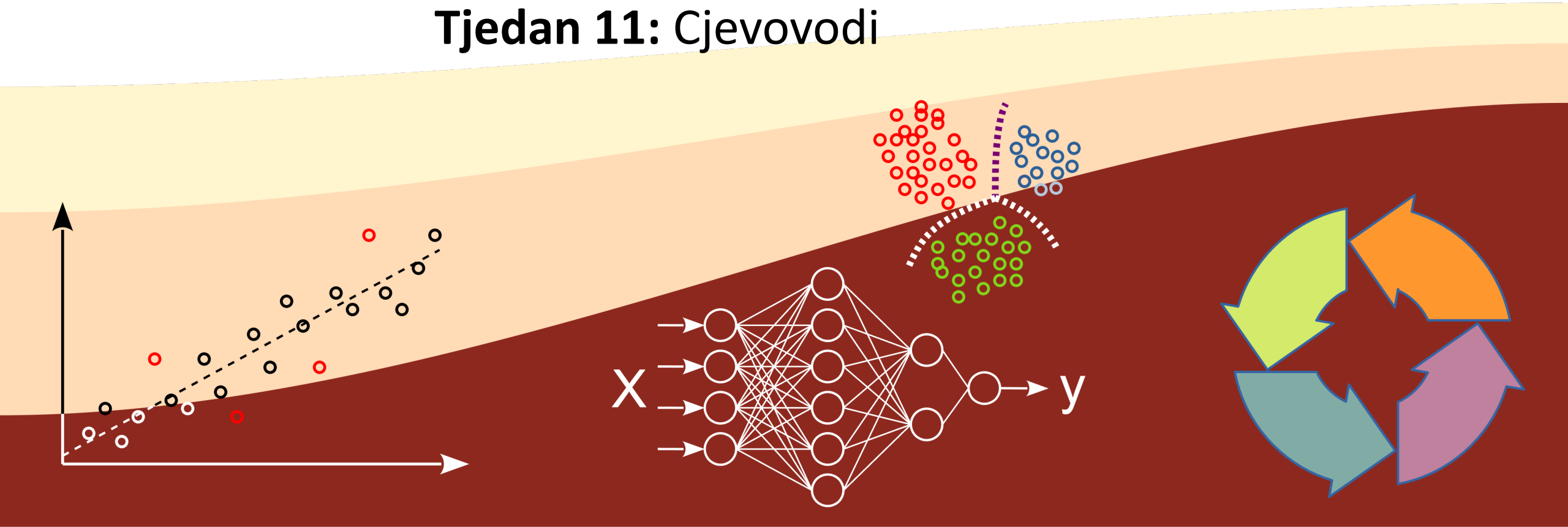
Arhitektura i Razvoj Inteligentnih Sustava

Tjedan 11: Cjevovodi



Creative Commons

- slobodno smijete:
  - dijeliti — umnožavati, distribuirati i javnosti priopćavati djelo
  - prerađivati djelo
- pod sljedećim uvjetima:
  - imenovanje: morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
  - nekomercijalno: ovo djelo ne smijete koristiti u komercijalne svrhe.
  - dijeli pod istim uvjetima: ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.

Općenito

- Cjevovod je proces kojim se
  - Uči model i instalira na poslužitelj modela – Cjevovod za učenje
  - Obogaćuje i transformira poziv mikroservisa modela – Servisni cjevovod

- Bilo kakva komponenta (poslužitelj) koja omogućava dugotrajne procese je adekvatna
  - *Stateful workflow*
  - *Long-running workflow*
  - Koncepti pasivizacije procesa i ponovnog pokretanja
  - Procesi trebaju biti persistirani

Apache Airflow

- Procesni server
  - Proces se piše u python programskom jeziku – DAG – python kod
  - Lokalna instalacija i pokretanje

```
pip install apache-airflow
airflow standalone
```

  - DAG je formiran od zadataka (taskova)
  - I DAG i taskovi se mogu označiti s anotacijama (dekoracijama)
  - Unutar DAG-a se definira slijed taskova

Instalacija

- Naivna instalacija može ići kroz generiranje vlastite *docker* slike
  - Ionako lokalna instalacija ima sve što treba za pokretanje
  - Ima smisla samo za razvojne okoline
  - Lako se podese svi parametri koji su potrebni *Apache Airflow-u*
    - Recimo *kubectl* i konfiguracija istog – za instalaciju modela

