Ava: From Data to Insights Through Conversation

Rogers Jeffrey Leo John Navneet Potti Jignesh M. Patel
Computer Sciences Department
University of Wisconsin-Madison
{rl, nav, jignesh}@cs.wisc.edu

ABSTRACT

Enterprises increasingly employ a wide array of tools and processes to make data-driven decisions. However, there are large inefficiencies in the enterprise-wide workflow that stem from the fact that business workflows are expressed in natural language but the actual computational workflow has to be manually translated into computational programs. In this paper, we present an initial approach to bridge this gap by targeting the data science component of enterprise workflows. In many cases, this component is the slowest part of the overall enterprise process, and focusing on it allows us to take an initial step in solving the larger enterprise-wide productivity problem. In this initial approach, we propose using a chatbot to allow a data scientist to assemble data analytics pipelines. A crucial insight is that while precise interpretation of general natural language continues to be challenging, controlled natural language methods are starting to become practical as natural interfaces in complex decision-making domains. In addition, we recognize that data science workflow components are often templatized. Putting these two insights together, we develop a practical system, called Ava, that uses (controlled) natural language to program data science workflows. We have an initial proof-of-concept that demonstrates the potential of our approach.

1. INTRODUCTION

It is now increasingly common for many aspects of business decisions to be guided in some form by data. This change in approach permeates through every level of the organization, from the CXOs down to the business analysts. However, the task of analyzing data resides in a select subset of employees, the so-called Data Science (DS) professionals. These professionals have the responsibility of translating complex business problems into a technical form, then applying their data analytics skills to solve these problems, and finally producing insights to improve business outcomes.

Our research community has built a plethora of efficient libraries and platforms that greatly simplify the individual tasks of the DS workflow. While we should continue to improve these tools, we must also be mindful of the larger organizational context for DS workflows. The real bottleneck is the turn-around time for the *entire* workflow. A crucial aspect of this holistic view is to improve the human productivity of the data scientists, and reduce their turn-around time in assembling DS pipelines. We note that our larger vision is to improve the productivity of the entire enterprise data-to-insights workflow, and democratize DS capabilities throughout an organization. In this paper, we start with the data scientists, as it provides a concrete starting point. In the future, we plan to target increasingly larger user-bases consisting of ever-less technical users, empowering them to directly construct and manage robust DS workflows.

To improve the productivity of data scientists, we begin with the observation that a natural language interface improves the efficiency of translating business problems into the technical domain, by reducing the impedance mismatch. Further, we note that the diversity of available tools, while a boon in many ways, has the detrimental effect of increasing the cognitive burden on a data scientist and steepens their learning curve, hindering productivity. Since these tools overlap in the problems that they solve, we propose using a natural language interface to *abstract* away the details of the underlying libraries and let a user cleanly specify their task at the conceptual level, automating the translation to executable code.

The second key observation in our work is that this translation to executable code is facilitated by the templatized nature of the workflow. Each individual task in the workflow (e.g., "Train a decision tree model with the given set of hyper-parameters.") maps to a short code sequence specific to the underlying library or platform in use. By maintaining a library of such task templates, we can translate a task specification in natural language into executable code by instantiating the template.

Further, we note that the DS workflow often proceeds through a well-defined sequence of logical stages, from data loading and cleaning, through feature engineering, to model training and visualization. This observation prompts us to build workflow tools that guide the data scientist through these stages by *composing* the templatized tasks at each stage. In this paper, we propose an interactive interface that eases the exploratory workflow in this way.

Finally, with an eye on the longer-term goal of expanding the target user-base of DS users beyond the typical data scientist community, we make the observation that this workflow tool should take the form of an intelligent, learning

8th Biennial Conference on Innovative Data Systems Research (CIDR '17)
January 8-11, 2017 , Chaminade, California, USA.

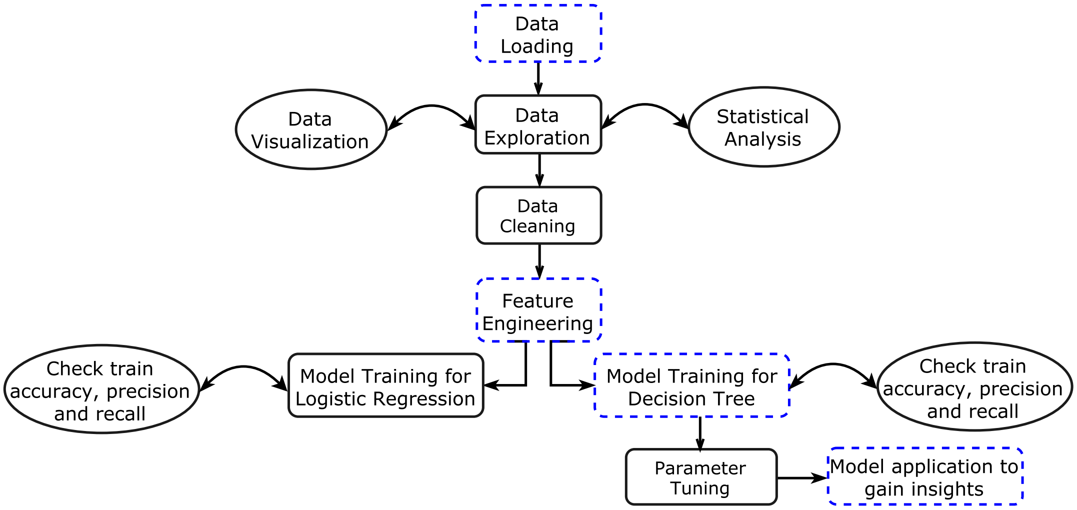


Figure 1: The Data Science workflow for Example 1. Tasks are shown in rectangles, and meta-tasks in ovals. The pipeline is highlighted in dotted blue boxes.

agent. Such an agent can be capable of learning from its interactions with data scientists by logging them and compiling a knowledge base for this context. This knowledge base includes such facts as which methods and tools are most effective, how optimal model hyper-parameter values relate to the dataset characteristics, what visualizations and metrics speed up the exploratory workflow, etc. The intelligent agent can then guide the user through the various stages of the workflow, recommending the most effective decisions the user can make at every step along the way. Thus, we believe we can pool together the experience and expertise of advanced data scientists, and democratize it to novices through an intelligent recommendation system.

Combining these ideas, we introduce Ava, an intelligent chatbot agent that interacts with data scientists using controlled natural language. Conversations in the controlled natural language are mapped to a state machine, and that state machine is used to unambiguously interpret the conversation, composing a workflow from individual templatized tasks. Thus, a key technical contribution of this work is proposing a way to unambiguously convert *conversations to compositions* for data analysis. In addition, we also propose an architecture for such an agent that not only uses predefined hard-coded rules to guide novice users through the workflow, but also learns from the interactions about what works and what doesn't. We demonstrate how our initial prototype based on this architecture is already capable of addressing some of the challenges we described above. Overall, we take an initial step in building conversational agents that in the future can be used by non data scientists too.

