

Ако имаме категорија "Да" (Yes) и "Не" (No), тогаш:

- Ако сите податоци се „Да“, Gini = 0 (чист сет).
 - Ако има 50% „Да“ и 50% „Не“, Gini е најголем (најголема нечистота).
-

7. Преприлагодување на одлучувачки дрвја (Overfitting in Decision Trees)

Големи дрвја → Overfitting

Ако дрвото има **премногу гранки**, тоа **може точно да ги запамети податоците**, но нема да функционира добро на нови податоци.

❖ Решение:

1. **Рано стопирање (Early Stopping)** – Ограничивање на **максимална длабочина**.
 2. **Кастрење на дрвото (Pruning)**
 - **Пред кастрење (Pre-Pruning)** → Запирање на растење кога нема значителен напредок.
 - **После кастрење (Post-Pruning)** → Се креира комплетно дрво, а потоа **се отстрануваат неважни гранки**.
-

8. Заклучок

- Полиноминалната регресија може да фати нелинеарни трендови, но води до overfitting.
- Регуларизацијата (L1/L2) помага да се контролира комплексноста на моделот.
- Крос-валидацијата е критична за мерење на перформансите на моделот.
- Одлучувачките дрвја се **моќни, но лесно претренираат**, па затоа се користат техники како **pruning**.

Ова е **детална анализа** на лекцијата "**Machine Learning 3**", со објаснување на сите главни концепти, примери и имплементации.

1. Ensemble Models (Модели со ансамбли)

Ансамбл моделите комбинираат повеќе алгоритми за да ја зголемат **точноста и стабилноста** на предвидувањата. Главните техники се:

- **Bagging (Bootstrap Aggregating)**
 - **Random Forest**
 - **Boosting (XGBoost, AdaBoost)**
-

2. Bagging (Bootstrap Aggregating)

📌 Што е Bagging?

- Техника што **комбинира повеќе модели** со цел да **намали варијансата**.
- **Идеја:** Ако имаме **нестабилен модел** (пример: Decision Tree), можеме да **тренираме повеќе верзии на истиот модел**, но со **различни податоци** (избрани со "Bootstrap Sampling").
- **Краен резултат:** Просек на резултатите (за регресија) или **гласање на повеќето модели** (за класификација).

📌 Чекори во Bagging

1. Генерирање на различни податоци (**Bootstrap Sampling**)
2. Тренирање на повеќе модели (обично **Decision Trees**)
3. Агрегирање на резултатите (**Majority Voting / Averaging**)

Пример: Класификација на пациенти врз основа на медицински податоци

Ако сакаме да предвидиме дали пациент има одредена болест:

1. Креираме **100 Decision Trees**, секој трениран со **различен Bootstrap Sample**.
2. За нов пациент, **се предвидува класа од секое дрво**.

3. Финалната предвидена класа е онаа што најчесто се појавува (**Majority Voting**).

📌 Python имплементација

python

CopyEdit

```
from sklearn.ensemble import BaggingClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.model_selection import train_test_split  
from sklearn.datasets import load_iris  
from sklearn.metrics import accuracy_score
```

Податоци

```
iris = load_iris()  
  
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3,  
random_state=42)
```

Bagging со Decision Trees

```
bagging = BaggingClassifier(DecisionTreeClassifier(), n_estimators=50, random_state=42)  
bagging.fit(X_train, y_train)
```

Тестирање

```
y_pred = bagging.predict(X_test)  
  
print("Accuracy:", accuracy_score(y_test, y_pred))
```

✓ **Bagging ја намалува варијансата**, но не ја решава целосно корелацијата помеѓу дрвјата.

3. Random Forest

Што е Random Forest?

- Напредна верзија на Bagging која **додава уште една компонента – Random Feature Selection.**
- Во секој чекор, секое одлучувачко дрво **се гради со случајно избран подмножество на предиктори.**
- Ова спречува **доминирање на одредени карактеристики** и дополнително **намалува корелацијата** меѓу дрвјата.

Пример: Предвидување на цената на куќа

- Креираме **100 Decision Trees**, но секое дрво **гледа само дел од променливите** (пример: квадратура, број на соби, локација).

Python имплементација

python

CopyEdit

```
from sklearn.ensemble import RandomForestClassifier
```

```
# Random Forest со 100 дрвја
```

```
rf = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=42)  
rf.fit(X_train, y_train)
```

```
# Тестирање
```

```
y_pred = rf.predict(X_test)  
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Предности на Random Forest:

- Подобра **генерализација** од Bagging.
- **Работи добро со голем број карактеристики.**
- **Може да одреди важност на карактеристиките.**

4. Boosting

📌 Што е Boosting?

- Наместо да ги комбинира моделите **паралелно (како Bagging)**, Boosting ги комбинира **секвенцијално**.
- Новите модели учат **од грешките** на претходните модели.

Главни типови на Boosting

1. **AdaBoost (Adaptive Boosting)** → Прва варијанта на Boosting.
2. **Gradient Boosting (XGBoost, LightGBM, CatBoost)** → Најпопуларна техника денес.

📌 Како работи Boosting?

1. Првиот модел се тренира со **почетните тежини**.
2. Новиот модел се тренира на **грешките на претходниот модел**.
3. Овој процес се повторува повеќе пати.

📌 Python имплементација на Gradient Boosting

python

CopyEdit

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
# Gradient Boosting со 100 итерации
```

```
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3,  
random_state=42)
```

```
gb.fit(X_train, y_train)
```

```
# Тестирање
```

```
y_pred = gb.predict(X_test)  
  
print("Accuracy:", accuracy_score(y_test, y_pred))
```

-  Boosting има ниска варијанса, но може да има висок bias ако не се конфигурира правилно.
-

5. XGBoost (Extreme Gradient Boosting)

Што е XGBoost?

