Chapter 8
Big Data Learning Analytics: A New Perpsective

Vivekanandan Suresh Kumar, Kinshuk, Thamarai Selvi Somasundaram, David Boulanger, Jérémie Seanosky and Marcello F. Vilela

Abstract Learners' attainment of academic knowledge in postsecondary institutions is predominantly expressed by summative approaches. Instead, recent advances in educational technology have hinted at a means to continuously measure learning attainment in terms of personalized learner competency, capacity, and effectiveness. Similarly, educational technology also offers guidelines to continuously measure instructional attainment in terms of instructional competency, instructional capacity, and instructional effectiveness. While accurate computational models that embody these attainments, educational and instructional, remain a distant and elusive goal, big data learning analytics approaches this goal by continuously observing study experiences and instructional practices at various levels of granularity, and by continually constructing and using models from these observations. This article offers a new perspective on learning and instructional attainments with big data analytics as the underlying framework, discusses approaches to this framework with evidences from the literature, and offers a case study that illustrates the need to pursue research directions arising from this new perspective.

V.S. Kumar (✉) · Kinshuk · D. Boulanger · J. Seanosky
Athabasca University, 1200, 10011-109 St, Edmonton, Alberta T5J 3S8, Canada
e-mail: vive@athabascau.ca

Kinshuk
e-mail: kinshuk@athabascau.ca

D. Boulanger
e-mail: david.boulanger@dbu.onmicrosoft.com

J. Seanosky
e-mail: jeremie@rsdv.ca

T.S. Somasundaram
CARE Lab, Anna University, MIT Campus, Chennai, India
e-mail: stselvi@annauniv.edu

M.F. Vilela
Universidade Federal Fluminense, 175, Oswaldo Terra St, Valença, RJ 27600-000, Brazil
e-mail: marcellovilela@id.uff.br

Kinshuk and R. Huang (eds.), *Ubiquitous Learning Environments and Technologies*,
Lecture Notes in Educational Technology, DOI 10.1007/978-3-662-44659-1_8

Keywords Big data learning analytics · Data mining · Artificial intelligence in education · Feedback · Assessment · Modeling · Collaboration · Social networks · Curriculum assessment

8.1 Data and Learning

Whether observed or not, data is being continually produced in all phases of learning. For example, a classroom lecture could generate new data about the presentation material, learners' level of comprehension, conceptual associations, and insights. An online discussion, for example, could induce data about learners' awareness of the topic under discussion, their emotive states, learners' peer networks, and instructional challenges of discussion procedures. A blended training exercise, for example, could create new data on learners' confidence, their reflections, and new regulations of the industry. Lifelong work scenarios, for example, could generate data on recognition of new skills, refinement of competencies, work—study relationships, and models of studies. Instructional design efforts generate their own sets of data concerning course design models, development workflow, course guidance, adaptation opportunities, course comprehensiveness, and course quality. Ubiquitous learning environments could raise new datasets on perspectives and new behavior from learners, instructors, and software agents. Irrespective of where, why, how, and when learning happens, learning environments do create new learning experiences and correspondingly new sets of data.

Each learning experience can be measured. Be it personalized learning, formal learning, adhoc learning, classroom learning, or mobile learning, different types of learning environments offer different types of learning and instructional attainment datasets. The associations that evolve among these datasets on a day-to-day basis have conveniently and predominantly been ignored since the beginning of modern era. For instance, from a learner perspective, summative assessments, augmented with formative assessments, are used to only estimate competency of the learner. The capacity of the learner and the effectiveness of learning processes undertaken by the learner are not measured normally. This is not because of the challenges involved in obtaining and validating data on capacity and effectiveness, but because of the pervasive nature of 'lethargy' in teaching.

Learner competence can be measured to a greater extent using a variety of assessments. Learning capacity, on the other hand, depends on a number of personal factors including working memory, motivation, support, learning material, and learning styles. Learning effectiveness is a measure that mainly involves competence and capacity. Capacity and effectiveness have generally been ignored by contemporary instruction attributing to the lethargy mentioned above. Expectation of the "bell curve" as the norm of classroom performance is another factor that attributes to lethargy in teaching. Constraints placed on learning outcomes force instructional processes to be satisfied mostly with summative feedback-based associations, offering further attributions to lethargy.

Learning processes/activities undertaken by individual learners from the time a concept to be learned is introduced up to the time instructional efforts cease to teach that concept to that learner can potentially be observed using a variety of techniques and technologies. For instance, the behavior of classroom interactions of students attending a lecture can be recorded and analyzed at real time to estimate the level of comprehension of a set of concepts introduced in the lecture. The behavior of online interaction of students in a discussion forum can be analyzed at real time to estimate their writing competence. The behavior of social interactions of students can also be analyzed to estimate their sentiments on specific services offered by the institution related to learning. These exclusive observations, along with other traditional data, can supply the ``raw data' for learning analytics.

One of the early studies on big data was found in a report on a course titled "very large scale modelling" offered at the Simon Fraser University in 2004 (Kumar et al. 2007). The study was conducted as part of this graduate course, where students attempted to integrate datasets from "the entire gamut of modeling, from mathematical modeling, data modeling, process modeling, symbolic modeling (AI), social interactions modeling, to modeling the computation (distributed, grid, and parallel)." Students worked on individual projects across these topics and attempted to relate various datasets to extract meaningful conclusions. Raw data is being produced at an alarming rate in other domains including transportation, healthcare, geographic advertising, retail, banking, and energy.[1] A report on online learning in Canada[2] outlines the market share of LMS (learning management systems) that already produces large quantities of data on learning and the inclusion of online learning as part of core business plan of academic institutions across Canada. An infrastructure that supports learning resource discovery, sharing, and amplification indicates to a large volume of metadata and data (Bienkowski et al. 2012). At risk, students can be recognized based on data about the type of risk, interventions based on predictive models can be designed to mitigate that risk, and the utility of the models can be gaged by the trace data on applied intervention (Essa and Ayad 2012). A number of data-driven analytics drivers to mine effectiveness of learning have been reported for use in learning and academic analytics (Ferguson 2012). Social learning analytics offers access to a slew of data on informal conversations among students and instructors (Shum and Ferguson 2012). Raw data is not confined to formal and informal datasets, but also includes adhoc datasets with marginal overlap with learning. Adhoc datasets include browsing patterns, reading habits, writing style (including freehand writing), coding, posture analyses, collaboration drivers, thinking protocols, chats, domain-specific tool traces, recording of cognitive and metacognitive traces, and knowledge traces.