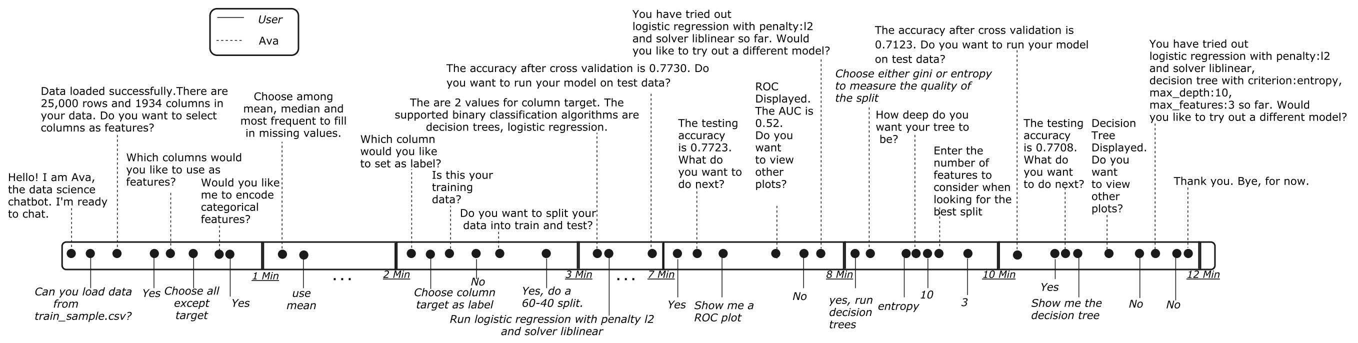
2. THE DATA SCIENCE WORKFLOW

In this section, we begin with an example and then formalize key terminology.

EXAMPLE 1. *A data scientist Daisy starts with a business problem, namely: "Predict which customers are most likely to respond to a direct email offer for a personal loan". As illustrated in Figure 1, she begins by loading some relevant data about customers from databases or files. She may use basic statistics and visualizations like histograms or scatter plots to better understand the data as well as gain intuition into the relevant features. She may find the need to impute missing postal Zip codes or bucket dates by month. She proceeds to carve out training and test datasets, and build a classification model using logistic regression. Finding that the model has poor precision and recall, she decides to use a decision tree instead. After confirming that the classification accuracy has improved, she fine-tunes various hyper-parameters such as tree depth or splitting criterion. She may iteratively visualize the tree or use an ROC curve to guide this model tuning. Finally, she settles on a model and applies it to current customer data to make her predictions about which customers are most likely to leave/churn.*

As illustrated in the example above, the DS **workflow** is composed of various logical **stages**, such as *Data Loading* and *Model Training*. At each logical stage, Daisy performs a number of physical **tasks**, such as loading one or more datasets from CSV files or databases, or training a *Decision Tree* model. In fact, while Daisy goes through roughly the same sequence of logical stages to model every business problem, she may perform very different tasks at each stage. For instance, when asked to predict inventory needs for the holiday season, she may go through a workflow with similar stages, but load data from different sources and train a regression model rather than a classification one.

Thus, the data scientist faces a number of choices of tasks at every stage, and must decide which task is best suited for this problem. These decisions are informed by prior experience and rules of thumb (e.g., decision trees might not work well if the number of observations is very small). In addition, the decisions are guided by **meta-tasks** such as statistical analysis and visualization. For instance, histograms and null-value counts are used to guide the imputation method used for data cleaning, as well as the choice of bucketing or encoding used for feature engineering. Similarly, metrics such as precision and recall (for classification) or R^2 (for regression), and visualizations such as Receiver Operating Characteristic (ROC) curves are used to guide model selection, including hyper-parameter tuning.



Hello! I am Ava, the data science chatbot. I'm ready to chat.

Can you load data from train_sample.csv?

Data loaded successfully.There are 25,000 rows and 1934 columns in your data. Do you want to select columns as features?

Yes

Which columns would you like to use as features?

Choose all except target

Would you like me to encode categorical features?

Yes

Choose among mean, median and most frequent to fill in missing values.

use mean

Which column would you like to set as label?

Choose column target as label

Is this your training data?

No

Do you want to split your data into train and test?

Yes, do a 60-40 split.

The are 2 values for column target. The supported binary classification algorithms are decision trees, logistic regression.

Run logistic regression with penalty l2 and solver liblinear

The accuracy after cross validation is 0.7730. Do you want to run your model on test data?

Yes

The testing accuracy is 0.7723. What do you want to do next?

Show me a ROC plot

ROC Displayed. The AUC is 0.52. Do you want to view other plots?

No

You have tried out logistic regression with penalty:l2 and solver liblinear so far. Would you like to try out a different model?

yes, run decision trees

Choose either gini or entropy to measure the quality of the split

entropy

How deep do you want your tree to be?

10

Enter the number of features to consider when looking for the best split

3

The accuracy after cross validation is 0.7123. Do you want to run your model on test data?

Yes

The testing accuracy is 0.7708. What do you want to do next?

Show me the decision tree

Decision Tree Displayed. Do you want to view other plots?

No

You have tried out logistic regression with penalty:l2 and solver liblinear, decision tree with criterion:entropy, max_depth:10, max_features:3 so far. Would you like to try out a different model?

No

Thank you. Bye, for now.

Figure 2: A timeline of an actual Ava conversation for the data science task in Example 1.

The outcome of this exploratory process is a DS **pipeline** that results in a model used to gather insights and solve the business problem at hand. The pipeline (shown in dotted blue boxes in Figure 1) excludes the extraneous steps (shown in black) that were incidental to the exploration, but will be cleaned up in the final output. For example, when asked to repeat the process for the same problem again, Daisy can use the already cleaned data (foregoing the *Data Cleaning* stage), directly train only the *Decision Tree* (skipping the mis-step of training the *Logistic Regression* model), and may even skip the meta-tasks like data and model visualization. Only this subset of tasks forms the pipeline in our terminology.

A **workflow** is thus a (usually cyclic) graph consisting of **tasks** and **meta-tasks**, conceptually grouped into **stages**. The **pipeline** is a subgraph of the **workflow**. The Data Science problem is one of **searching** for the best pipeline, and the exploratory steps involved make up the workflow.

2.1 Data Science Workflow is Templatized

The DS workflow is composed of a small, predefined set of stages and a well-defined sequence in which they occur in the workflow (possibly iteratively). Each stage is itself composed of a number of tasks.

The tasks in the workflow are typically performed by writing code in a general purpose programming language like Python, R or Scala, and using libraries like scikit-learn [12], caret [17] or Spark [38]. Usually, the code used to perform a certain task follows a corresponding template: a certain sequence of library calls. The listing below shows a code template for the task of training a decision tree using the Python scikit-learn library, wherein the data scientist is only required to select and set a few parameters, such as node splitting information criterion `criterion` and decision tree depth `max_depth`. Thus, the semantics of tasks typically lend themselves to a clean separation of specification (parameter values) and template, making it possible to compose the task code simply by substituting parameters into a pre-defined code template.

```
from sklearn import tree
classifier = tree.DecisionTreeClassifier(
                criterion={{criterion}},
                max_depth={{max_depth}})
model = classifier.fit(training_set, labels)
```

Observe that the DS workflow typically proceeds through a templated sequence of logical stages, each consisting of templatized tasks. This observation highlights an opportunity: We can build a user interface that is specifically tailored for this workflow, which is simpler than general purpose programming languages without significantly constraining the data scientist's capabilities.

