StreamDM: Advanced data science with Spark Streaming

Heitor Murilo Gomes and Albert Bifet
About me

- Heitor Murilo Gomes
- PhD in Computer Science
  - Adaptive Random Forests for evolving data stream classification
  - A Survey on Ensemble Learning for Data Stream Classification
- Researcher at Télécom ParisTech
- Contribute to StreamDM and MOA

- Website: www.heitorgomes.com
- Linkedin: www.linkedin.com/in/hmgomes/
Topics

- Batch learning X Stream learning
  - What is the difference?
  - What are the assumptions?

- StreamDM
  - Overview of the project
  - Example of how to get started
  - Discussion about extending/using StreamDM

- Wrap-up
Batch learning

Well defined training phase

Random access to instances

Challenges: missing data, noise, imbalance, high dimensionality, ...
Stream Learning

Sequential access only

Strict time/memory requirements

Non-stationary data distribution

Challenges: inherit those from batch + concept drifts, feature evolution, ...

Continuous flow

$X^0$, $X^1$, $X^t$, ...

Time

$u$
Training and Testing

- There are well-defined phases for training and validating your model
- In production you deploy a **trained model** (perform predictions)

- These phases are interleaved as the model and data (may) change over time
- In production you deploy a **trainable model** (predictions + updates).
StreamDM: overview

- Started in Huawei Noah’s Ark Lab
- Collaboration between Huawei Shenzhen and Télécom ParisTech
- Open source
- Built on top of Spark Streaming
- Does not depend on third-party libraries
- Can be extended to included new tasks/algorithms

- Website: http://huawei-noah.github.io/streamDM/
- GitHub: https://github.com/huawei-noah/streamDM
Spark Streaming

- Micro-batch and Discretized Streams (DStream)

Image source: https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html
StreamDM: micro-batches

- Micro-batches and StreamDM

- “So… you are not processing one instance at a time?!”
StreamDM

- **Stream readers/writers**
  - Classes for reading data in and outputting results.

- **Tasks**
  - Setting up the learning cycle (e.g. train/predict/evaluate).

- **Methods**
  - Supervised and unsupervised learning algorithms. Hoeffding Tree, CluStream, Random Forest, Bagging, …

- **Base/other classes**
  - Instance and Example representation, Feature specification, synthetic stream generators, parameter handling, …
StreamDM: Example

- **Task**
  - Price change in electricity market modeled as binary classification (up/down)

- **Input**
  - Simulated stream (file: electNormNew.arff) - it is available at the project git

- **Learner**
  - Hoeffding Tree

- **Output**
  - Basic classification performance per micro-batch
StreamDM: Example

1. git clone + sbt package
   https://github.com/huawei-noah/streamDM

2. cd /scripts and run this command line

   ./spark.sh "EvaluatePrequential -l (trees.HoeffdingTree) -s (FileReader -f ../data/ electNormNew.arff -k 4531 -i 45312) -e (BasicClassificationEvaluator -c -m) -h" 1> results_ht.csv

Demo
StreamDM: Example

./spark.sh
"EvaluatePrequential

-l (trees.HoeffdingTree)

-s (FileReader -f ../data/electNormNew.arff -k 4531 -i 45312)

-e (BasicClassificationEvaluator -c -m) -h"

1> results_ht.csv
class EvaluatePrequential extends Task {
/* attributes */
def run(ssc:StreamingContext): Unit = {

  val reader:StreamReader = this.streamReaderOption.getValue()
  val learner:Classifier = this.learnerOption.getValue()
  learner.init(reader.getExampleSpecification())

  val evaluator:Evaluator = this.evaluatorOption.getValue()
  evaluator.setExampleSpecification(reader.getExampleSpecification())

  val writer:StreamWriter = this.resultsWriterOption.getValue()

  val instances = reader.getExamples(ssc)

  if(shouldPrintHeaderOption.isSet)
    writer.output(evaluator.header())

  //Predict
  val predPairs = learner.predict(instances)
  //Train
  learner.train(instances)
  //Evaluate
  writer.output(evaluator.addResult(predPairs))
}
Task - Evaluate Prequential

class EvaluatePrequential extends Task {
/* attributes */
def run(ssc:StreamingContext): Unit = {

  val reader:StreamReader = this.streamReaderOption.getValue()

  val learner:Classifier = this.learnerOption.getValue()
  learner.init(reader.getExampleSpecification())

  val evaluator:Evaluator = this.evaluatorOption.getValue()
  evaluator.setExampleSpecification(reader.getExampleSpecification())

  val writer:StreamWriter = this.resultsWriterOption.getValue()

  val instances = reader.getExamples(ssc)

  if(shouldPrintHeaderOption.isSet)
    writer.output(evaluator.header())

  //Predict
  val predPairs = learner.predict(instances)

  //Train
  learner.train(instances)

  //Evaluate
  writer.output(evaluator.addResult(predPairs))
}
}
class EvaluatePrequential extends Task {
/* attributes */
def run(ssc: StreamingContext): Unit = {

  val reader: StreamReader = this.streamReaderOption.getValue()
  val learner: Classifier = this.learnerOption.getValue()
  learner.init(reader.getExampleSpecification())

  val evaluator: Evaluator = this.evaluatorOption.getValue()
  evaluator.setExampleSpecification(reader.getExampleSpecification())

  val writer: StreamWriter = this.resultsWriterOption.getValue()

  val instances = reader.getExamples(ssc)

  if (shouldPrintHeaderOption.isSet)
    writer.output(evaluator.header())

