

Final Project

Introduction to Databases
DataLab
CS, NTHU

Outline

- Project Goal
- Timeline
- Stages
- Final Presentation
- Demo
- Submission

Outline

- **Project Goal**
- Timeline
- Stages
- Final Presentation
- Demo
- Submission

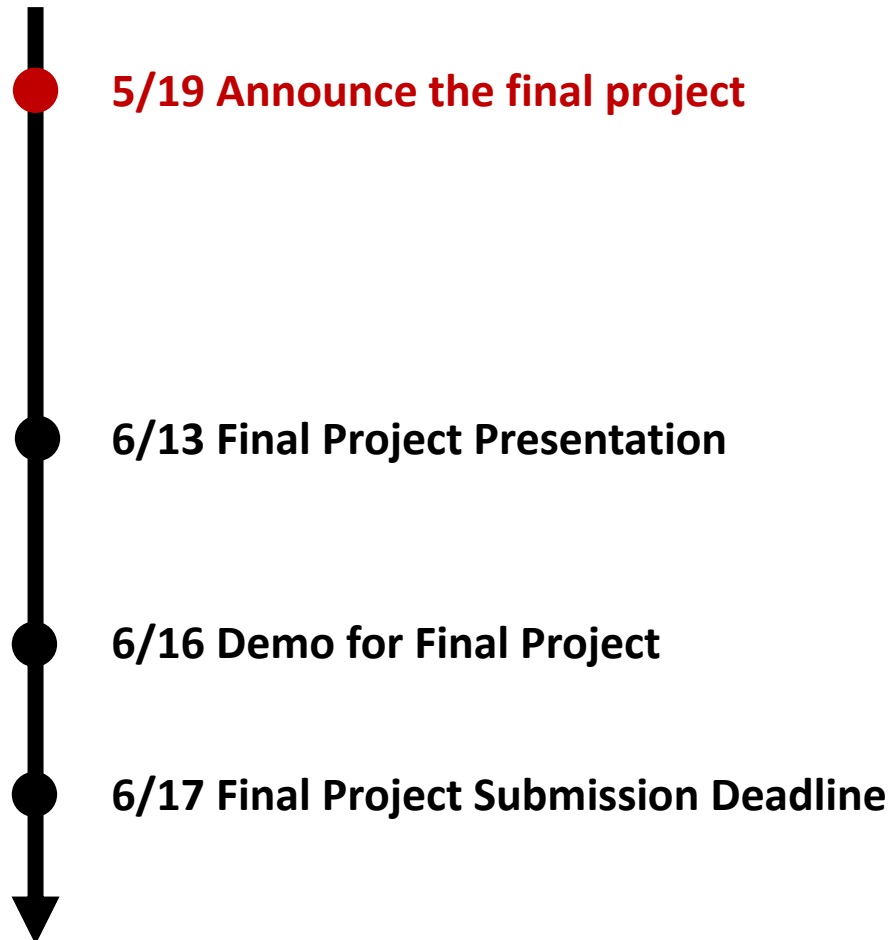
Project Goal

- This final project tests whether you are capable to figure out what DBMS components may affect transaction latency.
- In order to test this ability, you need to train a simple model and get better prediction accuracy.

Outline

- Project Goal
- **Timeline**
- Stages
- Final Presentation
- Demo
- Submission

Timeline



Outline

- Project Goal
- Timeline
- **Stages**
- Final Presentation
- Demo
- Submission

Stages

- Stage 1: Read the paper we provide
 - To learn the essential knowledge for how to do latency estimation.
- Stage 2: Trace the code of our latency estimator
 - To understand how to write an estimator.
- Stage 3: Write a feature collector
 - Before training the model, you should be able to collect features from VanillaCore.
- Stage 4: Improve the model accuracy
 - Collecting more features or optimizing VanillaCore to improve prediction accuracy.

Stages

- Stage 1: Read the paper we provide
 - To learn the essential knowledge for how to do latency estimation.
- Stage 2: Trace the code of our latency estimator
 - To understand how to write an estimator.
- Stage 3: Write a feature collector
 - Before training the model, you should be able to collect features from VanillaCore.
- Stage 4: Improve the model accuracy
 - Collecting more features or optimizing VanillaCore to improve prediction accuracy.

Assigned Reading for Stage 1

- SIGMOD'21 - MB2: Decomposed Behavior Modeling for Self-Driving Database Management Systems
 - To understand how to predict the DBMS's runtime behavior and resource consumption .