- Оптимизирана верзија на Gradient Boosting.
- Побрз, поефикасен и подобро го користи хардверот.

Карактеристики на XGBoost:

- L1/L2 регуларизација → Го спречува overfitting.
- Паралелно тренирање → Побрза обработка.
- Работи со големи податоци.

Python имплементација на XGBoost

python

CopyEdit

```
import xgboost as xgb
```

```
# XGBoost модел
```

```
xgb_model = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3,  
random_state=42)
```

```
xgb_model.fit(X_train, y_train)
```

```
# Тестирање
```

```
y_pred = xgb_model.predict(X_test)  
print("Accuracy:", accuracy_score(y_test, y_pred))
```

-  XGBoost е еден од најдобрите модели за табеларни податоци.
-

6. Bayes Classifier

📌 Што е Bayes Classifier?

- Пробабилистички класификатор кој **калкулира веројатности** за секоја класа.
- Го користи **Баесовиот теорем:** $P(C|X) = P(X|C)P(C)/P(X)$
- Ако **претпоставиме дека карактеристиките се независни**, го добиваме **Naïve Bayes Classifier**.

📌 Пример: Предвидување на е-пошта дали е спам

- Карактеристики: Број на одредени зборови во е-поштата.
- Баесовиот класификатор ќе **избере класата со највисока веројатност**.

📌 Python имплементација

python

CopyEdit

```
from sklearn.naive_bayes import GaussianNB
```

```
# Naïve Bayes модел
```

```
nb = GaussianNB()
```

```
nb.fit(X_train, y_train)
```

```
# Тестирање
```

```
y_pred = nb.predict(X_test)
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Naïve Bayes работи **добро со текст** и се користи за филтрирање на спам пораки.

Заклучок

Модел	Јака страна	Слаба страна
Bagging	Намалува варијанса	Не работи добро со многу променливи
Random Forest	Додава random feature selection	Бавен за големи податоци
Boosting	Најдобар за точност	Осетлив на overfitting
XGBoost	Брз и ефикасен	Потребен е fine-tuning
Naïve Bayes	Брз за текст	Прави погрешни претпоставки за независност

Ако имаш дополнителни прашања, пиши! 

40

Детална анализа на лекцијата „Deep Learning“

Оваа лекција покрива **вештачки невронски мрежи (Artificial Neural Networks - ANN)** и **длабоко учење (Deep Learning - DL)**, нивните компоненти, алгоритми и методи за тренинг.

1. Што се вештачки невронски мрежи (Artificial Neural Networks - ANN)?

📌 **Дефиниција:** ANN се **модели инспирирани од човечкиот мозок** кои обработуваат информации преку **поврзани неврони (nodes или units)**.

Клучни концепти во ANN:

- **Влезен слој (Input Layer)** → Ги прима податоците.
- **Скриени слоеви (Hidden Layers)** → Ги трансформира податоците со понатамошни пресметки.
- **Излезен слој (Output Layer)** → Ги дава конечните предвидувања.
- **Тежини (Weights) и пристрасност (Bias)** → Ги одредуваат врските помеѓу невроните.

- **Функции за активација (Activation Functions)** → Одредуваат како невронот реагира на влезот.

Пример за ANN во Python

python

CopyEdit

```
import tensorflow as tf  
from tensorflow import keras
```

```
# Креирање на едноставна ANN
```

```
model = keras.Sequential([  
    keras.layers.Dense(16, activation='relu', input_shape=(10,)), # Влезен слој  
    keras.layers.Dense(8, activation='relu'), # Скриен слој  
    keras.layers.Dense(1, activation='sigmoid') # Излезен слој (класификација)  
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

 **ANN работи со комплексни проблеми како предвидување, класификација и обработка на слики.**

2. Разлика помеѓу ANN и Deep Learning (DL)

 **Deep Learning (Длабоко учење) е подгрупа на машинското учење што користи длабоки невронски мрежи** (со повеќе скриени слоеви).

→ **Што го прави Deep Learning моќен?**

- **Учешењето на хиерархиски претстави** → Откривање комплексни обрасци во податоците.
- **Адаптивно учење** → Мрежата **сама ги открива најдобрите карактеристики** (без рачна инженеринг).

- **Самоорганизација** → Може да ги научи најдобрите претставувања на податоците автоматски.

📌 **Пример за разлика:**

Модел	ANN (плитка мрежа)	Deep Learning (длабока мрежа)
Број на скриени слоеви	1-2	5+
Флексибилност	Ограничена	Многу моќна
Перформанси со мали податоци	Добри	Слаби
Перформанси со големи податоци	Послаби	Одлични

3. Зошто Deep Learning е важен?

- Работи подобро од класични ML алгоритми на големи податоци.
- Овозможува обработка на комплексни структури (слики, текст, говор).
- Автоматски открива релевантни карактеристики без рачна инженеринг.

📌 **Пример: Обработка на слики**

Класичен пристап → Рачно дефинираме карактеристики како контури, форми.

Deep Learning пристап → Мрежата **сама ги учи карактеристиките**, започнувајќи од пиксели, преку ивици, до **цели објекти**.

4. Архитектура на невронска мрежа

📌 **Како функционира една ANN?**

1. **Секој влезен податок (X)** се множи со **тежина (W)** и се додава **пристрасност (Bias)**.
2. **Се применува активациона функција** за да се реши дали ќе се активира невронот.
3. **Податоците патуваат низ повеќе слоеви** за да стигнат до излезот.