A conversational interface simplifies the composition of workflows from task template libraries. The use of natural language allows a data scientist to provide the specification of the tasks without intimately familiarizing themselves with the specific details of the diverse and ever-changing ecosystem of libraries underlying the task templates.

2.2 Data Science Workflow is Exploratory

Another key feature of this workflow is that it is highly exploratory. At every stage of the workflow, the data scientist must choose the most suitable task and parameters from a large number of choices. Therefore, a simplified interface must respect this exploratory nature of the workflow and not constrain the data scientist to define the pipeline a priori. Further, there is an opportunity here to build an interface that is intelligent enough to make the search procedure more efficient by guiding the user to the most likely optimal choices at every stage. We believe that this exploratory aspect of the workflow is best served by an intelligent interactive agent.

3. INTRODUCING AVA

A key challenge in DS is the search problem of identifying the best pipeline for the business problem at hand. A plethora of languages, libraries and tools have simplified the individual tasks in the workflow. There are also tools available to build, validate, persist and manage pipelines. However, we believe that despite the remarkable advancements in this ecosystem, the data scientist is still left with the time-consuming steps of learning how to use the various tools (each with its own pros and cons) and gluing together tasks performed using these tools into a workflow, all while faced with the challenge of making decisions to navigate the search space of possible pipelines.

3.1 Overview and Example

Motivated by the arguments in Section 2, we introduce Ava, an intelligent chatbot that is aimed at simplifying this search process. It uses a natural language chat-based interface to abstract away the details of the underlying machine learning and data analysis/visualization environments, letting the user focus on the business problem. Ava interactively guides the data scientist through the process of deciding among the various tasks to be performed at every stage of the workflow. It does so using a conversation storyboard that mimics the decision-making process of a data scientist. Further, Ava automatically recommends tasks as well as presents statistical data and visualizations to aid the data scientist in the process of exploratory analysis. We envision that Ava will eventually manage data movement between tasks (orchestrating multiple subsystems such as databases and machine learning libraries) as well as between machines (intelligently spawning appropriate instances in the cloud on demand), thereby improving user productivity even further.

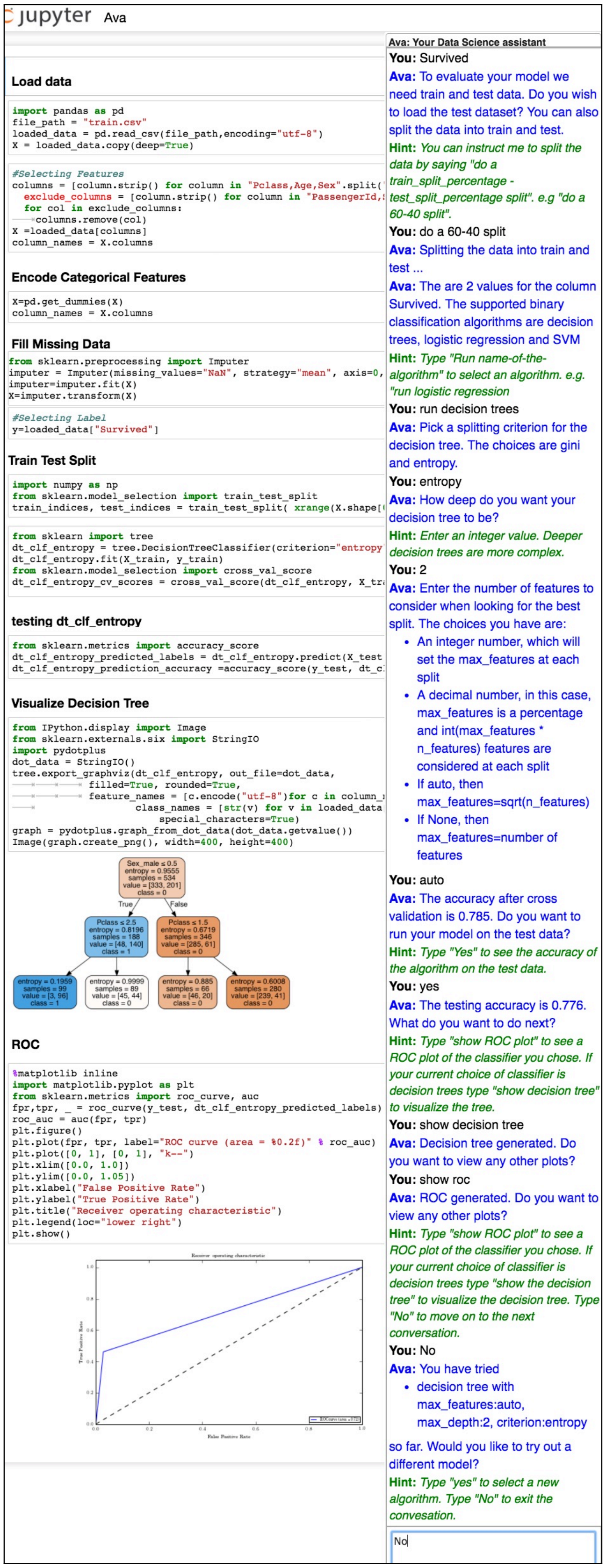
However, we acknowledge that the capabilities of such an interface will likely lag behind the fast-evolving libraries in the underlying ecosystem. To minimize any frustration on the part of the experienced data scientist, Ava *always* allows a user to seamlessly drop into a familiar programming interface (the Jupyter Notebook) at any stage in the workflow.

Throughout this process, Ava logs its interactions with the data scientist, producing a rich dataset that we will eventually compile into a growing knowledge-base for DS pipelines. We envision that this process of constantly learning from user interactions will make Ava more intelligent over time, helping data scientists find the optimal pipeline ever faster.

While we don't elaborate on this point further in this paper, we note that the chat logs provide a natural way to document the choices that are made. Chat logs are easier to read (compared to examining a stitched sequence of code), to review/revisit decisions, and to reuse portions of the decisions in building future pipelines.

