Marcello Trovati · Richard Hill
Ashiq Anjum · Shao Ying Zhu
Lu Liu *Editors*

Big-Data Analytics and Cloud Computing

Theory, Algorithms and Applications

Springer

Chapter 6
Data Science and Big Data Analytics at Career Builder

Faizan Javed and Ferosh Jacob

Abstract In the online job recruitment domain, matching job seekers with relevant jobs is critical for closing the skills gap. When dealing with millions of resumes and job postings, such matching analytics involve several Big Data challenges. At CareerBuilder, we tackle these challenges by (i) classifying large datasets of job ads and job seeker resumes to occupation categories and (ii) providing a scalable framework that facilitates executing web services for Big Data applications.

In this chapter, we discuss two systems currently in production at CareerBuilder that facilitate our goal of closing the skills gap. These systems also power several downstream applications and labor market analytics products. We first discuss Carotene, a large-scale, machine learning-based semi-supervised job title classification system. Carotene has a coarse and fine-grained cascade architecture and a clustering based job title taxonomy discovery component that facilitates discovering more fine-grained job titles than the ones in the industry standard occupation taxonomy. We then describe CARBi, a system for developing and deploying Big Data applications for understanding and improving job-resume dynamics. CARBi consists of two components: (i) WebScalding, a library that provides quick access to commonly used datasets, database tables, data formats, web services, and helper functions to access and transform data, and (ii) ScriptDB, a standalone application that helps developers execute and manage Big Data projects. The system is built in such a way that every job developed using CARBi can be executed in local and cluster modes.

6.1 Carotene: A Job Title Classification System

In many diverse domains such as bioinformatics and e-commerce (among others), content is organized according to concept hierarchies and taxonomies. A few examples of well-known taxonomies are Amazon.com, Ebay.com, the United Nations Standard Products and Services Code (UNSPSC), and Wikipedia (among

F. Javed (✉) • F. Jacob
Data Science R&D, CareerBuilder, Norcross, GA, USA
e-mail: faizan.javed@careerbuilder.com

M. Trovati et al. (eds.), *Big-Data Analytics and Cloud Computing*,
DOI 10.1007/978-3-319-25313-8_6

others). For large e-commerce websites, accurate item classification provides more relevant recommendations and a better item catalog search and browsing experience for end users. In many highly competitive e-commerce markets, an improved end-user experience can potentially result in an increase in sales transactions and repeat customers. Most of these websites handle Big Data and have deployed large-scale systems that can classify millions of items to thousands of categories in a fast, efficient, and scalable manner.

In the online recruitment domain, CareerBuilder (CB) operates the largest online job site in the USA with more than 24 million unique visitors a month (as of March 2015). One of CB's goals is to help close the skills gap. The skills gap is defined as the perceived mismatch between the needs of employers for talent and the skills possessed by the available workforce. At CB, accurate classification of job ads (job title, description, and requirements) and resumes to occupation categories is important for many downstream applications such as job recommendations and labor market analytics products. These applications contribute to CB's goal of helping job seekers find the jobs and training they need to be more competitive on the job market. The applications also deliver valuable insights to employers to help shape their recruitment strategies. Figure 6.1 shows some examples of CB applications and products that utilize Carotene.

Carotene is a large-scale job title classification system that leverages text document classification and machine learning algorithms. An automatic document classification system that uses machine learning algorithms requires documents labeled with predefined classes to create a set of training data. The training data is then used to induce a model that can generalize beyond the examples in the training data. The model is then used to classify new documents to one or more

