Integration and Automation of Data Preparation and Data Mining

Shrikanth Narayanan
Spatial Sciences Institute
Department of Computer Science
University of Southern California
Email: nara471@usc.edu

Ayush Jaiswal
Department of Computer Science
University of Southern California
Email: ajaiswal@usc.edu

Yao-Yi Chiang
Spatial Sciences Institute
University of Southern California
Email: yaoyic@usc.edu

Yanhui Geng
Huawei Technologies
Email: geng.yanhui@huawei.com

Craig A. Knoblock
Information Sciences Institute
Department of Computer Science
University of Southern California
Email: knoblock@isi.edu

Pedro Szekely
Information Sciences Institute
Department of Computer Science
University of Southern California
Email: pszekely@isi.edu

Abstract—**Data mining tasks typically require significant effort in data preparation to find, transform, integrate and prepare the data for the relevant data mining tools. In addition, the work performed in data preparation is often not recorded and is difficult to reproduce from the raw data. In this paper we present an integrated approach to data preparation and data mining that combines the two steps into a single integrated process and maintains detailed metadata about the data sources, the steps in the process, and the resulting learned classifier produced from data mining algorithms. We present results on an example scenario, which shows that our approach provides significant reduction in the time in takes to perform a data mining task.**

I. Introduction

While the quality and ease of use of data mining libraries such as in R [1] and Weka [2] is excellent, users must spend significant effort to prepare raw data for use in these libraries. In practice, the data preparation task is separate from the data mining process and very often requires data source dependent and repetitive manual work, such as applying Structured Query Language (SQL) statements to extract and aggregate database records, using Microsoft Excel to clean and normalize datasets of structured text files, and writing scripts to apply complex transformations (e.g., in Python). This practice does not scale when the data size becomes larger and the number of data sources increases.

In a previous work, Knoblock et al. [3] developed an interactive approach to extracting, modeling, and publishing data in a system called Karma.[1] Karma allows an end-user to solve their own integration tasks without having to program those tasks. Karma can semi-automatically build a semantic description (a model) of a data source. This semantic description makes it possible to rapidly convert a set of sources (represented in Extensible Markup Language (XML), Javascript Object Notation (JSON), structured text files, or databases)

into a shared domain model, which supports the integrated reasoning and processing of data across many sources. Once a user models the data sources, Karma can automatically converts the raw data into any of a variety of formats.

In this paper we present an integrated approach built on Karma for information integration and data mining. Our approach combines the steps in data preparation and data mining into a single integrated process and maintains detailed metadata about the data sources. We show that a user can use our approach to rapidly clean, normalize, restructure and integrate data collected from multiple mobile sensors, and then apply machine learning algorithms from a library of algorithms to perform a data mining task. We demonstrate these capabilities using a data mining task of predicting the mode of transport of a user given the sensor data collected via his/her cellphone. The challenge here is how to perform a data mining task, such that there is an sizable reduction in overall user time and effort in preparing the data set from its raw form and invoking the prediction service. We compare our approach with a baseline approach that uses Microsoft Excel for most of the data preparation tasks. We show how Karma supports the required steps for data preparation and data mining, and yields a significant reduction in time and effort.

The remainder of this paper is organized as follows. Section 2 describes a motivating data mining problem on prediction of mode of transport from mobile sensor data. Section 3 presents our integrated approach to data preparation and data mining. Section 4 describes the steps of using Excel to prepare raw data for data mining. Section 5 reports on our experimental results. Section 6 discusses the related work, and Section 7 presents discussion and future work.

II. Motivating Problem

As an example of a data mining task, we consider the problem of predicting the mode of transport of a mobile user. Mode of transport prediction is an interesting problem in data mining. It helps in providing contextual information about

---

IEEE computer society

Fig. 1: System block diagram

Trees algorithm with features such as the Discrete Fourier Transform (DFT) energy coefficients at 1Hz, 2Hz, and 3Hz of acceleration magnitude, the accelerometer variance, and the speed recorded by the GPS sensor. The features are derived from data collected from the accelerometer and the GPS sensors on a mobile device. Our method of predicting the mode of transport is similar to theirs as we use the Support Vector Machine (SVM) algorithm with similar features. We use GPS accuracy as a feature instead of accelerometer variance, keeping the rest of the features the same.

To acquire training and testing data, we use an application that collects accelerometer and GPS readings on mobile phones. We asked users to record their mode of transportation while using the application in order to acquire the data to build the prediction model. The data mining system derives the aforementioned features from the sensor data and merges the mode of transport labels noted by the user with the features for mining.

The collected data is hardly ever ready to use directly in real life scenarios. We have to prepare it for mining through multiple transformation steps, such as dealing with missing values, discretization, and normalization. Moreover, even if the data is in tabular form, in many cases, we cannot use the columns as features for data mining libraries directly. We have to compute useful features by using values in multiple columns. In the example case of mode of transport prediction, we calculate acceleration magnitude from the acceleration recorded by the accelerometer along the X, Y, and Z dimensions. We then calculate DFT energy coefficients at 1Hz, 2Hz, and 3Hz of the acceleration magnitude for every 1 second time window. We then discretize the data, which contains multiple entries every second, to the nearest second.

Many data mining tasks use data from multiple sources that have to be merged in order to derive features from them, such as those performed by large organizations on data collected from different branches and departments within them. The data used by the algorithm that we employ in our example case is derived from three sources, i.e., the accelerometer sensor, the GPS sensor, and the users mode of transportation notes. We merge the data by making use of timestamps contained in all three sources.

In general, after the data is prepared, machine learning algorithms are applied to it to train the prediction model and to test it, or to perform cross-validation. In our case, we train a prediction model using the SVM algorithm and test it. The final set of features consists of DFT energy coefficients at 1Hz, 2Hz, and 3Hz, speed recorded by the GPS sensor, and GPS accuracy.

These parts of the entire data mining system motivate the development of a complete end-to-end system that offers complex data transformations (as in-built features, extensions, or services), data merging capabilities, invocation of various machine learning algorithms for application and comparison, and display of results in a meaningful manner.