Among many others, raw data can be processed towards data mining (Romero et al. 2008), program evaluation (Hung et al. 2012), real-time classroom feedback

[1] http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation.

[2] http://www.contactnorth.ca/online-learning-canada.

(Wood et al. 2012), prediction (Watson et al. 2013; Abdous et al. 2013), visualization (Mazza and Dimitrova 2003; Kim and Lee 2013), and modeling (Medeiros et al. 2013), among others. Potential observations and analyses of the aforementioned categories would shed light on instructional processes that indicate introduction, deliberation, assimilation, evaluation, and application of a concept or skill to various degrees by different types of learners. However, a holistic view of the rates of changes in adaptations, repetitions, and refinements of instructional processes and instructional resources, as well as the rates of changes in learning processes and learner capabilities have never been observed as a matter of fact—till now, till the arrival of learning analytics as the science of analysis that concerns learning, and the suite of tools that observe, validate, and semiautomatically associate datasets related to learning and instruction.

Learning analytics is the science of analysis, discovery, and utilization of learning traces in emergent and related levels of granularity. Analysis could include techniques ranging from data mining to machine learning and to big data analysis. The discovery of new relations and the discovery of even new data include unconventional data, for examples, the family income of a politically competing region, inherent economic drivers influenced by a curriculum, and rate of changes in motivation levels of students with respect to weather. Relations of interest include sentiments among learners across collaborating groups, interinstitutional credit transfer policies among institutions, and mutual respect among instructors. Trace data refers to observable raw data of study activities such as reading, writing, conceptualizing, critically thinking, solving problems, storytelling, and visualizing, where a network of study activities lead to a measurable chunk of learning. For instance, the types of sentence openers used by a learner, the range of errors that the student can confidently correct, the level of trust exhibited by the student in sharing information in a forum, and the depth of understanding in a set of concepts are examples of learning traces where one could measure learning over time. In learning analytics, data is expected to arrive continuously, typically in an interleaved fashion, subject to interpretation at various levels of granularity.

In general, learning traces translate raw data into incoming data for learning analytics, where incoming data are typically big, unstructured, unrelated, and fits multiple models and possibly multiple theories. Importantly, learning traces capture highly personalized study experiences. For instance, consider the example of studying about the notion of a pointed object penetrating better than a blunt object. The study goal is to understand why this is so. A visual-oriented learner may choose to study and explain this phenomenon using visual tools, while a fellow student may workout the mathematics behind pressure and explain the results in terms of equations. Thus, learning traces capture different types of skills and background knowledge exhibited by individual learners, the extent to which learners learned new concepts while studying a particular material, the amount of time (efficiency) learners took to study the material, the effectiveness of assessments to conclude that the concept had been learned efficiently, evidences of learning activities/experiences (e.g., a video of the learner trying a sharp pencil on someone's palm and the learner's answer to the question on pressure in the examination), and resources used by learner (e.g., the pencil, simulation run by the learner). While each learning trace is measurable, there is no standard scale of measurement that applies across different types of traces. A learning trace is the least common denominator for a measure of learning.

Data mining offers techniques that typically operate on well-defined, medium-sized datasets (up to gigabytes). Intelligent tutoring systems typically operate on limited-sized data (up to megabytes). Learning analytics aims to operate on large volumes of data (at least in the order of terabytes, hence the name, bigdata) as well as large volumes of models produced from the data. The models are dynamic, if not emergent, because more often than not, processing of data would include newly arriving, marginally related datasets. Examples of emergent models include 'reading habits across domains,' 'writing styles across contexts,' 'application of problem solving skills across tools' (SPSS, R, Eclipse, Matlab, and so on), 'social distances while collaborating with peers', 'comfort zone when interacting with instructors,' and 'research potential'.

Data can be gleaned with sensors. Contemporary technologies only offer sensors that observe a tiny fraction of data associated with these new learning experiences. In learning analytics, one could conceive of three types of sensors—instructional sensors, context sensors, and internal sensors. Instructional sensors have explicit association with the study as seen in LMSs. Intelligent tutoring offers a wide variety of instructional sensors such as sensors to observe self-regulation activities of learners and sensors that observe the interactions between learners and anthropomorphic pedagogical agents. Context sensors include the observable environment in which learning takes place. Examples of context sensors include autonomous software agents that act independently on students' behalf and data traces that cluster students from across multiple institutions based on specific competencies. Internal sensors include methods that estimate learners' ability to think laterally and critically, ability to strategize in a game environment or regulation exercise, ability to use working memory and comprehension techniques, and the ability to pace oneself and proactively study. The observations of these three types of sensors and the interrelations among individual data streams produced by each sensor yield the raw data.

While big data has been a recognized field of exploration in many domains such as traffic regulation, healthcare, geographic advertising, retail, banking, public sector, consultancy, energy efficiency, and policing, applications of big data techniques in learning is still in its infancy. Learning attainment and instructional attainment need to synchronize with each other to offer an optimal approach to learning. In addressing this very goal, this chapter highlights the application of learning analytics with a case study. The case illustrates the collection of raw data, the transforming the raw data into learning traces, the application of learning analytics techniques on learning traces, the use of results of analytics towards balancing learning and instructional attainment, and the measurement of impact of analytics.

