ModelDB: a system for managing ML models

Manasi Vartak, PhD Candidate
MIT Database Group

mvartak@csail.mit.edu | @DataCereal
Why Model Management?
IMDB Prediction Task

- Given data about movies (e.g. year made, studio, genres, actors)
- Predict IMDB_score
// process data in
val (preprocessedData, featureVectorNames, _) = // Train a Linear Regression model.
  process(
    "color", "content_rating",
    "country", "first_genre", "second_genre",
    "num_critic_for_reviews", "gross",
    "num_user_for_reviews", "title_year",
    "num_voted_users"
  )

val labelCol = "imdb_score"
val featuresCol = "features"
val predictionCol = "prediction"
val lr = new LinearRegression()
  .setMaxIter(10)
  .setLabelCol(labelCol)
  .setPredictionCol(predictionCol)
  .setFeaturesCol(featuresCol)
  .setRegParam(0.3)
  .setElasticNetParam(0.1)
val lrModel = lr.fit(train)
lrModel.save("imdb_simple_lr"

// Evaluate the model.
val eval = new RegressionEvaluator()
  .setMetricName("rmse")
  .setLabelCol(labelCol)
  .setPredictionCol(predictionCol)

val predictions = lrModel.transform(test)
val score = eval.evaluate(predictions, lrModel)
// process data in
val (preprocessedData, featureVectorNames, _) = 
// Train a Linear Regression model.
  process(
    "color", "content_rating",
    "country", "first_genre", "second_genre",
    "num_critic_for_reviews", "gross",
    "num_user_for_reviews", "title_year",
    "num_voted_users"
  )

val labelCol = "imdb_score"
val featuresCol = "features"
val predictionCol = "prediction"
val lr = new LinearRegression()
  .setMaxIter(10)
  .setLabelCol(labelCol)
  .setPredictionCol(predictionCol)
  .setFeaturesCol(featuresCol)

val paramGrid = new ParamGridBuilder()
  .addGrid(lr.regParam, Array(0.1, 0.3, 0.5))
  .addGrid(lr.elasticNetParam, Array(0.1, 0.3, 0.8))
  .build()

val lrCv = new CrossValidator()
  .setEstimator(lr)
  .setEvaluator(eval)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(3)

val lrCvModel = lrCv.fitSync(train)
lrCvModel.saveSync("imdb_exploratory_lr")
val lrPredictions = lrCvModel.transformSync(test)
val (preprocessedData, featureVectorNames, _) = // Train a Linear Regression model.

val labelCol = "imdb_score"
val featuresCol = "features"
val predictionCol = "prediction"

val rf = new RandomForestRegressor()
  .setNumTrees(20)
  .setFeaturesCol(featuresCol)
  .setLabelCol(labelCol)

val eval = makeEvaluator()

val paramGrid = new ParamGridBuilder()
  .addGrid(rf.featureSubsetStrategy, Array("log2", "auto", "variance"))
  .addGrid(rf.maxDepth, Array(5, 7, 9))
  .build()

val rfCv = new CrossValidator()
  .setEstimator(rf)
  .setEvaluator(eval)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(3)

val rfCvModel = rfCv.fitSync(train)
rfCvModel.saveSync("imdb_exploratory_rf")
val rfPredictions = rfCvModel.transformSync(test)
val (preprocessedData, featureVectorNames, _) = // Train a Linear Regression model.
val labelCol = "imdb_score"
val featuresCol = "features"
val predictionCol = "prediction"
val rf = new RandomForestRegressor()
  .setNumTrees(20)
  .setLabelCol(labelCol)
val rf = new RandomForestRegressor()
  .setNumTrees(20)
  .setLabelCol(labelCol)
val paramGrid = new ParamGridBuilder()
  .addGrid(rf.featureSubsetStrategy, Array("log2", 
  .addGrid(rf.maxDepth, Array(5, 7, 9))
  .build()
val paramGrid = new ParamGridBuilder()
  .addGrid(rf.featureSubsetStrategy, Array("log2", 
  .addGrid(rf.maxDepth, Array(5, 7, 9))
  .build()
val rfCv = new CrossValidator()
  .setEstimator(rf)
  .setEvaluator(eval)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(3)
val rfCvModel = rfCv.fitSync(train)
rfcvModel.saveSync("imdb_exploratory_rf")
val rfPredictions = rfCvModel.transformSync(test)
// Train a Linear Regression model.
val labelCol = "imdb_score"
val featuresCol = "features"
val predictionCol = "prediction"
val gbt = newGBTRegressor()
  .setMaxIter(10)
  .setFeaturesCol(featuresCol)
  .setLabelCol(labelCol)