III. APPROACH

In general, a data mining task will have the following steps - data collection, data integration and cleaning, transformation
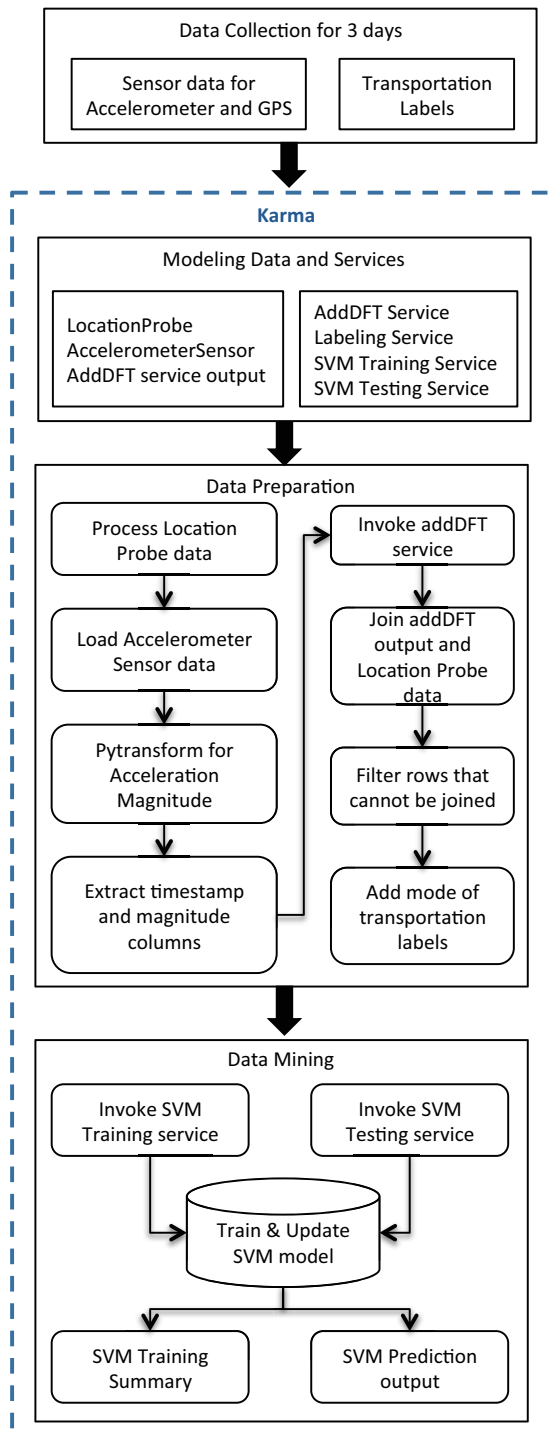
the user that can be used in building intelligent smartphone applications.

Reddy et al. [4] describe such a method for predicting the mode of transport of a mobile user by applying the Decision

and selection, data mining and evaluation [5]. It has been estimated that data preparation - integration, cleaning, selection and transformation, accounts for a significant portion of the time spent on a data mining project. In our approach to demonstrate the end-to-end process of data preparation and data mining, we select the task of predicting the mode of transport for a user, given the sensor data collected by the mobile phones. Our approach enables users to integrate data from a variety of sources without exposing them to the underlying complexity of data integration and exchange. Figure 1 shows the different blocks of system.

A. Data Collection

We collect the mobile sensor data using a custom Android application built using the funf[2] sensing framework. The application records the readings from the Accelerometer and the Global Positioning System (GPS) sensors, archives and uploads the collected data to a Dropbox[3] account every 24 hours. Our users collected data for three days, yielding datasets consisting of Comma Separated Value (CSV) files. Each dataset is comprised of 3 files:

1) *LocationProbe.csv*: Contains positioning data from the GPS sensors and has 47 columns describing the user's location, speed, bearing, etc.
2) *AccelerometerSensor.csv*: Contains coordinate values from the accelerometers
3) *TransportationLabels.csv*: The user collecting the data has to label each mode of transport that they use and the specific intervals of the day when they use it
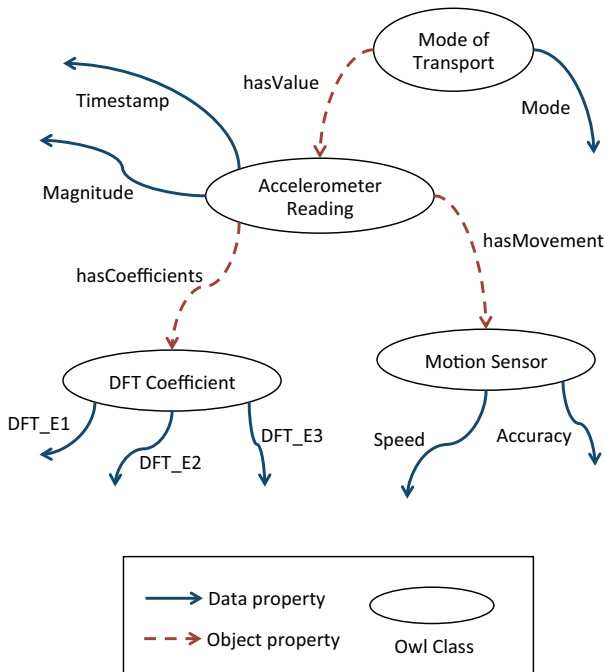

Fig. 2: Ontology used in for the mode of transportation data

B. Processing with Karma

The steps using Karma are divided into two parts:

1) **Karma setup**: These are tasks that are performed only the first time. They include modeling the three web services, addDFT, getLabel, and svmTraining, which are explained later, as well as modeling the two raw datasets AccelerometerSensor and LocationProbe. All transformations and processing done here is recorded by Karma and can be played automatically for the other datasets.
2) **Karma execution**: The Karma execution tasks are ones that are repeated for each datasets. They mainly include tasks such as service invocation, joining datasets, and publishing data in the Resource Description Framework (RDF).

