COM3600
Building Interactive Machine Learning Visualizations

Vasily Shelkov
Supervisors: Neil Lawrence, Mike Smith

This report is submitted in partial fulfilment of the requirement for the degree of BSc in Computer Science with Year of Employment Experience by Vasily Shelkov

University of Sheffield
Faculty of Computer Science
May 4th, 2016

Signed Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name:.....

Signature: .....

Date:.....

Acknowledgments

Without the help and excellent guidance of my project supervisors Neil Lawrence and Mike Smith, this project and my newfound knowledge in machine learning would be non existent.

Without my fellow coursemates and friends, I would not have had the stimulating discussions, the sleepless nights and all the fun we have in the past year.

Without my family, I would not have had the unconditional support throughout the BSc and in my life.

For that I thank you all.

Abstract

Machine learning is becoming a real thing of science fiction in the public's eyes. Recent media reports have caused misunderstanding and, in some cases, fear about what the outcomes of machine learning could become. Also, possibilities of machine learning have frequently been misrepresented.

'Cold reading', in essence, is the statistical analysis of basic information about a given person to produce insights that are revealing to them. A prototype application called the Scikic already exists which acts as a cold reader using a Bayesian Network and Collaborative Filtering.

The following project will look at visualizing the current Scikic's machine learning techniques. This will be done by implementing a more effective conversational interface than the prototype, where each question answered will visually represent how machine learning is being used to produce personal insights for any of its users.

The final implemented application can be found at http://scikic.org/.

Contents

List of Figures vi

1 Introduction 1
- 1.1 The Current Scikic(psychic) . . . 2
- 1.2 Motivation behind building a web interface . . . 3
- 1.3 Overall Aim . . . 3
- 1.4 Report Overview . . . 3

2 Literature Review 6
- 2.1 Machine learning visualizations . . . 6
  - 2.1.1 Bayesian networks . . . 6
  - 2.1.2 Collaborative filtering . . . 8
  - 2.1.3 Available visualization libraries . . . 10
  - 2.1.4 Conclusion . . . 12
- 2.2 Conversational user experience . . . 13
  - 2.2.1 What is a conversation? . . . 13
  - 2.2.2 How to help the conversation be successful . . . 13
  - 2.2.3 Solutions of conversational interfaces . . . 14
  - 2.2.4 Conclusion . . . 17

3 Requirements and Analysis 18
- 3.1 How does the Scikic visualize data? . . . 18
  - 3.1.1 Current data sources . . . 18
  - 3.1.2 The Scikic's required input data . . . 19
  - 3.1.3 Information about the Scikic's baysian network from returned inference data . . . 20
  - 3.1.4 Most appropriate visualisation library . . . 21
  - 3.1.5 Conclusion . . . 21
- 3.2 How the Scikic talks to users . . . 22
  - 3.2.1 Scikic's prototype conversational interface key features . . . 22
  - 3.2.2 Comparison to other conversational interfaces . . . 25
  - 3.2.3 Conclusion . . . 26
- 3.3 Final requirements . . . 27
  - 3.3.1 Visualization module . . . 27
  - 3.3.2 Rebuilding the prototype conversational interface module . . . 28
- 3.4 Evaluating the Scikic's web interface . . . 28

4 Design 29
- 4.1 Visualization module design . . . 29
  - 4.1.1 Choosing a final design . . . 33
- 4.2 Conversational interface module design . . . 34
  - 4.2.1 Final design . . . 36
- 4.3 Combining the visualizations and conversational interface modules . . . 37
  - 4.3.1 Final layout design . . . 38
- 4.4 Designing the user's possible interactions with the new web interface . . . 39

5 Implementation and Testing 41
- 5.1 Implementing the final design with components using ReactJS . . . 41
  - 5.1.1 InfoPages Component . . . 42

5.1.2 Chat Component . . . 43
5.1.3 Visualizations Component . . . 44
5.1.4 Using D3 in React for the Visualization component . . . 45
5.2 Implementing the user interactions with ReduxJS . . . 46
5.2.1 Chat module's Actions and how they affect its Reducer . . . 47
5.2.2 Visualization module's Actions and how they affect its Reducer . . . 48
5.2.3 Conclusion . . . 49
5.3 Keeping the code quality high during development . . . 49
5.4 Testing . . . 49
5.4.1 Verifying the data's is in the correct format using automated unit tests . . . 49
5.4.2 Analysis of the user testing . . . 51
5.4.3 Conclusion . . . 53

6 Results and Discussion **54**
6.1 Different approaches to integrating D3 and React . . . 54
6.2 Did the project meet all of its aims and requirements? . . . 55
6.2.1 Visualization module's requirement analysis . . . 56
6.2.2 Conversational interface module's requirement analysis . . . 56
6.2.3 Conclusion . . . 57
6.3 What has visualizing the Scikic's data achieved ? . . . 57
6.4 The future of the Scikic . . . 58

7 Project Conclusion **59**

References **61**

List of Figures

1.1 Machine Learning Trend 1

1.2 Prototype Scikic conversational web interface 2

2.1 Bayesian Network establishing relations between events on the burglary-earthquake-alarm domain, together with complete specifications of all probability distributions[81]. 7

2.2 An example of different items (like videos, images, games) user ratings using collaborative filtering and wanting to know a given user's likely rating[63]. 8

2.3 An example of predicted of the user's rating using collaborative filtering[63]. 8

2.4 An example of a Bayesian Network representation of Collaborative Filtering[63]. 9

2.5 Comparison of the popular opensource JS data-driven visualisation repository by number of stars it has. (as of 06/12/2015) 10

2.6 Comparison of the commits/issues/releases and contributors between D3[19] and Raphael[73] (as of 01/05/2016) 11

2.7 An extreme example of how the form reflects the way we converse making interaction with it more natural by engaging with the user[41]. 14

