package com.twitter.search.earlybird.ml;

import java.io.IOException;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Optional;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.file.AbstractFile;

import com.twitter.search.common.file.FileUtils;

import com.twitter.search.common.metrics.SearchStatsReceiver;

import com.twitter.search.common.schema.DynamicSchema;

import com.twitter.search.common.util.ml.prediction\_engine.CompositeFeatureContext;

import com.twitter.search.common.util.ml.prediction\_engine.LightweightLinearModel;

import com.twitter.search.common.util.ml.prediction\_engine.ModelLoader;

import static com.twitter.search.modeling.tweet\_ranking.TweetScoringFeatures.CONTEXT;

import static com.twitter.search.modeling.tweet\_ranking.TweetScoringFeatures.FeatureContextVersion.CURRENT\_VERSION;

/\*\*

\* Loads the scoring models for tweets and provides access to them.

\*

\* This class relies on a list of ModelLoader objects to retrieve the objects from them. It will

\* return the first model found according to the order in the list.

\*

\* For production, we load models from 2 sources: classpath and HDFS. If a model is available

\* from HDFS, we return it, otherwise we use the model from the classpath.

\*

\* The models used for default requests (i.e. not experiments) MUST be present in the

\* classpath, this allows us to avoid errors if they can't be loaded from HDFS.

\* Models for experiments can live only in HDFS, so we don't need to redeploy Earlybird if we

\* want to test them.

\*/

public class ScoringModelsManager {

private static final Logger LOG = LoggerFactory.getLogger(ScoringModelsManager.class);

/\*\*

\* Used when

\* 1. Testing

\* 2. The scoring models are disabled in the config

\* 3. Exceptions thrown during loading the scoring models

\*/

public static final ScoringModelsManager NO\_OP\_MANAGER = new ScoringModelsManager() {

@Override

public boolean isEnabled() {

return false;

}

};

private final ModelLoader[] loaders;

private final DynamicSchema dynamicSchema;

public ScoringModelsManager(ModelLoader... loaders) {

this.loaders = loaders;

this.dynamicSchema = null;

}

public ScoringModelsManager(DynamicSchema dynamicSchema, ModelLoader... loaders) {

this.loaders = loaders;

this.dynamicSchema = dynamicSchema;

}

/\*\*

\* Indicates that the scoring models were enabled in the config and were loaded successfully

\*/

public boolean isEnabled() {

return true;

}

public void reload() {

for (ModelLoader loader : loaders) {

loader.run();

}

}

/\*\*

\* Loads and returns the model with the given name, if one exists.

\*/

public Optional<LightweightLinearModel> getModel(String modelName) {

for (ModelLoader loader : loaders) {

Optional<LightweightLinearModel> model = loader.getModel(modelName);

if (model.isPresent()) {

return model;

}

}

return Optional.absent();

}

/\*\*

\* Creates an instance that loads models first from HDFS and the classpath resources.

\*

\* If the models are not found in HDFS, it uses the models from the classpath as fallback.

\*/

public static ScoringModelsManager create(

SearchStatsReceiver serverStats,

String hdfsNameNode,

String hdfsBasedPath,

DynamicSchema dynamicSchema) throws IOException {

// Create a composite feature context so we can load both legacy and schema-based models

CompositeFeatureContext featureContext = new CompositeFeatureContext(

CONTEXT, dynamicSchema::getSearchFeatureSchema);

ModelLoader hdfsLoader = createHdfsLoader(

serverStats, hdfsNameNode, hdfsBasedPath, featureContext);

ModelLoader classpathLoader = createClasspathLoader(

serverStats, featureContext);

// Explicitly load the models from the classpath

classpathLoader.run();

ScoringModelsManager manager = new ScoringModelsManager(hdfsLoader, classpathLoader);

LOG.info("Initialized ScoringModelsManager for loading models from HDFS and the classpath");

return manager;

}

protected static ModelLoader createHdfsLoader(

SearchStatsReceiver serverStats,

String hdfsNameNode,

String hdfsBasedPath,

CompositeFeatureContext featureContext) {

String hdfsVersionedPath = hdfsBasedPath + "/" + CURRENT\_VERSION.getVersionDirectory();

LOG.info("Starting to load scoring models from HDFS: {}:{}",

hdfsNameNode, hdfsVersionedPath);

return ModelLoader.forHdfsDirectory(

hdfsNameNode,

hdfsVersionedPath,

featureContext,

"scoring\_models\_hdfs\_",

serverStats);

}

/\*\*

\* Creates a loader that loads models from a default location in the classpath.

\*/

@VisibleForTesting

public static ModelLoader createClasspathLoader(

SearchStatsReceiver serverStats, CompositeFeatureContext featureContext)

throws IOException {

AbstractFile defaultModelsBaseDir = FileUtils.getTmpDirHandle(

ScoringModelsManager.class,

"/com/twitter/search/earlybird/ml/default\_models");

AbstractFile defaultModelsDir = defaultModelsBaseDir.getChild(

CURRENT\_VERSION.getVersionDirectory());

LOG.info("Starting to load scoring models from the classpath: {}",

defaultModelsDir.getPath());

return ModelLoader.forDirectory(

defaultModelsDir,

featureContext,

"scoring\_models\_classpath\_",

serverStats);

}

}