Формула за неврон

$$Y = f(W_0 + \sum_{i=1}^d W_i \cdot X_i) Y = f(W_0 + \sum_{i=1}^d W_i \cdot X_i) Y = f(W_0 + \sum_{i=1}^d W_i \cdot X_i)$$

Каде што:

- $X_i X_i$ → Влезни вредности
- $W_i W_i$ → Тежини
- f → Активациона функција
- Y → Излез

📌 Пример за предвидување на примање на студент на пракса

Ако имаме податоци за студент: **GPA, година на студии, број на активности**, тогаш:

$$X = [GPA, Year, Activities] X = [GPA, Year, Activities] X = [GPA, Year, Activities]$$

$$Y = f(W_1 \cdot GPA + W_2 \cdot Year + W_3 \cdot Activities + B) Y = f(W_1 \cdot GPA + W_2 \cdot Year + W_3 \cdot Activities + B) Y = f(W_1 \cdot GPA + W_2 \cdot Year + W_3 \cdot Activities + B)$$

Мрежата треба **да научи** оптимални WWW и BBB за најдобри предвидувања.

5. Функции за активација

Функциите за активација ги одредуваат излезните вредности на невроните.

Функција	Формула	Проблеми	Решение
Sigmoid	$\sigma(x) = \frac{1}{1 + e^{-x}}$	Губење на градиент	ReLU
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	Vanishing gradient	Leaky ReLU
ReLU	$\max(0, x)$	Dying neurons	Leaky ReLU
Leaky ReLU	$\max(0.01x, x)$	-	-

📌 Пример: Користење на ReLU во Keras

python

CopyEdit

```
from tensorflow.keras.layers import Dense, Activation  
layer = Dense(64, activation='relu')
```

6. Тренинг на невронски мрежи

📌 Како невронската мрежа учи?

- **Forward Propagation** → Податоците минуваат низ мрежата и се добива излез.
 - **Loss Function** → Мери колку излезот се разликува од вистинската вредност.
 - **Backpropagation** → Ги ажурира тежините врз основа на грешките.
 - **Gradient Descent** → Го минимизира **Loss Function** со промена на тежините.
-

7. Решавање на проблеми во тренингот

1. Overfitting (преприлагодување)

- Мрежата премногу ги учи тренинг податоците и **не работи добро на нови податоци**.
- **Решенија:**
 - **Dropout** → Случајно исклучување на неврони.
 - **Регуларизација (L1/L2)** → Казнување на големи тежини.
 - **Рано стопирање (Early Stopping)** → Прекинување на тренинг кога перформансите се стабилни.

2. Vanishing/Exploding Gradients

- Проблем при користење на **многу слоеви**.
 - **Решение:**
 - Користење на **ReLU** **наместо Sigmoid/Tanh**.
 - **Batch Normalization** за нормализирање на податоците.
-

Заклучок

- **Deep Learning** е револуционерен пристап што автоматски учи **сложени карактеристики**.

- **ANNs и Deep Learning** работат со **слики, текст, говор.**
- **Backpropagation и Gradient Descent** се клучни за тренирање.
- **Правилното избирање на активациони функции, нормализација и Dropout** помага за стабилност.

Прашања?

40

Детална анализа на лекцијата „Advanced Neural Networks“

Оваа лекција покрива **напредни концепти на невронски мрежи**, вклучувајќи перцептрони, длабоки невронски мрежи, оптимизација, градиентен спуст, регуляризација, свиени невронски мрежи (CNN), рекурентни невронски мрежи (RNN) и долгочочна краткорочна меморија (LSTM).

1. Основи на невронските мрежи

1.1 Перцепtron (Perceptron)

Што е перцепtron?

- **Наједноставен вид на невронска мрежа**, кој може да се користи за **бинарна класификација**.
- Се состои од **еден слој на тежини (weights)**, **пристрасност (bias)** и **активациона функција**.
- Излезот се добива преку функцијата: $y=g(W_0+\sum_{i=1}^m w_i \cdot x_i)$ каде што g е активационата функција (на пример, **Sigmoid**).

Пример за перцепtron во Python

python

CopyEdit

```

import numpy as np

def perceptron(X, W, b):
    z = np.dot(X, W) + b
    return 1 if z >= 0 else 0

# Пример со две влезни вредности
X = np.array([2, -1])
W = np.array([3, -2])
b = 1

y = perceptron(X, W, b)
print("Излез:", y)

```

Перцепtronот може да решава само линеарно одделиви проблеми.

2. Длабоки невронски мрежки (Deep Neural Networks - DNN)

Што е длабока невронска мрежа (DNN)?

- Проширување на еднослојниот перцепtron со повеќе скриени слоеви.
- Користи **сложени неврони** кои обработуваат податоци низ **повеќе нивоа**.

Пример: Разлика меѓу плитка и длабока мрежка

Тип	Број на слоеви	Моќност
Едноставен перцепtron	1	Може да решава само линеарни проблеми.
Длабока невронска мрежка	3+	Може да учи комплексни обрасци.

-  Длабоките мрежи работат подобро за проблеми како обработка на слики и текст.
-

3. Оптимизација во невронски мрежки

3.1 Функција на загуба (Loss Function)

Што е функција на загуба?

- Мери колку моделот греши при предвидувањето.
- Честа метрика за регресија: $MSE = \frac{1}{n} \sum (Y_{true} - Y_{pred})^2$
- Честа метрика за класификација: $Cross-Entropy = -\sum y \log(\hat{y})$

3.2 Градиентен спуст (Gradient Descent)

 Градиентниот спуст е алгоритам што ја минимизира функцијата на загуба преку итеративно ажурирање на тежините.

Формула за ажурирање на тежините:

$$W = W - \lambda \partial J(W) / \partial W$$

- λ → брзина на учење (learning rate).
- Проблеми:
 - Ако λ е премал, конвергенцијата е спора.
 - Ако λ е преоголем, може да ја прескочи оптималната точка.