1) Karma Setup: The work on Karma [3] shows how to map structured data to RDF using an ontology. Karma uses a two step process to map data to ontology. The mapping of columns to semantic types and specifying relationships between the semantic types is demonstrated in the work on the Smithsonian dataset [6]. When we map our services to our ontology, we attach additional properties to the model along with the semantic type mappings that enables Karma to mark them as a service model. A service model is used to invoke a web service using the columns mapped as inputs to the service. We create an ontology for the mode of transportation data and use it to model our services and data sources. We do not add the ontology creation time in the setup time because such ontologies address a larger domain model and evolve over a period of time. Hence we assume that the ontology was created beforehand. We discuss our ontology and modeling of the services and data sources in the remaining part of the section.

Figure 2 shows the mode of transportation ontology. It contains 4 classes:

1) *ModeOfTransport*: Contains a data property for the mode of transportation label
2) *AccelerometerReading*: Contains data properties for timestamp and acceleration magnitude. The object property — 'hasMovement' connects it to MotionSensor. Similarity, 'hasCoefficient' object property connects the AccelerometerReading class to DFT_Coefficient
3) *MotionSensor*: Contains data properties for speed and accuracy components of the location probe data
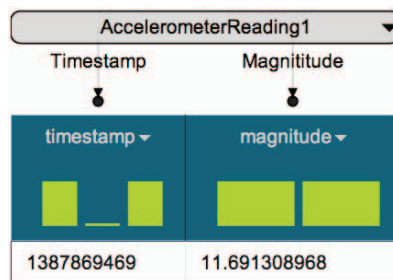4) *DFT_Coefficient*: Contains data properties for the DFT energy coefficients at 1Hz, 2Hz and 3Hz

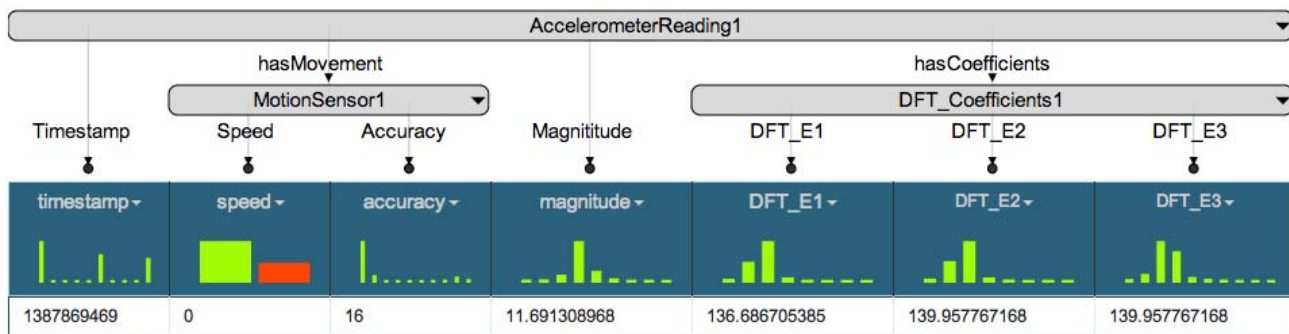Fig. 3: Semantic model for addDFT service

1078

Fig. 4: Semantic model for getLabel service

We will model three services that we require in our prediction task. Figures 3, 4 and 5 show the models for these services.

The 'addDFT' service is a Hypertext Transfer Protocol (HTTP) POST service that calculates the DFT energy coefficients for acceleration magnitude at 1Hz, 2Hz and 3Hz. The input to this service is a CSV file that contains two columns — timestamp and acceleration magnitude. The DFT values are calculated over a time window of one second. The output generated is a CSV file having the columns — timestamp, magnitude and the DFT energy coefficients. To model this service, we map the timestamp and magnitude column of our sample CSV file to the appropriate classes. As shown in Figure 3, we set the semantic type of the timestamp column to 'timestamp', which is a property of the 'Accelerometer-Reading' class. We then map the magnitude column using the 'Magnitude' property of the 'AccelerometerReading' class. We set the service URL for the addDFT service and publish the model.

The 'getLabel' service is a HTTP POST service that adds the mode of transportation label provided by the user for each row in the input file, using the timestamp values. The output produced is a CSV file containing all the columns of the input with an additional column for labels. To model this service, we map all the columns of our sample CSV file to the appropriate classes excluding the mode column. Figure 4 shows the column mappings and relationships between the classes, displayed on the Karma interface. We map the speed and accuracy columns to the MotionSensor class using the

respective data properties. The timestamp and magnitude are mapped to the AccelerometerReading class. After setting all the semantic types, we set the service URL and other options that are required while invoking the service and publish the model.

The SVM service has two parts — training and testing. Both services take the same set of inputs and have identical semantic types for their corresponding columns. Figure 5 shows the semantic model for the SVM training and testing services. The model is very similar to the getLabel service shown in Figure 4. For the SVM service, we map the 'mode' column that contains the mode of transportation labels to the 'ModeOfTransport' class. The rest of the mappings are discussed in the previous paragraph. We use different service URLs when we publish the SVM training and testing service models. These services also differ in the output that they produce. The training service returns a summary of the SVM model that was trained. In order to distinguish our prediction models, we specify a unique tag in the URL when we train the model. This tag serves as an identifier when we test the model. The testing service produces the prediction output along with the confusion matrix. The output for both the training and testing service is in JSON format. They both consume CSV data in the POST payload.

After modeling the services, we model our data sources because the raw data needs to be cleaned and transformed before it can be fed to our services. Karma records the transformations that we perform while modeling the data and replays it when we apply the model on a new file from our
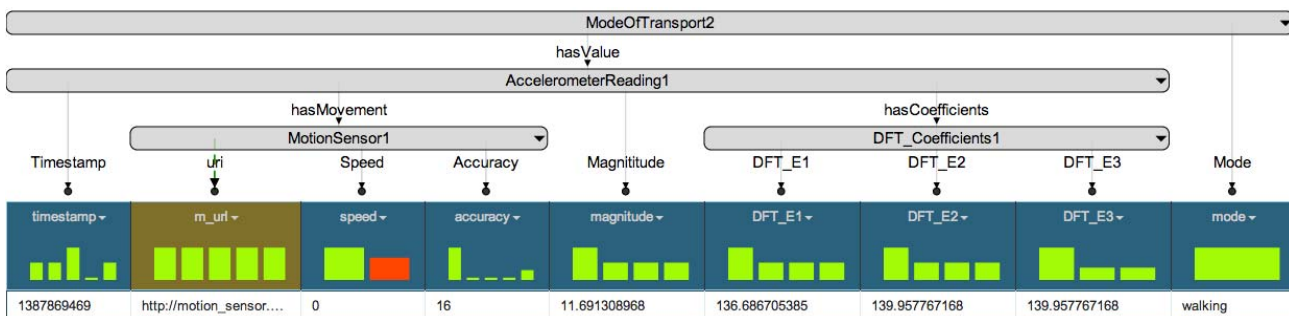