2.8 Facebook's chat typing awareness indicator[79]. 15

2.9 WhatsApp's chat interface[99]. 15

2.10 Skypes chat interface[56]. 15

2.11 Dominos pizza tracking interface[21]. 16

2.12 Typeform's typing indicator where each letter is typed individually[93]. 16

2.13 Deliberate "thinking and pointing..." screen despite actually being instantaneous[70]. . . 17

3.1 Scikic initial screen with option to Facebook login and description of the application. . . 22

3.2 Scikic's free flowing answer format. 23

3.3 Scikic's selection based answer format. 23

3.4 Scikic's fading away previous history. 24

3.5 Scikic's insights after performing inference. 24

3.6 Scikic checks the accuracy of its insights to improve its accuracy for future users. 25

3.7 Scikic asks users to choose what to do with their data. 25

4.1 A simple BN which the alternative designs will be based on. 30

4.2 Visualizations module alternative design 1 31

4.3 Visualizations module alternative design 2 32

4.4 Visualizations module alternative design 3 33

4.5 Scikic's Skype like profile. 34

4.6 Proposed Facebook login after a user has shown interest in using the Scikic. 35

4.7 Scikic's Skype like profile. 35

4.8 Basic menu design for the Scikic. 37

4.9 UML diagram showing the possible user interactions from entering the interface to showing the user's insights 39

5.1 The new interface overlayed showing what components it includes 42

5.2 All the different info pages which contain extra information about the Scikic. 42

5.3 The components in the implemented conversational interface otherwise known as the Chat Component. 43

5.4 The InitialQuestion component rendered in the QuestionList component asking the user "Interested?". 44

5.5 The component that is rendered if the user chooses they are not interested in using the Scikic. 44

5.6 The components in the implemented Visualizations Component. 45

5.7 Data in the Flux architecture flows in a single direction.[33] 46

5.8 Unit tests in the web interface run by Mocha pass successfully. 50

5.9 The 10 questions and answer results from the user testing using `https://www.surveymonkey.com`. 51

1 Introduction

Throughout the late 20th and 21st century, popular culture such as the "Terminator" series and the more recent film "Ex-Machina" have portrayed machine learning as something that will outgrow the human race and take over the world[102]. Even leading influential figures in the industry such as Stephen Hawking have said "Computers will overtake humans with AI at some point within the next 100 years" [12]. As a result, Hollywood especially, has a habit of depicting machine learning as a threat. In reality, machine learning is a tool that you can choose to accept and put to use[91]; it is something that extends your capabilities, not the machine's.

This has partly been caused by machine learning becoming a more popular buzzword over time. What "most people" are not aware of is that, in 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed"[87]. *Figure 1.1* shows how the field has grown since 1983 by comparing the total scholarly articles per year containing "machine learning". Despite having early roots, it has only been in the general public's awareness since the 21st century.

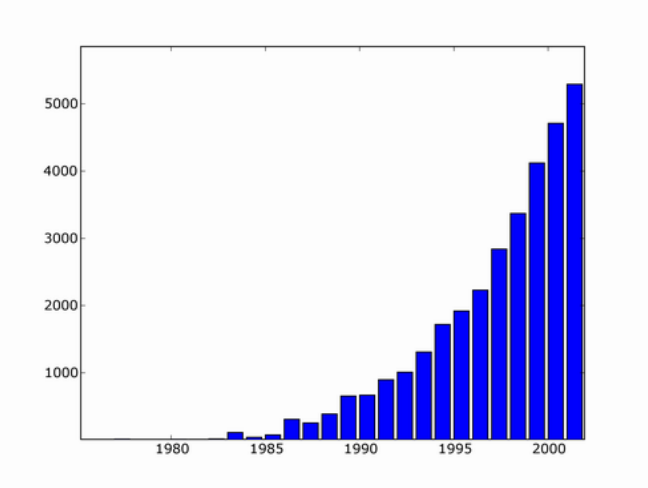


Figure 1.1: Number of publications containing the phrase "machine learning" by year, indexed by Google Scholar[10]

Machine learning is intertwined with our daily lives more than people realise. When people open their emails, "Machine learning techniques now[a]days are used to automatically filter the spam e-mail at a very successful rate"[26]. One of the most used social networks and websites in the world, *Facebook*, features posts on your news feed based on your closest friends, likes, and a long list of other interactions that you have had within the network[22]. There is a survey showing that since May 2015, "two-thirds of Americans are not completely averse to self driving cars"[50]. The survey suggests to us that many people who do have concerns about machine learning do not necessarily realize that self-driving cars are driven using machine learning!

The Scikic has been created to attempt to resolve the misconception that popular culture has created about machine learning.

This project will look at implementing a flexible and extensible chat interface web application to replace Scikic's proof of concept web interface(see figure 1.2) and augment it by visualizing the Scikic's Bayesian

Network that produces insights about the user.

1.1 The Current Scikic(psychic)

A psychic is defined in the Cambridge dictionary as: "a person who has a special mental ability, for example being able to know what will happen in the future". Following on from that, some psychics use a technique called "Cold Reading" which suggests to the psychic more information about the person than the psychic actually knows[24]. The general idea of cold reading, is that the psychic gets some basic information such as name, age or birth town about a person and from that, performs some practiced subconscious statistical analysis which suggests further insights about the given person. It is subconscious to make the insights seem like "special mental ability".

From the concept of a psychic, the Scikic's name was born for this application. The core of the Scikic is a stateless web Application Programming Interface (API). A web API is an open communication channel where web interfaces can request work to be done and information sent to the web API through a web URL. The Scikic's API gets sent a list of questions and answers about a user from any authorized web interface. The list then produces insights using machine learning pattern recognition techniques (Bayesian network and collaborative filtering) on open data sets such as the UK Census. The Scikic is stateless because the insights it returns are entirely based on the list of answered questions the API gets sent and does not rely on anything previously sent by the user. This makes the web API flexible to produce insights in different ways. For example by sending just one answered question at a time, or ten questions all at once.

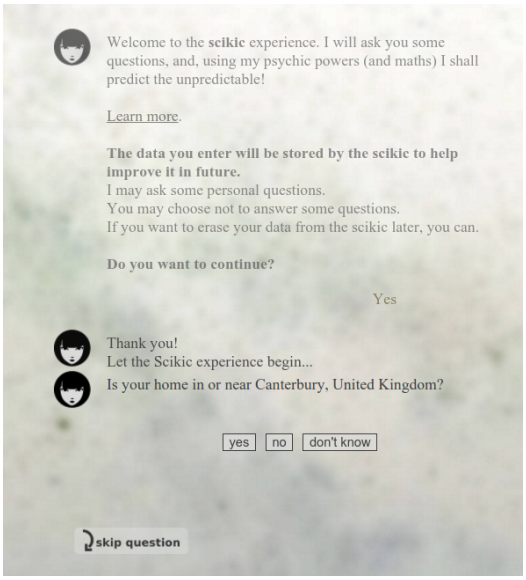


Figure 1.2: The prototype interface available at http://scikic.org

The insights are interesting for the user because it is based on their personal answers which changes the insights depending on those answers. Personalized insights are an external reward to users. This is also known an extrinsic motivation[82] and tries to encourage users to engage with the Scikic. Extrinsic motivation has proven results of online engagement in the past such as how gamers decoded AIDS protein that stumped researchers for 15 years in just 3 weeks [49].