8.2 Big Data Coding Analytics

Students taking an introductory computer programming class in an online environment, or in any course with a significant digital component, have access to learning activities associated with an array of media—including slideshows, reading material, audio/video lectures, interactive tutors, and automated tests. And, as they work through the course material in this digital environment, students generate learning activity traces that track their interaction with the course material, much like a web analytics system tracks visitors to a website. Similarly, as students complete the associated computer programming assignments, software development traces are generated which can be used to track the student success and coding competency. The same traces can then be used to assess the effectiveness of instruction.

Here we present a study that uses a combination of software tools to collect data generated from 240 students at the Madras Institute of Technology (Chennai, India) while they completed a course on "Introduction to Programming in C." The students interacted with the Moodle online education platform in addition to a number of software tools. The quantity of data collected using the additional software tools is much larger than the data that are traditionally collected from learning management systems. Thus, it approaches the realms of big data learning analytics. We present the types of data we collected, followed by details of the underlying technology, and some outcomes of the study.

8.2.1 Data Collection

A Moodle-based course was created to supplement the classroom instruction for the course on "introduction to programming in C." Student interactions with the Moodle content were captured from the Moodle database. Students also completed a built-in survey questionnaire in Moodle to indicate their experience with the Moodle course. Moodle's reporting module compiles learning activity analytics allowing for tracking of all logged on student interactions with the learning material.

Being a programming course, it required students to write programs in C language. Software development traces are generated by students as they solve computer programming problems, such as the number of compiles, warnings and errors encountered during these compiles, ability of the student to tackle these warnings and errors, the ensuing design changes, documentation efforts, and the testing of code can be captured from their development environments, along with the student learning traces from the digital course materials accessed for the duration of the course.

The limit to what can potentially be captured is determined only by the functionality of the Integrated Development Environments (IDE) and the data streams accessible in the IDE. The data generated from each learner's activities is owned solely by the learner. Learners have to explicitly consent to sharing this data with the researchers in each and every coding session. Learners also have the ability to

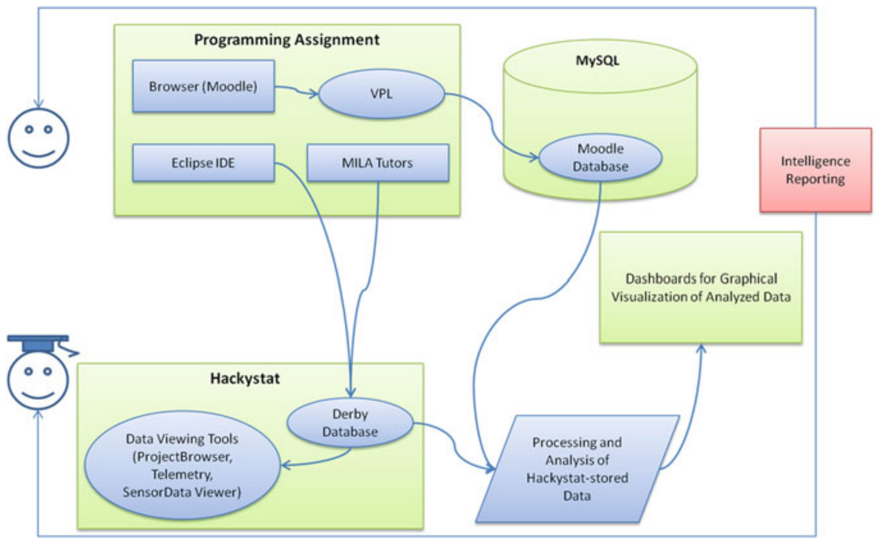

Programming Assignment
Browser (Moodle)
VPL
Eclipse IDE
MILA Tutors
MySQL
Moodle Database
Intelligence Reporting
Dashboards for Graphical Visualization of Analyzed Data
Hackystat
Derby Database
Data Viewing Tools (ProjectBrowser, Telemetry, SensorData Viewer)
Processing and Analysis of Hackystat-stored Data

Fig. 8.1 EIDEE system architecture

withdraw any and all of their data from the repository at any time of their choice. Further, the system will only publish aggregated results and not individualized results. The individualized results are only shared with the individual concerned. Such a strict ethical guideline is important because of the implications that can be derived from the observed data on competencies and how these implications could be used (or misused) in a person's life.

A software system called Hackystat[3]—an open source project originally developed by Researchers in the University of Hawaii—acts as a central repository for collecting information on software development activities and logs from a variety of development environments, including Eclipse, Emacs, and Visual Studio.

An open source Moodle plug-in developed by University of Las Palmas de Gran Canaria called Virtual Programming Lab (VPL)[4] was altered to perform a similar data collection task from within Moodle itself for less intensive programming exercises.

Further, highly personalized software tutors called MI-LATTE (mixed-initiative learning analytics tutoring and training environment) have also been developed. Each MI-LATTE tutor targets a single assignment problem, where learners are expected to solve the problem by writing "one line of code" at a time. The tutor analyzes this single line of code and assesses the skill associated with the writing of that line of code. Data collected from learner interactions with each MI-LATTE along with any identified skills are collected.

[3] http://code.google.com/p/hackystat/.

[4] http://vpl.dis.ulpgc.es/.

Finally, learner interaction data with the Eclipse IDE Extension (EIDEE) captures data concerning code design steps (e.g., UML design), code writing, code debugging, code documenting, code testing, code review/reflection, and code optimizing. Figure 8.1 depicts the system architecture.

Using a combination of static analysis toolkits, depending on the language in use (for example, Java program analysis uses a combination of Checkstyle, FindBugs, and Junit, while C programs use a combination of Lint, and Virtual Programming Lab customizations), and manual grading by Teaching Assistants, the work of the students are graded and the results obtained from these tools are logged—along with the final grade on the assignment problem. These final grades and the marks scored by a student corresponding to each rubric in each assignment problem are also collected as raw data.

8.2.2 Technology and Datasets

Figure 8.2 presents the underlying technology used to collect continuous sets of raw data.

