use log::debug;

use std::fs;

use serde\_json::{Map, Value};

use crate::error::SegDenseError;

use crate::mapper::{FeatureInfo, FeatureMapper, MapWriter};

use crate::segdense\_transform\_spec\_home\_recap\_2022::{self as seg\_dense, InputFeature};

pub fn load\_config(file\_name: &str) -> Result<seg\_dense::Root, SegDenseError> {

let json\_str = fs::read\_to\_string(file\_name)?;

// &format!("Unable to load segdense file {}", file\_name));

let seg\_dense\_config = parse(&json\_str)?;

// &format!("Unable to parse segdense file {}", file\_name));

Ok(seg\_dense\_config)

}

pub fn parse(json\_str: &str) -> Result<seg\_dense::Root, SegDenseError> {

let root: seg\_dense::Root = serde\_json::from\_str(json\_str)?;

Ok(root)

}

/\*\*

\* Given a json string containing a seg dense schema create a feature mapper

\* which is essentially:

\*

\* {feature-id -> (Tensor Index, Index of feature within the tensor)}

\*

\* Feature id : 64 bit hash of the feature name used in DataRecords.

\*

\* Tensor Index : A vector of tensors is passed to the model. Tensor

\* index refers to the tensor this feature is part of.

\*

\* Index of feature in tensor : The tensors are vectors, the index of

\* feature is the position to put the feature value.

\*

\* There are many assumptions made in this function that is very model specific.

\* These assumptions are called out below and need to be schematized eventually.

\*

\* Call this once for each segdense schema and cache the FeatureMapper.

\*/

pub fn safe\_load\_config(json\_str: &str) -> Result<FeatureMapper, SegDenseError> {

let root = parse(json\_str)?;

load\_from\_parsed\_config(root)

}

// Perf note : make 'root' un-owned

pub fn load\_from\_parsed\_config(root: seg\_dense::Root) -> Result<FeatureMapper, SegDenseError> {

let v = root.input\_features\_map;

// Do error check

let map: Map<String, Value> = match v {

Value::Object(map) => map,

\_ => return Err(SegDenseError::JsonMissingObject),

};

let mut fm: FeatureMapper = FeatureMapper::new();

let items = map.values();

// Perf : Consider a way to avoid clone here

for item in items.cloned() {

let mut vec = match item {

Value::Array(v) => v,

\_ => return Err(SegDenseError::JsonMissingArray),

};

if vec.len() != 1 {

return Err(SegDenseError::JsonArraySize);

}

let val = vec.pop().unwrap();

let input\_feature: seg\_dense::InputFeature = serde\_json::from\_value(val)?;

let feature\_id = input\_feature.feature\_id;

let feature\_info = to\_feature\_info(&input\_feature);

match feature\_info {

Some(info) => {

debug!("{:?}", info);

fm.set(feature\_id, info)

}

None => (),

}

}

Ok(fm)

}

#[allow(dead\_code)]

fn add\_feature\_info\_to\_mapper(

feature\_mapper: &mut FeatureMapper,

input\_features: &Vec<InputFeature>,

) {

for input\_feature in input\_features.iter() {

let feature\_id = input\_feature.feature\_id;

let feature\_info = to\_feature\_info(input\_feature);

match feature\_info {

Some(info) => {

debug!("{:?}", info);

feature\_mapper.set(feature\_id, info)

}

None => (),

}

}

}

pub fn to\_feature\_info(input\_feature: &seg\_dense::InputFeature) -> Option<FeatureInfo> {

if input\_feature.maybe\_exclude {

return None;

}

// This part needs to be schema driven

//

// tensor index : Which of these tensors this feature is part of

// [Continious, Binary, Discrete, User\_embedding, user\_eng\_embedding, author\_embedding]

// Note that this order is fixed/hardcoded here, and need to be schematized

//

let tensor\_idx: i8 = match input\_feature.feature\_id {

// user.timelines.twhin\_user\_follow\_embeddings.twhin\_user\_follow\_embeddings

// Feature name is mapped to a feature-id value. The hardcoded values below correspond to a specific feature name.

-2550691008059411095 => 3,

// user.timelines.twhin\_user\_engagement\_embeddings.twhin\_user\_engagement\_embeddings

5390650078733277231 => 4,

// original\_author.timelines.twhin\_author\_follow\_embeddings.twhin\_author\_follow\_embeddings

3223956748566688423 => 5,

\_ => match input\_feature.feature\_type {

// feature\_type : src/thrift/com/twitter/ml/api/data.thrift

// BINARY = 1, CONTINUOUS = 2, DISCRETE = 3,

// Map to slots in [Continious, Binary, Discrete, ..]

1 => 1,

2 => 0,

3 => 2,

\_ => -1,

},

};

if input\_feature.index < 0 {

return None;

}

// Handle this case later

if tensor\_idx == -1 {

return None;

}

Some(FeatureInfo {

tensor\_index: tensor\_idx,

index\_within\_tensor: input\_feature.index,

})

}