The project is in collaboration with a company called *CitizenMe*. They would like to use the resulting application without the visualizations as a method of on-boarding users to their application, replacing the traditional account registration process. This suggests that the conversational interface and visualizations need to be independent of each other to meet *CitizenMe*'s needs.

1.2 Motivation behind building a web interface

To interact with the Scikic API, there is a prototype proof of concept web interface(*figure 1.2*). The interface is used to answer questions which creates the list of answered questions that are inputted into the stateless Scikic API. Once complete, the Scikic API returns insights based on the answered questions and the web interface displays the produced text insights.

While there is already a web chat interface application to interact with the Scikic API, it was built with the intention to prove the concept of the Scikic. The project's code reflects that the web interface is a prototype and therefore has not been built to be extensible and is difficult to improve the interface's features. This is the main motivation to rebuilding the chat interface using proven concepts by the industry.

Even though people are talking about machine learning frequently and most likely using it on a daily basis, there are few interactive educational tools available that help people understand the possibilities of machine learning better. The Scikic uses machine learning techniques on every question the user answers, but the current interface shows no indication of how it produces those insights to help people's understanding.

1.3 Overall Aim

There are two main aspects of the web interface that this project will design and implement:

1. A conversational interface that engages the user. The prototype has been a successful proof of concept but the code base is not extendable to improve its features. The project will aim to rebuild a strong code base foundation to be able to easily extend and change the new conversational interfaces functionality.

2. To show real-time visualisations of the machine learning techniques being used by the Scikic which returns personalized insights about the user. This is to try and help people's understanding of machine learning.

1.4 Report Overview

The project has two main aims, the report's chapters will reflect this by being broken down into two parts:

1. Displaying machine learning visualizations based on how the Scikic API works.

2. Rebuilding the chat interface to improve the current prototype interface

Literature Review
The literature review will help gain an understanding of needed concepts for the new web interface's implementation by surveying:

1. What Bayesian networks are and how collaborative filtering works. Based on that knowledge, what technologies could be appropriate to attempt to implement visualizations for these machine learning techniques.

2. Explore what a conversation actually is and what criteria is there to make a conversational user experience effective. Using that criteria, research how some of the industry's web applications go beyond the criteria to help design an effective conversational interface for the Scikic.

Requirements and Analysis
An analysis of the current Scikic's web interface with the aim of realizing the requirements to improve the prototype's web interface and how to visualize machine learning techniques. This is intended to help come up with designs for these two domains in the next chapter.

1. Since the Scikic API has already been built, there is a need to understand what data sources are used by the Scikic to produce insights, what input does the Scikic stateless API expect to be able to produce the insights and what data the Scikic API returns. Does the Scikic return all the necessary data required to create visualizations in the new web interface? Based on the research in the *Literature Review* on the possible visualization libraries, assess, and justify what visualization library would be most appropriate for this project.

2. Understanding how the current Scikic web interface works and comparing it to the conversational interfaces that were found in the *Literature Review* to discuss which features would be required to make the prototype's conversational interface more effective.

Design
Based on the identified requirements, discuss potential alternative designs and choose a final design to implement for:

1. The Visualization module.

2. The Conversational interface module.

Finally, design and explain a unified modeling language activity diagram to show the dynamic behaviour of the new web interface and the possible user interactions which meets all the requirements for the two modules.

Implementation and Testing
Present the key technologies that were chosen to implement the new web interface from the proposed final design:

- How ReactJS is used to structure and encapsulate the project's two modules into 'components'.

- How ReduxJS is used to implement the unified modeling language activity diagram, representing the user interactions with the new web interface from the *Design* chapter.

- The development tools that have been used to ensure the highest quality of code, making the new web interface's code base extendable and readable.

Finally, the two types of testing that were used which includes unit testing to make sure that the user interactions in the project's new web interface function as expected and user testing. User testing involves asking an independent user group ten questions to both evaluate that the Scikic works but also to get ideas for future improvements that could make the new web interface more effective in helping users understand Bayesian networks.

Results and Discussion
The chapter starts by discussing how the best approach to integrating D3 with ReactJS was discovered for the new web interface's Visualization module. Followed by an analysing the number of requirements and aims that have been achieved by the project. The analysis includes which requirements were implemented, and out of those, which of them were verified successfully with tests. There is then a

discussion around what being able to visualize the Scikic's machine learning techniques has actually achieved. Finally, what direction is the Scikic likely to follow in the future.

Final Conclusion
A final overview of the project's final outcomes and achievements. What has the Scikic gained from this project's results and how will they drive the Scikic forward in progressing its motivations.

2 Literature Review

The *Literature Review* is designed to give an understanding of the two different problem domains, as well as the current technologies and techniques available to solve them.

Visualizing machine learning to the every day person is no easy feat, otherwise it would have been done many times over already. Despite that, the web is full of visualization libraries that could be used to help represent machine learning, in particular *Bayesian networks* graphically.

An online conversational user experience is a well explored domain, whether it is between at least two people or between person and machine. The review will look to define what a human conversation is and comparing it with an online conversational experience. Researching the expectations or guidelines for a conversational interface followed by presenting examples of successful conversational interface techniques from industry leading companies' implementations that will help inspire requirements for an effective design of the project's new chat interface.

2.1 Machine learning visualizations

The core of the project is to visualise the machine learning used in the Scikic in an approachable way. The project will look to provide users of the Scikic with real-time visual feedback which attempts to present to users how machine learning is being used to predict information about them in the Scikic. This is with the aim of adding to the user's understanding of machine learning concepts.

This section will look at understanding the types of machine learning techniques the Scikic is using, how they can be visualised and which libraries could be appropriate to implement the visualizations.