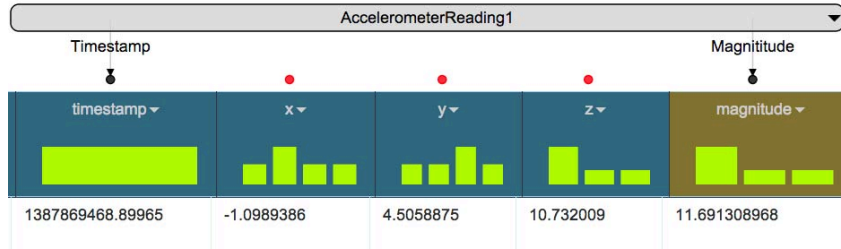Fig. 5: Semantic model of SVM — training and testing services

1079

Fig. 6: Semantic model for the AccelerometerSensor dataset.

data set. After modeling the data, we publish the resulting RDF. Figures 6, 7, and 8 show the models for our data sources.

Starting with the AccelerometerSensor file, we add the magnitude column using a Python transformation, which is available as a feature in Karma. In this transformation, we calculate the magnitude values using acceleration along x, y, and z coordinates available in the raw data, and by using standard library functions in Python. Figure 6 shows the resulting model in Karma. The magnitude column appears in yellow color because it is added using a Python transformation and indicates that it is not a part of the original file that is uploaded in Karma. After mapping the semantic types and generating URLs for the classes, we publish the model.

The next data source we model is the output of the addDFT service which is shown in Figure 7. The output is a CSV file with the columns — timestamp, magnitude and DFT energy coefficient values at 1Hz, 2Hz and 3Hz. The addDFT service appends three columns to the input file having headers — 'DFT_E1', 'DFT_E2' and 'DFT_E3', as shown in Figure 7. We map the three columns containing the DFT energy coefficients to the 'DFT_Coefficient' class. The timestamp and magnitude columns are mapped to the 'AccelerometerReading' class. We add an additional column, 'acce_url', using a Python transformation and populate it by appending the timestamp values to a base Uniform Resource Identifier (URI). The 'acce_url' column is mapped as the URI for the 'AccelerometerReading' class. It is not necessary to have URIs for every class in the model. We add the URI for 'AccelerometerReading' class because it is required by Karma to perform join operations.

The LocationProbe file contains 47 columns, some of them are — *id, device, timestamp, mAccuracy, mAltitude, mBearing, mLatitude, mLongitude, mProvider, mSpeed, mTime, timestamp, etc*. Out of these we only use 'timestamp', 'mSpeed' and 'mAccuracy'. As shown in Figure 8, we map the accuracy and speed columns to the 'MotionSensor' class and the timestamp to 'AccelerometerReading' class. We generate two additional columns — 'acce_url' and 'm_url' to add URIs for the 'AccelerometerReading' and 'MotionSensor' classes. The URIs are generated using a Python transformation, by appending the timestamp value to a base URI. This completes the Karma setup process.

2) Karma Execution: Once we have modeled all our data sources and services, we start with the Karma execution steps to process the mode of transportation data. Our goal is to integrate all the datasets to produce a CSV file that can be fed to the SVM algorithm. In Karma, we do this by first modeling each dataset according to the ontology, then publishing the data as RDF, and finally using join operations to merge the data.

We load the LocationProbe file, apply the 'LocationProbe' model, and publish the resulting RDF. For the accelerometer sensor data, we start by loading the data collected for the first day. We apply the 'AccelerometerSensorModel' and publish the RDF. To invoke the addDFT service, we select the 'Invoke Service' option from the drop down menu that appears after clicking the AccelerometerReading class bubble. Karma distinguishes between service models and data models and shows the list of services that could be invoked. Karma determines that a service can be invoked if the semantic type mappings for all the input columns along with the relationships between the classes
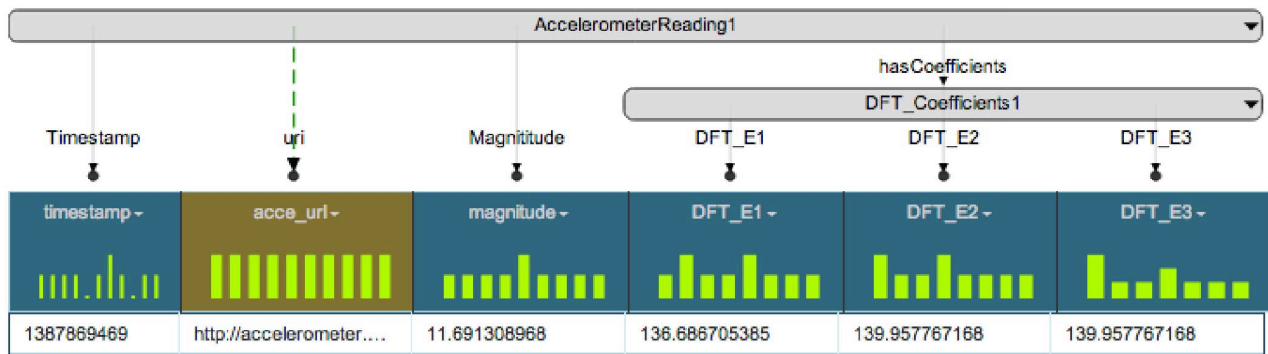