Example 2. *We have used our current Ava prototype to execute the workflow in Example 1 in an initial user study. Figure 2 shows a timeline view of the resulting conversation for a sample user. Ava interactively guides Daisy through the successive stages of the workflow. At each stage, Ava uses natural language questions to clarify Daisy's intent and then executes the appropriately instantiated code template. Ava also proactively reports meta-information that might help Daisy take the next action, like automatically reporting the cross-validation accuracy after training a model. In future, Ava will also learn to make recommendations, such as proposing that a decision tree may be a more accurate model than logistic regression in a similar context.*

Note that all the interaction with Ava is through natural language. Internally, in this initial prototype, Ava loads data from a database or from a file into a Pandas data frame, uses the Pandas library [24] for data transformation, visualizes data using Matplotlib [15], and uses scikit-learn [12] for machine learning. (An initial implementation to use Tensorflow [9] is also in the works.) Ava seamlessly orchestrates all these systems and libraries together without requiring any such knowledge on the part of the user. In our current implementation, Ava can run the machine learning tasks either on the user's device (typically a laptop) or on a remote instance in our private academic cloud (which we can easily extend to public clouds like Amazon AWS or Microsoft Azure). Currently, Ava uses simple heuristics that we have encoded to make recommendations. In future, Ava will use statistical analysis of the datasets at hand to find relevant information in its knowledge base (consisting of expert knowledge, thumb-rules, and historical logs) to make its recommendations.



Load data

```
import pandas as pd
file_path = "train.csv"
loaded_data = pd.read_csv(file_path,encoding="utf-8")
X = loaded_data.copy(deep=True)
```

```
#Selecting Features
columns = [column.strip() for column in "Pclass,Age,Sex".split('
  exclude_columns = [column.strip() for column in "PassengerId,$
  for col in exclude_columns:
    columns.remove(col)
X =loaded_data[columns]
column_names = X.columns
```

Encode Categorical Features

```
X=pd.get_dummies(X)
column_names = X.columns
```

Fill Missing Data

```
from sklearn.preprocessing import Imputer
imputer = Imputer(missing_values="NaN", strategy="mean", axis=0,
imputer=imputer.fit(X)
X=imputer.transform(X)
```

```
#Selecting Label
y=loaded_data["Survived"]
```

Train Test Split

```
import numpy as np
from sklearn.model_selection import train_test_split
train_indices, test_indices = train_test_split( xrange(X.shape[(
```

```
from sklearn import tree
dt_clf_entropy = tree.DecisionTreeClassifier(criterion="entropy'
dt_clf_entropy.fit(X_train, y_train)
from sklearn.model_selection import cross_val_score
dt_clf_entropy_cv_scores = cross_val_score(dt_clf_entropy, X_tr
```

testing dt_clf_entropy

```
from sklearn.metrics import accuracy_score
dt_clf_entropy_predicted_labels = dt_clf_entropy.predict(X_test
dt_clf_entropy_prediction_accuracy =accuracy_score(y_test, dt_c
```

Visualize Decision Tree

```
from IPython.display import Image
from sklearn.externals.six import StringIO
import pydotplus
dot_data = StringIO()
tree.export_graphviz(dt_clf_entropy, out_file=dot_data,
                    filled=True, rounded=True,
                    feature_names = [c.encode("utf-8")for c in column_
                            class_names = [str(v) for v in loaded_data
                                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png(), width=400, height=400)
```

ROC

```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
fpr,tpr, _ = roc_curve(y_test, dt_clf_entropy_predicted_labels)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, label="ROC curve (area = %0.2f)" % roc_auc)
plt.plot([0, 1], [0, 1], "k--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic")
plt.legend(loc="lower right")
plt.show()
```

You: Survived
Ava: To evaluate your model we need train and test data. Do you wish to load the test dataset? You can also split the data into train and test.
Hint: *You can instruct me to split the data by saying "do a train_split_percentage - test_split_percentage split". e.g "do a 60-40 split".*
You: do a 60-40 split
Ava: Splitting the data into train and test ...
Ava: The are 2 values for the column Survived. The supported binary classification algorithms are decision trees, logistic regression and SVM
Hint: *Type "Run name-of-the-algorithm" to select an algorithm. e.g. "run logistic regression*
You: run decision trees
Ava: Pick a splitting criterion for the decision tree. The choices are gini and entropy.
You: entropy
Ava: How deep do you want your decision tree to be?
Hint: *Enter an integer value. Deeper decision trees are more complex.*
You: 2
Ava: Enter the number of features to consider when looking for the best split. The choices you have are:

- An integer number, which will set the max_features at each split
- A decimal number, in this case, max_features is a percentage and int(max_features * n_features) features are considered at each split
- If auto, then max_features=sqrt(n_features)
- If None, then max_features=number of features

You: auto
Ava: The accuracy after cross validation is 0.785. Do you want to run your model on the test data?
Hint: *Type "Yes" to see the accuracy of the algorithm on the test data.*
You: yes
Ava: The testing accuracy is 0.776. What do you want to do next?
Hint: *Type "show ROC plot" to see a ROC plot of the classifier you chose. If your current choice of classifier is decision trees type "show decision tree" to visualize the tree.*
You: show decision tree
Ava: Decision tree generated. Do you want to view any other plots?
You: show roc
Ava: ROC generated. Do you want to view any other plots?
Hint: *Type "show ROC plot" to see a ROC plot of the classifier you chose. If your current choice of classifier is decision trees type "show the decision tree" to visualize the decision tree. Type "No" to move on to the next conversation.*
You: No
Ava: You have tried

- decision tree with max_features:auto, max_depth:2, criterion:entropy

so far. Would you like to try out a different model?
Hint: *Type "yes" to select a new algorithm. Type "No" to exit the convesation.*

No

Figure 3: Ava in action. The right panel is the chat window. The conversation between the user and Ava is carried out in this chat window. Actual code is created and run in the left panel using the regular Jupyter Notebook interface.

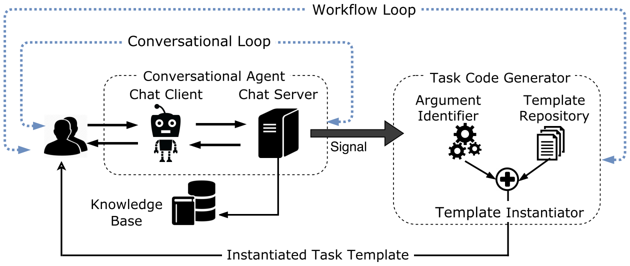


Figure 4: Overview of Ava's control flow architecture.

3.2 Client-Server Jupyter Notebook Design

The conversational nature of Ava naturally lends itself to using a notebook style of interaction. Ava currently uses the Jupyter Notebook environment. This notebook style also allows a data scientist to "take over" from Ava and switch to their current mode of operations. Thus, Ava is a productivity accelerator to existing data science workflows.

The Jupyter Notebook [26] is an interactive computing environment that is widely used by data scientists. As the user types in code (typically in Python or R), the code can be executed, and results, including tables and figures, are produced inline with the interface. Thus, a notebook can be produced with the code, intermediate results, and the final results. Juypter also has a client-server architecture. In the server side of the Jupyter Notebook framework, there is a programming language specific kernel. This kernel manages the interactive context and controls code executions that are triggered by the client. The client communicates with the kernel using a REST-based API. It is possible to run the client and the server on different machines. Ava taps into this existing architecture of Jupyter as described in more detail in Section 3.3.