Motivation of MB2

- The main steps to configure a DBMS automatically:
 - Workload forecasting
 - Predicting the future workload.
 - **Behavior modeling**
 - Predicting the runtime behavior relative to the target objective (latency, throughput) given the predicted workload.
 - Decision planning
 - Selecting the actions that improves the objective.

MB2's Problem Formulation

- Given:
 - The workload
 - The system state
 - An action (e.g., adding an index, changing the number of threads)
- Predict:
 - Query Performance (Latency)
 - Resource consumption (CPU time, memory usage) for the action and other queries
- However, it has some assumptions.

MB2 v.s. VanillaCore

- MB2's assumptions
 - In-memory DBMS with optimistic MVCC
 - Its model cannot predict disk-based DBMSs.
 - Does not consider aborts due to data conflicts.
- However, VanillaCore is much more complicated.
 - It's disk-based
 - S2PL may arbitrarily abort transactions

How Can We Solve Those Problems

- To avoid the aborts due to locking, we can use deterministic conservative locking protocol.
 - No more random aborts
 - It ensures transaction order so that the latencies become more predictable
- We can use a large buffer pool to avoid most I/O.
 - I/O latencies are usually much harder to be predicted.
 - Note that I/O may still happen (e.g., logging or appending files)

What Should We Do?

- However, even though we have solved those issues, it is still not easy to predict latencies on VanillaCore due to...
 - Latches
 - Buffer Pools
 - Indices
 - Locks
- So, it is your job to find out how to predict latencies on VanillaCore.

We Provide

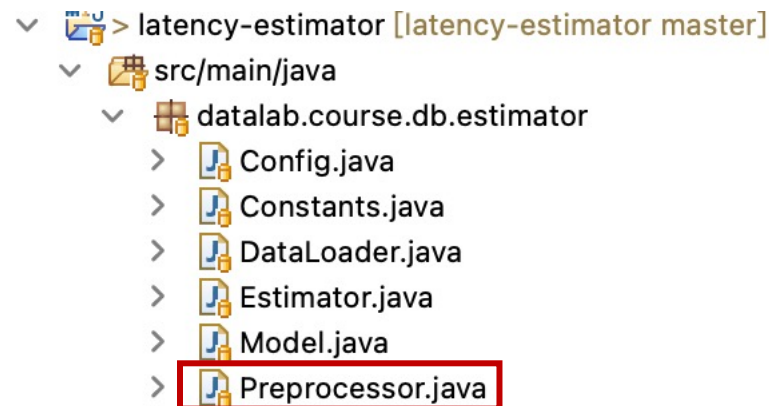
- Codebase
 - VanillaCore
 - VanillaBench
 - A latency estimator
 - An estimator that trains a random forest for latency prediction.
- Clone from here:
 - <https://shwu10.cs.nthu.edu.tw/courses/databases/2022-spring/db22-final-project>

Stages

- Stage 1: Read the paper we provide
 - To learn the essential knowledge for how to do latency estimation.
- Stage 2: Trace the code of our latency estimator
 - To understand how to write an estimator.
- Stage 3: Write a feature collector
 - Before training the model, you should be able to collect features from VanillaCore.
- Stage 4: Improve the model accuracy
 - Collecting more features or optimizing VanillaCore to improve prediction accuracy.