2.1.1 Bayesian networks

Also known as "*Bayes network*, *belief network*, *Bayes model* or *probabilistic directed acylic graphical model*"[48] is a graphical model that represents probabilistic relationships among variables of interest[38]. Since it is a set standard for a graphical model, it is important to understand what a Bayesian Network(herein BN) consists of to be able to accurately evaluate which visualization library is the most appropriate to use in the *Requirements and Analysis* chapter. This also means that there will be no need to review variations of visualizing a BN since by their definition it is a set standard.

How Bayesian networks are represented
Before talking about how a BN offers a number of advantages over conventional statistical techniques[100] in data analysis, it is worth understanding the required form of the graphical model. Formally, BNs are directed acyclic graphs, meaning that their edges have a direction associated with them and have no directed cycles. As a result each node does not have to have a uniform in/out degree.

Nodes represent the probability(degree of belief) of random variables about "an entity [and] can consist of both discrete(e.g. Gender=Male, Female) and continuous(e.g. age)"[62] probabilities.

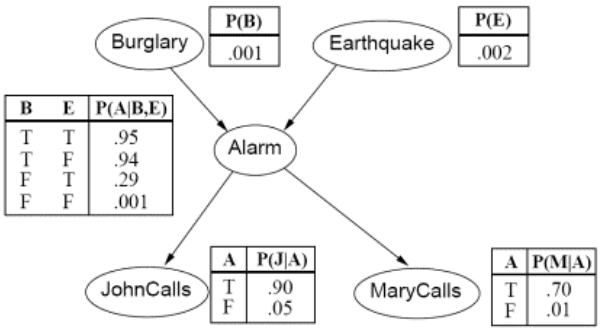


Figure 2.1: Bayesian Network establishing relations between events on the burglary-earthquake-alarm domain, together with complete specifications of all probability distributions[81].

Edges represent conditional dependencies and can show causal relationships. Equally, nodes that are not connected(where there is no path from one of the variables to the other) shows that the nodes are conditionally independent from each other. It is important to note that edges "do not necessarily represent direct cause and effect"[78]. BNs are robust where information is missing, not because they can suggest direct cause and effect, but because they facilitate making the best possible prediction using whatever information is available.

The graph represents the relationships and usually have tables to store the conditional probability of each node representing their degree of beliefs as shown in *figure 2.1*. Instead of using nodes to show relationships between entities, you could use a table with each of the nodes as a column and list the probabilities based on whether they are true(1) or false(0). This is called a full joint distribution table which could be used as an equivalent representation of *figure 2.1*. A full joint distribution table would require 2^5 (32 probabilities) since there are 5 binary values. *Figure 2.1*'s BN only requires 10 probabilities showing that BNs "provide a concise way [to show a full join distribution] by representing **conditional independence**"[81]. As a result, it is intuitively easier for a human to understand direct dependencies and local distributions when compared with full joint distributions.

How Bayesian networks are trained

There are two main ways that the model and conditional probability tables(herein CPTs) in BNs are trained. The first is having a domain expert which might "provide prior knowledge specifying conditional independence among variables...[and even] about the values of certain parameters in the CPTs"[67]. This could be seen as a bottleneck, since knowledge acquisition to become a domain expert could be seen as an expensive process. The second more common way is using observational data due to the rapid growth of the Internet, it makes it easy to collect data on a large scale which is often too large for processing[23].

Inference

Once a BN has been constructed, applying new evidence to the nodes affects other nodes. Combining new evidence prior to degree in beliefs to make a prediction is known as inference. The reason we apply inference is to know the "probability of an event given observations of the other variables"[38].

Several researchers have developed probabilistic inference algorithms for BNs with discrete variables. The most commonly used and practical inference methods for BNs are those "based on the *clique tree* algorithm(Lauritzen and Spiegelhalter,1988; Jensen et al.,1990; and Dawid,1992) for discrete variables"[53]. The algorithm transforms the BN into a *tree* like structure where each node in the tree corresponds to a subset of the variables for an entity. The algorithm then is based on two operations: *collectEvidence* (messages are passed from leaves to root) and *distributeEvidence* (messages are passed from root to leaves).

The *clique tree* works efficiently well for moderately size networks[53]. This becomes even more com-

plicated when looking at BNs with continuous variables. It is well known that, in general, the "exact inference algorithms are either computationally infeasible for dense networks or [currently] impossible for mixed discrete-continuous networks"[13]. Despite that, continuous variable networks are still being used in a wide range of applications today, including "fault detection (e.g., U. Lerner and Koller, 2000), modeling of biological systems (e.g., Friedman et al., 2000) and medical diagnosis (e.g., Shwe et al., 1991)"[25].

2.1.2 Collaborative filtering

The main idea of collaborative filtering(herein CF) is a rating system that can help recommend items to users based on other users' opinion[60] ratings. CF is the process of filtering for information or patterns using techniques involving the collaboration of multiple datasources. CF is the most successful recommendation technique to date[83].

The main use of CF is attributed to the growth of the Internet has resulted in a "tremendous amount of information available and a vast array of choices for consumers"[60]. CF is used throughout industry to present people with recommendations in ways other than by *word of mouth* from friends and people we know telling us what they like. This includes industry leading companies such as *Amazon*[54], *MovieLens*[52] and *Google News*[20] to name a few.

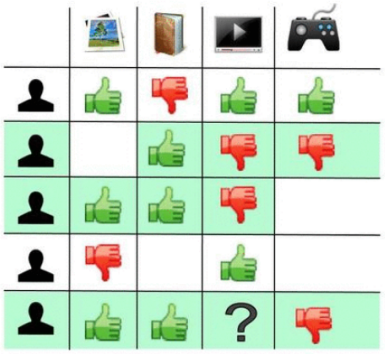


Figure 2.2: An example of different items (like videos, images, games) user ratings using collaborative filtering and wanting to know a given user's likely rating[63].

Figure 2.2 and *2.3* shows us how CF can potentially classify an item (top row) to a given user (left column) based on a database of information that we know about other users' ratings of the items.

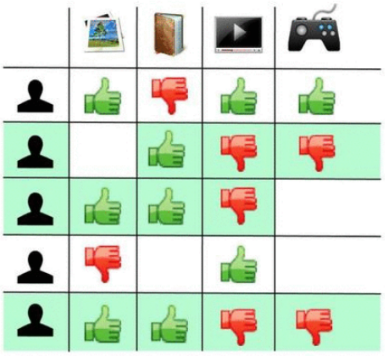


Figure 2.3: An example of predicted of the user's rating using collaborative filtering[63].