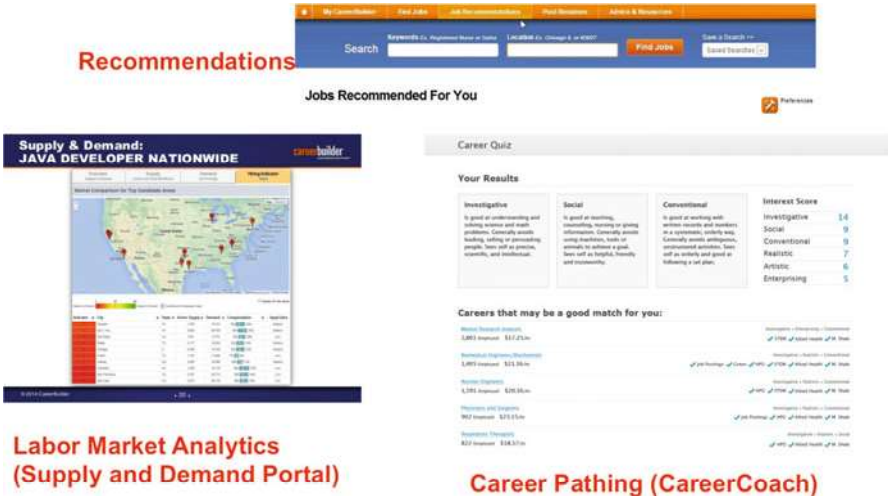

Fig. 6.1 Carotene downstream applications and products

of the predefined classes. There are many related works in both industry and academia that discuss classification systems that categorize entities to domain-specific taxonomies. In [1], a Support Vector Machine (SVM) [2] was used as both a coarse and fine-grained classifier as a cascade to classify large heterogeneous web content. A relevance vector machine (RVM)-k-nearest neighbor (kNN) classifier cascade to detect cancer from 3D medical images is discussed in [3]. In this system, data samples are first grouped into similar clusters using clustering. An RVM is used as a coarse-level classifier to prune data samples far from the classification boundary, while kNN is used as the fine-level classifier for the final classification. EBay's work on large-scale item categorization [4] uses a cascade of kNN-SVM classifiers to classify products to a homegrown taxonomy. In the online recruitment domain, LinkedIn's approach to job title classification [5] is based on a heavily manual phrase-based system that relies on the near-sufficiency property of short text. The system has offline components that build a controlled vocabulary and a list of standardized job titles as well as a component for crowdsourced labeling of training samples.

6.1.1 Occupation Taxonomies

For the online recruitment industry in the USA, the Occupational Information Network (O*NET) taxonomy is the primary source of occupation categories. O*NET is an extension of the Standard Occupational Classification (SOC) system developed by the US Bureau of Labor Statistics and was developed through the sponsorship of the US Department of Labor/Employment and Training Administration (USDOL/ETA). The O*NET taxonomy is a four-level hierarchy composed of 23 major groups, 97 minor groups, 46 broad occupations, and 1,110 detailed occupations. Table 6.1 shows the taxonomy breakdown for O*NET code 15-1132.00 which represents the *Software Developers, Applications* occupation.

The O*NET taxonomy is not granular enough for our suite of products and applications. For example, consider the niche and emerging software engineering titles such as *Big Data Engineer* and *Machine Learning Engineer*. These job titles are represented alongside more general software engineering titles such as *Java Engineer* by the *Software Developers* subcategories 15.1132 (*Applications*) and 15-1133 (*Systems Software*). The lack of fine-grained job title granularity inherent in the O*NET

Table 6.1 O*NET taxonomy groups for software developers, applications

Taxonomy group	Code and description
Major	15 – Computer and mathematical occupations
Broad	1130 – Software developers and programmers
Minor	1132 – Software developers, applications
O*NET extension	00 – Software developers, applications

and SOC taxonomies adversely affects our ability to drill down on metrics and analytics insights to the level of detail expected by our end users. These taxonomies cannot accommodate emerging job categories and titles because of long lead times between taxonomy updates. These taxonomies will be updated every 10 years.

6.1.2 The Architecture of Carotene

Classifiers for large-scale taxonomies are usually hierarchical cascade classifiers or flat classifiers. A flat classifier has to make a single classification decision that is more difficult than for a cascade classifier. A flat classifier also encompasses all leaf nodes of the taxonomy as classes that for large-scale taxonomies can be in the thousands. A hierarchical classifier is preferred for highly unbalanced large-scale taxonomies [6] because even though a hierarchical classifier has to make multiple decisions, there is a less severe imbalance at each level of the hierarchy. However, hierarchical classifiers are susceptible to error propagation because the final classification decision relies on the accuracy of previous decisions made in the hierarchy. Pruning the taxonomy can mitigate error propagation in hierarchical classifiers.