The key data in this flow is the continuously arriving student source code. Using the three tools, Eclipse extension (EIDEE), mixed-initiative tutors (MI-LATTE), and embedded coding interface (VPL), we collected a large quantity of data on students' coding habits. We also collected data from Moodle on browsing patterns, formal assessments (e.g., quiz and exams), student survey results, and student discussion contributions in forums.

To collect the raw data in a central repository, Hackystat sensors are embedded within Eclipse, within VPL, within EIDEE, within MI-LATTE, and within Moodle. All data are sent to the Hackystat sensorbase. The sensorbase, a Derby database, contains all the data collected from Hackystat sensors. Hackystat provides a helpful layer of abstraction through its Java API to query and display the data in a more intuitive way.

The raw data is processed continuously to populate two ontologies to discover and maintain a coding profile for each student and to track changes to this profile. Each ontology captures the mastery level of the concepts of a specific area within the coding domain. For instance, given a specific programming problem, the abstract syntax tree of the student source code will be time-stamped and stored in a first ontology. This ontology records the state of the abstract syntax trees generated by the student's code during the problem solving process. A second ontology captures all the errors made by the student. This ontology will store the results generated by code analysis tools such as Eclipse JDT[5]/CDT[6] compilers, FindBugs, CppCheck, Checkstyle, etc. To handle ontologies programmatically, we use the

[5] http://www.eclipse.org/jdt/.

[6] http://www.eclipse.org/cdt/.

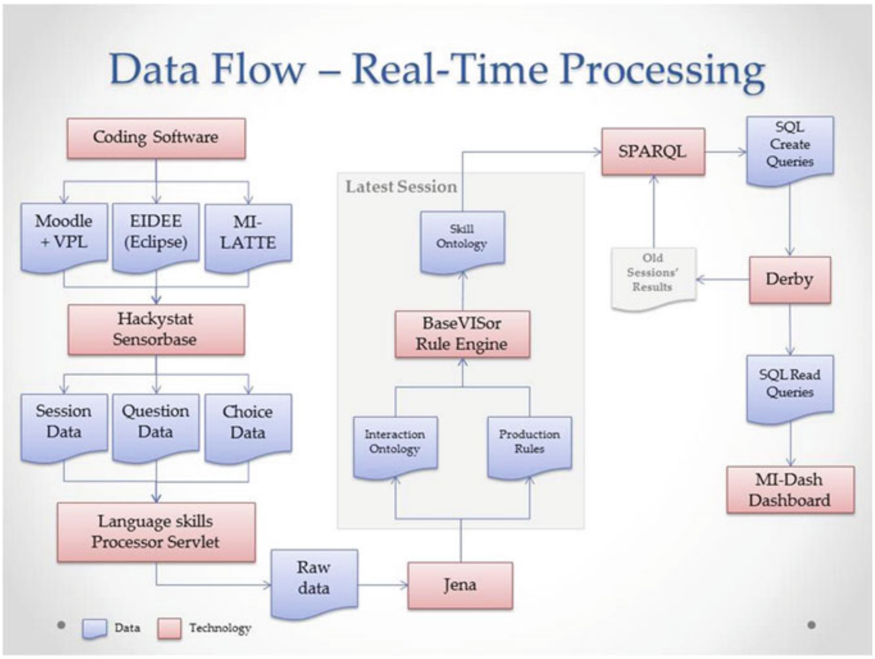


Fig. 8.2 Dataflow of the EIDEE system

Apache Jena framework. Ontologies are written in the RDF or OWL format depending on the needs of expressivity or power of inference engines.

Further, we built a knowledge base to interpret the performance of students and the results from code analysis tools in terms of skills, mastery of concepts, cause-effect relationships, and cognitive models. For instance, if Hackystat receives information about the student's struggles in debugging a particular type of error, the instances of struggles (from the second ontology) are recorded in the ontology along with the coding context (from the first ontology) in which these struggles were observed. This leads to an estimate of time it would take the student to solve a particular type of error could be estimated. This estimate is an inferred competency and further changes to such estimates are also recorded. BaseVISor,[7] an RDF-based forward-chaining inference engine, implements all the production rules in an XML format and records the state of the working memory of the student. The results of the inferences made by BaseVISor are stored in new ontologies.

These results serve as the contents of a dashboard called MI-DASH. The dashboard, we hope, will empower students to manage their own coding habits by interacting with their profiles. Ontologies can be queried with SPARQL, an RDF query language, and will be translated into a SQL database which will help in creating the contents displayed in the dashboard.

[7] http://www.vistology.com/basevisor/basevisor.html.

8.2.3 Hackystat Sensors

Hackystat provides a range of sensors to collect data from tools, such as Eclipse, Ant, Checkstyle, Clover, FindBugs, JUnit, Visual Studio, tools generating XML data, etc. Hackystat also provides a sensor shell Java API which supports the creation of customized sensors to increase the range of tools monitored by Hackystat sensors.

The main advantage of the sensor shell is that it has a built-in data integrity mechanism; that is, the sensor sends packets of data to Hackystat at specified time intervals or when an event occurs (clicking a button, pressing the return key, etc.) via a network (be it intranet or internet). However, if for some reason network connectivity is lost while data being sent, the sensor shell will temporarily store the data in XML files on the user's local machine until connectivity is restored. Then, any local sensor data is automatically pushed on the Hackystat sensorbase. This ensures that no data is lost due to some unexpected loss of connectivity.

Since Hackystat is open source, we modified its Eclipse sensor to suit the particular needs of our experiments. Essentially, we kept the data types originally designed with the sensor and added new ones. The sensor was also modified to fit various versions of Eclipse for various packages (Galileo, Indigo). Finally, we created our own Eclipse package with our customized Hackystat Eclipse sensor embedded. That package also contains course assignment projects ready to be monitored by the sensor. The purpose of this package is to hide the complexity of the sensor configurations from the students.