Python имплементација

python

CopyEdit

`W = 0.5 # Почетна тежина`

`learning_rate = 0.1`

`gradient = 2 # Претпоставен градиент`

```
# Ажурирање на тежината  
W = W - learning_rate * gradient  
print("Нова тежина:", W)
```

 **Градиентниот спуст се користи за ажурирање на тежините на невронските мрежи.**

4. Регуларизација (Regularization)

 **Регуларизацијата помага да се избегне overfitting** (преприлагодување на моделот кон тренинг податоците).

4.1 Dropout

- Случајно **исклучува дел од невроните** за време на тренингот.
- Ги намалува зависностите меѓу невроните.

 **Пример за Dropout во Keras**

python

CopyEdit

```
from tensorflow.keras.layers import Dropout
```

```
dropout_layer = Dropout(0.5) # Исклучува 50% од невроните
```

4.2 Early Stopping

- **Го прекинува тренингот** кога моделот почнува да се претренира.

 **Регуларизацијата ја подобрува генерализацијата на моделот.**

5. Напредни архитектури

5.1 Сврткани невронски мрежи (Convolutional Neural Networks - CNN)

 **CNN се користат за обработка на слики.**

- Користат **конволуциски слоеви** кои бараат обрасци во податоците.
- **Pooling слоеви** намалуваат димензионалност.

📌 **Пример: CNN во Keras**

python

CopyEdit

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

```
conv_layer = Conv2D(32, (3,3), activation='relu')
```

```
pooling_layer = MaxPooling2D(pool_size=(2,2))
```

✓ **CNN се користат за препознавање лица, автономни возила, медицински дијагнози.**

5.2 Рекурентни невронски мрежи (RNN)

📌 **RNN се користат за обработка на секвенцијални податоци (текст, говор).**

- Чува **меморија** од претходните состојби.
- Користи **повратни врски (feedback loops)**.

📌 **Пример: RNN во Keras**

python

CopyEdit

```
from tensorflow.keras.layers import SimpleRNN
```

```
rnn_layer = SimpleRNN(50, activation='relu')
```

✓ **RNN се користи за машински превод, анализа на чувства, автоматски транскрипции.**

5.3 Долгорочна краткорочна меморија (Long Short-Term Memory - LSTM)

❖ **LSTM** е подобрена RNN која избегнува проблеми со „vanishing gradients“.

- Се состои од **гейт механизам** кој одлучува што да задржи и што да заборави.

❖ **Пример: LSTM во Keras**

python

CopyEdit

```
from tensorflow.keras.layers import LSTM
```

```
lstm_layer = LSTM(50, activation='tanh')
```

✓ **LSTM** се користи за предвидување на временски серии, обработка на јазик, генерирање на текст.

Заклучок

- **DNN** се посложени верзии на **ANN** со повеќе скриени слоеви.
- **Градиентен спуст и Backpropagation** се клучни за учење.
- **CNN** е најдобар за слики, **RNN/LSTM** за **текст и времески податоци**.
- **Регуларизација (Dropout, Early Stopping)** помага да се спречи overfitting.

Детална анализа на лекцијата „Natural Language Processing (NLP)“

Оваа лекција покрива **основни и напредни концепти на природната обработка на јазикот (NLP)**, вклучувајќи **методи за претставување на текст, машинско учење во NLP, трансформер модели, Word2Vec, POS-тагирање, именувано препознавање на ентитети (NER) и алгоритми за обработка на граматика и синтакса**.

1. Што е Natural Language Processing (NLP)?

📌 **Дефиниција:** NLP е област во вештачката интелигенција која се фокусира на разбирање и обработка на природниот јазик (англиски, македонски, француски итн.) со помош на компјутери.

📌 **Примери за NLP во секојдневниот живот:**

- **Spell check** (Автоматска проверка на правопис).
- **Autocomplete** (Довршување на зборови во пребарување).
- **Гласовни аистенти** (Siri, Alexa, Google Assistant).
- **Филтрирање на спам пораки.**

✓ **NLP надминува човечката способност во обработка на големи количини на податоци.**

2. Како бизнисите го користат NLP?

- **Анализа на повратни информации од корисници** (преку социјални медиуми, е-пораки, коментари).
- **Четботи и автоматизирани поддржувачки системи** (намалување на потребата за човечка интервенција).
- **Олеснување на пребарувањата** (подобрување на корисничкото искуство со автокомплетирање).

✓ **Компаниите користат NLP за подобрување на корисничката поддршка, анализа на текст и автоматизација на задачи.**

3. Технички концепти во NLP

3.1 Topic Modeling (Моделирање на теми)

📌 **Што е модел на тема?**

- **Статистичка техника** која автоматски **открива скриени теми во голем корпус на текстови.**
- **Latent Dirichlet Allocation (LDA)** → Најпознат модел кој групира текстови по теми.

📌 **Пример за употреба:**

- Компанија сака да анализира **прегледи на производи** и да открие **најчестите теми** (цена, квалитет, услуга итн.).

 **Моделите на теми можат да помогнат во анализа на јавното мислење, анализа на новински статии и категоризација на документи.**

3.2 Named Entity Recognition (NER)

 **NER (Препознавање на именувани ентитети)** е метод кој идентификува **имиња на луѓе, компании, локации и други ентитети** во текст.

 **Пример за NER во NLP:**

- „Microsoft е основан од Bill Gates во Сиетл.“
 - Microsoft → Организација (ORG)
 - Bill Gates → Личност (PERSON)
 - Сиетл → Локација (GPE - Geopolitical Entity)

 **NER е корисен во финансиски анализи, пребарувачки алгоритми и обработка на вести.**

3.3 Open Information Extraction (OIE)

 **OIE (Отворено извлекување на информации)** автоматски екстрагира факти од текстови без потреба од претходно дефинирани структури.

 **Пример:**

- „Алберт Ајнштајн, германски теоретски физичар, ја објави теоријата на релативност во 1915.“
 - **Субјект:** Алберт Ајнштајн
 - **Предикат:** ја објави
 - **Објект:** теоријата на релативност

 **OIE помага во автоматска анализа на статии, документи и пребарување на знаење.**

4. Методи за претставување на текст

4.1 Bag of Words (BoW)

📌 Претставува текст како вектор од број на зборови, без редоследот на зборовите.