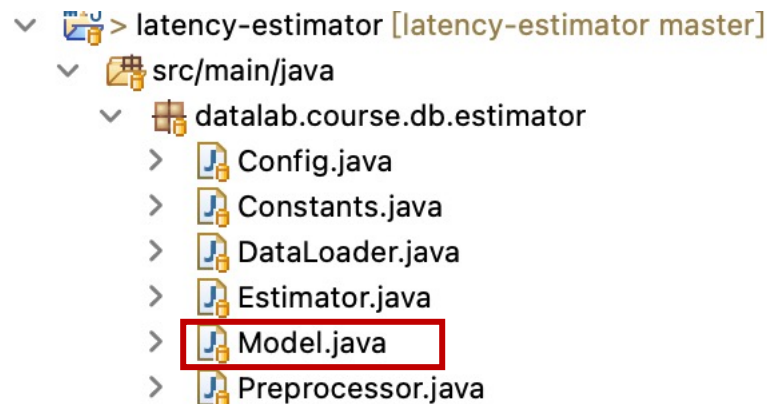
The Latency Estimator

- In the latency estimator, we provide:
 - A data preprocessor



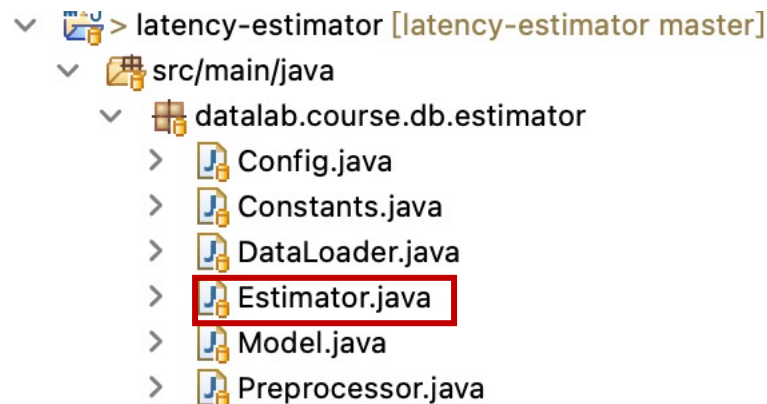
Latency Estimator

- In the latency estimator, we provide:
 - A data preprocessor
 - A random forest model
 - <http://haifengl.github.io/api/java/smile/regression/RandomForest.html>



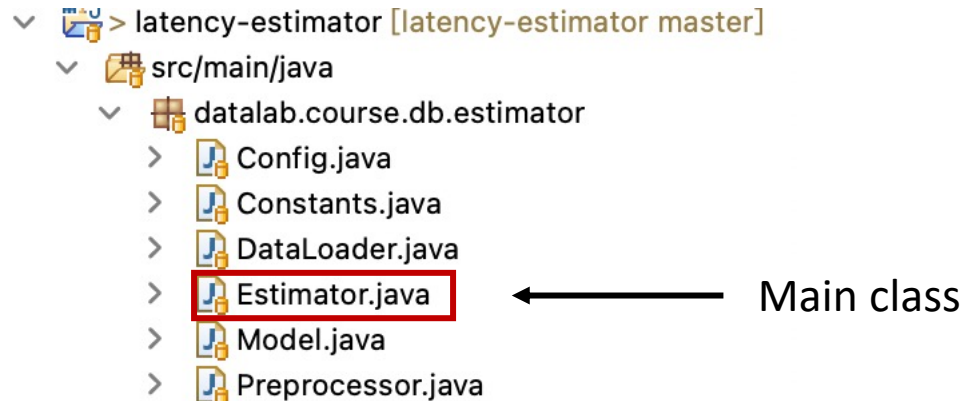
Latency Estimator

- In the latency estimator, we provide:
 - A data preprocessor
 - A random forest model
 - <http://haifengl.github.io/api/java/smile/regression/RandomForest.html>
 - An estimator which combines above together