The Eclipse sensor enables the collection of different data types during code development. The three types of data captured by the sensor are Edit, Build, and Debug. It tracks any operations made on a file such as opening, saving, or closing a file. It also records any addition, renaming, removal, and move of programming constructs. Any failed compilation is also sensed and sent to the sensorbase. The sensor also reports debugging activities such as the start and termination of debugging sessions, the setting and removal of breakpoints, and the passage of the debugger into or over a block of code. We also modified the sensor, so that the source code may also be collected as the student types. The sensor collects the source code at 1-s intervals or whenever the student compiles his/her code (saves the code file), so that we might reconstruct his/her coding efforts and the evolution of the code.

As for the MI-LATTE tutor, we built a new sensor using the sensor shell API. The MI-LATTE tutor records the student interaction within its interface and forwards them to Hackystat. Each tutor is downloadable and consists of a client-side Java Web Start application and a server-side Java servlet. The Java Web Start application stores data collected from the student in a buffer and sends the data over HTTP post messages to the Java servlet. The servlet contains a customized sensor which sends to the Hackystat sensorbase all the instances of data types collected by the client application. The servlet also parses the final version of the student's source code to generate the corresponding abstract syntax tree and compiles the code using the Eclipse JDT/CDT compiler. Eventually, every capture of source code is built and parsed, so that we can track the evolution of errors during problem solving (for instance, which errors occurred and were solved properly by the student in the process and how much time it took him/her to solve them). Any error reported after a failed compilation is sent to the Hackystat sensorbase. Source code is captured as a DevEvent sensor data type, while errors are captured as CodeIssue sensor data types.

Developing the Moodle web sensors was a challenge. The Hackystat sensor shell (as well as all of Hackystat) is built in Java, but since Moodle is built using PHP, JavaScript, and AJAX, we used a framework called PHP/Java Bridge[8] which is designed to allow the integration of Java libraries inside PHP code and vice versa. Therefore, we used the sensor shell Java API inside the Moodle PHP code to build a Hackystat sensor for Moodle that uses the data integrity feature of the Java sensor shell. We created event handlers in Moodle's code that incorporate the Web Hackystat sensors which send data about the event that just occurred (action type, timestamp, action context, any user input, etc.) to the Hackystat sensorbase.

8.2.4 Feedback Mechanism

We define learning analytics as the science of analysis, discovery, and utilization of learning traces in emergent and related levels of granularity. Learning analytics is also defined as "the measurement, collection, analysis, and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs."[9] The data collected are being processed and analyzed in order to improve the student's experience and increase the quality of tutoring services through human and software agents.

At this stage, the MI-LATTE tutor performs three tasks: provides instructions, points out errors, and offers corrective actions. The role of the human agent with the software agent will be to provide it with a set of alternative solutions. An alternative solution may follow a different algorithm, a different sequence of programming constructs, different variable names, etc.

The feedback from the software agent is optional. The student can solve the problem by himself/herself and submit his/her work for evaluation without resorting to any automated feedback from the software agent. Feedback will only be given on demand. Examples of types of MI-LATTE tutor feedback messages are given below:

- All the written code so far is correct. Here are the instructions for the next statement.
- The statement at Line 5 is incorrect. Please look at the instructions again.

8 http://php-java-bridge.sourceforge.net/pjb/index.php.

9 https://tekri.athabascau.ca/analytics/.

- The statement at Line 5 is incorrect. Please look at the instructions again. You can also look at the reported compiler error message.
- The statement at Line 5 is incorrect. Please replace this section of your code with the following string.
- Here is the code section corresponding to what you have written so far. Please study it and try to solve the remaining part(s) of the problem.
- Here is the whole code. Please study it and try to solve the problem a second time by yourself.

Moodle and VPL also offer opportunities for providing instantaneous tutoring/feedback to students. This Hackystat-based technology allows us to customize Moodle to include feedback messages for the students as they work on a quiz or write a forum posting or view learning material on Moodle, or complete a programming assignment in VPL. By storing personalized messages in a database and even creating generic event handling sensor code that Moodle administrators can add or remove from the actual Moodle code at will to suit their individual needs, one can create a very powerful interactive tutoring feedback-based Moodle/VPL environment which, in addition to collecting data for later analysis and feedback, also provides just-in-time feedback to the student.

8.2.5 Ontologies

When a student works on a programming exercise, two ontologies are populated. The first ontology contains the abstract syntax tree of the final version of his/her source code while the second ontology stores the corresponding compile-time errors. We call the first ontology the AST (Abstract Syntax Tree) ontology and the second ontology the Error ontology. We currently use Protégé,[10] an ontology editor, to view the contents of the ontologies.

In the AST ontology, we can view the different programming constructs employed by a student, the relationships between the various nodes in the AST as well as the code corresponding to a specific node in the tree. We can also view graphically the abstract syntax tree through the OntoGraf[11] module, a plug-in installed by default in Protégé.

In the error ontology, we can view all the errors encountered by a student. The recorded information about a specific error will include the type of the error, the source code line number, the severity of the problem, the start and end character positions of the error, and an explanative error message. In brief, we can retrieve all the information generated by the Eclipse JDT/CDT compiler.

[10] http://130.88.198.11/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf.

[11] http://protegewiki.stanford.edu/wiki/OntoGraf.

Since we propose to build the student's source many times per session, we want also to find a way to identify errors and keep track of them to study how students resolved them.

We manage the ontologies programmatically using the Jena framework. One of the most interesting aspects in the overall framework is the possibility to customize actions that will be carried out on the occurrence of a specific programming construct when the source code is parsed. The Eclipse JDT/CDT APIs offer a library of Java classes enabling to tailor the parsing process. Hence, we use the Jena framework to generate the AST ontology on the occurrence of all programming constructs. Interestingly, this enables us to build Hackystat sensors for Eclipse for the C and C++ languages as well as any other tools such as the MI-LATTE tutors which make use of the Eclipse API to implement its own features in a standalone Java application.

These ontologies play an important role in the analysis and report of the data collected from the students. These ontologies could be processed at multiple levels of granularity focussing on the assignment problem, or the student, or the course, or the institutions, and so on.