📌 Пример:

- Реченица: „The cat sat on the mat.“
- BoW: {'the': 2, 'cat': 1, 'sat': 1, 'on': 1, 'mat': 1}

📌 Проблеми со BoW:

- Го губи значењето на зборовниот редослед.
- Големината на векторот расте со зголемување на вокабуларот.

✓ BoW се користи за класификација на текст, препораки и анализа на тоналност.

4.2 N-grams

📌 N-grams земаат секвенци од N зборови за да зачуваат дел од редоследот на зборовите.

📌 Пример:

- **Bigram (2-gram):** („The cat“, „cat sat“, „sat on“, „on the“, „the mat“).
- **Trigram (3-gram):** („The cat sat“, „cat sat on“, „sat on the“, „on the mat“).

✓ N-grams се корисни за машински превод, автокорекција и анализи на јазик.

4.3 Word2Vec (Embedding на зборови)

📌 Word2Vec е техника која ги претставува зборовите како вектори во векторски простор базиран на контекстот на нивната употреба.

📌 Пример:

- „King - Man + Woman = Queen“
- 📌 Предности:
- Зборови со слично значење имаат слични вектори.

- Овозможува **кластерирања на зборови** според нивната семантика.

 Word2Vec се користи во пребарување, анализа на тоналност и преводи.

5. Современи NLP модели (Transformers)

 Современите NLP модели се базираат на трансформер архитектурата.

 Најпознати модели:

- BERT (Bidirectional Encoder Representations from Transformers)
- GPT (Generative Pre-trained Transformer)
- XLNet, RoBERTa, T5

 Клучни придобивки на трансформер моделите:

- Можат да обработуваат контекст во двете насоки (лево-десно и десно-лево).
- Можат да бидат предтрениирани на огромни количини текст.

 Примени на трансформер модели:

- Машински превод (Google Translate).
 - Генерирање на текст (GPT-3).
 - Прашања-одговор системи (BERT во Google Search).
-

6. Transfer Learning во NLP

 Преносното учење (Transfer Learning) дозволува реупотреба на веќе обучени модели на нови задачи.

 Пример:

- BERT трениран на Wikipedia + Fine-tuning на медицински текстови → Медицински BERT модел.

 Transfer Learning овозможува помала потреба од податоци и побрза адаптација на модели.

Заклучок

- ✓ NLP овозможува автоматска обработка на природен јазик.
- ✓ Методи како **BoW**, **N-grams**, **Word2Vec** и **Transformers** се клучни за модерното NLP.
- ✓ **Апликации:** машински превод, пребарување, анализа на тоналност, автоматизирани четботи.

◆ Прашања? 

Анализа на Презентацијата: "NLP и Transformers"

Оваа презентација опфаќа различни концепти во областа на Природно Јазично Обработување (NLP) и трансформер-модели, како што се GPT, BERT, и нивните примени. Подолу е детална анализа на сите важни точки.

1. Јазично Моделирање (Language Modelling)

- Јазичен модел (Language Model) предвидува веројатност за секвенца од зборови.
- **Пример:** Ако дадената реченица е "*Eleni was late for class*", моделот пресметува $P(X) = P(\text{"Eleni was late for class"})$.

Примена на јазично моделирање:

- Генерирање текст
- Автоматско дополнување (Auto-complete)
- Препознавање говор (Speech-to-text)
- Преведување (Machine Translation)
- Четботови и прашање-одговор системи

Bigram Модел

- Наместо да користи целосни реченици, моделот гледа само парови на последователни зборови.

- **Проблеми:**
 - Ограничена контекст (недоволно информации за целосно значење).
 - Проблем со **Out-of-Vocabulary (OOV)** – ако зборот не постои во податоците, моделот нема веројатност за него.
-

2. Рекурентни Неуронски Мрежи (RNNs)

- RNN користи **скриени слоеви (hidden layers)** за да меморира контекст од претходните зборови.
- Проблеми со RNN:
 - **Градиентно исчезнување (Vanishing gradient)** – моделот заборава далечни зборови.
 - **Градиентна експлозија (Exploding gradient)** – тежините стануваат преголеми.

Решение: LSTM (Long Short-Term Memory)

- LSTM користи дополнителна **мемориска ќелија** што му помага на моделот да „запомни“ подолги секвенци.
-

3. Bi-Directional LSTMs и ELMo

- Bi-LSTM користи информации **од левата и десната страна** (не само од минатото).
 - **ELMo (Embeddings from Language Models):**
 - Креира подобри претстави за зборови базирани на контекст.
 - Ги надминува стандардните вградени зборови како Word2Vec и GloVe.
-

4. Sequence-to-Sequence (Seq2Seq) и Attention

- **Seq2Seq модели** користат **енкодер-декодер архитектура**:
 - Енкодерот прима влезен текст и го конвертира во **контекстуален вектор**.

- Декодерот користи тој вектор за да создаде нова секвенца (на пример, превод).

Проблем: Енкодерот мора да „запамети“ цела реченица!

- **Решение: Attention Mechanism**

- Декодерот може **селективно** да фокусира делови од влезниот текст.
 - Ова значително ги подобрува резултатите во **машински превод и резимеизација**.
-

5. Transformers

- Трансформерите користат **Self-Attention Mechanism** наместо рекурентни врски.
- **Предности:**
 - Побрза обработка бидејќи дозволува **паралелно** процесирање.
 - Подобро справување со долги зависности.

Компоненти на трансформер:

1. **Self-Attention:**

- Дозволува моделот да „гледа“ во целиот текст при обработката на секој збор.

