#include "internal/thrift.h"

#include "internal/error.h"

#include <twml/HashedDataRecordReader.h>

#include <twml/utilities.h>

#include <twml/functions.h>

#include <cmath>

namespace twml {

bool HashedDataRecordReader::keepId(const int64\_t &key, int64\_t &code) {

auto it = m\_keep\_map->find(key);

if (it == m\_keep\_map->end()) return false;

code = it->second;

return true;

}

bool HashedDataRecordReader::isLabel(const int64\_t &key, int64\_t &code) {

if (m\_labels\_map == nullptr) return false;

auto it = m\_labels\_map->find(key);

if (it == m\_labels\_map->end()) return false;

code = it->second;

return true;

}

bool HashedDataRecordReader::isWeight(const int64\_t &key, int64\_t &code) {

if (m\_weights\_map == nullptr) return false;

auto it = m\_weights\_map->find(key);

if (it == m\_weights\_map->end()) return false;

code = it->second;

return true;

}

void HashedDataRecordReader::readBinary(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_SET, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

int32\_t length = readInt32();

record->extendSize(length);

int64\_t id, code;

for (int32\_t i = 0; i < length; i++) {

id = readInt64();

if (keepId(id, code)) {

record->addKey(id, id, code, DR\_BINARY);

} else if (isLabel(id, code)) {

record->addLabel(code);

}

}

}

void HashedDataRecordReader::readContinuous(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_MAP, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_DOUBLE, "value\_type");

int32\_t length = readInt32();

record->extendSize(length);

int64\_t id, code;

for (int32\_t i = 0; i < length; i++) {

id = readInt64();

if (keepId(id, code)) {

double value = readDouble();

if (!std::isnan(value)) {

record->addKey(id, id, code, DR\_CONTINUOUS, value);

}

} else if (isLabel(id, code)) {

record->addLabel(code, readDouble());

} else if (isWeight(id, code)) {

record->addWeight(code, readDouble());

} else {

skip<double>();

}

}

}

void HashedDataRecordReader::readDiscrete(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_MAP, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "value\_type");

int32\_t length = readInt32();

record->extendSize(length);

int64\_t id, code;

for (int32\_t i = 0; i < length; i++) {

id = readInt64();

if (keepId(id, code)) {

int64\_t transformed\_key = mixDiscreteIdAndValue(id, readInt64());

record->addKey(id, transformed\_key, code, DR\_DISCRETE);

} else {

skip<int64\_t>();

}

}

}

void HashedDataRecordReader::readString(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_MAP, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_STRING, "value\_type");

int32\_t length = readInt32();

record->extendSize(length);

int64\_t id, code;

for (int32\_t i = 0; i < length; i++) {

id = readInt64();

if (keepId(id, code)) {

const uint8\_t \*begin = nullptr;

int32\_t str\_len = getRawBuffer<uint8\_t>(&begin);

int64\_t transformed\_key = mixStringIdAndValue(id, str\_len, begin);

record->addKey(id, transformed\_key, code, DR\_STRING);

} else {

int32\_t str\_len = readInt32();

skipLength(str\_len);

}

}

}

void HashedDataRecordReader::readSparseBinary(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_MAP, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_SET, "value\_type");

int32\_t length = readInt32();

record->extendSize(length);

int64\_t id, code;

for (int32\_t i = 0; i < length; i++) {

id = readInt64();

if (keepId(id, code)) {

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_STRING, "set:key\_type");

int32\_t set\_length = readInt32();

for (int32\_t j = 0; j < set\_length; j++) {

const uint8\_t \*begin = nullptr;

int32\_t str\_len = getRawBuffer<uint8\_t>(&begin);

int64\_t transformed\_key = mixStringIdAndValue(id, str\_len, begin);

record->addKey(id, transformed\_key, code, DR\_SPARSE\_BINARY);

}

} else {

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_STRING, "set:key\_type");

int32\_t set\_length = readInt32();

for (int32\_t j = 0; j < set\_length; j++) {

int32\_t str\_len = readInt32();

skipLength(str\_len);

}

}

}

}

void HashedDataRecordReader::readSparseContinuous(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_MAP, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_MAP, "value\_type");

int32\_t length = readInt32();

record->extendSize(length);

int64\_t id, code;

for (int32\_t i = 0; i < length; i++) {

id = readInt64();

if (keepId(id, code)) {

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_STRING, "map::key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_DOUBLE, "map::value\_type");

int32\_t map\_length = readInt32();

for (int32\_t j = 0; j < map\_length; j++) {

const uint8\_t \*begin = nullptr;

int32\_t str\_len = getRawBuffer<uint8\_t>(&begin);

int64\_t transformed\_key = 0;

switch(m\_decode\_mode) {

case DecodeMode::hash\_fname\_and\_valname:

transformed\_key = mixStringIdAndValue(id, str\_len, begin);

break;

default: // m\_decode\_mode == DecodeMode::hash\_valname == 0 is default

twml\_get\_feature\_id(&transformed\_key, str\_len, reinterpret\_cast<const char \*>(begin));

}

double value = readDouble();

if (!std::isnan(value)) {

record->addKey(id, transformed\_key, code, DR\_SPARSE\_CONTINUOUS, value);

}

}

} else {

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_STRING, "map::key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_DOUBLE, "map::value\_type");

int32\_t map\_length = readInt32();

for (int32\_t j = 0; j < map\_length; j++) {

int32\_t str\_len = readInt32();

skipLength(str\_len);

skip<double>();

}

}

}

}

void HashedDataRecordReader::readBlob(

const int feature\_type,

HashedDataRecord \*record) {

CHECK\_THRIFT\_TYPE(feature\_type, TTYPE\_MAP, "type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_I64, "key\_type");

CHECK\_THRIFT\_TYPE(readByte(), TTYPE\_STRING, "value\_type");

int32\_t length = readInt32();

int64\_t id;

for (int32\_t i = 0; i < length; i++) {

// Skips the BlobFeatures if they are defined or not in the FeatureConfig

id = readInt64();

int32\_t str\_len = readInt32();

skipLength(str\_len);

}

}

} // namespace twml