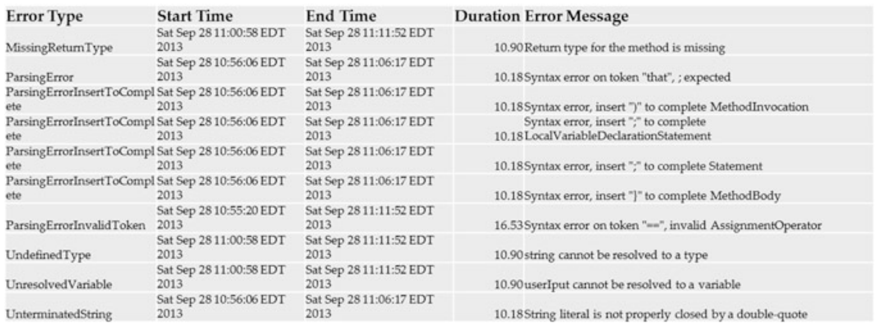
BaseVISor along with SPARQL provide key tools in performing large-scale inferencing and a detailed statistical analysis (using R) to help discover new patterns relating to a programming context. Based on the results of inferencing and statistical analyses, one could determine cause-effect relationships between various occurrences of specific constructs and/or errors. These could then be translated back into BaseVISor production rules to expand the knowledge base and enhance the results of later analyses.

The coding profiles of students have been assessed and modeled in the BaseVISor inference engine. Changes to the state of the working memory are brought back into the ontology. The resulting ontology can be translated into a SQL database by querying the ontology with SPARQL, an RDF query language, and translating SPARQL result sets into SQL queries to create SQL tables and insert values into those tables. A SQL database will be the repository of processed data where dashboards can retrieve information.

8.2.6 Sample Datasets and Analyses

Dataset collected through the Eclipse IDE and the MI-LATTE tutors are classified and stored in the Hackystat sensorbase. The data are then merely gathered and converted to ontologies for extensive inferencing in BaseVISor. Data are classified according to the tools which generated them and their type. Two data types exist at this point: DevEvent and CodeIssue. The DevEvent data type stores source code. The CodeIssue data type stores error instances generated by the Eclipse JDT/CDT compiler.

By time-stamping, the builds of a student's source code as well the error messages generated for the identified errors (see Fig. 8.3), an instructor could be



Error Type	Start Time	End Time	Duration	Error Message
MissingReturnType	Sat Sep 28 11:00:58 EDT 2013	Sat Sep 28 11:11:52 EDT 2013	10.90	Return type for the method is missing
ParsingError	Sat Sep 28 10:56:06 EDT 2013	Sat Sep 28 11:06:17 EDT 2013	10.18	Syntax error on token "that", ; expected
ParsingErrorInsertToComplete	Sat Sep 28 10:56:06 EDT 2013	Sat Sep 28 11:06:17 EDT 2013	10.18	Syntax error, insert ")" to complete MethodInvocation
ParsingErrorInsertToComplete	Sat Sep 28 10:56:06 EDT 2013	Sat Sep 28 11:06:17 EDT 2013	10.18	Syntax error, insert ";" to complete LocalVariableDeclarationStatement
ParsingErrorInsertToComplete	Sat Sep 28 10:56:06 EDT 2013	Sat Sep 28 11:06:17 EDT 2013	10.18	Syntax error, insert ";" to complete Statement
ParsingErrorInsertToComplete	Sat Sep 28 10:56:06 EDT 2013	Sat Sep 28 11:06:17 EDT 2013	10.18	Syntax error, insert "]" to complete MethodBody
ParsingErrorInvalidToken	Sat Sep 28 10:55:20 EDT 2013	Sat Sep 28 11:11:52 EDT 2013	16.53	Syntax error on token "==", invalid AssignmentOperator
UndefinedType	Sat Sep 28 11:00:58 EDT 2013	Sat Sep 28 11:11:52 EDT 2013	10.90	string cannot be resolved to a type
UnresolvedVariable	Sat Sep 28 11:00:58 EDT 2013	Sat Sep 28 11:11:52 EDT 2013	10.90	userIput cannot be resolved to a variable
UnterminatedString	Sat Sep 28 10:56:06 EDT 2013	Sat Sep 28 11:06:17 EDT 2013	10.18	String literal is not properly closed by a double-quote

Fig. 8.3 Errors types observed during coding

provided with a report (see Fig. 8.4) on errors made by a student when solving a specific problem and how long it took him/her to solve an error and which errors remained unsolved at the submission of his/her assignment.

Further, data on the number of compiles, the number of errors, the types of errors, the error correction patterns, documentation patterns, and test plan patterns can also be collected.

Figure 8.4 shows patterns of code submitted by students in VPL. We can go further to look at each student's submissions across multiple coding sessions. Within each session, we can investigate the code written by the student, the types of errors made by the student, the error correction behavior of the student (e.g., time to debug different types of errors), the pace of writing code, the types of competencies being gained (e.g., use of abstract data types), the ebb and flow of these competencies over time, and so on.

We classified the types of data being collected to course traces and student traces. Course traces offer analytics on how the course material and students are performing overall. This could include a semester-over-semester trend of some

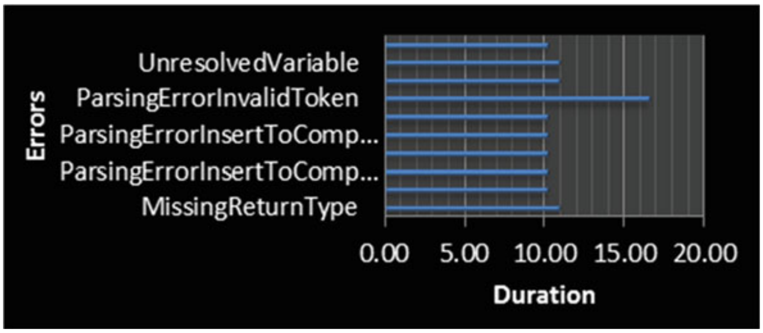


Fig. 8.4 Debug duration versus error types

measurements (e.g., competency growth), or as a yardstick against similar courses —an introductory C programming course against an introductory Java course. Some example course-wide traces include.