Fig. 7: Semantic model of the output generated by addDFT service.
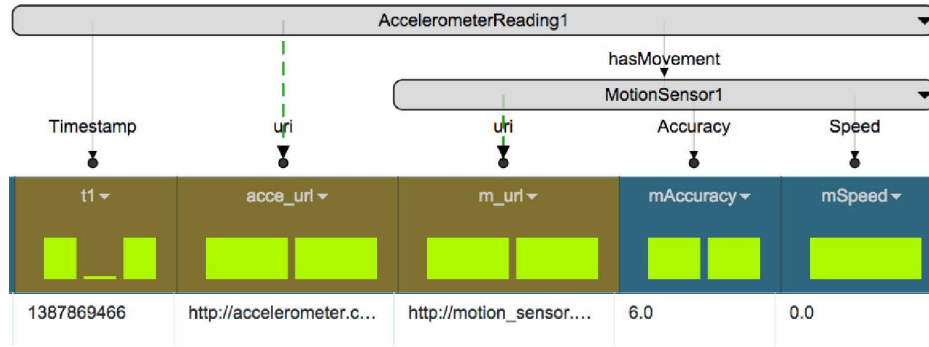
1080

Fig. 8: Semantic model for the Location probe dataset

match with the model on the current worksheet. From the list of services shown, we select the addDFT service and then select the corresponding RDF graph where the accelerometer data was published. The results of the addDFT service are loaded in a new worksheet. We apply the model that was created for the addDFT output to the new worksheet and publish RDF. After we have published the data from the addDFT service, we perform a join operation with the LocationProbe data that was published previously. Starting from the topmost class in the model, i.e., AccelerometerReading, we use the 'Augment Data' option to perform the join.

Karma explores the properties that could be added to the selected class, in this case the AccelerometerReading class, which are not currently present in the worksheet. Karma fetches the additional properties from the available RDF data in the triple store. In our ontology, the AccelerometerReading class has an object property 'hasMovement' that connects it to the MotionSensor class. We select this property to be added in the worksheet. Karma adds a URL column for the MotionSensor class and populates it with the joined values. We repeat the join process for the MotionSensor class and add the speed and accuracy properties. The final augmented worksheet is shown in Figure 9. The yellow colored columns are added through transformations and the red columns are the ones that get attached by joining the location probe data. The columns in blue are part of the original model. We then publish the augmented model and the RDF in a new graph. We need to publish this model because the model shown in Figure 9 is

not the original model and contains additional properties that were added after the join operation.

We now invoke the getLabel service to add the mode of transport column. The service creates a new worksheet to which we apply the SVM model as shown in Figure 5, and publish the RDF. Since this is the first file that we process, we will invoke the SVM training service and use the processed data as training set. For the succeeding files, we first invoke SVM testing service and evaluate our model for the accuracy. Then we publish RDF for the test dataset to append it to the same graph that we used in the previous iteration. We use the merged data as a training set and again train our SVM model by invoking the SVM training service. We do this iteratively for the data for all the days that follow. Thus, as we process more data files, and add to the training set, our model improves in the prediction accuracy. The JSON results of the service invocation are parsed and loaded in a new worksheet.

IV. ALTERNATE APPROACH: PROCESSING WITH EXCEL

Microsoft Excel is one of the most widely used applications for data cleaning and preparation. It is not only easy to learn but also powerful to perform numerous computations across multiple rows and columns. Other proprietary packages that are available for data preparation are much more complex to use and usually require user training. Despite Excel being very popular amongst users who are not tech savvy and do not possess the skills to write scripts for data preparation,
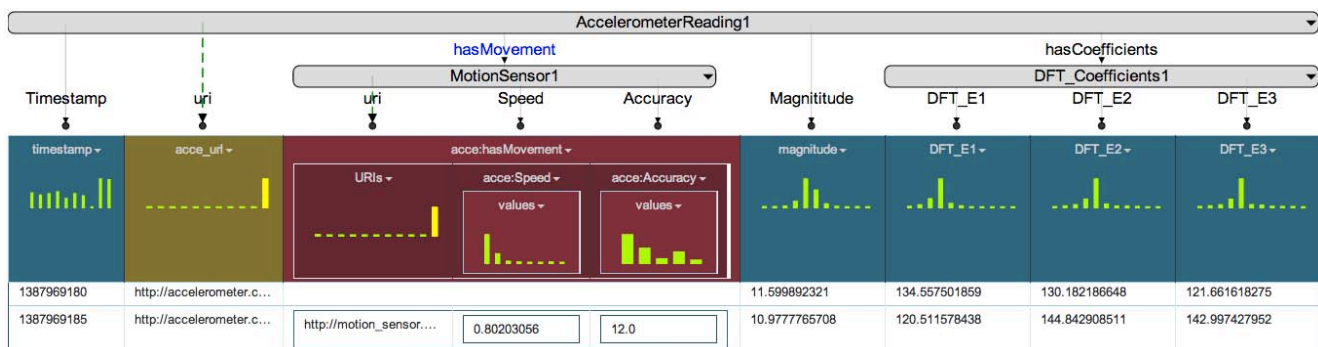

Fig. 9: Karma worksheet after joining the accelerometer and location datasets

1081

we illustrate several shortcomings of using Excel for data preparation. Using Karma the user can publish models as linked data which are accessible globally for others to reuse, which is not possible using Excel files.

TABLE I: Sample accelerometer data after adding magnitude

timestamp	x	y	z	magnitude
1387969178.5018	0.7374141	4.096479	10.688913	11.47073585
1387969180.1531	-0.6596026	4.035427	10.85531	11.59989232
1387969180.1649	-0.533907	4.3382936	10.539276	11.40974087
1387969180.2361	-0.8535329	4.820725	8.850166	10.11401731

One of the most important and rather annoying aspects of data preparation is its iterative nature. Users repeat the same steps to process each file in the data set. Even if the user is able to automate some parts of the data preparation using Excel, they still need to invoke the data mining services and feed in the processed files for training and testing. Moreover, using Excel does not let users share the transformations in a way that allows other users to reuse the steps recorded previously by someone.