At a high-level, Ava is built on the Jupyter Notebook framework. An extended IPython kernel serves as the chat server. The chat server is responsible for parsing the chat messages, extracting the semantic information about the task to be performed, identifying when further clarification is required, and finally generating the executable code. The Ava chat client has an instant messaging-like interface, and runs above the regular notebook client, as shown in the screenshot in Figure 3. The user interacts directly with the chat client, providing instructions and answering questions using chat messages. When the specification for some task is completed, the Ava chat server triggers Python code generation. After executing this code, Ava updates the Jupyter Notebook with both the code and its result. Thus, a user can take over from Ava at any point, and directly write code in the notebook. This aspect is crucial, as it allows advanced users to jump back to the usual mode of "driving" the data analysis process, minimizing frustration when the task at hand can't be done using Ava. (Although, at this point the one-to-one correspondence between a chat and the associated code is lost.)

3.3 Control Flow Architecture

Figure 4 shows Ava's control flow architecture. The user chats with the *conversational agent* in a Controlled Natural Language. The conversational agent consists of a *chat client* and a *chat server*, and matches the input with patterns in the AIML files (more details about how Ava uses AIML is presented below in Section 3.5). The conversational agent is responsible for steering the conversation towards a full task specification.

The chat server consists of a pair of processes, one that is the AIML interpreter (a Java process), and the other that is the iPython kernel.

The chat client sends the messages that are typed by the user to the IPython kernel, using IPython's REST-based communication API. The IPython kernel then passes the chat message that it received to the AIML interpreter (which is part of the chat server). The IPython kernel communicates with the AIML interpreter using JPype [5]. Once the AIML interpreter matches the chat message with a pattern in the AIML files, it sends a response back to the IPython kernel using JPype. The IPython kernel, then relays back the response to the Ava chat client using the kernel's REST-based API. The conversation between Ava and the user continues until a full task specification is reached.

Once a specification is complete, the system signals the *task code generator*, which in turn looks up its *template repository* to identify the template that best matches the task specification. Its *argument identifier* extracts the template parameters from the chat messages, and the *template instantiator* fills out the chosen template with the user-specified parameter values. The generated code is now executed on the underlying DS platform. Both the code and the results are also presented back to the user in the Jupyter Notebook client. The chat server logs all interactions to later be compiled into a *knowledge base*, and consults it to guide the workflow towards the optimal pipeline.

Note that the discussion above described how the template code generates Python code to run in the Jupyter Notebook client. This same mechanism can be used to call out to other machine learning platforms, by simply having the template code invoke APIs in the machine learning platform (e.g. TensorFlow), which can be run as a separate process. Ava is currently packaged as a Docker [25] image, which also allows for easy packaging and deployment of all the associated components.

3.4 Controlled Natural Language (CNL)

While natural language is highly expressive and capable of representing arbitrary concepts, this expressiveness comes with ambiguity and poses a challenge for automatic semantic inference. CNLs [29] provide a middle ground that balances the expressiveness of natural language with ease of semantic inference by restricting the vocabulary and grammar in the language. Thus a CNL constrains the input to be more concise, with a less general sentence structure, making it easier to parse and extract meaning from the input sentences. (CNLs have been used successfully in many domains, including air traffic control [30].)

The sentence structure for Ava's CNL is: `<ACTION VERB> <NOUN> <PREDICATE>`. For example, a data scientist can say *Run logistic regression with penalty l2 and solver liblinear* to instruct Ava to fit a logistic regression classifier. Here, "Run" is the `ACTION VERB`, "logistic regression" is the `NOUN` and "with penalty l2 and solver liblinear" is the `PREDICATE`. This simple sentence structure makes it easy for Ava to identify the action to be taken in response to the user's input as well as the parameters associated with the action.

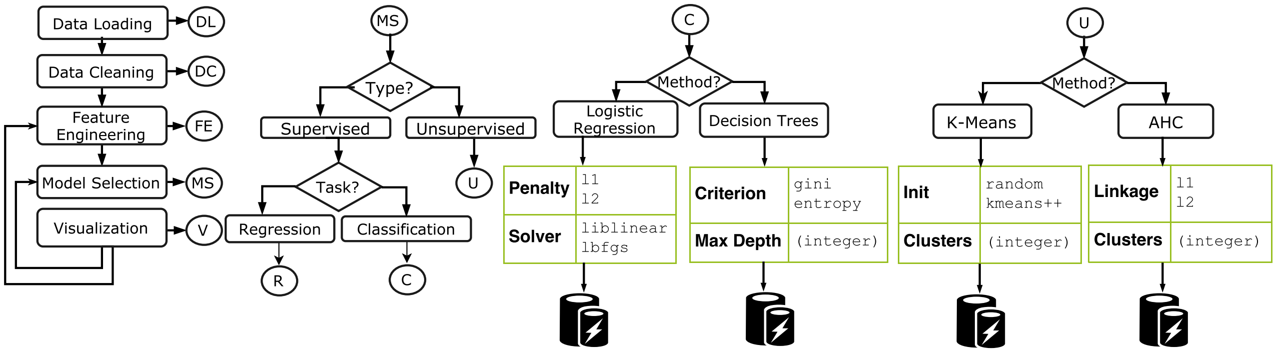


Figure 5: The storyboard powering Ava, excerpted and truncated for brevity.

3.5 Artificial Intelligence Markup Language

In addition to CNL processing, Ava has to maintain a conversational context. We briefly describe our approach here.

The conversational context is represented by a state machine, which has the following key elements:

- The current state of the conversation, e.g., *What stage of the workflow are we in? What dataset and model variables are in scope?*
- The allowable inputs that will trigger a state transition, e.g., *What can the user say at this stage in the workflow to choose a task or move to the next stage?*
- The corresponding actions that Ava must take when the user effects a state transition, e.g., *What code should be executed when the user asks to train a decision tree?*

The state machine is wrapped up with a layer that implements the actual chat/conversation. We use the popular XML dialect, Artificial Intelligence Markup Language (AIML) [34], to represent the state machine for our chatbot. Each unit of conversation in AIML, called a `category`, consists of a `pattern` that represents a possible user input and a `template` that represents an action that the chatbot can take. An example listing is shown below:

```
<aiml>
  <category>
    <pattern>HELLO</pattern>
    <template>Hi,there!</template>
  </category>
</aiml>
```

We have created various AIML files to script each possible interaction that Ava can have. We also use existing open-source libraries for parsing AIML, and running a conversational agent, using developer-defined callbacks for processing user input and effecting state transitions.

3.6 Task Template Repository

As we observed in Section 2.1, the code for executing a DS workflow task is highly templatized. In Ava, each such template is represented using a text file with placeholders for the various parameters that can be set. We maintain a repository of such templates, along with a mapping from specific tasks, such as *Model Training for Decision Tree*, to the corresponding code template file. These templates are specific to the underlying libraries or tools that Ava uses, and are manually created by us in this initial implementation. In the future we plan to augment it using static analysis techniques to automatically generate templates from the source code and documentation of the underlying library.