Carotene is a semi-supervised job title classification system that also has a clustering component for taxonomy discovery. The classification component is a multi-class, multi-label cascade classifier that for a query text returns a list of job titles ranked by confidence scores. Carotene only takes into consideration the top (SOC major) and O*NET code levels of the four-level O*NET taxonomy. To create the training dataset for Carotene, we collected 3.6 m job ads posted on CareerBuilder.com. The job ads distribution at the SOC major level is skewed towards occupation categories such as 15 (Computer and Mathematical) and 41 (Sales) among others. Job ads from occupation categories such as 45 (Farming, Fishing, and Forestry) and 23 (Legal) comprise the long tail of the overall job ads distribution. We tackle the class imbalance problem in the dataset by under sampling all the classes to a base count of around 150 k jobs. This reduced the training dataset size to 2 m jobs.

Since crowdsourced labeling of datasets is susceptible to systematic and consistency errors, we used Autocoder[1] to label the training dataset. Autocoder is a third-party tool that classifies recruitment industry content such as resumes and job ads to O*NET codes. Autocoder's classification accuracy of 80 % for job titles and 85 % for job ads puts a theoretical upper bound on Carotene's potential accuracy. However, we proceeded with using Autocoder labels for our dataset because one of our long-term goals is to meet or exceed Autocoder's accuracy and because of the cost and efficiency gains realized compared to third-party crowdsourcing alternatives. Figure 6.2 gives an overview of Carotene.

[1] http://www.onetsocautocoder.com/plus/onetmatch?action=guide

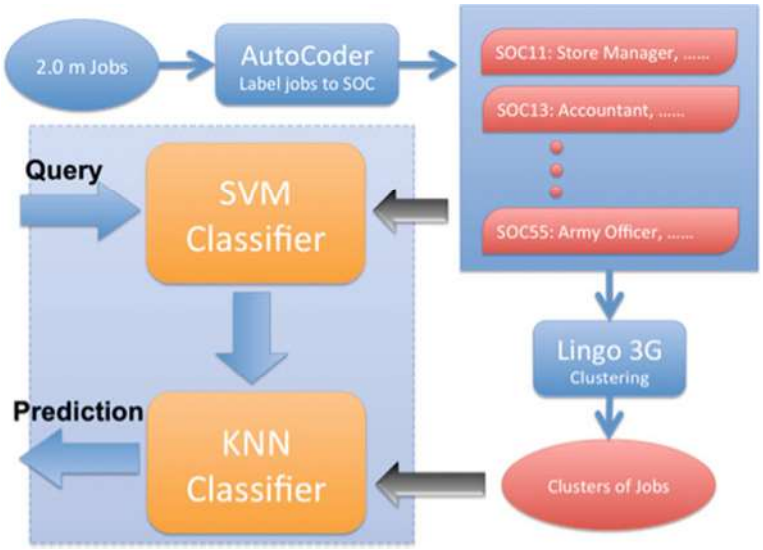

Fig. 6.2 Architecture of Carotene

6.1.2.1 Taxonomy Discovery Using Clustering

The first phase of Carotene uses a clustering process to discover job titles. The training dataset is organized by SOC major into dataset segments and preprocessed by removing extraneous markup characters and noise. For clustering, we use Lingo3G [7], a proprietary clustering algorithm. Lingo3G identifies meaningful cluster label that assists in inferring a job title taxonomy from a dataset. Some of the Lingo3G parameter settings that we empirically set were *max-cluster-size*, *min-cluster-size*, and *merge-threshold* among others. We run Lingo3G on these dataset segments to create *verticals*. These verticals are clusters of job titles for a SOC major. Every cluster in a vertical has a *glabel* that is the representative normalized label of the raw job titles in that cluster. We also applied several post-processing steps such as glabel validation and cluster merging and deletion. These steps require domain expertise and human validation. To facilitate such post-processing operations, we use an in-house tool that provides a user-friendly interface and abstraction to automate most of these tasks. The tool helped us reduce our original estimate of 3 months for creating the taxonomy to 1 month. The fine-level classifier in the classification component of Carotene uses the appropriate vertical to return the classification for the input query text. There are 4,076 job titles in the Carotene taxonomy. Figure 6.3 shows a Physical Therapy Assistant cluster from the SOC-31 vertical.

```
22 | Physical Therapy Assistant (871)
   |    *Physical Therapist Assistant - Orlando - Acute Care - Per Diem
   |    *Physical Therapist Asst. - STE
   |    *Physical Therapy Assistant - Deaconess Hospital
   |    *Physical Therapy Assistant - Hillside Rehabilitation Hospital
   |    *Physical Therapy Assistant- Per Diem- Northside Medical Center
```