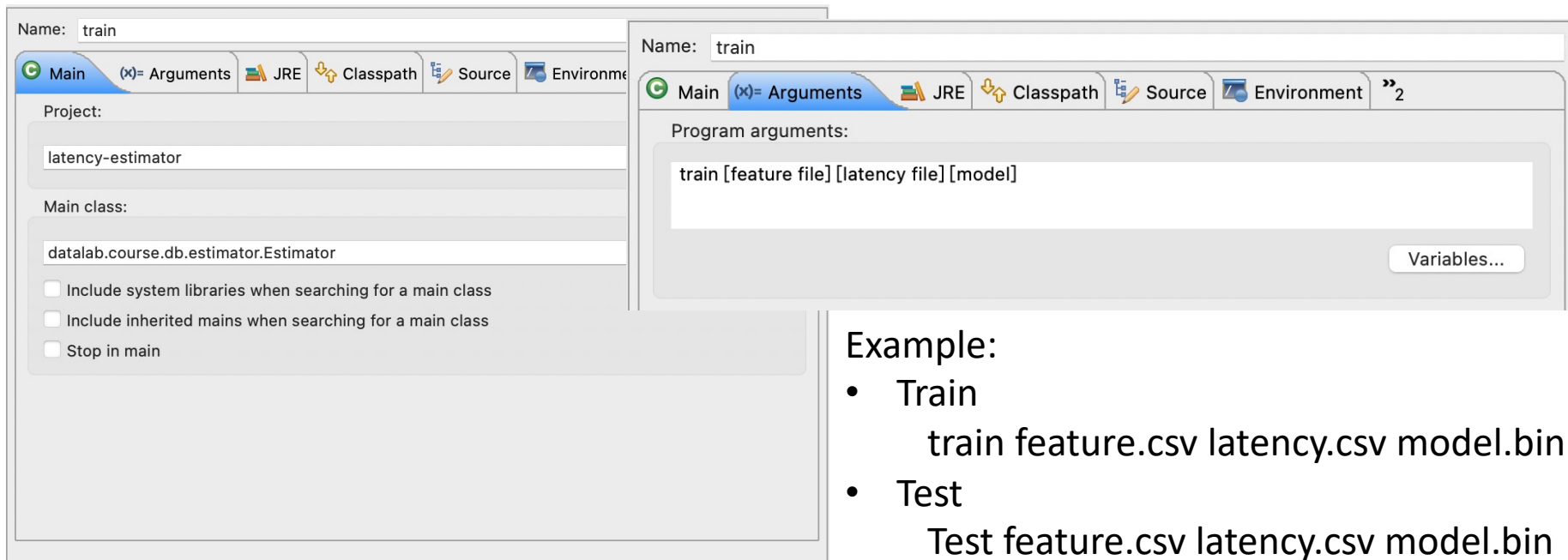
How to Use

- There are two ways to use latency estimator
 - Eclipse
 - Command line



How to Use

- There are two ways to use latency estimator
 - Eclipse
 - Command line

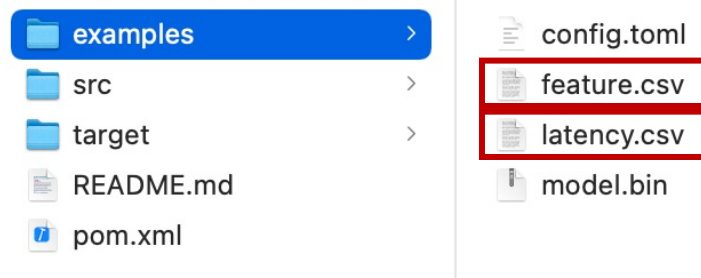


The image shows two overlapping Eclipse IDE windows. The left window is titled 'train' and shows the 'Main' tab. The 'Project' field contains 'latency-estimator' and the 'Main class' field contains 'datalab.course.db.estimator.Estimator'. There are three checkboxes: 'Include system libraries when searching for a main class', 'Include inherited mains when searching for a main class', and 'Stop in main'. The right window is also titled 'train' and shows the 'Program arguments' tab. The 'Program arguments' field contains 'train [feature file] [latency file] [model]'. There is a 'Variables...' button in the bottom right corner of the right window.

Example:

- Train
train feature.csv latency.csv model.bin
- Test
Test feature.csv latency.csv model.bin

How to Use

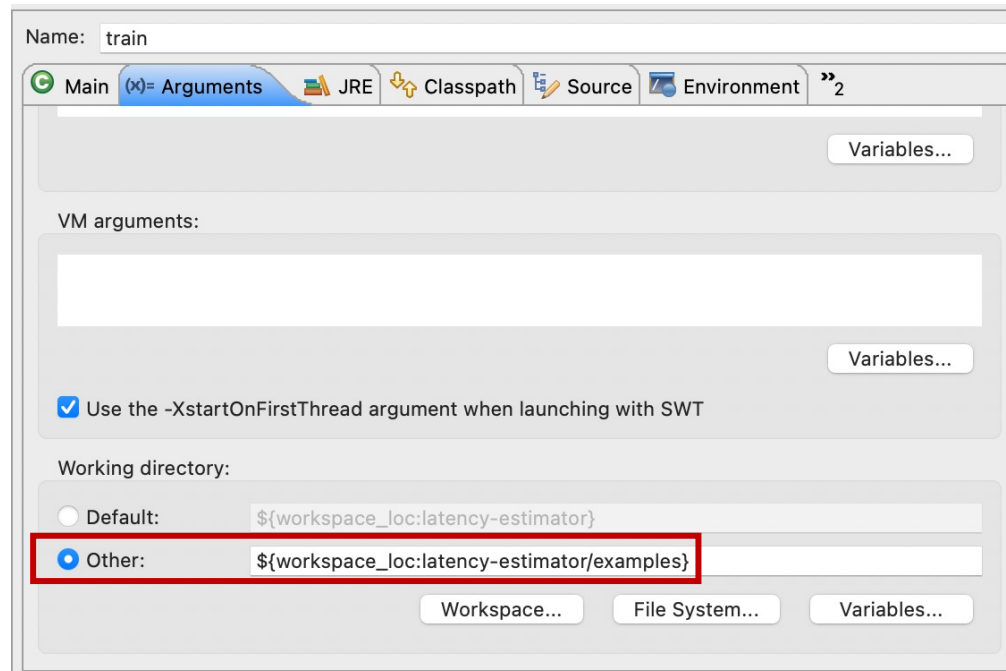


```
feature.csv
Transaction ID, Start Time, Read Count, Write Count
1, 871, 0, 0
2, 6804032, 5, 4
3, 6804032, 5, 4
4, 6804444, 29, 29
5, 6804560, 33, 33
6, 6842170, 5, 4
7, 6848203, 25, 25
8, 6857980, 5, 4
9, 6860231, 5, 4
10, 6864443, 5, 4
11, 6870689, 33, 33
12, 6873909, 5, 4
13, 6879729, 23, 23
14, 6886243, 17, 17
15, 6886308, 5, 4
16, 6893886, 5, 4
17, 6897247, 5, 4
18, 6904411, 17, 17
19, 6908675, 5, 4
20, 6916981, 5, 4
21, 6934994, 25, 25
22, 6940147, 21, 21
23, 6947138, 17, 17
24, 6950429, 19, 19
25, 6955148, 5, 4
```

```
latency.csv
Transaction ID, Latency
1, 5260723
3, 13449
5, 34124
6, 11401
7, 10315
8, 2663
9, 6368
10, 6345
11, 7510
12, 10834
13, 5225
14, 3375
15, 6120
16, 7005
17, 6795
18, 9454
19, 19302
20, 14570
4, 139843
21, 11155
23, 6123
```