3.7 Conversation Storyboard

Ava's conversations with the user are guided through a *storyboard*, which is represented as a set of AIML files. The storyboard describes the dialogue between the data scientist and Ava in CNL, and the actions that Ava must take in response. These responses are often questions that Ava must ask the user in order to disambiguate their intent and arrive at a complete task specification.

Figure 5 visualizes an excerpt of the storyboard in the form of a set of flowcharts. The first flowchart is a high-level view of the different stages of the DS workflow. Corresponding to each stage, the figure shows a separate sub-flowchart. The *Model Selection (MS)* flowchart describes how Ava uses multiple prompts to identify whether the user is interested in a classification, regression or clustering method. The flowcharts for *Classification (C)* and *Unsupervised Learning (U)* further detail how Ava talks to the user to identify the specific method (say *Decision Trees*) and its parameters (information criterion and maximum tree depth). When the task specification is complete, the storyboard also instructs Ava to load the corresponding task template and execute it.

A particular workflow therefore plays out as a conversation with Ava, which is modeled as a walk through the flowchart graph. Note that the storyboard is also a template for the entire DS workflow (in contrast to the task-specific template library described in Section 3.6). Over time, as Ava grows its knowledge base, we envision that much of the storyboard will be automated, reducing the need for user prompts. For instance, Ava may eventually learn what decision tree hyper-parameters to use, so that those prompts in the storyboard are replaced by lookups in the knowledge base.

3.8 Salient Aspects

The finite state machine (FSM) framework is crucial to Ava's ability to unambiguously convert from conversations in CNL to executable code. The FSM states, through the use of CNL, make interpretation of the conversation tractable and unambiguous. The FSM transitions allow Ava to drive the conversation towards a complete task specification. Finally, the task templates allow Ava to generate and execute the code for a task, decoupling underlying libraries from the chat engine.

Ava's modular architecture allows implementation of each component using the appropriate technologies. This modularity also allows us to easily extend Ava's capabilities, such as adding new machine learning methods or libraries.

4. EXPERIMENTS

We designed single-subject and multiple-subject trials to analyze the efficiency of Ava in constructing data science pipelines. We present our experimental design and results in this section.

4.1 Single-Subject Trial

We conducted this experiment with a single user playing the role of Daisy in the DS workflow in Example 1. We chose the specific Kaggle task [7] where the goal is to identify customers for direct marketing. For training, we used a 15% sample of the 1GB customer dataset, consisting of 25000 rows and 1934 columns. This experiment was run on a MacBook with a dual-core Intel i7 processor and 8GB of memory. Ava was configured to generate code using the Pandas, Matplotlib and scikit-learn libraries. Our current Ava prototype has a small number of hand-written templates in its template repository. The knowledge base consists of a set of hand-written rules providing user assistance (including recommendations) based on prefix-based exact matching of the state of the current conversation. Figure 2 visualizes the conversation flow in the form of a timeline with timings measured from our user study for an actual user. Figure 3 shows a screenshot of the interaction with Ava. All the interaction between Ava and the user occurs through chat messages in (controlled) natural language, although Ava also presents the generated code and graphs in the Jupyter Notebook interface alongside the conversation window.

A key objective in this experiment was to study Ava's *response time*, which we define as the time it takes for Ava to respond to a given user request, not including any time spent in executing generated code for a task. Our measurements showed that the average response time was 93 ms, whereas the maximum was 558 ms. Thus, we believe that our CNL/AIML framework in Ava has fast response times, which is crucial for a chatbot framework. We note that the total time to assemble the final DS pipeline in this case was 12 minutes (this includes the time to actually run the machine learning models in scikit-learn). Of these 12 minutes, only 2 seconds were spent in Ava's client-server engine. We expect that novice users faced with this DS problem would likely spend many more minutes putting together the same pipeline by writing code. Additionally, since an expert user can always drop down to the notebook programming interface, Ava does not significantly impair their productivity even in the worst case (2 seconds in this case). We believe that Ava's proactive presentation of meta-information and recommendations would likely improve productivity for even the experts.

4.2 Multiple-Subject Trials

Inspired by the results of the single-subject trial, we conducted a more detailed study to test our hypothesis that Ava significantly improves productivity of its non-expert users, with only a small learning curve. In this study, we asked users to complete the same data science task using both Ava as well as Python packages, and measured various metrics of productivity.

Participants. We recruited graduate students from the Computer Sciences Department at the University of Wisconsin-Madison for this experiment. All students in this user group had exposure to machine learning and data science through their coursework, but were not otherwise experienced in these domains. We administered a Python proficiency test and picked 16 participants who passed this test.

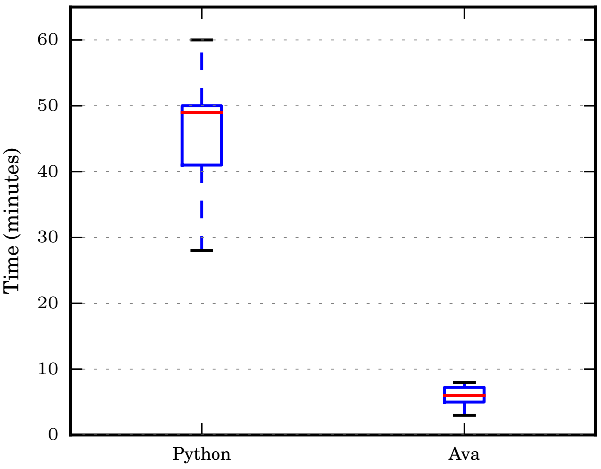


Figure 6: Distribution of the time taken by participants to complete the first model.

Task and data description. The task that we chose for this study was one of predicting customer churn for a telecommunications firm [4]. The objective of the task is to develop a model to classify which customers are most likely to refrain from renewing their contract when it ends. We obtained a dataset consisting of historical details and account usage information for customers of the telecommunications firm, along with labels that indicate whether or not they renewed their contract. These labels, however, were skewed in that only 433 of the 3333 customers had terminated their contract. We sampled a subset consisting of 2300 customer records and provided it as training data to the participants. The remaining 1033 customer records were used as a held-out test set to evaluate the models developed by the participants. All participants used the same machine, which was a MacBook with 2.8 GHz Intel Core i7 processor and 16GB of memory.

Each participant performed this task once using Ava and once using the Python packages (pandas and scikit-learn). In both cases, the participants were required to follow a 10-step procedure to develop a data science pipeline. The pipeline included steps for feature selection, model training, cross-validation, and visualizing data that described the accuracy of the constructed model(s). For simplicity, we limited the choice of machine learning algorithms to logistic regression, support vector machines and decision trees. We limited the duration of the task to 30 minutes when using Ava and to 60 minutes when using Python. Participants were instructed to explore as many models (combination of learning algorithm, features and hyper-parameter choices) as they could within this duration. They were free to use any documentation, tutorials or code samples from the Internet to help them complete the task. Note that the participants were given no training or instruction on the usage of Ava.

More details about the dataset as well as the exact task description are available at [2, 3].