Obviously the abstract example data set is overly simplified to show how CF works. In reality, the "industry implementations of CF are all multi-class data sets"[88]. This is where a BN model is more concise than a matrix based CF system. The reason is a similar concept to how the number of probabilities are more concise in a BN than a full joint distribution. The gaps in the matrices shown in *figures 2.2 and 2.3* represent the conditional independence that is represented by a lack of directed edge to the particular nodes in BNs(*figure 2.4*.

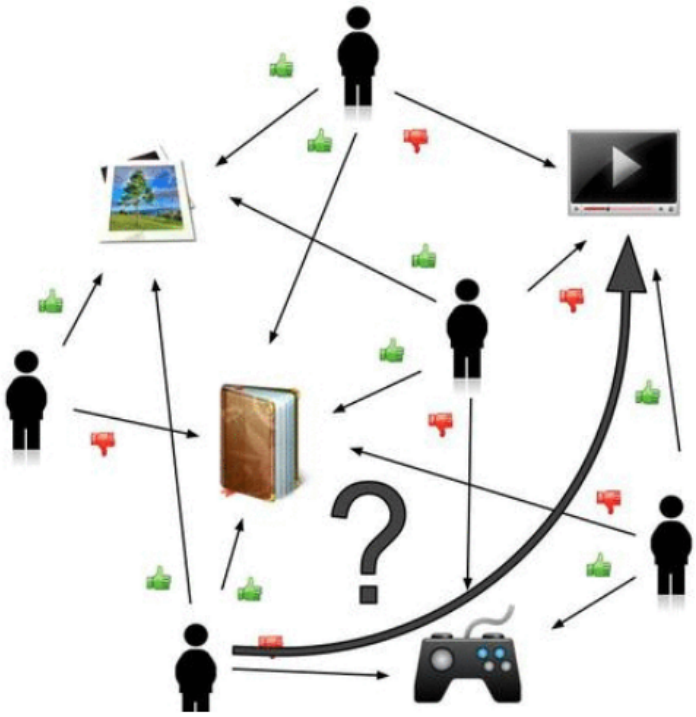


Figure 2.4: An example of a Bayesian Network representation of Collaborative Filtering[63].

As discussed in the BN Literature Review section, there is an increased computational complexity involved when making inferences, however, there are convincing conclusions to "accurately CF with BNs using real world data from *MovieLens* (a web based recommender system with 43,000 users and their ratings for over 3,900 movies"[88]) is feasible and achievable in reasonable time. The results are not as important as the fact it shows that these techniques are feasible and clearly being used in the industry.

2.1.3 Available visualization libraries

Creating visualizations in web applications is something that has been around since at least 2005, with *Prefuse*[71] being one of the first available visualization toolkits that required the usage of *Java*[39] to render it. One of the most significant events to overcome the limitations of clicking links and scrolling pages to give users rich interactivity was the integration and adoption of *JavaScript*(herein JS) as the "language of the web"[37]. For that reason, it makes sense to limit the scope to only JS libraries. This project is not funded and is not intended to provide any kind of revenue. The project's intention is also to show a completely transparent implementation for machine learning visualizations. For those reasons, it makes sense to only investigate open source JS libraries.

Given rich interactivity being a key requirement for the project, it could be appropriate to use very flexible JS graphical rendering engines such as *Famous*[32] for animations and interfaces. After some investigation of the libraries available, there are a number of higher level abstractions available which avoid coupling the code so tightly, are less complex and would make the project more robust compared to using a 3d graphical engine. The JS libraries this section investigates focus on providing the tools for creating interactive data-driven visualizations such as Bayesian networks.

It is important that the comparison looks at JS libraries that can create interactive visualizations of Bayesian networks(since this is the basis of the current Scikic). A good place to search is *GitHub*[35] since it has become "the industry standard"[89]. *Github*[35] has useful indicators of the most popular, well-maintained and well-documented production ready JS libraries. Based on that, this section discusses the following most starred JS repositories that could be potentially used for the machine learning visualizations:

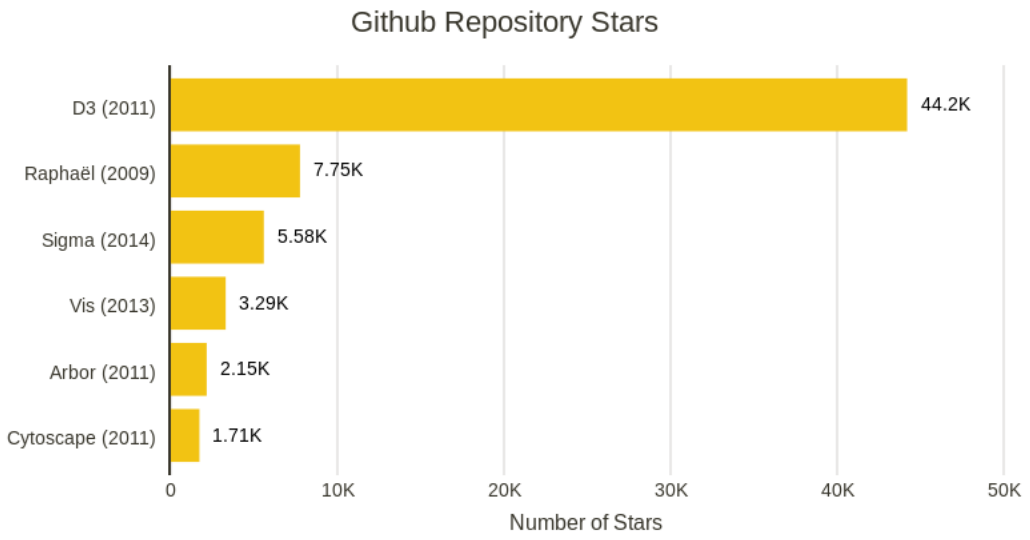


Figure 2.5: Comparison of the popular opensource JS data-driven visualisation repository by number of stars it has. (as of 06/12/2015)

The comparison of *Github*[35] repositories(*figure 2.5*) show some interesting results. Despite *Raphael*[73] being the oldest data visualisation library out of the comparison set, *D3*[19] clearly distinguishes itself as the front runner in terms of what developers choose to use. Another impressive library is *Sigma*[86] which has almost overtaken Raphael in terms of popularity in the space of a year which could be an indicator of the library's quality. Here the section will take a closer look at why these are the most popular libraries and what features they provide.