Fig. 6.3 Physical therapy assistant cluster

6.1.2.2 Coarse-Level Classification: SOC Major Classifier

Carotenes' classification component is a coarse and fine-level hierarchical cascade. Our coarse-level classifier is a SOC major classifier. At the coarse level, jobs are classified to a SOC major which is then used by the fine-level classifier to narrow the space of candidate job title labels for the input query. We use the LIBLINEAR [8] implementation of the SVM classifier as the SOC major classifier. We chose the L2-regularized L2-loss SVC (dual) solver for training the model because this is a large-scale, sparse data-learning problem where the number of features is far greater than the number of training instances. The training dataset had 5.8 m features with approximately 280 nonzero features per instance. It took 1.5 h to train the SOC major classifier on an m3.2xlarge EC2 instance with 30GB of RAM. While the SOC major classifier is a multi-class, multi-label classifier that returns a list of classifications ranked by confidence, in Carotenes' current cascade architecture, only the top-ranked label is sent to the fine-level classifier for further processing. We estimated the accuracy of the SOC major classifier using tenfold cross validation where the average precision, recall, and F-scores were 95.54 %, 95.33 %, and 95.43 %, respectively. Global accuracy was 95.8 % and coverage was 99 %.

6.1.2.3 Fine-Level Classification: Proximity-Based Classifier

Proximity-based classifiers such as k-nearest neighbor (kNN) find the k-nearest neighbors of a query instance or pre-process the data into clusters and use cluster meta-data as *k*-nearest neighbor documents to improve classification accuracy. Carotene uses kNN as a multi-class, multi-label proximity-based fine-level classifier with k empirically set to 20. Standard implementations of kNN are considered slower than other classification techniques because of kNNs' lazy learning characteristics. We use a Lucene[2]-based implementation of kNN that results in classification response time of less than 100 ms. The kNN classifier has access to 23 verticals and chooses the appropriate vertical based on the classification it receives from the SOC major classifier. The first version of Carotene, CaroteneV1, had a classification component that was a single-level kNN classifier that queried a

[2] http://lucene.apache.org

single vertical composed of the initial set of 1,800 titles. Compared to its majority voting-based implementation in CaroteneV1 which is susceptible to skewed class distributions, the kNN classifier in Carotene has an improved voting strategy where the absolute values of Lucene scores are used to assign a weight to each neighbor. In CaroteneV1, only the raw job titles in verticals were indexed and searched on. Raw job titles are noisier than normalized titles. In Carotene, the verticals also have a normalized job title called glabel for each cluster; hence, both the raw titles and glabels are indexed as *title* and *label* fields, respectively. Carotene uses a Lucene multi-field query with a default boost factor of 1.0 and a boost factor of 0.9 for the title and label fields, respectively. Indexing the glabel improves the quality of classification in close boundary decisions.

6.1.3 Experimental Results and Discussion

Development teams at CB use Carotene for classifying large text such as job ads and resumes as well as short text consisting of job titles. Hence, we used two datasets to compare the performance of Carotene to CaroteneV1: job ads and normalized titles. The job ads dataset contains 1,000 randomly sampled jobs posted on CareerBuilder.com. We ensured that the job ads selected were from all the 23 major SOC categories. For the normalized title dataset, we chose 570 job titles that exist in both CaroteneV1 and Carotene taxonomies. Table 6.2 shows the performance of CaroteneV1 and Carotene on the two datasets. We observe a 27.7 % increase in accuracy at classifying job ads with Carotene compared to CaroteneV1. For the job title dataset, there is a 12 % increase in accuracy from 68 % to 80 %. The more comprehensive taxonomy along with the cascade architecture of the classification component of Carotene gives better job title classification results.

