System Overview

Erik Brännström

December 17, 2012

1 Introduction

This document aims to provide an overview of the design of the ad estimation system. For implementation specifics, please refer to the Javadoc and source code.

The system is started by running the Main class. Upon first launch it will use the Setup class to initialize the the database. Setup could be extended to for example create a directory and set its read/write properties, or any other tasks that should be run on first launch.

Main will then initialize the GUI class, which mostly utilizes other components in the GUI package to create the interface and handle user interaction. The rest of the packages will get a brief explanation below, and in the following sections all packages (except Evaluation) will get a little bit more detailed description.

- Core contains the most important application logic, e.g. for estimation.
- Data has classes for dealing with ad data as well as for interacting with the knowledge base and managing configuration settings.
- Evaluation was developed in order to statistically analyze different performance aspects of the system and is not used by the user application.
- IO has classes for importing and exporting data to and from the application.

The final section will describe resources that the system requires, which are located in the resources folder.

2 Core

The most important classes in the Core package are CombinationAdFactory and Estimator.

The CombinationAdFactory is the only implementation of the AdFactory interface, and it is used to create non-existing combinations of ad properties. These combinations can then be put through an Estimator to predict their performance.

The Estimator is really an abstract class which has two subclasses, one for numerical predictions and one for nominal predictions. The most common way to create an Estimator of either class is however to use the factory method in the abstract super class. It will automatically decide which type of estimator is required for the specified classification algorithm.

3 Data

The Ads class in the Data package is a subclass of the Weka Instances class, with some additional convenience methods.

Config and KnowledgeBase are both classes for easily interacting with the underlying database for storing and fetching data of their respective type. To use the correct database they both require the DatabaseHelper. The DatabaseHelper is also used when testing so that all database interactions are performed against a test database.

The AdDatabaseStorage is an implementation of AdStorage, and its purpose is to make it more convenient to use Weka's built in database manager to both fetch and store data. The interface exists in case other persistence layers are to be developed, such as for example file system storage.

Finally, KnowledgeBaseContainer is simply a class which can hold a single KnowledgeBase instance, and to which observers can register in order to be notified when the instance is replaced. Multiple GUI classes register to this to be able to update their components when the data set changes.

4 GUI

As mention in the introduction, the application's user interface is initialized in the GUI class. It depends on four custom views, which exist in the subpackage with the same name.

The AdsTable is a JTable extension which is responsible for displaying the ad data to the user. The table data can be filtered using the FilterPanel, which directly manipulates the table. The data which is shown in the table is selected based on the targeting options the user inputs in the TargetPanel. Finally, there is the Menu class which is a JMenu extension and contains all menu bar actions.

The Models subpackage contains the AdsTableModel, which contains the actual data that is displayed by the table. There are also two interfaces for targeting and selection which are used to pass data between different parts of the system.

Finally, the Controllers subpackage contains action listeners for the export action in the menu (ExportActionListener) and for the buttons which the user click on in order to reload the data in the table (DataActionListener). The DataActionListener is where the estimator is created and is thus a very central part of the application. The current classifier used is the IBk algorithm, chosen because of its speed and relatively high accuracy.

5 IO

The Exporter class uses a template file and key-value lists to export data on a specific format. The FacebookDataParser is used to handle the importing of Facebook reports, since they cannot be handled directly by Weka due to problems with character encoding and delimitation.

6 Resources

The resources folder contains files that are vital to the application. It is here that the Setup class will initialize the database for example.

It is also where the export-template.csv file exist, which is the main template used when exporting data. The top line in this file is the header which will be copied directly to the output file, and the second line will be repeated for as many times as needed. This line, the template line, can contain placeholders on the form {Key}, where the whole key including curly brackets will be replaced by the matching value for the key in the provided map.

Under weka/props is the DatabaseUtils.props file. This is where the database to be used is defined so that Weka can access it.

Finally, the resources directory contain a folder for files used when running the test suite for the system.

7 Deploying on Mac

The application can be executed from its JAR form, but to give it a more native feel we can bundle the application into a Mac application.

Xcode comes with an application called the Jar Bundler, which can be used for exactly this purpose. Launch the Jar Bundler and on the first tab, select the main class as the AdEstimator.jar file. Here you can also set the icon to be used. Next step is on the Classpath and Files tab, where you need to add all libraries to the application. Finally, on the Properties tab, add *-Dfile.encoding=UTF8* to the VM Options field in order for the application to correctly handle Unicode characters and check the box next to *Set Working Directory to Inside Application Package.*

Now press Create Application and select a name and location to store it. We still need to modify the contents of this application in order to add the required resources. Simply right-click the app and choose Show Package Contents. Go into Contents - Resources - Java and paste a copy of the DatabaseUtils.prop file in this folder. Also create a folder named *resources* and add the *export-template.csv* file to it.

The app is now ready to be deployed.