2. **Position Encoding:**

- Бидејќи нема редослед во трансформерите како кај RNN, позицијата на зборовите се додава преку **синусоидални функции**.
-

6. GPT, BERT и Нивните Разлики

GPT (Generative Pre-trained Transformer)

- **Еднонасочен трансформер** – учи текст од лево кон десно.
- Се користи за **генерирање на текст**.

BERT (Bidirectional Encoder Representations from Transformers)

- **Двонасочен трансформер** – гледа и лево и десно од зборот.
 - Се користи за **разбирање на јазикот** (на пример, класификација, sentiment analysis).
-

7. Примени на Големи Јазични Модели (LLMs)

- **GPT-2 и GPT-3** – моделите за текст генерација.
 - **BERT, RoBERTa, T5** – модели за разбирање на текст.
 - **Hugging Face** – платформа за споделување NLP модели.
-

Заклучок

Оваа лекција опфаќа клучни концепти во NLP и трансформер-модели. Главни поенти:

1. Јазично моделирање е основа за NLP.
2. RNN модели се корисни, но имаат ограничувања.
3. Attention механизам решава многу проблеми во Seq2Seq.
4. Transformers се најнапредни за NLP задачи.
5. GPT и BERT имаат различни примени – **генерација vs разбирање**.

Анализа на Презентацијата: "Unsupervised Learning" (Неконтролирано учење)

Оваа презентација обработува **неконтролирани техники на учење**, фокусирајќи се на **кластерирања и автоценкодери**. Детално ја анализирам содржината за да добиеш подобро разбирање за испитот.

1. Вовед во Кластерирања

Кластерирањата е техника за групирање на податоци според нивната сличност. Основната идеја е **да се минимизира растојанието помеѓу податоците во ист кластер и да се максимизира растојанието помеѓу различните кластери**.

Примени на кластеризација

- Групирање на **документи** за пребарување.
 - Групирање на **гени и протеини** според нивната функција.
 - Групирање на **акции** со слична динамика на цена.
-

2. Техники за Кластеризација

Главни методи:

1. Partitioning (Разделувачки) Методи:

- **K-Means** – поделба на податоците во K кластери.
- **K-Medoids** – слично на K-Means, но користи реални податоци како центроиди.

2. Hierarchical (Хиерархиски) Методи:

- **Agglomerative clustering** – започнува со секој податок како посебен кластер и ги спојува најблиските.
- **Divisive clustering** – започнува со еден кластер и го дели.

3. Density-Based (Методи базирани на густина):

- **DBSCAN** – идентификува густи области на податоци и ги групира.
-

3. K-Means Кластеризација

- K-Means е **разделувачки метод** кој бара претходно да се дефинира бројот на кластери K.
- Алгоритам:
 1. Случајно избира K почетни центроиди.
 2. Секој податок се додава во најблискиот центроид.
 3. Центроидите се ажурираат според просекот на податоците во кластерот.
 4. Повторување додека нема промена на центроидите.
- **Предности:** Брз и едноставен.

- **Недостатоци:** Осетлив на почетните центроиди, не работи добро со кластери од различна големина или форма.

Решенија за проблемот со почетните центроиди:

- **K-Means++:** Подобрен метод кој паметно ги избира почетните центроиди.
-

4. Hierarchical Кластерирање

- **Agglomerative Clustering (Агломеративна кластерирање):**
 1. Започнува со секој податок како посебен кластер.
 2. Се спојуваат најблиските кластери според растојание.
 3. Процесот се повторува сè додека остане само еден кластер.
- **Divisive Clustering (Деливна кластерирање):**
 - Обратен процес – започнува со сите податоци во еден кластер и постепено ги дели.

Мерки за растојание во хиерархиска кластерирања:

1. **Single Linkage (Мин. растојание)** – најблиските точки од два кластери.
2. **Complete Linkage (Макс. растојание)** – најоддалечените точки.
3. **Average Linkage (Просечно растојание)** – просечно растојание меѓу сите точки.
4. **Ward's Method** – минимизира вкупната варијанса.

Предности:

- Не треба однапред да се дефинира бројот на кластери.
- Дава подобар увид во структурата на податоците.

Недостатоци:

- **Бавен** ($O(n^2)$ или $O(n^3)$).
 - Не може да ги „поправи“ претходните одлуки.
-

5. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- Кластеризација базирана на густина на податоци.
- **Дефиниции:**
 - **Core Point** – има доволен број на блиски точки.
 - **Border Point** – близку до core point, но нема доволно блиски точки.
 - **Noise Point** – не припаѓа на ниту еден кластер.

DBSCAN алгоритам:

1. Означи ги core точките.
2. Формирај кластери околу нив.
3. Додај ги border точките во соодветните кластери.
4. Игнорирај ги noise точките.

Предности:

- Детектира кластери од различни форми.
- Отпорен на шум (noise).

Недостатоци:

- Не функционира добро ако кластерите имаат **различна густина**.
- Осетлив на изборот на **Eps** и **MinPts** параметрите.

6. Валидација на Кластерите

Бидејќи **кластеризацијата не е контролирана**, нема точен одговор за тоа кој резултат е „правилен“. Потребни се метрики за евалуација:

1. **Sum of Squared Errors (SSE)** – помал SSE значи подобри кластери.
2. **Silhouette Score** – мерка за тоа колку се блиски точките во кластер во однос на други кластери.
3. **Dunn Index** – колку се разликуваат кластерите меѓу себе.
4. **Rand Index** – споредба со „златен стандард“ (ако постои).

7. Автоенкодери (Autoencoders)

Автоенкодерите се вид **невронски мрежи** што учат **компресирани претстави (latent representations)** на податоците.

Компоненти на автоенкодер:

- **Енкодер** – компресира влезни податоци во латентен простор.
- **Декодер** – ја реконструира влезната слика од латентниот простор.