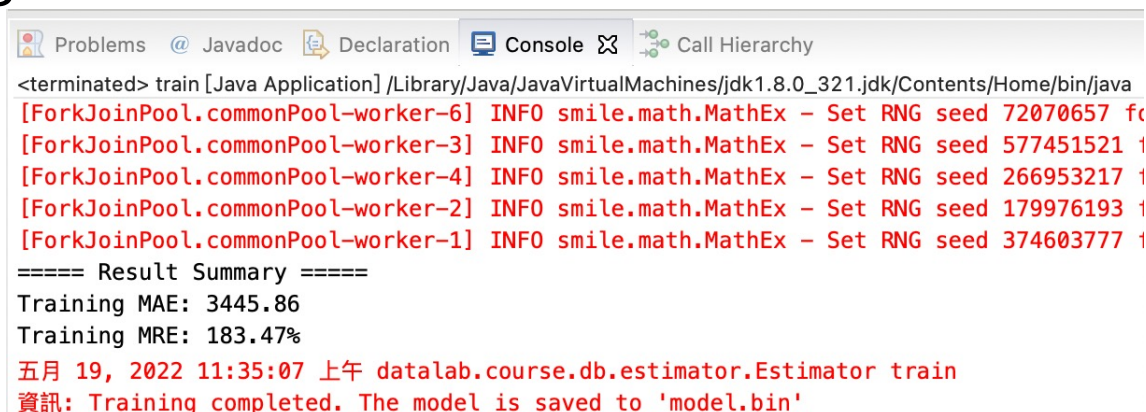
How to Use

- You can change your working directory to the folder you store your data.