Results: Productivity Improvement. We used three different metrics to measure productivity. These metrics are: a) the time taken to complete the first model, b) the number of models that were constructed in a fixed amount

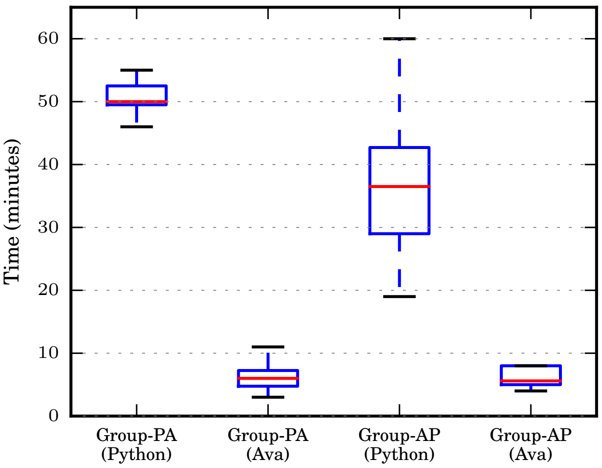


Figure 7: Distribution of the time taken by participants to complete the first model.

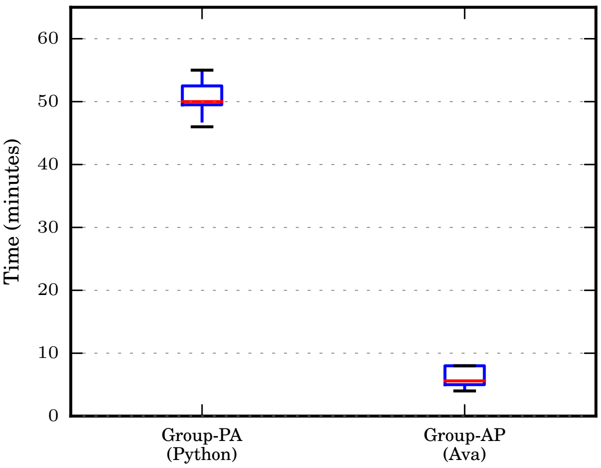


Figure 8: Distribution of unbiased task completion times.

of time, and c) the time taken to complete successive models after the first model was constructed. As a overview of our findings (described in detail below), with Ava we saw big productivity improvements in each of these three metrics.

Note that all time measurements include the actual processing time for the analytics steps (within the underlying machine learning and the visualization libraries), in addition to the time spent by participants in manually proceeding through the steps in the analysis task.

First, we consider the time taken by the participants to train their first working machine learning model. The distribution of the results using this metric is shown as box plots [31] in Figure 6. Of the 16 participants, only 13 were able to complete their first model using Python within the allotted task duration of 60 minutes. On average, these participants took 45 minutes to do so. On the other hand, all 16 participants completed their first model using Ava within 14 minutes, with an average time of about 7 minutes. Thus, we observed a improvement of more than 6× in this productivity metric.

Second, we consider the number of models that the participants were able to train within the allotted duration using each approach. Only one of the participants was able to train more than one model using the Python packages within 60 minutes. On the other hand, every participant was able to explore at least 7 models using Ava, and on average, each participant explored about 11 models within the allotted 30 minutes. Thus, Ava produced at least an order of magnitude improvement in productivity by this metric.

Finally, we note that the participants spent less than 3 minutes training each successive model after the first one using Ava. This time was about half a minute faster than the average time per successive model using Python for the single participant that completed training of more than one model.

Results: Learning Curve. Of the 16 participants in the study, half were randomly chosen to perform the task using Python packages before doing so with Ava (we denote this group using the label *Group-PA*). The remaining half were asked to perform the task using Ava before doing so with Python (denoted as *Group-AP*). Comparing the productivity differences between these groups using the two approaches allows us to reason about the learning curves with these two approaches. The distributions of the time taken to train the first model for these two groups are shown in Figure 7.

When performing the task using Python packages, the 8 participants in Group-PA (who used Python before Ava) took, on average, 51 minutes to train their first model. On the other hand the 8 participants in Group-AP (who used Ava before Python) took 7 minutes to train their first model. Figure 8 compares the time taken by the participants to train the first working model in Python and Ava, when they did not have any prior knowledge of the task.

Also, familiarity with the task as well as the data science process (by completing the task in Ava first) allowed the participants in Group-AP to take only 27 minutes to reach the same milestone. Contrast this 14 minute (27%) improvement in productivity with the negligibly small 1 minute improvement (from 7 minutes down to 6 minutes) to train the first model using Ava in Groups-AP and -PA respectively. Further, we saw only a negligibly small difference in the average number of models trained by the participants using Ava in the two groups.

We consider these observations as indicating that novice users can be nearly equally productive with Ava regardless of their prior experience in or familiarity with the data science process. We cannot, of course, extend these conclusions directly to expert data scientists since they were not part of this study.

Other Observations. Since we held out a subset of the labeled data from the participants in the study, we were able to compare the accuracy of their models on this (unseen) test set. A surprising observation was that the "best" model trained by participants using their cross-validation accuracy scores often had far lower accuracy scores on the test set. Clearly, empowering users with the ability to explore a large number of models in a short time had the unintended consequence of encouraging them to create highly over-fitted models. Furthermore, nearly all the participants seemed to have overlooked the skew in the labels (class frequencies) in the datasets, thereby obtaining models that had much higher prediction accuracies for one class than for the other.

These and other observations indicate that, despite data science training and some amount of hand-holding, users can still make mistakes that impair the accuracy and usability of the DS pipelines that they create. A data science workflow tool such as Ava must be built with sanity checks and knowledge of best practices to guide non-expert users away from such mistakes. In particular, we are looking to add checks against over-fitting as well as detection of class frequency imbalance into our Ava storyboard to address the specific issues that we observed. We note that this aspect of providing expert guidance or cautionary flags is an additional interesting aspect of Ava. Analogous to how automation helps drivers of cars with autonomous features, with Ava we can build safe-guards that warns its user when the user starts to tread into an “unsafe” territory.

5. RELATED WORK

There has been significant research in understanding natural language, dating back to the “Imitation Game” proposed by Turing in 1950 [32]. The quest to demonstrate an Artificial Intelligence (AI) capable of behaving indistinguishably like humans spurred research into early chatbots such as ELIZA [36], PARRY [13] and ALICE [35]. The Loebner Prize competition [6], conducted annually since 1991, was the first formal instantiation of a Turing test.

In recent years, the explosion of mobile web usage as well as improvements in natural language parsing have led to a resurgence in chatbots. These chatbots are designed with much more pragmatic ends in mind, such as virtual assistants that help book flight tickets or order pizza delivery. Many recent language understanding services such as Microsoft LUIS [37],Watson Conversation [8] and Amazon Lex [1] are designed to ease development of such chatbots. Due to their closed-source nature and licensing restrictions however, we chose to build our research prototype Ava using the open-source AIML [34] chatbot framework instead.