We also compared Carotene to Autocoder by running a user survey of users registered on CareerBuilder.com. We used surveys as a way of measuring their comparative performances because the two systems have different taxonomies. The most recent work history listed in a user's resume was classified using both Carotene and Autocoder, and the users were then requested to rate the classifications on a scale of 1–5, where 5 is for a perfect classification and 1 for a classification that totally missed the mark. We surveyed 2 M active users, and Carotene had a classification precision of 68 % versus 64 % for Autocoder. We observed that in some instances, Autocoder's classification was more likely to be given a higher rating because of the more general nature of its classifications especially

Table 6.2 Accuracy of Carotene and CaroteneV1 on job ads and job titles

	Accuracy	
Dataset	*Carotene V1* (%)	*Carotene* (%)
Normalized titles	68	80
Job ads	39.7	67.4

when the more fine-grained Carotene classification did not exactly match the user's job title. For example, for "Embedded Systems Software Engineer," the Autocoder classification "Software Developers, Systems Software" is more likely to be rated higher by users than the Carotene classification of "C++ Developer." As we iteratively improve Carotene's classification accuracy as well as expand the coverage of its taxonomy, we expect such discrepancies in user ratings to decrease over time.

As a system currently in production at CB and actively being used by several internal clients, there are several improvements and extensions to Carotene that we are working on such as (1) internationalization, support for Carotene in international markets, (2) extending the architecture to support classification at the O*NET code level, and (3) enrichments vectors and semantic kernels for more semantically aligned classifications.

6.2 CARBi: A Data Science Ecosystem

CARBi is a Big Data/Data Science ecosystem designed and implemented to satisfy the growing needs of data scientists at CB. With CARBi, data scientists have support for developing, executing, and managing Big Data solutions. The core components of CARBi are explained in the following subsections.

6.2.1 Accessing CB Data and Services Using WebScalding

WebScalding was initially developed to process high volumes of resume and job data (most of our data processing are implemented as scalable webservices, hence the name "WebScalding"). The project evolved to an umbrella of many reusable components and encompasses several development best practices. Today, WebScalding is a collective name for a group of projects that are designed for Big Data solutions. A project can be a WebScalding project or a WebScalding-compatible project, and each of them is explained in the following subsections:

- *WebScalding project*: A project that uses the WebScalding core library that is a Big Data library developed using Twitter's Cascading [9] interface called Scalding.[3] In addition to the core project, a WebScalding project can use any WebScalding compatible projects as well.
- *WebScalding-compatible project*: Any project that exposes APIs satisfying two main requirements: (1) independent execution, each execution of the API should be completely independent of any another execution thus enabling parallel execution of the API, and (2) independent existence, every execution is self-contained

[3] https://github.com/twitter/scalding

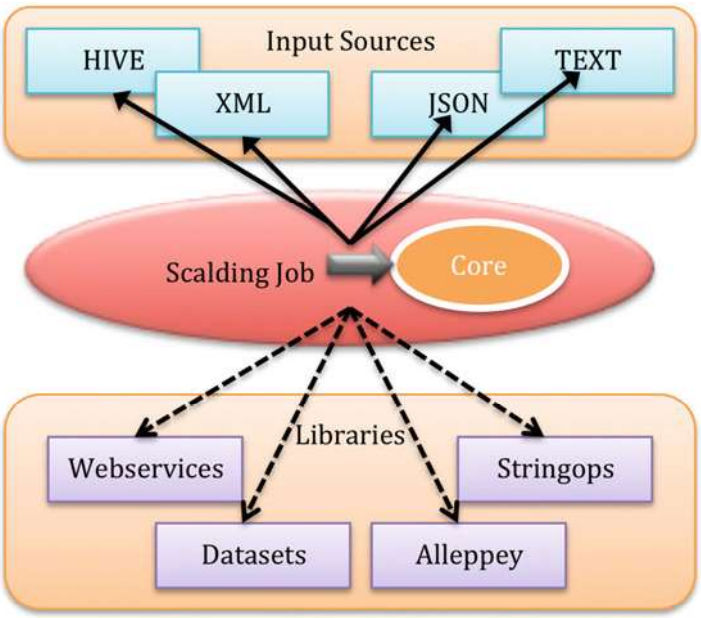

Fig. 6.4 Overview of WebScalding

such that it can create data if required. As an example, if an API uses an index, the WebScalding-compatible implementation of the API can create the index if the index doesn't exist.