How to Use

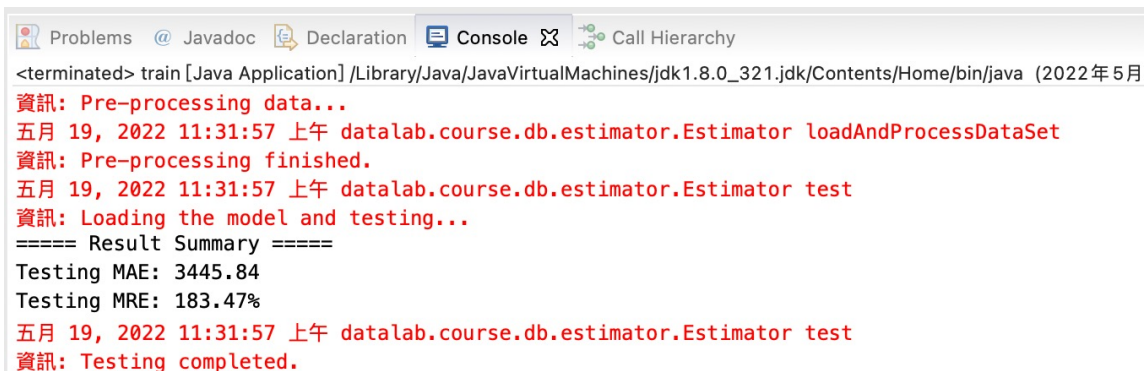
- Training result



The screenshot shows an IDE console window with the following text:

```
<terminated> train [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_321.jdk/Contents/Home/bin/java
[ForkJoinPool.commonPool-worker-6] INFO smile.math.MathEx - Set RNG seed 72070657 fo
[ForkJoinPool.commonPool-worker-3] INFO smile.math.MathEx - Set RNG seed 577451521 1
[ForkJoinPool.commonPool-worker-4] INFO smile.math.MathEx - Set RNG seed 266953217 1
[ForkJoinPool.commonPool-worker-2] INFO smile.math.MathEx - Set RNG seed 179976193 1
[ForkJoinPool.commonPool-worker-1] INFO smile.math.MathEx - Set RNG seed 374603777 1
===== Result Summary =====
Training MAE: 3445.86
Training MRE: 183.47%
五月 19, 2022 11:35:07 上午 datalab.course.db.estimator.Estimator train
資訊: Training completed. The model is saved to 'model.bin'
```

- Testing result



The screenshot shows an IDE console window with the following text:

```
<terminated> train [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_321.jdk/Contents/Home/bin/java (2022年5月
資訊: Pre-processing data...
五月 19, 2022 11:31:57 上午 datalab.course.db.estimator.Estimator loadAndProcessDataSet
資訊: Pre-processing finished.
五月 19, 2022 11:31:57 上午 datalab.course.db.estimator.Estimator test
資訊: Loading the model and testing...
===== Result Summary =====
Testing MAE: 3445.84
Testing MRE: 183.47%
五月 19, 2022 11:31:57 上午 datalab.course.db.estimator.Estimator test
資訊: Testing completed.
```

How to Use

- There are two ways to use latency estimator
 - Eclipse
 - Command line
 - <https://shwu10.cs.nthu.edu.tw/courses/databases/2022-spring/db22-final-project/-/blob/master/latency-estimator/README.md>

Latency Estimator

▼  > latency-estimator [latency-estimator master]

▼  src/main/java

▼  datalab.course.db.estimator

>  Config.java

>  Constants.java

>  DataLoader.java

>  Estimator.java

>  Model.java

>  Preprocessor.java

```
public static DataFrame preprocess(DataFrame features, DataFrame latencies, Config config) {  
    Stream<Tuple> featStream = features.stream();  
    Stream<Tuple> latStream = latencies.stream();
```

```
    // Filter warm-up transactions in features  
    featStream = filterWarmUpTransactions(featStream, config.warmupEndTime());
```

```
    // Sort  
    featStream = sortById(featStream);  
    latStream = sortById(latStream);
```










```
    // Filter warm-up transactions in latencies  
    DataFrame[] dfs = leaveMatchedRecords(featStream, latStream);
```

```
    // Drop id and start time columns  
    features = dfs[0].drop(Constants.FIELD_NAME_ID, Constants.FIELD_NAME_START_TIME);  
    latencies = dfs[1].drop(Constants.FIELD_NAME_ID);
```

```
    // Merge two data frames  
    return features.merge(latencies);  
}
```

<http://haifengl.github.io/api/java/smile/data/DataFrame.html>

Latency Estimator

- ▼  > latency-estimator [latency-estimator master]
- ▼  src/main/java
 - ▼  datalab.course.db.estimator
 - >  Config.java
 - >  Constants.java
 - >  DataLoader.java
 - >  Estimator.java
 - >  Model.java
 - >  Preprocessor.java

Latency Estimator

```
@Command(name = "train", mixinStandardHelpOptions = true)
public int train(
    @Parameters(paramLabel = "FEATURE_FILE", description = "path to the feature file") File featureFile,
    @Parameters(paramLabel = "LATENCY_FILE", description = "path to the latency file") File latencyFile,
    @Parameters(paramLabel = "MODEL_SAVE_FILE", description = "output path for saving the model") File modelSaveFile
) {
    // Load the configurations
    Config config = Config.load(configFile);

    // Load and process the data set
    DataFrame dataSet = loadAndProcessDataSet(featureFile, latencyFile, config);

    // Fit the model
    Model model = Model.fit(dataSet);

    // Report the results
    double trainingMae = model.testMeanAbsoluteError(dataSet);
    double trainingMre = model.testMeanRelativeError(dataSet);

    System.out.println("==== Result Summary =====");
    System.out.println(String.format("Training MAE: %.2f", trainingMae));
    System.out.println(String.format("Training MRE: %.2f%%", trainingMre * 100));

    // Save the model
    model.saveToFile(modelSaveFile);

    if (logger.isLoggable(Level.INFO))
        logger.info("Training completed. The model is saved to '" + modelSaveFile + "'");

    return 0;
}
```