An important observation is that practical conversational interfaces can function reasonably well while understanding only a restricted subset of natural language. Such a “controlled natural language” (CNL) removes ambiguity in interpretation, thereby easing development of such interfaces. This approach has therefore found increasing use, including in air traffic control [18, 30]. Some CNLs can also be mapped to first-order logic for unambiguous interpretation (e.g. [14, 29]), and we build on this insight to control Ava’s chat language.

There has been a large body of research aimed at simplifying data access for non-programmers through the use of natural language, dating back to late 1960s. We direct the interested reader to the comprehensive survey [10] of natural language interfaces for databases (NLIDBs) for a review of the progress made over several decades. In the early NLIDBs, the acceptable input query grammar was usually tailored to the specific data schema, reducing their general applicability. More recent work, including Microsoft English [11], Precise [27], Nalix [22], Nalir [21], Athena [28] and Nauda [20], has sought to overcome this shortcoming by translating to SQL in a generic fashion. While systems such as Microsoft English [11] and Athena [28] rely on a semantic model of the database to correctly interpret natural language queries, others like Precise [27] and Nalir [21] use only the names of database elements and their synonyms. Most recently, EchoQuery [23] demonstrated a conversational, speech-based interface. Ava too advocates using natural language as the interface to data systems. We believe research on NLIDBs to be complementary to our own, since we target the broader data science domain rather than the database interface.

There is broad recognition that various aspects of DS pipelines must be optimized (e.g. [16, 19, 33]), and we have a similar overall goal. We advocate a specific approach for part of this problem using chatbots; many other aspects in previous work, such as smarter parameter selection, are complementary with our proposed approach.

6. CONCLUSIONS AND FUTURE WORK

The true democratization of data requires dramatic simplification of the methods that are used to extract insights from data. In this paper we present a small step in this direction. We present the initial version of Ava, which targets improving the productivity of data scientists. Ava uses controlled natural language as the “programming” interface, and has an architecture for building a knowledge base that can improve the overall productivity in performing data analysis. There are numerous open research challenges even in this initial version, including managing and mining the knowledge base, and even larger issues in making the Ava approach accessible to non-data scientists.

7. ACKNOWLEDGMENTS

This work was supported in part by grants from the National Science Foundation (NSF) under grants IIS-1250886 and IIS-0963993, and by the National Institutes of Health (NIH) under grant U54AI117924. We also thank CloudLab (https://www.cloudlab.us, NSF grant CNS-1419199) for providing part of the computational infrastructure that was used for this work. We thank the anonymous graduate students for participating in the user study that is described in this paper. We also thank Jean-Michel Ané, Victor Cabrera, Mark Craven, Pamela Herd, David Page, Federico E. Rey, Garret Suen, and Kent Weigel for providing valuable feedback on earlier versions of this paper.

8. REFERENCES

[1] Amazon lex. https://aws.amazon.com/lex/, 2016.
[2] Churn analysis using ava. http://go.wisc.edu/784l5o, 2016.
[3] Churn analysis using python. http://go.wisc.edu/h2ex60, 2016.
[4] Customer churn prediction. https://www.knime.org/knime-applications/churn-prediction, 2016.
[5] Jpype - java to python integration. http://jpype.sourceforge.net/, 2016.
[6] Loebner prize. http://www.loebner.net/Prizef/loebner-prize.html/, 2016.
[7] Springleaf marketing response. https://www.kaggle.com/c/springleaf-marketing-response/, 2016.
[8] Watson conversation. https://www.ibm.com/watson/developercloud/conversation.html, 2016.
[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng.

Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[10] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[11] A. Blum. Microsoft english query 7.5: Automatic extraction of semantics from relational databases and olap cubes. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 247–248, 1999.

[12] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[13] K. M. Colby. *Artificial Paranoia: A Computer Simulation of Paranoid Processes*. Elsevier Science Inc., New York, NY, USA, 1975.

[14] N. E. Fuchs, S. Höfler, K. Kaljurand, F. Rinaldi, and G. Schneider. Attempto controlled english: A knowledge representation language readable by humans and machines. In *Proceedings of the First International Conference on Reasoning Web*, pages 213–250, 2005.

[15] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science and Engg.*, 9(3):90–95, May 2007.

[16] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *Sixth Biennial Conference on Innovative Data Systems Research (CIDR)*, 2013.

[17] M. Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(1):1–26, 2008.

[18] T. Kuhn. A survey and classification of controlled natural languages. *Comput. Linguist.*, 40(1):121–170, Mar. 2014.

[19] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *SIGMOD Rec.*, 44(4):17–22, May 2016.

[20] D. Küpper, M. Storbel, and D. Rösner. Nauda: A cooperative natural language interface to relational databases. *SIGMOD Rec.*, 22(2):529–533, June 1993.

[21] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, Sept. 2014.

[22] Y. Li, H. Yang, and H. V. Jagadish. Nalix: An interactive natural language interface for querying xml. In *Proceedings of the 2005 ACM International Conference on Management of Data (SIGMOD)*, pages 900–902, 2005.

[23] G. Lyons, V. Tran, C. Binnig, U. Cetintemel, and T. Kraska. Making the case for query-by-voice with echoquery. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, pages 2129–2132, 2016.

[24] W. McKinney. pandas: a foundational python library for data analysis and statistics.

[25] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), Mar. 2014.

[26] F. Perez and B. E. Granger. Ipython: A system for interactive scientific computing. *Computing in Science and Engg.*, 9(3):21–29, May 2007.

[27] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI)*, pages 149–157, 2003.

[28] D. Saha, A. Floratou, K. Sankaranarayanan, U. F. Minhas, A. R. Mittal, and F. Özcan. Athena: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.*, 9(12):1209–1220, Aug. 2016.

[29] R. Schwitter. Controlled natural languages for knowledge representation. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters (COLING)*, pages 1113–1121, 2010.

[30] D. Troegner. Grammar for natural language recognition: Adaptation to air traffic phraseology. 2011.

[31] J. W. Tukey. *Exploratory Data Analysis*. Behavioral Science: Quantitative Methods. Addison-Wesley, Reading, Mass., 1977.

[32] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[33] M. Vartak, H. Subramanyam, W.-E. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. Modeldb: A system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*, pages 14:1–14:3, 2016.

[34] R. Wallace. The elements of aiml style. *Alice AI Foundation*, 2003.

[35] R. S. Wallace. *The Anatomy of A.L.I.C.E.*, pages 181–210. Springer Netherlands, Dordrecht, 2009.

[36] J. Weizenbaum. Eliza – a computer program for the study of natural language communication between man and machine. *CACM*, 9(1):36–45, Jan. 1966.

[37] J. Williams, E. Kamal, M. Ashour, H. Amr, J. Miller, and G. Zweig. Fast and easy language understanding for dialog systems with microsoft language understanding intelligent service (luis). In *Proceedings of 2015 SIGDIAL Conference*, 2015.

[38] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In S. D. Gribble and D. Katabi, editors, *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, 2012.