An overview of the WebScalding system is shown in Fig. 6.4. Currently, the system can read/write data from/to four formats: (1) HIVE, (2) XML, (3) TEXT, and JSON. In the figure, four WebScalding-compatible projects are shown and are explained in the following subsections:

1. *Web Services*: We designed the web services component to enable us to call our scalable web services for high volumes of data. Using the web services component, users can define a web service as an executable Hadoop job reading inputs from any of the supported formats and writing the output to one of the supported format.
2. *StringOps*: Our primary sources of data are job postings and resumes. Consequently, text/string processing is an important functionality for WebScalding applications. The StringOps component supports many text operations such as stemming, string matching, string sorting, language detection, and parsing several text formats.
3. *Alleppey*: Our commonly used machine learning libraries and tools are available in the Alleppey project. This project enables us to quickly try out different machine learning models for the same data. We currently support several popular machine learning toolkits (e.g., WEKA [10], LibShortText [11]).

4. *Datasets*: We have several projects (e.g., sCooL [12], SKILL [13]) that use knowledge bases (KBs) built using our data as well as open source datasets. We have developed projects that provide access to open source datasets such as Wikipedia and Freebase.[4]

To illustrate how WebScalding can help data scientists in designing and executing Big Data solutions, we have included an example to show how HIVE tables are accessed in a WebScalding project. For WebScalding, any HIVE table is a file location, so it is possible to have multiple WebScalding table definitions for the same file location. This is very useful for cases where the actual HIVE table has many columns, but the user is interested in only some of them for a given job. To access any HIVE table in a WebScalding project, we have to first define a WebScalding table and then load that WebScalding table in a Hadoop job.

WebScalding Table Definition An example of WebScalding table definition is shown in Fig. 6.5. We require every WebScalding job to be executable in two modes: (1) local and (2) cluster. In local mode, the job is executed locally with local files as the input and output are written to local files as well. In cluster mode, the job reads from Hadoop HDFS and writes to the HDFS as well. To make this possible for a Hadoop job involving HIVE tables, we require the users to create a sample table that can simulate the HIVE table. In the example in Fig. 6.5, our goal is to read a table

```
1  case class UnifiedProfile(edgeId: String, profiles: String,
2                            parentId: String)
3
4  object UnifiedProfileTable extends HiveTable[UnifiedProfile] {
5
6    def tableName: TableName = TableName("local.txt", "remote")
7
8    def localColumns(x: String) = {
9      val parts = x.split("\t", -1)
10
11     UnifiedProfile(parts(0).trim, parts(4).trim(), parts(13).trim)
12   }
13
14   def hdfsColumns(x: String) = {
15     val parts = x.split(HIVEDelimiter, -1)
16     UnifiedProfile(parts(0).trim, parts(4).trim(), parts(13).trim)
17
18   }
19
20   def columns = Columns(localColumns, hdfsColumns)
21
22 }
```

Fig. 6.5 HIVE table in WebScalding

[4]Freebase, http://www.freebase.com

```
1 class GenerateEdgeNetworkLinks(args: Args)
2                   extends Job(args) with HiveAccess  {
3
4    getHiveContent(UnifiedProfileTable)
5 }
```

Fig. 6.6 Using HIVE table in a Scalding job

and create `UnifiedProfile` object (line 1) for each row. Users have to provide the information how to convert a line in the file location to a `UnifiedProfile` object. From our experience, this parsing can be different (e.g., the delimiters can be different, number of columns can be different) for cluster and local modes; hence, users has the flexibility to provide the conversion process separately for local (lines 8–12) and remote (lines 14–18) modes.

WebScaling Table Access Figure 6.6 shows an example of accessing a WebScalding Table in a Hadoop job. A WebScalding table can be accessed in a Hadoop job using the trait `HiveAccess` (line 2) which includes a method `getHiveContent(WebScaldingTable)` (line 4). This returns a `TypedPipe`[5] of type `UnifiedProfile`.

To summarize, we achieved two main goals by introducing WebScalding: (1) we raised the level of abstraction, enabling data scientists to focus more on the problem at hand than on the low level implementation details of MapReduce or Hadoop, and (2) we empowered users to develop "write once and run on local and cluster modes" code, enabling users to test the data flow locally before executing it on a cluster.