Latency Estimator

```
@Command(name = "train", mixinStandardHelpOptions = true)
public int train(
    @Parameters(paramLabel = "FEATURE_FILE", description = "path to the feature file") File featureFile,
    @Parameters(paramLabel = "LATENCY_FILE", description = "path to the latency file") File latencyFile,
    @Parameters(paramLabel = "MODEL_SAVE_FILE", description = "output path for saving the model") File modelSaveFile
) {

    // Load the configurations
    Config config = Config.load(configFile);

    // Load and process the data set
    DataFrame dataSet = loadAndProcessData();

    // Fit the model
    Model model = Model.fit(dataSet);

    // Report the results
    double trainingMae = model.testMeanAbsoluteError(dataSet);
    double trainingMre = model.testMeanRelativeError(dataSet);

    System.out.println("==== Result Summary =====");
    System.out.println(String.format("Training MAE: %.2f", trainingMae));
    System.out.println(String.format("Training MRE: %.2f%%", trainingMre * 100));

    // Save the model
    model.saveToFile(modelSaveFile);

    if (logger.isLoggable(Level.INFO))
        logger.info("Training completed. The model is saved to '" + modelSaveFile + "'");

    return 0;
}
```

```
public static Model fit(DataFrame trainingSet) {
    RandomForest forest = RandomForest.fit(Formula.lhs(Constants.FIELD_NAME_LATENCY),
        trainingSet, 200, trainingSet.ncols() / 3, 2, 100, 5, 1.0);
    return new Model(forest);
}
```


Latency Estimator

```
@Command(name = "train", mixinStandardHelpOptions = true)
public int train(
    @Parameters(paramLabel = "FEATURE_FILE", description = "path to the feature file") File featureFile,
    @Parameters(paramLabel = "LATENCY_FILE", description = "path to the latency file") File latencyFile,
    @Parameters(paramLabel = "MODEL_SAVE_FILE", description = "output path for saving the model") File modelSaveFile
) {

    // Load the configurations
    Config config = Config.load(configFile);

    // Load and process the data set
    DataFrame dataSet = loadAndProcessDataSet(featureFile, latencyFile, config);

    // Fit the model
    Model model = Model.fit(dataSet);

    // Report the results
    double trainingMae = model.testMeanAbsoluteError(dataSet);
    double trainingMre = model.testMeanRelativeError(dataSet);

    System.out.println("==== Result Summary");
    System.out.println(String.format("Training MAE: %f", trainingMae));
    System.out.println(String.format("Training MRE: %f", trainingMre));

    // Save the model
    model.saveToFile(modelSaveFile);

    if (logger.isLoggable(Level.INFO))
        logger.info("Training completed. The model is saved to '" + modelSaveFile + "'");

    return 0;
}
```

```
public void saveToFile(File savePath) {
    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(savePath))) {
        out.writeObject(forest);
        out.flush();
    } catch (Exception e) {
        throw new RuntimeException("error while writing the model to file: " + savePath, e);
    }
}
```

Latency Estimator

```
@Command(name = "test", mixinStandardHelpOptions = true)
public int test(
    @Parameters(paramLabel = "FEATURE_FILE", description = "path to the feature file") File featureFile,
    @Parameters(paramLabel = "LATENCY_FILE", description = "path to the latency file") File latencyFile,
    @Parameters(paramLabel = "MODEL_SAVE_FILE", description = "path to the pre-trained model") File modelFile
) {

    // Load the configurations
    Config config = Config.load(configFile);

    // Load and process the data set
    DataFrame dataSet = loadAndProcessDataSet(featureFile, latencyFile, config);

    if (logger.isLoggable(Level.INFO))
        logger.info("Loading the model and testing...");

    // Load the model
    Model model = Model.loadFromFile(modelFile);

    // Report the results
    double testingMae =
    double testingMre =
    System.out.println("
    System.out.println(S
    System.out.println(S
    public static Model loadFromFile(File modelFile) {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(modelFile))) {
            RandomForest forest = (RandomForest) in.readObject();
            return new Model(forest);
        } catch (Exception e) {
            throw new RuntimeException("error while reading the model from file: " + modelFile, e);
        }
    }
    if (logger.isLoggable
        logger.info("Testing completed.");

    return 0;
}
```