- Course interactions—how many actions (viewing pages, logging in/out, errors, adding forum postings, etc.) performed by each student ID on general course material
- Coding interactions—total actions related to coding in Moodle VPL (new code construct, compiling, deleting code, testing code, running code, etc.) performed by students in the setting of a course
- Course Assessment Performances Answered—successful completion of a programming assignments using VPL or EIDEE, performance in the quiz/ assignment, and classification of forum interactions.

Course traces also include clues about the attractiveness and effectiveness of course content including lessons, assessments, and tools.

Student-based traces look specifically at how the student is performing in the course. This could be compared against the same metric for other students, or to look at different actions by the same student (i.e., were there fewer coding errors when the student first looked at the learning material, or is there a relationship between how long he looks at the learning material, and the number of errors made). Student-based traces include:

- All Moodle Actions—superset of all actions performed by each student on the Moodle website, including code-related actions
- Successful Submissions—total of all successful assignment submissions for all students collectively or each student individually
- Successful Submissions by Programming Problem—number of successful programming assignment submissions per programming problem in VPL
- All Coding Actions—all of a student's code-related actions in VPL
- Coding Actions by Programming Problem—number of code-related actions per programming problem
- Raw submitted/attempted program files—actual source (program) files students worked on both for submitted and attempted programming assignments

Student-specific questions might zero in on specific behaviors (number of actions, number of coding actions, time on Moodle, etc.), or even subdivide the course by section, and look at each section as a component to determine how helpful it was to the students.

Analysis includes relationships discovery across datasets. For instance, is there a reduction in coding errors over time when students first study the entire lesson for a significant time before attempting to solve the program? What are the predominant types of errors and how well they are addressed in the content? Are there different learning styles emerging from the Moodle browsing patterns among students who perform (based on marks scored) at different levels? Is there a significant improvement in coding submissions over time, as the student becomes more proficient, when instructors are informed about potentially demotivated students? Is there a relation between student performances and student self-regulation behavior (browsing their own profiles)? Is the level of student documentation an indication of their coding competency? Are there clusters of students who require special attention from the instructor? Are there clusters of students who progress faster through the course related to other students? Does the learner competency growth align well with the curricular goals of the institution? Do group study students perform better self-regulation than individual study students? Can performances of students be predicted based on initial patterns of study and progress? Is there a significant match between student competency levels and industry expectations?

The types of analyses, information, and data one could extract from learning related datasets is quite large and varied. The dashboards need to be customized to extract only the types that are of use and preferred by individuals. The progress reports of students generated by the dashboards need to not only show the current progress but also the means to reach a better competency growth. Shown below is one of results of our first controlled study in learning analytics outlined in Sect. 2.1.

A total of 75 first year students (batch 3) taking the C programming course were given access to learning analytics tools from within Moodle, in addition to traditional resources, at the end of Assessment 1. The rest, a total of 692 students, taking the same C programming course studied with access only to traditional resources such as lecture notes, text book, lab support, and assessment feedback. Table 8.1 shows the performance of batch 3, the experimental students and Table 8.2 shows the performance of the rest, the control students.

Comparison of Tables 8.1 and 8.2 indicate that, overall, the learning analytics tools, had a significant impact in improving grades of students in assignments 2 and 3. Further, learning analytics tools have also been instrumental in significantly reducing the percentage of failed students in assignments 2 and 3.

The theory and the practical exams target the full scope of the course and the impact of the learning analytics tools on these two assessments is inconclusive. Another study to investigate impact of individual analytics tools on student performances is currently underway.

Table 8.1 Performance of control students

	Assess 1	Assess 2	Assess 3	Theory	Practical
Average marks	64.44	60.96	59.55	6.41	7.44
No. of fails	146	221	151	99	24
% of fails	21.10	31.94	21.82	14.31	3.47

Table 8.2 Performance of experimental students

	Assess 1	Assess 2	Assess 3	Theory	Practical
Average marks	62.51	73.51	73.91	7.09	7.67
No. of fails	22	2	6	7	3
% of fails	29.73	2.70	8.10	9.46	4.05

8.3 Lessons Learned and Future Potential

One of the key challenges in big data learning analytics is data integrity that ensures the authenticity of the arriving data and the ethical nature of the data.

Data can be processed by any technique, whether it is a data mining technique or a machine learning technique, or a big data clustering technique, or a sentiment analysis technique. The point is, students and instructors should be in control of the choice of analyses and the corresponding techniques.

In addition to the students initiating interactions with the analytics tools, the tools themselves could also be initiating interactions with students.

Performance indicators, by themselves, offer reflection opportunities. Big data analytics enables students and instructors to proceed further towards regulation opportunities.

Analytics being offered are available without "any" intrusion on the student's course interactions. There is no controlled experiment as such. Instead, data are being collected in the background with students' explicit permission.

Students are welcome to relate different datasets to arrive at insightful analytics. The datasets and the associated analytics techniques are open for use by students. Students can request additional datasets or techniques for other types of meaningful analytics. Students and instructors can share their individual analytics dashboards with others.

Competency growth in programming can be observed within a specific language (say, Java) across multiple courses. Equally, competency growth in programming can be observed across multiple languages (say Java and C++). Competency growth in programming can be observed at the individual student level, or the course level, or the departmental level, or even at the institutional level.

Key inferences about intervention opportunities could be made available to instructors. These opportunities arise out of analytics with substantiating evidence. Intervention opportunities could be tuned towards learning efficiency of individual students and groups of students.

One of the substantiating evidences is the use of causal modeling in arriving at analytics. With sufficient data, coding activities, study activities, performances, and reflection/regulation activities can be causally connected. Such causal connections can be open to students and instructors to help determine intervention opportunities.

In addition to assessment outcomes, instructional effectiveness can be continually measured based on successful interventions and growth patterns of student performances.