6.2.2 ScriptDB: Managing Hadoop Jobs

The main motivation of ScriptDB is management of multiple Hadoop jobs associated with a single project. Some of these jobs are interconnected to form a pipeline while others stay independent (e.g., the Recruitment Edge project[6] currently has more than 100 Hadoop jobs). Such Hadoop jobs usually require different configuration parameters, input arguments, and other details that are very specific to the project and the job in question. In an effort to keep all the Hadoop jobs available to run, we have the tedious task of maintaining script files specific to each Hadoop job. Any refactoring in the project or the job can lead to making changes in these script files. Moreover, changing versions or platforms (e.g., Hadoop versions, Hadoop to Scalding conversions) can cause significant changes in the script files for each Hadoop job execution.

[5] TypedPipe, http://twitter.github.io/scalding/index.html#com.twitter.scalding.typed.TypedPipe

[6] Recruitment Edge, http://edge.careerbuilder.com

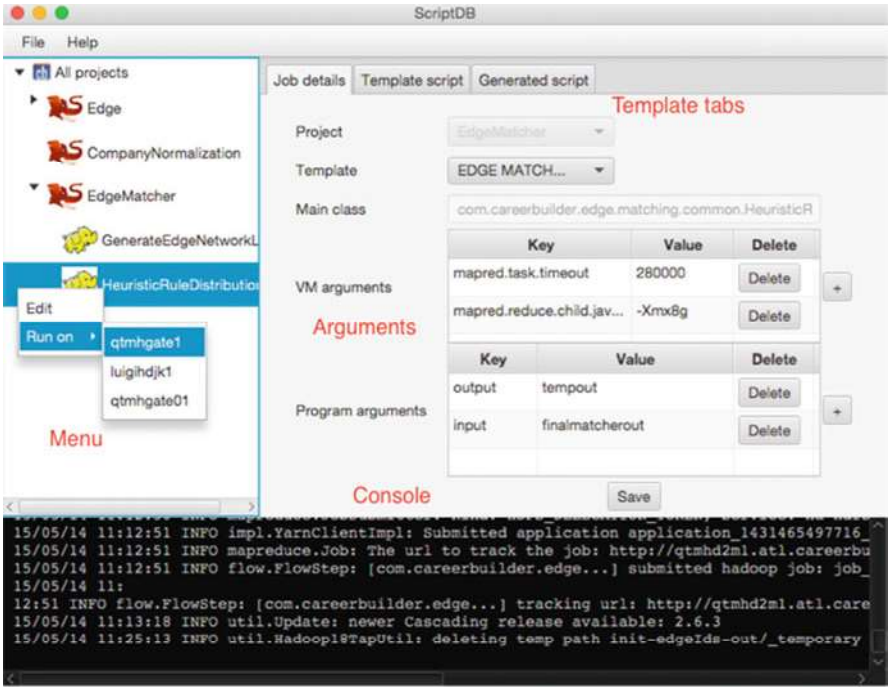

Fig. 6.7 ScriptDB app to execute Hadoop jobs

We developed ScriptDB so that data scientists can (1) browse through the available Hadoop jobs and if required submit the job to a given cluster and (2) add/update a Hadoop job independent of the language or platform. A screenshot of ScriptDB is shown in Fig. 6.7. To define a Hadoop job using ScriptDB, the user has to specify the project, a script template, the main class for the Hadoop job, and arguments. The key parts of the ScriptDB are explained in the following subsections:

- *Menu*: Hadoop jobs are organized by projects. This is to facilitate an ordering and also to allow project-specific templates for job submission.
- *Arguments*: Using ScriptDB, users can provide two kinds of arguments: (1) program arguments, the arguments the Scalding job/Hadoop job need to run the job, and (2) virtual machine (VM) arguments, similar to VM arguments for a JAVA program, the configuration parameters for a MapReduce job (e.g., mapred.task.timeout).
- *Console*: Console prints out the information messages and error messages for the user. This is also useful if user wants to execute a command in a server, in that case, console serves as "System.out."
- *Template tabs*: The key feature of the ScriptDB is the template. The template mechanism makes ScriptDB language independent. The user is given the ability to create/update templates and to view the generated script file from the template for a given job.

```
template1(main,arguments,jarname,options,projectname,projectlocation) ::=<<
 #!/bin/bash
JAR_NAME=<projectlocation>/<jarname>
RUNNER=com.careerbuilder.edge.main.JobRunner
MAIN_CLASS=<mainclass>
cd <projectlocation>
git pull origin yarn-changes
sbt 'project matcher' clean assembly
chmod a+r <jarname>
ARGUMENTS=<arguments:{u|--<u.key> <u.value> }>
hadoop jar $JAR_NAME  $RUNNER $MAIN_CLASS --hdfs $ARGUMENTS &
```