Latency Estimator

```
@Command(name = "test", mixinStandardHelpOptions = true)
public int test(
    @Parameters(paramLabel = "FEATURE_FILE", description = "path to the feature file") File featureFile,
    @Parameters(paramLabel = "LATENCY_FILE", description = "path to the latency file") File latencyFile,
    @Parameters(paramLabel = "MODEL_SAVE_FILE", description = "path to the pre-trained model") File modelFile
) {

    // Load the configurations
    Config config = Config.load(configFile);

    // Load and process the data set
    DataFrame dataSet = loadAndProcessDataSet(featureFile, latencyFile, config);

    if (logger.isLoggable(Level.INFO))
        logger.info("Loading the model and testing...");

    // Load the model
    Model model = Model.loadFromFile(modelFile);

    // Report the results
    double testingMae = model.testMeanAbsoluteError(dataSet);
    double testingMre = model.testMeanRelativeError(dataSet);

    System.out.println("==== Result Summary =====");
    System.out.println(String.format("Testing MAE: %.2f", testingMae));
    System.out.println(String.format("Testing MRE: %.2f%%", testingMre * 100));

    if (logger.isLoggable(Level.INFO))
        logger.info("Testing completed.");

    return 0;
}
```

Stages

- Stage 1: Read the paper we provide
 - To learn the essential knowledge for how to do latency estimation.
- Stage 2: Trace the code of our latency estimator
 - To understand how to write an estimator.
- Stage 3: Write a feature collector
 - Before training the model, you should be able to collect features from VanillaCore.
- Stage 4: Improve the model accuracy
 - Collecting more features or optimizing VanillaCore to improve prediction accuracy.

Data Set Format Requirements

- Both feature and latency files should be in CSV format.
- Feature file should include the following fields:
 - Transaction ID
 - Start time
- Latency file should include the following fields:
 - Transaction ID
 - Latency

Stages

- Stage 1: Read the paper we provide
 - To learn the essential knowledge for how to do latency estimation.
- Stage 2: Trace the code of our latency estimator
 - To understand how to write an estimator.
- Stage 3: Write a feature collector
 - Before training the model, you should be able to collect features from VanillaCore.
- Stage 4: Improve the model accuracy
 - Collecting more features or optimizing VanillaCore to improve prediction accuracy.

Improve the Model Accuracy

- Collect more features
 - The features mentioned in MB2 can be tried.
 - Or other features you think are importance for tx latency.
- Change your model
 - You can only use either **random forest** or **linear regression**.
 - Changing model parameters is also allowed.
- (Optional) Do some optimizations to VanillaCore to improve predictability

Outline

- Project Goal
- Timeline
- Stages
- **Final Presentation**
- Demo
- Submission

Presentation

- 6/13 (Mon.) 15:00 @ Google Meet
- 5 mins presentation + 2 mins QA for each group
- TAs will announce the group order of presentation soon.

How to Present

- The presentation should break down into several parts.
 - Idea
 - What features should be considered and why?
 - What's your model (structure & model type)?
 - Experiments
 - On baseline workload and your custom workloads
 - Feature importance
 - Improvement in MAE and MRE score
 - Conclusion

How to Evaluate Improvement

- The baseline model & features
 - In your evaluation, you must compare your results with our original estimator with baseline features {Read Count, Write Count}.
- Workloads
 - The baseline workload
 - The default parameters of the TPC-C
 - Custom workloads
 - A workload to emphasis the strength of your model

Evaluation

- We will evaluate each group based on these 3 dimensions
 - Insight
 - Experiment
 - Presentation

Award

- Best Improvement Award
 - Consider both MAE and MRE score improvement
 - Let us know your improvement before presentation
 - <https://forms.gle/L4ca2PPXg2ChtDTr9>
- Best Presentation Award
- Each team that wins these awards can get a bonus score.

Outline

- Project Goal
- Timeline
- Stages
- Final Presentation
- **Demo**
- Submission

Demo

- **6/16** (Thu.) @ Google Meet
- Please fill in your team number to reserve demo time before 6/15 23:59
 - https://docs.google.com/spreadsheets/d/1TK_rilTIBLh3RY-YGzihzw_IWhlpElb1G4cQLB4hok8/edit?usp=sharing
- We will ask you to train your model on some new workloads, so make sure your model is ready before the demo.

Outline

- Project Goal
- Timeline
- Stages
- Final Presentation
- Demo
- **Submission**

Submission

- Requirements
 - You have to write a report as usual.
 - Include our suggestions for your presentation.
- The details for submission will be on our GitLab.
- Deadline: 2022/6/17 23:59



Good Luck !