val eval = makeEvaluator()
val paramGrid = new ParamGridBuilder()
  .addGrid(gbt.lossType, Array("squared", "absolute"))
  .addGrid(gbt.maxDepth, Array(5, 7, 9))
  .build()
val gbtCv = new CrossValidator()
  .setEstimator(gbt)
  .setEvaluator(eval)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(3)
val gbtCvModel = gbtCv.fitSync(train)
gbtCvModel.saveSync("imdb_exploratory_gbt")
val gbtPredictions = gbtCvModel.transformSync(train.take(10))
Why is this a problem?

- No record of experiments
- Insights lost along the way
- Difficult to reproduce results
- Cannot search for or query models
- Difficult to collaborate

Did my colleague do that already?

How did normalization affect my ROC?

What params did I use?

Where is the prod version of the model for churn?

How does someone review your model?
ModelDB: an end-to-end model management system

- Ingest models, metadata
- Store and version modeling artifacts
- Query
- Collaborate, Reproduce results
ModelDB Architecture

Scala
- spark.ml

Python
- scikit-learn

... Light Client

ModelDB Backend
- Storage

thrift

Events

ModelDB Frontend:
- vis + query
Demo
ML Infrastructure

- DBMSs
- Spark
- Hive
- CSV
- Spark.ml
- sklearn
- R
- DL frmks
- H2O

Data Processing

Model Training

Model Management

Serving

+ A/B testing
+ Model Retraining

Monitoring

Custom

+ Visualizations
+ Interpretability
+ Debugging

Custom
## Benefits of model management

<table>
<thead>
<tr>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Developer Productivity</strong></td>
<td><strong>Model Monitoring</strong></td>
</tr>
<tr>
<td>+ Provenance</td>
<td>+ Model performance over time</td>
</tr>
<tr>
<td>+ Reproducibility</td>
<td>+ Anomaly detection</td>
</tr>
<tr>
<td>+ Meta-analyses</td>
<td>+ Trigger retraining</td>
</tr>
</tbody>
</table>

**Increased Transparency**

| + What models have been built |
| + How well do models work? |
| + Auditability | + How was this model built? |
| + What has changed? |
At last NIPS

- Initial version of ModelDB with sklearn, spark.ml support
- Early adopters (banks, financial firms), early feedback
- Focus on developer productivity
Since last NIPS!

- Initial release of ModelDB in Feb early 2017
- Adoption/evaluation at Adobe, banks, financial institutions, and tech companies
- Won AIGrant for open-source projects
- See papers at SIGMOD, NIPS workshops
Since last NIPS!

- Easy installation: docker, pip
- Light clients (R, YAML, packages outside of sklearn)
- Flexible metadata storage
- Collecting metrics over time
- Fine-grained visualizations
- In the (research) pipeline
- Data and intermediate storage
- Model diagnosis
ModelDB so far

• Incredible inbound interest
  • Banks, finance, insurance, tech
  • Lots of feature requests (e.g. monitoring, diagnosis, DL). *More than research resources can handle :)*

• Validation
  • Every data scientist building > 10 models needs model management and is looking for these tools

• Vision: Industry standard tool for managing ML models and metadata
Moving to Apache Incubation

- With MIT, Adobe, other partners (*MLSys community)
- Open development to wider community
- Contributions across industry
- Roadmap
  - Multiple storage backends, DL frameworks, R
  - Monitoring capabilities
Call for Contributions!

• Community over code

• Build once, reuse many times

• Why?

• It will measurably improve your workflow

• Pay it forward

• Be part of larger open-source project
How to Contribute

• Test it out and give feedback

• Share: teams, meetups, data science meetings, blogs

• Documentation

• Code:
  • Lots of issues on GitHub
  • Add support for your favorite ML frameworks
Informal Meeting at MLSys

• Interested in testing/adopting ModelDB?

• Did you build such a system, can you share lessons?

• Open-source Contributors!

• How/when

  • Whova app (“Model Management Meetup”)

  • mvartak@csail.mit.edu

• Poster
ModelDB

https://github.com/mitdbg/modeldb

http://modeldb.csail.mit.edu

Manasi Vartak  I  @DataCereal