D3[19]
What makes this library interesting is the flexibility and potential creativity that is given to developers

with this fully-featured library. There are many public demos and tutorials of unique data-driven visualizations which are eye capturing, making it no surprise that many developers have chosen to adopt D3. Its powerful API allows the developer to create rich interactive visualizations that is almost only bounded by the developer's creativity. That fact is, I would argue, the reason why it appeals to such a wide target audience and why there is now such a large community of developers surrounding D3.

A developer is limited by their determination to learn how to utilize D3 since there is a steeper learning curve than the other similar libraries. To have more complicated capabilities, naturally, the library also becomes more complicated to use. As developers have begun to realise that, an ecosystem has appeared with alternative reusable D3-based JS libraries such as *C3*[11] and *d3-shape*[58] to lower the learning curve required to use the library for more common visualizations.

Raphael[73]
Despite *D3*'s[19] exponentially higher popularity than the rest of the JS libraries, Raphael is the next most popular. That could be attributed to the library being developed the earliest. A key difference between Raphael and the other libraries is that its focus is giving the developer the ability to "create vector graphics on the web"[73] as opposed to creating data-driven visualizations based on the data that is inputted to the library. This suggests it may be less suitable to create the visualizations for the project.

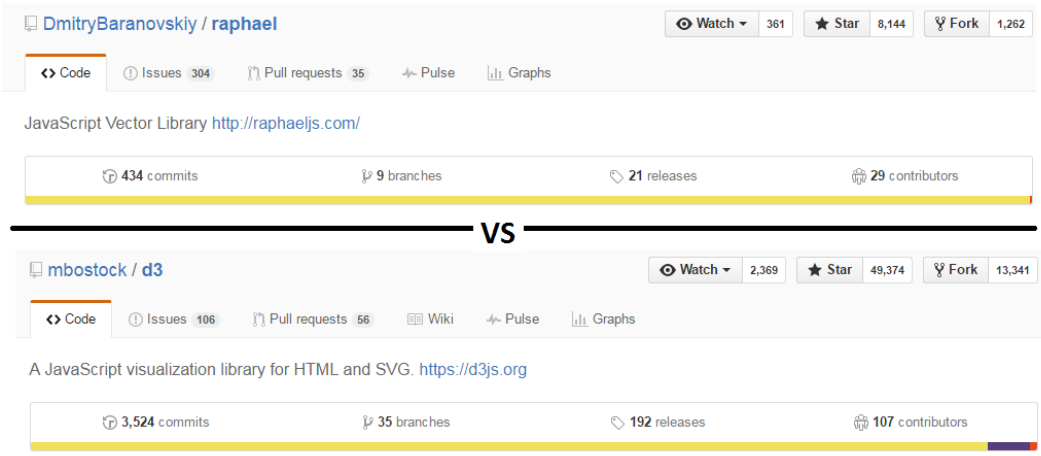


Figure 2.6: Comparison of the commits/issues/releases and contributors between D3[19] and Raphael[73] (as of 01/05/2016)

The level of development activity is clearly low for this library, shown by the infrequent commit history and the backlog of issues which are higher than *D3*'s[19](*figure 2.6*). This makes Raphael more risky to use, unless the report can confirm that the library meets 100% of of the project's requirements without bugs due to lack of ongoing support.

Sigma[86]
The appeal of this library is the pure focus on creating high performance large network visualizations while at the same time providing a highly extendable API. This gives the developer full control of its interactivity and also allows developers to add custom functions to render the nodes and edges exactly how they want them to appear.

Even though the library has only been publicly released since 2015 and has a very focused target audience, its high adoption rate is a good indication of its quality, lower boundary to entry(compared to *D3*[19]) and flexibility.

The library also provides many plugins with graph analysis in mind which shows that in such a short period of time it has managed to build and grow an ecosystem of its own. This makes the Sigma a key

contender for this project.

Vis[95]
This library is not as focused in its functionality as *Sigma*[86], however, is somewhere in between that and *D3*[19]. It is less complicated than *D3*[19] because it limits itself to only 5 different modules: Network, Timeline, Graph2d, Graph3d and Dataset. Having the limited number of possible modules has allowed those particular modules to have a high number of structured variations without having to invest a large amount of time in understanding how to create them.

For this project, a small subset of the available modules would actually be necessary, perhaps suggesting that this is not the most appropriate library compared to the libraries discussed above. Having been released in 2013, the example modules do also look rigid, outdated and not as appealing as the other libraries.

This has not stopped Vis' community creating some complex projects[96] with this library. A particularly notable one is *Pfizer*[90] which explores an interactive relational network of patent references made by startups for an industry report. The showcases clearly prove that complex networks are possible to make.

Arbor[3]
This library which again similarly to *Sigma*[86], focuses on only providing a way of creating a node-arc graph network. Further investigation into the library's thorough but short documentation, it is apparent that the library is a way of creating visually impressive networks but do not have extensive nor editable interactions.

Arbor is a quick solution for developers that want no hassle in setting up a simple network, only needing and being able to change the layout and node parameters of the network. Unfortunately, it is clear that this library is not suited to the project, due to the lack of interaction extendability to meet the requirements of this project.

Cytoscape[14]
Its opening statement is that it "is an open-source graph theory (a.k.a network) library... for graph analysis and visualisation". Clearly, being the least popular on *Github* is not reflective of the functional capability of this library which judging by its documentation is comparable, if not greater than *Sigma*[86].

Where the library does not quite match up to the other available libraries is how animations in their visualizations are more jittery, suggesting the example implementations are not performant when rendering which is most likely why this library is not more popular.

2.1.4 Conclusion

This section has given an in-depth enough understanding of Bayesian networks and collaborative filtering be able to design real-time visualizations of machine learning techniques used by the Scikic. It has also discussed and compared a variety of appropriate open source JS libraries that could be used to implement the visualizations. The fact that there were so many confirms that the project's visualization aim is achievable and realistic.

2.2 Conversational user experience

This section of the literature review will first look into what a conversation is and techniques used to create a successful conversational situation. Then the review will look into current conversational interface solutions that exist in today's industry implementations using those techniques.

2.2.1 What is a conversation?

Conversation generally has no existing accepted definition, beyond the fact that it is a "form of interaction (normally spoken) involving at least two people"[97]. Instead of focusing on what a conversation is, it is often defined in terms of what it is not. Some commonly occurring examples of what it is not include:

- A ritualized exchange such as a mutual greeting or an interaction that includes a marked status differential(such as a boss giving orders) is not a conversation[97].
- An interaction with a tightly focused topic or purpose is also generally not considered a conversation[97].

