use std::collections::BTreeSet;

use std::collections::BTreeMap;

use bpr\_thrift::data::DataRecord;

use bpr\_thrift::prediction\_service::BatchPredictionRequest;

use thrift::OrderedFloat;

use thrift::protocol::TBinaryInputProtocol;

use thrift::protocol::TSerializable;

use thrift::transport::TBufferChannel;

use thrift::Result;

fn main() {

let data\_path = "/tmp/current/timelines/output-1";

let bin\_data: Vec<u8> = std::fs::read(data\_path).expect("Could not read file!");

println!("Length : {}", bin\_data.len());

let mut bc = TBufferChannel::with\_capacity(bin\_data.len(), 0);

bc.set\_readable\_bytes(&bin\_data);

let mut protocol = TBinaryInputProtocol::new(bc, true);

let result: Result<BatchPredictionRequest> =

BatchPredictionRequest::read\_from\_in\_protocol(&mut protocol);

match result {

Ok(bpr) => logBP(bpr),

Err(err) => println!("Error {}", err),

}

}

fn logBP(bpr: BatchPredictionRequest) {

println!("-------[OUTPUT]---------------");

println!("data {:?}", bpr);

println!("------------------------------");

/\*

let common = bpr.common\_features;

let recs = bpr.individual\_features\_list;

println!("--------[Len : {}]------------------", recs.len());

println!("-------[COMMON]---------------");

match common {

Some(dr) => logDR(dr),

None => println!("None"),

}

println!("------------------------------");

for rec in recs {

logDR(rec);

}

println!("------------------------------");

\*/

}

fn logDR(dr: DataRecord) {

println!("--------[DR]------------------");

match dr.binary\_features {

Some(bf) => logBin(bf),

\_ => (),

}

match dr.continuous\_features {

Some(cf) => logCF(cf),

\_ => (),

}

println!("------------------------------");

}

fn logBin(bin: BTreeSet<i64>) {

println!("B: {:?}", bin)

}

fn logCF(cf: BTreeMap<i64, OrderedFloat<f64>>) {

for (id, fs) in cf {

println!("C: {} -> [{}]", id, fs);

}

}