Примена:

- Намалување на димензионалноста (dimensionality reduction).
 - Детекција на аномалии (anomaly detection).
 - Генерирање на нови податоци (image generation).
-

8. Типови на Автоенкодери

1. Основен Автоенкодер

- Се состои од **еден енкодер и еден декодер**.
- Врши компресија и реконструкција.

2. Длабок Автоенкодер

- Користи **повеќе слоеви** за подобра репрезентација.

3. Варијационен Автоенкодер (VAE)

- Користи **стохастичка латентна претстава** за подобра генерација на податоци.
- Подобро зачувува детали од оригиналните податоци.

4. Конволутивен Автоенкодер (CAE)

- Наместо целосно поврзани слоеви, користи **конволутивни слоеви** за обработка на слики.
-

9. Генеративни модели и историја

- **2014-2017:** Појавата на **VAE** и **GANs** за генерирање на слики.
- **2018-2019:** Епа на **Transformers** (GPT, BERT).
- **2020+:** Големи модели како **GPT-3, DALL-E, Stable Diffusion**.

Заклучок

Оваа лекција покрива две важни теми:

1. **Кластерирација** – K-Means, DBSCAN, хиерархиски методи.
2. **Автоенкодери** – основни, длабоки, варијациони и конволутивни.

Со оваа анализа имаш јасен преглед на материјалот. Ако ти треба подетално објаснување за некој дел, пиши ми! 

PRASANJA – ODGOORI

Eve ti gi tocnite odgovori za sekoe prasanje:

1. Razlika pomegu RNN i LSTM

- LSTM е подобрена верзија на RNN која ги решава проблемите со експлодиратко/заникакво размножување на градиентот, користјќи посебни ћелии со "gate" механизми.
- LSTM може да памети подолги секвени, додека RNN има проблем со долгосеквенчни зависности.

2. Feed Forward Neural Network (FFNN)

- Точно е дека податоците се движат во една насока, од влезниот слој кон скриините слоеви и понатака кон излезниот слој, без било какви loop-ови или повратни врзки (ова е токен одговор).

3. Slika i clustering metod

- На слика е приказан **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**.
- Тоа е правилниот одговор меѓу понудени DBSCAN, Agglomerative, KNN и уште еден.

4. Model za bolnica so 1000 statii i medicinski zborovi

- **Fine-tuning na ClinicalBERT** e najdobar izbor, bidejki ClinicalBERT e specijalno obucen na medicinski podatočni sklopovi, sto go pravi podobro prilagoden za tekstovi od bolnici.

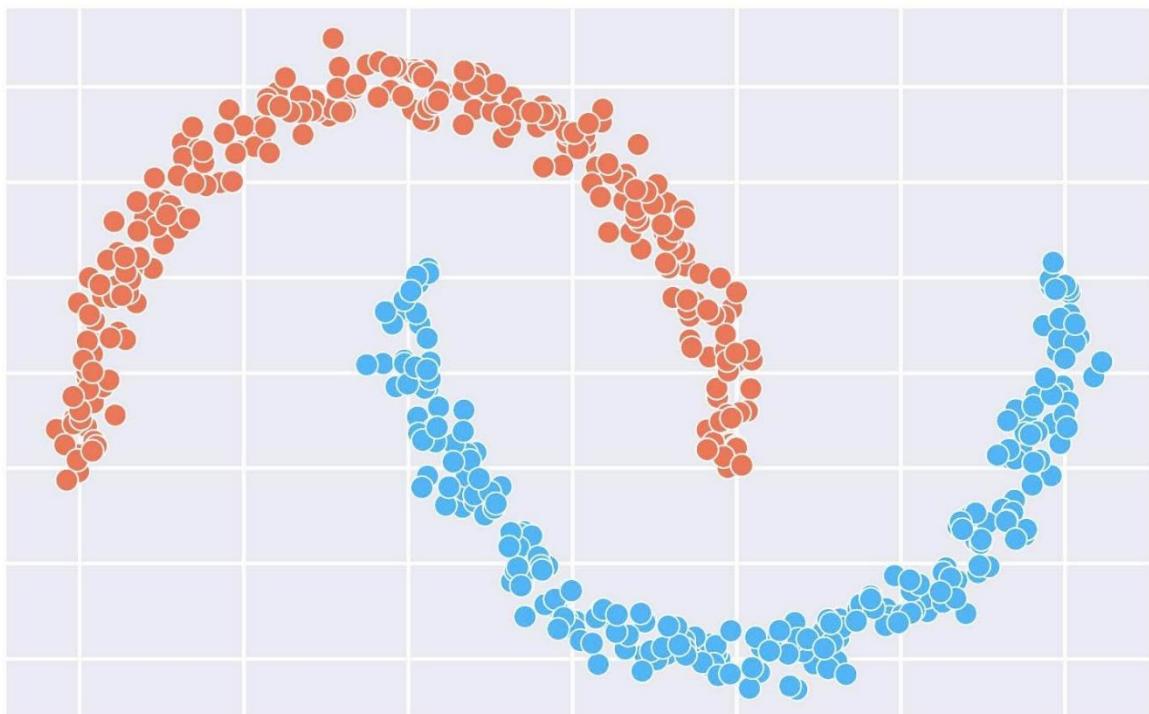
5. Lagovi vo vremenski serii

- Točno: **lagovite mora da zimaat vrednosti od minatoto** (toa e pravilniot odgovor).

6. LSTM vo Keras - return_sequences=True

- Ako go postavime return_sequences=True, toa znaci deka LSTM ke vrakja sekvenca kako izlez za секој vremensки корак, а не само finalniот резултат.
Ова се користи кога имаме пoveкje sloevi на LSTM.