Summarizing the above, a conversation is the kind of speech that happens informally, symmetrically, and for the purposes of communicating information while "establishing and maintaining social ties"[84].

In this project, it is key that there will be an interaction between two entities rather than two people. More specifically, between a person and computer. The ability to generate conversation that cannot be distinguished from a human participant has been one way to test for successful artificial intelligence, known as the Turing Test[92]. For this project however, the Scikic does not need to be indistinguishable from a human to be a successful implementation that visualizes machine learning techniques in an attempt to eventually help users understand machine learning better.

The Scikic only needs to emulate a conversation to some extent in order to help the interaction feel more informal. The purpose of a conversational interface is for the user to feel less like they are having information pushed at them and interrogated. The model that the project should follow is that "*this is a conversation, not an interrogation*"[51].

2.2.2 How to help the conversation be successful

The difference between a real life conversation and 'conversing' with a web interface, is that generally there is a social obligation to stay active in the conversation. Online, a user can leave and never come back to the web interface with the single click of a button[9].

There are two main criteria that make users more likely to answer questions in an online conversation. The first is "*trust*"[98]; users will not answer questions if there is not any trust, in the same way as if a stranger was to approach you and ask personal questions in a human conversation. The way to help ensure there is trust, is to clearly show information about what the web interface is doing, how it uses data and what the web interface is trying to achieve. The more information given to the user before asking for personal data, the more trustworthy the web interface will be[80]. The second is "*having a reason*"[9]; asking for personal data before it is needed can have an impact on whether personal data is given. For example, if an e-commerce web interface asks for a physical address on account sign up, a user may feel their privacy is being violated because they have not made a purchase yet and may not make one as a result. In an alternative situation, the e-commerce web interface asks for a physical

address only when checking out the user's purchase which gives the reason for asking for that piece of personal data. The address can then be saved for convenience if the user so wishes achieving no less than requiring it during account sign up.

There are some "useful user experience guidelines"[57] during interaction with a conversational interface that can encourage users to answer questions:

- Order labels logically reflecting the natural flow of a conversation.
- Group related information where one set of questions to the next better resemble a conversation.
- Each question should address one topic at a time.
- Introduce white space to indicate natural pauses in a conversation.
- Get rid of background noise and remove clutter such as banners and unnecessary navigation that might distract users.

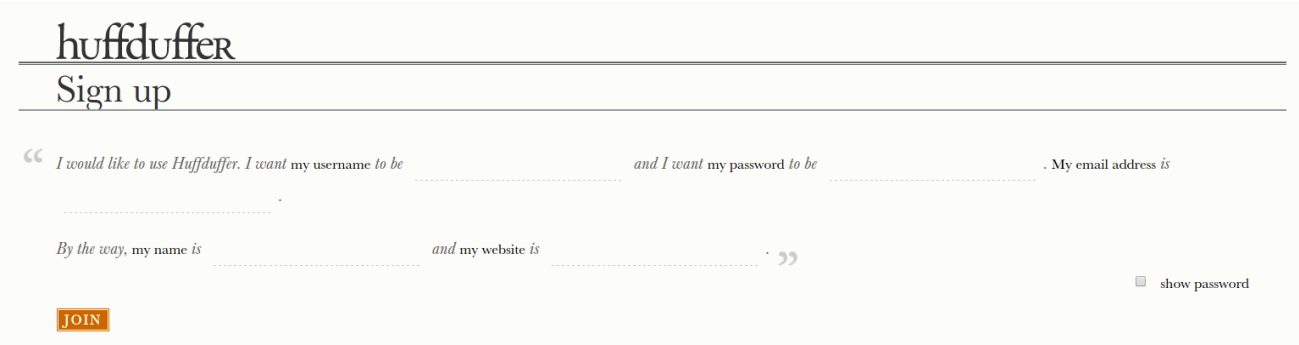


Figure 2.7: An extreme example of how the form reflects the way we converse making interaction with it more natural by engaging with the user[41].

Not applying these concepts can make the experience more like an interrogation and therefore more probable for users to leave the web interface before using the Scikic. The project does not aim to virtualise a conversation but to provide a conversation like user experience, to which there is a difference.

2.2.3 Solutions of conversational interfaces

Unlike in the visualizations section, there is no selection of open source JS technologies available to use or a 'holy grail' of conversational interfaces. The current solutions will instead look at the different industry leader's implemented user experience techniques as inspiration for what makes a good conversational web interface.

It is worth noting, that all the conversational interfaces discussed make sure to adhere by the guidelines mentioned in the previous section of how to make a conversation successful. The focus of these implemented techniques is to research what can be further done to contribute to an effective conversational experience and keep the user from leaving the conversational web interface before using it properly.

Facebook chat typing awareness indicator

An the indicator that is used by *Facebook M*'s[46] artificial intelligence(*figure 2.8*) and more commonly known by users of *Faceboook's Messenger* application[31]. When the other party to the user is typing a message, an indicator appears that they are engaging in the conversation. This makes the user feel more like they are in a conversation where they know whether the other person/entity is responding or

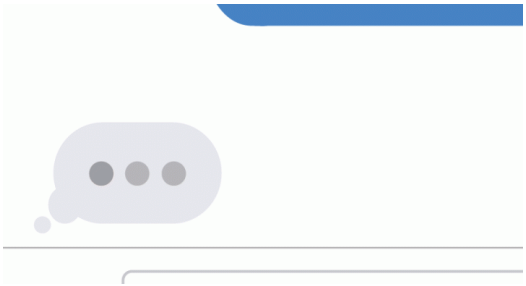


Figure 2.8: Facebook's chat typing awareness indicator[79].

waiting for the user to say something further[4].

WhatsApp[99] separating the user's replies

WhatsApp[99] is not the only chat to clearly distinguish which user communicated which part of the conversation but it does offer a good example of clearly doing so(*figure 2.9*). It does that by using clear spacing between conversational points, colouring and distinguishing each of the different user's replies.