TABLE II: Sample output after running the addDFT script

timestamp	magnitude	DFT_E1	DFT_E2	DFT_E3
1387969180	11.59989232	134.5575019	130.1821866	121.6616183
1387969185	10.97777657	120.5115784	144.8429085	142.997428
1387969214	11.56426981	133.7323362	140.4465978	150.3014305
1387969282	4.014597704	16.11699473	16.11699473	22.75025583

We start by processing the AccelerometerSensor file for the first day. The timestamp column in the file contains decimal values; hence we change the default column format in excel, to contain decimal point upto 4 places. We then add a new column for the acceleration magnitude, that is calculated as the sum of squared values of the x,y and z coordinates. The coordinate values are fetched from their respective columns in the same file as shown in TableI. We apply a formula to get the magnitude value for the first row, and repeat it across the entire worksheet to calculate acceleration magnitude for the remaining rows. Once we have the magnitude, we prepare the file for DFT calculation. We select two columns — timestamp and magnitude, and create a new file having only these columns. The addDFT Python script is executed from the command line with the newly created file as input. The script generates an output file having the DFT energy coefficients at 1Hz, 2Hz, and 3Hz of the acceleration magnitude for every 1 second time window.

The Accelerometer sensor produces a large number of readings when compared to the GPS sensor. Before using the LocationProbe file, we merge all of the files from three days into one because we only perform join operation using the LocationProbe data with the addDFT service output, and no other transformations. The accelerometer data cannot be merged because it contains continuous time-series data and the DFT calculations are performed over a time window. Merging the accelerometer data files will create gaps in the time window leading to incorrect DFT calculations. The merging of LocationProbe file is done in both our approaches.

In the LocationProbe file, we select and copy 3 columns 'timestamp', 'mSpeed', 'mAccuracy' into a new worksheet.

We format the timestamp column to have decimal values and add a new column, 'absTimestamp', that will contain the absolute values of timestamp. We apply the 'Round' formula to populate the 'absTimestamp' column. In the next step, we load the output file generated from the addDFT script into a new worksheet in the current workbook. We join the two worksheets to combine the speed and accuracy columns with the DFT energy coefficients. We then add two new columns for speed and accuracy and apply the VLookUp formula to join the LocationProbe data by using the 'absTimestamp' column as the key.

TABLE III: Processed LocationProbe data

timestamp	absTimestamp	mSpeed	mAccuracy
1387969150.56400	1387969151.00000	0	32
1387969173.56800	1387969174.00000	1.3263288	12
1387969185.71600	1387969186.00000	0.80203056	12
1387969282.09300	1387969282.00000	1.0898862	24

We add separate VLookUp formulas for each of the speed and accuracy columns to populate the respective values. For the rows that could not yield any value after the join, either for speed or for the accuracy column, we delete them and save the file as a CSV file. The missing values in the data set arise due to failure of probes — accelerometer or GPS, while collecting data for a particular timestamp. We now execute the addLabel Python script using the previously joined file as input and generate our final processed file with the mode of transportation labels. The addLabel script attaches the mode of transportation label to each row based on the time intervals specified by the user. Table IV shows the structure of the processed CSV file that is used to train and test the SVM model.

Since this is the first file that we process, we will execute the svmTraining script and use the processed data as training set. For the files from the succeeding days, we first invoke SVM testing script and evaluate our model for the accuracy. We merge the test data set with the training data set and again train our SVM model. We do this iteratively for the data from all the days to follow. Thus, as we process more data files, and add to the training set, our model improves in the prediction accuracy.

V. EVALUATION

We evaluated our approach by measuring reduction in the time and effort required to perform data preparation and data mining for the mode of transport prediction task. We used a two parameter evaluation strategy (time and accuracy) to provide us with summary data to show the impact of our approach. In the following sections we discuss our experiment setup, the test user's skill level, and the evaluation criteria used to summarize the assessment chart.

A. Experiment Setup

In our experiment, we evaluated the steps using Microsoft Excel and Karma for processing the entire dataset. Three sets of the AccelerometerSensor and LocationProbe dataset were collected (one set for each day). After each step, the time required for the user to perform the task was noted along with any system processing time. For certain tasks

1082

TABLE IV: Structure of the processed file

Timestamp	Magnitude	DFT_E1	DFT_E2	DFT_E3	Speed	Accuracy	Mode
1387869469	0	16	11.691308968	136.686705385	139.957767168	139.957767168	walking
1387870904	0	16	12.1850014516	148.474260375	146.964104434	148.759649196	auto
1387872050	0	24	11.3212865392	128.171528903	136.300006172	133.300527309	bus
1387874848	0	858	12.080955151	145.94947736	145.866842087	143.819565165	stationary

that involved programmatic computation like calculating DFT energy coefficient values for the acceleration data, or labeling the mode of transport for individual time ranges, we used Python scripts in both the approaches. Therefore, the time taken for writing these scripts and executing them was the same in both the approaches. Hence we excluded those timings in the evaluations. We had one user to perform all the tasks in the experiment. The proficiency level of the test user was advanced, i.e., the user was an expert in using Karma and Excel.

In order to demonstrate our objective, we asked the user to perform two trials for processing the collected data set and noted their timings. The user first prepared the collected data from day one and generated an SVM model. Then he prepared the collected data from day two and tested the day-two data with the day-one model. Finally, the user prepared the collected data from day three and tested it with the SVM model built using the data from day one and day two. When using Karma, there were one-time setup tasks in which the user first modeled all of our services by adding semantic types to the service inputs. The user modeled the following data sources and services:

1) *Accelerometer sensor data*: To calculate the acceleration magnitude and extract the timestamp.
2) *Accelerometer data with DFT energy coefficients* The result of the addDFT service
3) *LocationSensor data*: To model the speed, accuracy and timestamp columns
4) *AddDFT service*: Here we model the inputs for the addDFT service along with other meta data like service url, invocation method (GET or POST) etc.
5) *GetLabel service*: This service gets us the mode of transport for the given timestamp
6) *SVM Training*: This service model defines the SVM training and its inputs
7) *SVM Testing*: This service model defines the SVM testing and its inputs

Once the user modeled all the services, he uploaded each file of the collected data and applied the semantic models. Karma automatically applied all the transformations defined in the models, so no manual effort was required. At the end of processing each file, the user invoked the SVM training and testing services. When Excel was used, the user did not need to setup the application. Table VI lists the user actions and time required to perform the task in Excel and Table V shows the timings when using Karma.