DBSCAN



Ovie prasanja se dosta opfatni, па ќе ги одговориме детално и јасно:

Prv del – Direktni odgovori

1. Normalizacija

- **Zašto se pravi?**

- Normalizacijata se pravi za da gi dovede site karakteristiki vo slicen opseg, so cel da se izbegnat dominacii od odredeni atributi so pogolemi vrednosti.
 - Toa e klucno za algoritmi kako KNN, k-means, i gradient-based metodi kako logistic regression i neural networks.
- **Dali bi ja pravile ako barame performansi a ne točnost?**
- Da, zatoa sto performansite vklucuvaat i brzina na treniranje i konvergencija na modelot.
 - Bez normalizacija, mnogi algoritmi (kako SVM, gradient descent) bi bile pobavni i ponemokni da najdat optimalni resenija.

2. Real-life problem so logistička regresija

Dataset so mnogu redici i koloni, target binaren – što ke iskoristime i zosto?

- **One-hot encoding** ako imame kategoriski promenlivi.
 - **Feature selection** za da gi otstranime nerelevantnите koloni.
 - **StandardScaler ili MinMaxScaler** za normalizacija.
- **Regularizacija (L1/L2)** za da izbegneme overfitting ako datasetot e gole.
 - **Cross-validation** za stabilni predviduvanja.

3. Polinomialna regresija – Zadaca

- **Polinomialna regresija se koristi koga imame nelinearni relacii pomegu inputot i outputot.**
- Se generiraat polinomialni termi (x^2, x^3, \dots) za da se zgolemi fleksibilностa na modelot.

- Se koristi **sklearn PolynomialFeatures** za da se generiraat povekje stepeni na promenlivite.
-

4. Regresija za pogoduvanje pari

- Verovatno se misli na **Pairwise Regression**, kade modelot uci odnosи pomegu parovi na podatoci (na primer, koi dva proizvoda se slicni).
 - Se koristi vo **recommender systems**.
 - Algoritmi: RankNet, LambdaMART, pairwise LSTM modeli.
-

Vtor del – Pojmovi i algoritmi

1. KNNImputer, X so k=2, Y so moda, Xmiss i Ymiss

- **KNNImputer** se koristi za popolnuvanje na nedostignati vrednosti koristejki sosedi.
 - **Ako X e numericki i k=2**, znaci se koristi prosek od 2 najbliski vrednosti.
 - **Ako Y e kategoricen, moda se koristi za da se popolni so najcestata vrednost.**
 - **Xmiss i Ymiss** se odnosuва на nedostignati vrednosti vo datasetot.
-

2. Bagging – Koga e dobar, koga ne pravi razlika

- **Dobar e za modelite so visoka varijansa (decision trees, neural networks).**
 - **Ne pravi golema razlika ako se koristi so stabilni modeli kako linear regression.**
-

3. Klasifikacija za kredit zemanje

- **Feature engineering:** istoriski podatoci, mesecni prihodi, broj na krediti.
 - **Modeli:** Logistic regression, Decision Trees, Random Forest, Neural Networks.
-

4. Heatmap za recall

- Heatmap se koristi za da se prikaze preciznosta na modelot vo razlicni klasi.
 - **Najlos slučaj: 50*3** (site pogresno klasificirani).

- Najdoabar slučaj: 50 (samo diagonalata so točni predviduvanja).
-

Treti del – Matematici, klasifikacija i evaluacija

1. Matrica so TP, FP, TN, FN i Recall/Precision

- **Recall = $TP / (TP + FN)$** – pokazuva kolku od vistinskite pozitivni sluaci se otkrieni.
 - **Precision = $TP / (TP + FP)$** – pokazuva kolku od predvidenite pozitivni se tocni.
-

2. Banka so 90k normalni, 70k drugi, 10k maliciozni – model so 90% accuracy

- Verovatno e problem so **imbalanced dataset**.
 - Modelot verovatno gi predviduva samo cestite klasi, ignorirajki gi malicioznite.
 - **Accuracy e los metriki tuka, treba Precision/Recall/F1 Score.**
-

3. Decision Tree so 10 koloni i 300 redici

- Najdoabar broj na listovi zavisi od pruning i hyperparameters.
 - **Najlos: 300 listovi (preterano spesificni).**
 - **Najdoabar: Nekolku listovi so generalizacija.**
-

4. Pearsonova korelacija – grafik

- **Slaba korelacija (0.1 - 0.3).**
 - **Silna korelacija (nad 0.8).**
-

Četvrti del – Esejski prasanja

1. Zesto dlabokite mrezi se koristat namesto plitki?

- Moze da naucat kompleksni funkcionalni odnosi.
- Namaluva potreba za manualen feature extraction.

- Moze da modelira hijerarhija od apstraktni karakteristiki.
-

2. K-means problemi i primeri

- Moze da se zaglavi vo lokalni optimum.
 - Osetliv na outliers i shuma.
 - Primer: analiza na korisnici vo e-trgovija – k-means moze da gi grupira pogresno.
-

Petti del – DBSCAN, Feed Forward, Hierarchical Clustering

1. DBSCAN vs K-means za outliers

- DBSCAN gi ignorira outlier-ite kako noise.
- K-means gi vklucuva outlier-ite vo klasteri.

Koga DBSCAN e podobro?

- Koga imame neuniformni gustini na podatoците (GPS podatoci).
-

2. Hierarchical Clustering i losi performansi

- Ako ne se normaliziraat podatocite, karakteristiki so golemi skali moze da dominiraat.
 - Primer: visina vo metri i prihod vo dolari ke imaat razlicna tezina.
-

Šesti del – Transformer arhitektura

1. Feed Forward Neural Networks vo Transformers

- Se koristi za da se zacvrstat reprezentaciите posle self-attention mehanizmot.
 - **Pomaga vo ucenje kompleksni transformacii na inputot.**
-

Sedmi del – Bias/Variance i Deep Networks

1. Universal Approximation Theorem – Zosto povekje skrieni sloevi?

- Podobrata generalizacija na kompleksni podatoci.
- Dlabokite mrezi ucat modularno – moznost za preucuvanje e pomala.