- Produkcijska instalacija na Kubernetes *cluster* – kroz helm
  - Mogu se razdvojiti komponente koje koristi *Apache Airflow*

Operatori

- Imamo posebne operatore koji predstavljaju tipove zadataka
  - *PythonOperator* – python tip zadatka
    - Čista python *@task* dekorirana funkcija završava kao *PythonOperator*
    - Recimo samo učenje modela će vrlo često biti ovakav tip zadatka
  - *EmptyOperator* – ne radi ništa, grupira druge zadatke
  - *BranchPythonOperator* – sadrži python logiku koja omogućava grananje u procesu
  - *BashOperator* – sadrži *bash shell* komandu koja se izvodi u *shell-u* kontejnera ili lokalno (ako je lokalna instalacija) – npr *kubectl* ...
  - *PostgresOperator* – specijalni operator koji radi s PostgreSQL bazom podataka
    - Definira mu se konekcija na bazu i SQL koji mora izvesti
    - *PostgresHook* – možemo koristiti u python operatoru kako bismo iz čistog python koda pristupili bazi podataka

Operatori (2)

- Generalno se *hook* koristi kako se pročitala neka definicija i koristila – recimo konekcija
  - Tako *PostgresHook* se koristi za spajanje na *PostgreSQL* bazu
  - Pa recimo *KubernetesHook* – Kubernetes *cluster* API klijent

- Slijed zadataka se definira unutar DAG-a
  - Osim *branching* operatora koristimo i

```
task1 >> task2
```

    - što znači da *task2* slijedi nakon *task1*

```
[task1 task2] >> task3
```

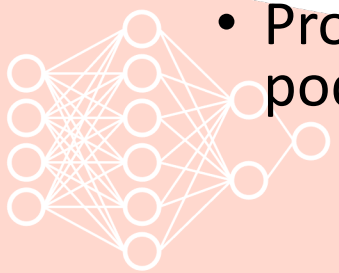
    - *task3* slijedi nakon taskova *task1* i *task2*

Slijed

- Možemo pisati i

```
task1 >> func()
```

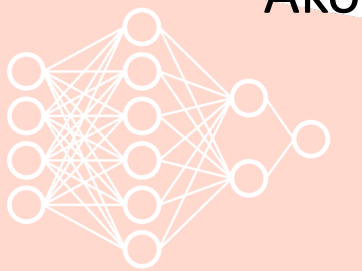
  - Što znači da je funkcije *func()* nakon *task1*
  - Funkcija je ionako implicitni *PythonOperator*
  - Kod slijeda u zadatku možemo odrediti i pravila za okidanje
    - Recimo okidamo samo ako su svi prethodni zadaci bili uspješni
    - Ili ako su svi prethodni zadaci završeni
- Pokretanje procesa može biti na više načina
  - REST API
  - Vremenski – *scheduled*
  - Ručno
  - Promjenom na skupu podataka – Apache Airflow može pratiti određeni skup podataka recimo na minIO, pa ako se datoteka promijeni starta se DAG



Instalacija

- Instalacija dodatnih modula u virtualnu okolinu *Apache Airflowa*
  - Podesite varijablu okoline _PIP_ADDITIONAL_REQUIREMENTS
  - Recimo *mlflow*, *s3fs* i slično
  - *s3fs* možemo koristiti kako bismo se spojili na *minIO* i manipulirali datoteke

- *Apache Airflow* ima specifično mjesto na koje gleda DAG-ove
  - Recimo *github*, *minIO*
  - Lokalni direktorij ~/airflow/dags
  - U trenutku kada uoči python datoteku, kreće parsiranje
  - Ako se radi o DAG-u, upisuje ga u svoju bazu, te pokreće ako je tako definirano



Dodatni detalji

- Zadaci su izolirani i svaki je svijet za sebe
  - To omogućava da se svaki zadatak pokreće na posebnom pod-u Kubernetes *clustera*
  - Pravi paralelizam kad su zadaci paralelni
  - Prijenos podataka kroz specifične persistentne varijable

```
Variable.set('run_id', run.info.run_id)
run_id=Variable.get('run_id')
```

  - *Getter* može i kao

```
{{ var.value.run_id }}
```