B. Evaluation Criteria

Our evaluation metric had two components – efficiency and accuracy. The efficiency component included user interaction time and system processing time. The user interaction time

was the time taken by the user to complete a task using the interface of the tool, for example publishing RDF in Karma or formatting a column in Excel. The system processing time was the time taken by the tool to complete the computation and render the results after the user has issued the command. In terms of accuracy, we marked a data preparation task as correct if the user was able to process the data correctly and the resultant file could be used to invoke the SVM service.

TABLE V: Stepwise time chart for processing one file using Karma

Step	Task	User Time (sec)	System Process-ing Time (sec)	Total Elapsed Time
1	Modeling LocationProbe data	34	18	0:52
2	Publish RDF for LocationProbe	12	6	1:10
3	Modeling AccelerometerSensor data	18	5	1:34
4	Publish RDF for AccelerometerSensor	11	9	1:54
5	Invoke addDFT service	8	2	2:04
6	Modeling DFT service output	10	2	2:16
7	Publish RDF for DFT output	11	6	2:33
8	Join with LocationProbe RDF	12	5	2:50
9	Publish the augmented model	15	3	3:08
10	Publish RDF for joined data	10	6	3:24
11	Invoke getLabel service	8	2	3:34
12	Filter our 'NA' mode of transport	31	3	4:08
12	Model mode_of_transport data - the result of add label service	6	3	4:17
13	Publish RDF for Model of transport data	20	4	4:41

C. Experiment Results

Table V gives us the time taken for each of the data preparation tasks executed using Karma in one of the trials. The first step took the highest processing time because the combined location probe data file that was modeled, contained data from all of the three days. The transformations were executed for each step when the model was applied to this data set and since the number of rows was higher, it took more time to process. We observed that when a new worksheet is created in Karma, either by uploading a file or generated as a result of invoking a service, two steps were performed, namely applying a model and publishing the RDF. Together, these steps took roughly 20 to 25 seconds, depending upon the size of the dataset. After performing all the data transformations, RDF was published for the processed data using a new graph so that each data set can be individually used to invoke the SVM training and testing services.

Table VI displays the time taken for the data preparation tasks executed using Excel. We observe that the user spends significant time switching between different application windows as well as the switching between the use of a mouse and keyboard. These switches were likely to introduce the most number of errors, which adds to the overall processing time.

After processing the data for one day, it was stored in a new file to be used as training and testing sets for the SVM prediction task.

Table VII summarizes the timings for all of the trials. There are a number of observations that can be drawn from the table. When Karma was used, the user took the longest time to process the data set for the first day. As previously mentioned, the first iteration took more time to finish compared to the other trials since the LocationProbe data set was processed only once. However, when Excel was used, the data processing time for the first file was very close to the avarage time taken by Excel. Since most of the processing was automated in Karma, the user was relieved from repeating the commands. The time taken to invoke the SVM services for training and testing remained consistent throughout both the approaches. The Karma setup time was calculated only once as it was a one-time process. The user was also not expected to get a better result or a faster time.

TABLE VI: Stepwise time chart for processing one file using Excel

Step	Task	User Time (sec)	System Processing Time (sec)	Total Elapsed Time
1	Process AccelerometerSensor data — add magnitude and set timestamp column to be 4 decimal places	44	0	0:44
2	Extract timestamp and Magnitude in new worksheet and save as CSV	41	0	1:25
3	Invoke addDFT script	8	2	1:35
4	Process addDFT output file — format timestamp column to be 4 decimal places	12	0	1:48
5	Copy timestamp, speed and accuracy columns from LocationProbe data into a new worksheet	41	0	2:29
6	Process timestamp column to be 4 decimal places, and add a new column to round off the decimal	25	0	2:54
7	Add vLookUp formulae in the AccelerometerData worksheet for Speed	27	0	3:21
8	Add vLookUp formulae in the AccelerometerData worksheet for Accuracy	34	0	3:55
9	Apply filter to remove unmatched — NA rows after join and delete them.	43	0	4:38
10	Save this accelerometer with DFT data for input to labeling service	19	0	4:57
11	Invoke the labeling service over the exported CSV file	12	1	5:09
12	Filter data to remove NA columns	32	0	5:41
13	Save the file as ProcessedData file	6	0	5:48
14	Copy the ProcessedData file to the required location for SVM invocation	10	0	5:58

The accuracy for our experiment is the number of data preparation tasks completed correctly. We evaluated the correctness of the task as discrete Boolean values. At the end of processing a data file, if we were able to invoke the SVM training or testing service with no errors, we marked it as a successful completion. In our experiment, having 100% accuracy at the end of all the trials for Excel and Karma does not indicate that the task is easy to perform, but it highlights the reliability and availability of using Karma to replay the transformations any number of times and always yield correct results. When using Excel, the user could undo/redo a certain transformation to rectify any error and make sure the final result was correct it. Hence it yields 100% accuracy while using Excel.

It took 9:30 minutes to complete the Karma setup phase and 22:39 minutes to complete two trials of data preparation and mining in the execution phase. The execution time, as shown in Table V and VI for Karma and Excel respectively, is the sum of user time and system processing time and, does not include any setup time. We observed that the system processing time is approximately 20% of the total execution time for Karma. When using Excel, the total execution time to complete the same set of trials was 40:20 minutes. We compare Karma's execution time with the execution time for Excel since both involve repetition all of the tasks for every file. We will exclude Karma's setup time in the comparison and address it later in the discussion.

We observed a 42.14% reduction in the execution time by using Karma to process six data files. Every time we picked a new file to process, Karma took approximately one minute less than Excel. Assuming the user had collected data for 50 days and at the end of each day the user wanted to perform the data mining tasks on the newly collected data, he/she would need to apply Karma/Excel again. The files could not be merged into one file because the data was collected for each day and labelled separately and, the DFT calculations were performed on the data for a given day. The cumulative time spent processing the data over 50 days could be approximately 336 minutes. However the process could be completed in approximately 118 minutes using Karma, which is a 43% reduction in time.