Fig. 6.8 Stringtemplate script for executing a Hadoop job

```
 1 #!/bin/bash
 2 JAR_NAME=/home/fjacob.site/Recruitment-Edge/target/scala-2.11/EdgeHadoop-job.jar
 3 RUNNER=com.careerbuilder.edge.main.JobRunner
 4 MAIN_CLASS=com.careerbuilder.edge.emsi.GenerateEdgeNetworkLinks
 5 cd /home/fjacob.site/Recruitment-Edge/scalding/Edge
 6 git pull origin yarn-changes
 7 sbt 'project matcher' clean assembly
 8 chmod a+r matcher/target/scala-2.11/EdgeHadoop-job.jar
 9 ARGUMENTS=--output init-edgeIds-out --input init-edgeIds
10 hadoop jar $JAR_NAME  $RUNNER $MAIN_CLASS --hdfs $ARGUMENTS &
```

Fig. 6.9 Code generated using the ScriptDB template engine

ScriptDB uses Stringtemplate[7] as the template engine, and hence, the templates have to be specified using the Stringtemplate syntax. In Fig. 6.8, all the StringTemplate-specific syntax is highlighted in blue. An example code generated using the template is shown in Fig. 6.9.

References

1. Dumais S, Chen H (2000) Hierarchical classification of web content. In: Proceedings of ACM SIGIR'00. New York, USA, pp 256–263
2. Joachims T (1999) Transductive inference for text classification using support vector machines. In: Proceedings of ICML 1999. San Francisco, USA, pp 200–209
3. Liu M, Lu L, Ye X et al (2011) Coarse-to-fine classification via parametric and nonparametric models for computer-aided diagnosis. In: Proceedings of ACM CIKM'11. New York, USA, pp 2509–2512
4. Shen D, Ruvini J-D, Sarwar B (2012) Large-scale item categorization for e-commerce. In: Proceedings of ACM CIKM'12, pp 595–604

[7]Stringtemplate, http://www.stringtemplate.org

5. Bekkerman R, Gavish M (2011) High-precision phrase-based document classification on a modern scale. In: Proceedings of the 17th ACM SIGKDD-KDD'11, pp 231–239
6. Babbar R, Partalas I (2013) On flat versus hierarchical classification in large-scale taxonomies. In: Proceedings of the neural information processing systems (NIPS), pp 1–9
7. Osiński S, Weiss D (2005) A concept-driven algorithm for clustering search results. IEEE Intell Syst 3(20):48–54
8. Fan RE et al (2008) Liblinear: a library for large linear classification. J Mach Learn Res 9:1871–1874
9. Nathan P (2013) Enterprise data workflows with cascading, 1st edn. O'ReillyMedia, Sebastopol. Sebastopol, USA
10. Hall M, Frank E, Holmes G et al (2009) The WEKA data mining software: an update. ACM SIGKDD Explor Newsl 11(1):10–18
11. Yu H-F, Ho C-H, Juan Y-C et al (2013) LibShortText: a library for short-text classification and analysis. Department of Computer Science, National Taiwan University, Taipei, Technical report. http://www.csie.ntu.edu.tw/ncjlin/papers/libshorttext.pdf
12. Jacob F, Javed F, Zhao M et al (2014) sCooL: a system for academic institution name normalization. In: 2014 international conference on collaboration technologies and systems, CTS. Minneapolis, USA, pp 86–93
13. Zhao M, Javed F, Jacob F et al (2015) SKILL: a system for skill identification and normalization. AAAI 2015. Austin, USA, pp 4012–4018