The ontology-based analytics framework lends itself to develop analytics measures for learning efficiency and instructional effectiveness. The underlying mechanism that yields these measures (e.g., data mining, machine learning, or big data analytics) can be open for investigation but can be comfortably ignored by students and instructors.

The development of a comprehensive coding traces platform reporting on the efficiency of coding and effectiveness of coding instruction is a key contribution of this research. It reports not only on learner habits but also relates instructor habits.

The traces platform is quite conducive to the development of software agent-oriented guidance that can monitor the data sources on a student-by-student basis, to identify successful patterns of learning, based on a student's interaction with the digital learning environment, and suggest corrective actions, materials and behaviors that might improve performance, based on the past results of similar student behaviors.

The underlying datasets approach the quantities expected in the context of big data analytics. The platform that we have developed supports the inclusion of Hadoop[12]-based analytics that can handle large volumes of data. The platform also supports the use of causal modeling technologies (e.g., Tetrad IV) that can generate large volumes of potential competency models.

Coding efficiency encompasses aspects of coding style, metacognitive scaffolds, peer interactions, and interactions with instructors, among others. Instruction on effective coding encompasses aspects of intervention, guidance, content material, sensitivity to learner capacity, and timeliness of feedback, among others. Datasets corresponding to each of these aspects can be observed and relations among these datasets can be established using the platform.

With big data as the premise, we developed a sensor-based big data software platform that is open, inclusive, adaptable, and precise. It shows tremendous potential towards accountability of students' study tasks. Equally, the accountability and performance of instructors are also continually analyzed. Matching of these two accountabilities would yield optimal learning opportunities.

Acknowledgments This research was funded in part by NSERC, iCORE, Xerox, research related gift funding by Mr. A. Markin, NSERC Discovery Grants, SSHRC INE (LearningKit), Shastri Indo-Canadian Institute (SICI), and Athabasca University, Canada.

References

Abdous, M., He, W., & Yen, C. J. (2013). Using datamining for predicting relationships between online question theme and final grade. *Educational Technology & Society, 15*(3), 77–88.

Bienkowski, M., Brecht, J., & Klo, J. (2012). The learning registry: Building a foundation for learning resource analytics In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge* (pp. 208–211).

Essa, A., & Ayad, H. (2012). Improving student success using predictive models and data visualisations. In *Proceedings of the ALT-C 2012 Conference* (pp. 58–70).

Ferguson, R. (2012). Learning analytics: Drivers, developments and challenges. *International Journal of Technology Enhanced Learning (IJTEL), 4*(5/6), 304–317.

[12] http://hadoop.apache.org.

Hung, J. L., Hsu, Y. C., & Rice, K. (2012). Integrating datamining in program evaluation of K-12 online education. *Educational Technology & Society, 15*(3), 27–41.

Kim, M., & Lee, E. (2013). A multidimensional analysis tool for visualizing online interactions. *Educational Technology & Society, 15*(3), 89–102.

Kumar, V., Chu, S., & Groeneboer, R.C. (2007). Personalizing instructional design processes to empower learners, case study, In D. Stockley & J. Mighty(Eds.), *Teaching More Students* (pp. 74–78). Canada: Queen's University Press.

Mazza, R., & Dimitrova, V. (2003). CourseVis: Externalising student information to facilitate instructors in distance learning. In *Proceedings of the International Conference in Artificial Intelligence in Education (AIED 2003)* (pp. 279–286).

Medeiros, F., Gomes, A.S., Amorim, R., & Medeiros, G. (2013) Architecture for social interactions monitoring in collaborative learning environments as a support for the teacher's awareness. In *Proceedings of International Conference on Advanced Learning Technologies (ICALT)* (pp. 123–127).

Romero, C., Ventura, S., & Garcia, E. (2008). Datamining in course management systems: Moodle cases study and tutorial. *Computers & Education, 51*(1), 368–384.

Shum, S. B., & Ferguson, R. (2012). Social learning analytics. *Educational Technology & Society, 15*(3), 3–26.

Watson, C., Li, F.W.B., & Godwin, J.L. (2013). Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *Proceedings of International Conference on Advanced Learning Technologies (ICALT)* (pp. 319–323).

Wood, E., Zivcakova, L., Gentile, P., Archer, K., Pasquale, D. D., & Nosko, A. (2012). Examining the impact of off-task multi-tasking with technology on real-time classroom learning. *Computers & Education, 58*(2012), 365–374.

Authors Biography

Vivekanandan Suresh Kumar is Associate Professor in the School of Computing and Information Systems. His research interests are in applications of causal modelling in big data learning analytics, artificial intelligence in education, collaborative learning, self-regulated learning, co-regulated learning and mixed-initiative interactions using anthropomorphic pedagogical agents.

Kinshuk holds the NSERC/iCORE/Xerox/Markin Research Chair for Adaptivity and Personalization in Informatics. He is also Full Professor in the School of Computing and Information Systems and Associate Dean of Faculty of Science and Technology, at Athabasca University, Canada. Areas of his research interests include learning analytics; learning technologies; mobile, ubiquitous and location aware learning systems; cognitive profiling; and, interactive technologies.

Thamarai Selvi Somasundaram is Full Professor and Dean of the MIT Campus at Anna University, India. Areas of her research interests include high performance computing, educational technology, cognitive modelling, and instructional design.

David Boulanger is a research assistant and undergraduate student at Athabasca University. His areas of research interests include bigdata collection, ontologies, analytics, and visualization of learning-oriented data.

Jeremie Seanosky is an undergraduate student and research assistant at Athabasca University. His areas of research interests include bigdata analytics, learning analytics sensors, failsafe data collection, retrieval, analysis, visualization, and automated assessment.

Marcello F. Vilela is an engineering student at the Universidade Federal Fluminense in Brazil. He was a Globalink Mitacs scholar at Athabasca University. During his stay in Edmonton, Canada, he explored bigdata framework as a common platform for traffic analytics and learning analytics.