  // Predict
  val predPairs = learner.predict(instances)

  // Train
  learner.train(instances)

  // Evaluate
  writer.output(evaluator.addResult(predPairs))
}
}
class EvaluatePrequential extends Task {
/* attributes */
def run(ssc:StreamingContext): Unit = {

    val reader:StreamReader = this.streamReaderOption.getValue()
    val learner:Classifier = this.learnerOption.getValue()
    learner.init(reader.getExampleSpecification())

    val evaluator:Evaluator = this.evaluatorOption.getValue()
    evaluator.setExampleSpecification(reader.getExampleSpecification())

    val writer:StreamWriter = this.resultsWriterOption.getValue()

    val instances = reader.getExamples(ssc)

    if(shouldPrintHeaderOption.isSet)
        writer.output(evaluator.header())

    // Predict
    val predPairs = learner.predict(instances)

    // Train
    learner.train(instances)

    // Evaluate
    writer.output(evaluator.addResult(predPairs))
    
}
Learner - Hoeffding Tree

- Incremental Decision Tree learning algorithm
- Hoeffding trees are the cornerstone of supervised learning for data streams
- Used (a lot) to build ensemble models
- StreamDM implementation
  - horizontal partitioning
  - handle numeric and nominal features
  - binary / multi-class
  - Naive bayes at leaves

- Theoretical details: Mining High-Speed Data Streams by Pedro Domingos and Geoff Hulten
Output - Basic Classification Performance

- Outputs different metrics (e.g. accuracy, fbeta-score, …)
- Binary and multi-class evaluation per micro-batch

<table>
<thead>
<tr>
<th>Training Time</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F(beta=1.0)-score</th>
<th>Specificity</th>
<th>TP</th>
<th>FN</th>
<th>FP</th>
<th>TN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.577</td>
<td>0.791</td>
<td>0.846</td>
<td>0.576</td>
<td>0.685</td>
<td>0.77</td>
<td>1032</td>
<td>188</td>
<td>760</td>
<td>2551</td>
</tr>
<tr>
<td>1.763</td>
<td>0.711</td>
<td>0.956</td>
<td>0.382</td>
<td>0.546</td>
<td>0.657</td>
<td>787</td>
<td>36</td>
<td>1272</td>
<td>2436</td>
</tr>
<tr>
<td>2.606</td>
<td>0.714</td>
<td>0.932</td>
<td>0.441</td>
<td>0.599</td>
<td>0.649</td>
<td>967</td>
<td>70</td>
<td>1225</td>
<td>2269</td>
</tr>
<tr>
<td>3.571</td>
<td>0.668</td>
<td>1</td>
<td>0.199</td>
<td>0.332</td>
<td>0.638</td>
<td>373</td>
<td>0</td>
<td>1504</td>
<td>2654</td>
</tr>
<tr>
<td>4.57</td>
<td>0.639</td>
<td>1</td>
<td>0.096</td>
<td>0.174</td>
<td>0.624</td>
<td>173</td>
<td>0</td>
<td>1637</td>
<td>2721</td>
</tr>
<tr>
<td>5.549</td>
<td>0.621</td>
<td>0.983</td>
<td>0.094</td>
<td>0.171</td>
<td>0.606</td>
<td>177</td>
<td>3</td>
<td>1714</td>
<td>2637</td>
</tr>
<tr>
<td>6.537</td>
<td>0.624</td>
<td>0.827</td>
<td>0.051</td>
<td>0.097</td>
<td>0.619</td>
<td>91</td>
<td>19</td>
<td>1683</td>
<td>2738</td>
</tr>
<tr>
<td>7.531</td>
<td>0.652</td>
<td>0.953</td>
<td>0.104</td>
<td>0.188</td>
<td>0.638</td>
<td>183</td>
<td>9</td>
<td>1570</td>
<td>2769</td>
</tr>
<tr>
<td>8.533</td>
<td>0.716</td>
<td>0.901</td>
<td>0.416</td>
<td>0.569</td>
<td>0.668</td>
<td>849</td>
<td>93</td>
<td>1192</td>
<td>2397</td>
</tr>
<tr>
<td>9.553</td>
<td>0.704</td>
<td>0.954</td>
<td>0.363</td>
<td>0.526</td>
<td>0.653</td>
<td>744</td>
<td>36</td>
<td>1303</td>
<td>2448</td>
</tr>
</tbody>
</table>
StreamDM, MLlib and MOA

- Using Hoeffding Tree as a MLlib streaming algorithm
- For the same electricity data
  - StreamingLogisticRegressionWithSGD
  - Hoeffding Tree (StreamDM)
  - Hoeffding Tree (MOA)

- Implementation:
  - From Example to LabeledPoint
  - “Schema” specification
  - Adhering to coding standard
Wrap-up

- Brief overview of learning from data streams
- How to set up StreamDM (you should try it out in your own data)
- Basic concepts of how to extend StreamDM
  - Adding new tasks/methods
  - Using it in your code
- If you develop something please consider contributing it to StreamDM
Upcoming

- More supervised learning algorithms (e.g. Random forest)
- Task and algorithms for pattern mining, multi-label and concept drift detection
- StreamDM + Structured Streaming (Strata NY 2018)
  - Machine learning for non-stationary streaming data using Structured Streaming and StreamDM
Thanks!

https://github.com/huawei-noah/streamDM