Karma's setup time is a one time, non-iterative process to model all the data and services required for the given task. In our experiment, when the setup time is included, the total time reduction by using Karma decreases from 42.14% to 20.28%. Therefore, in our experiment, Karma requires 4 iterations of data processing to amortize the cost of its setup. As the number of files to be processed increases, for example 50 files, the reduction ratio is more or less the same even if the Karma setup time is included or excluded.

Here the benefit of modeling data source and services to an ontology using Karma, is the ability to share models across users. We will discuss another important scenario where multiple users of the same skill level are involved in our experiment to process the data. For example, we consider 10 users participating in our experiment to process 60 files. If every user is using Karma individually, they will all roughly need 9:30 minutes to setup Karma and 22:39 minutes to process 6 files. A total of 5.3 person hours is spent in data preparation. By sharing the models created by one user the setup time can be eliminated for all the other users. Therefore, if only one user spent his 9:30 minutes in setting up Karma, the data preparation task can be completed in 3.7 hours, saving 26% of the time that was required previously. Based on the observations, we can conclude that Karma improves efficiency in data preparation and data mining on large data sets by reducing time taken and manual efforts.

TABLE VII: Timing summary

		Karma			Excel		
		Time	TotalSeconds	Accuracy	Time	TotalSeconds	Accuracy
Trial 1	Dataset 1	4:41 min	281	Correct	5:58 min	358	Correct
	Invoke SVM training	0:12 min	12	Correct	0:35 min	35	Correct
	Dataset 2	3:20 min	200	Correct	5:50 min	350	Correct
	Invoke SVM testing on dataset2	0:14 min	14	Correct	0:30 min	30	Correct
	Invoke SVM training using dat set 1 and 2 combined	0:29 min	29	Correct	1:20 min	80	Correct
	Dataset 3	3:13 min	193	Correct	5:43 min	343	Correct
	Invoke SVM testing on dataset3	0:12 min	12	Correct	0:29 min	29	Correct
Trial 2	Dataset 1	3:18 min	198	Correct	5:36 min	326	Correct
	Invoke SVM training	0:11 min	11	Correct	0:31 min	31	Correct
	Dataset 2	2:49 min	169	Correct	5:25 min	325	Correct
	Invoke SVM testing on dataset2	0:16 min	16	Correct	0:28 min	28	Correct
	Invoke SVM training using dat set 1 and 2 combined	0:25 min	25	Correct	1:43 min	103	Correct
	Dataset 3	3:05 min	185	Correct	5:53 min	353	Correct
	Invoke SVM testing on dataset3	0:14 min	14	Correct	0:29 min	29	Correct
Analysis	Total	22:39 min	1359		40:20 min	2420	
	Total Reduction excluding karma setup				17:41 min		42.14%
	Total Reduction including karma setup (+9:30 min)				8:11 min		20.28%
	Accuracy with Karma						100.00%
	Accuracy with Excel						100.00%

VI. RELATED WORK

Preparing raw data for a data mining task typically requires multiple data transformation steps [5]. Many of these tasks are common across a large number of data mining applications, such as dealing with missing values, discretization, and normalization. However, in many cases, these are not sufficient and other transformation steps are required which are specific to the problem being solved.

Popular data mining software suites like RapidMiner[4] and KNIME[5] allow most of the common transformations like string operations, invoking remote web services, etc. However, when it comes to performing data transformation tasks that are specific to the problem at hand, for example DFT calculations, a majority of these suites do not provide built in support for such specialized transformations. Using remote services to perform special transformation is supported by most of the tools but they lack a semantic definition of the services that could be published and shared by using Karma.

There are also data preparation tools like DataPreparator[6] that offer operations like merging data from different files, but these tools are just for data preparation and mining has to be done separately, unlike our end-to-end approach. Moreover, these tools do not provide support for problem-specific custom data transformation tasks.

Google Refine[7] is another data transformation tool which has data cleaning capabilities, and supports some commonly used data transformation tasks. It also supports custom text transformations to modify text in data cells. Like in the case of other data preparation tools, data mining has to be done separately as Google Refine does not have data mining capabilities.

Our work provides an end-to-end integrated approach for data preparation and mining, with support for problem-specific transformation tasks as features, extensions, or services. Moreover, our work allows automation of data preparation tasks, which reduces the time taken to prepare data for mining.

---

[4] http://rapidminer.com/products/rapidminer-studio
[5] http://www.knime.org/knime
[6] http://www.datapreparator.com
[7] https://code.google.com/p/google-refine/

VII. DISCUSSION

This paper presented a unified approach to data preparation and mining using the data integration system, Karma. We discussed an end-to-end approach of data preparation and data mining, which consists of various transformations applied on a data set, publishing the data as RDFs, joining different data sets based on their mapping to the ontology, modeling the data mining services to a domain model, and invoking theses services with the published data. Our work on Karma provided us with a platform to record the various transformations that the user performs in preparing the raw data set to its desired form. We achieved a significant improvement in reducing the time taken for this iterative process of data preparation by replaying the transformations the user performed on the first file on all the other files in our data set. We also showed how services could be modeled using semantics and reused by publishing them as linked data.

REFERENCES

[1] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[2] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[3] C. A. Knoblock, P. Szekely, J. L. Ambite, S. Gupta, A. Goel, M. Muslea, K. Lerman, M. Taheriyan, and P. Mallick, "Semi-automatically mapping structured sources into the semantic web," in *Proceedings of the Extended Semantic Web Conference*, Crete, Greece, 2012.

[4] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. Sen. Netw.*, vol. 6, no. 2, pp. 13:1–13:27, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1689239.1689243

[5] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Advances in knowledge discovery and data mining," U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1996, ch. From Data Mining to Knowledge Discovery: An Overview, pp. 1–34. [Online]. Available: http://dl.acm.org/citation.cfm?id=257938.257942

[6] P. Szekely, C. A. Knoblock, F. Yang, X. Zhu, E. Fink, R. Allen, and G. Goodlander, "Publishing the data of the smithsonian american art museum to the linked data cloud," *International Journal of Humanities and Art Computing (IJHAC)*, vol. 8, pp. 152–166, 2014.