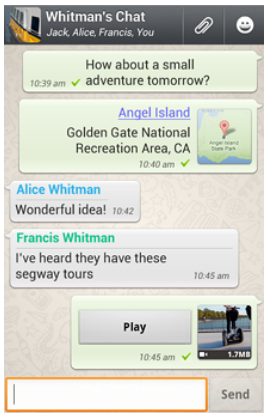


Figure 2.9: WhatsApp's chat interface[99].

Skype[56] showing the recipient's user profile

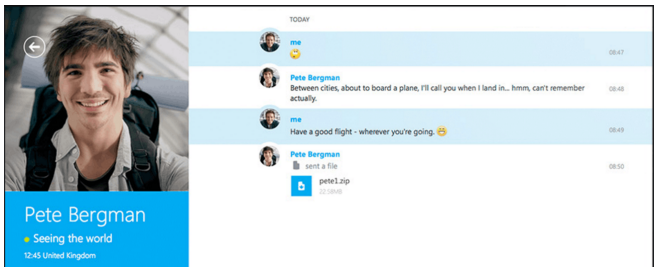


Figure 2.10: Skypes chat interface[56].

Again *Skype*[56] is not the only chat service to use this functionality. It shows the user profile of the recipient of the conversation to remind the sender user that they are speaking to a real person that they may know(as seen on the left hand side of *figure 2.10*).

Dominos[21] Pizza "embodied agent"

Good features extend beyond the online chat services. *Dominos*[21] pizza delivery has a tracker to show the user how close their pizza is to being delivered. In the centre of *figure 2.11*, there is an image of

a robot which speaks more informally during updates, giving customers a more human conversational experience.

A graphical representation that interacts with the user is "commonly referred to as an *embodied agent* or *interface agent*"[85]· Research has found that users "prefer a non-verbal visual indication of an embodied system's internal state"[55] because it provides a "social dimension to the interaction ascribing social awareness"[66], making users slightly more inclined to stay on the web interface (see *How to Help the Conversation be Successful section 2.2.2*).

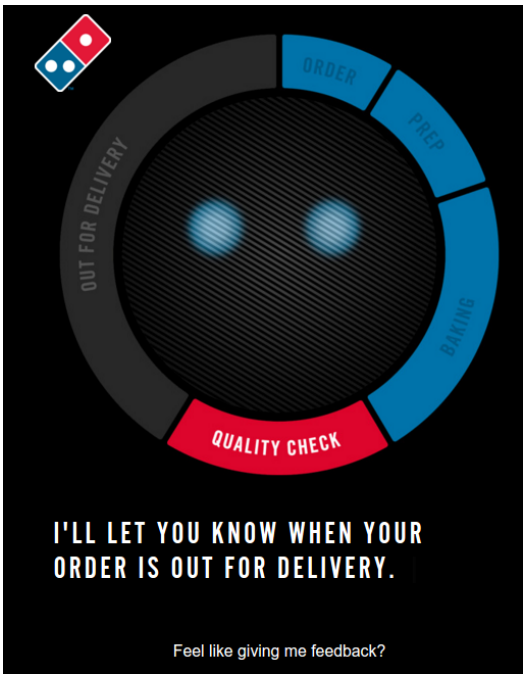


Figure 2.11: Dominos pizza tracking interface[21].

Typeform's[93] letter by letter typing

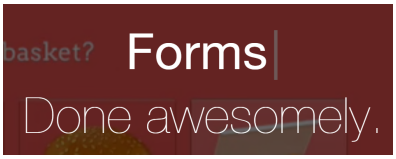


Figure 2.12: Typeform's typing indicator where each letter is typed individually[93].

Typeform[93] uses a text cursor animation to type out letters individually which better emulates a conversation, where words are spoken one by one as opposed to all spoken instantaneously.

PointerPointer's[70] deliberate processing time emulating human thought

PointerPointer[70] and others such as *KittenMouse*[101] are web applications where a collection of photos with an entity is directed in the place where your mouse is positioned on the web interface.

This is a very simple selection of pre-defined images and loading of photos which in reality is instantaneous[43]. The point of adding a loading screen(*figure 2.13*). despite having no calculation time makes the interaction more conversational like, replicating the pauses taken by humans during thought in between conversation.

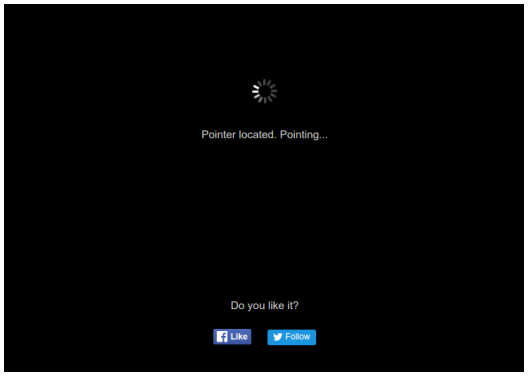


Figure 2.13: Deliberate "thinking and pointing..." screen despite actually being instantaneous[70].

2.2.4 Conclusion

This section has broadly covered what a conversation is. It also covered how the project is to successfully get questions answered by users by making the conversation feel less like an interrogation. It has also covered useful user experience guidelines, as well as some leading industries' techniques used to create a more conversation like experience beyond the described basic guidelines.

The industry leading companies are clearly putting in effort into their conversational interfaces. This is a good indicator of how user's expectations of conversational interfaces are made high by these companies. There is value in using their ideas as inspiration when designing a new web interface to meet the industry's high standards and ultimately give Scikic's users a better experience making it more likely for them to use it.

3 Requirements and Analysis

This chapter is split into two independent sections focusing on the two overall aims(*Introduction section 1.3*): Scikic's view on data and how the Scikic's prototype web interface talks to its users. This chapter will not show any designs and is meant to specifically show the process of defining clear requirements to meet the overall aims based on the research in the *Literature Review*.

The analysis will begin by describing the Scikic's Bayesian Network(herein BN). This will be done by understanding what the current Scikic's external data sources are, what input data the API expects to perform inference and the data that is returned by the API that will enable the new web interface to visualize the Scikic's BN. After analyzing the current Scikic's API, this section will be able to justify which visualization library is the most appropriate based on the survey(*Literature Review section 2.1.3*).

The second section of the analysis will critically review the current prototype conversational interface's features as to whether they adhere to the previously presented user experience guidelines(*Literature Review section 2.2.2*), by following a user's typical experience using the interface. This section will then go on to discuss which of the presented industry leaders' user experience techniques(*Literature Review section 2.2.3*) would be appropriate or already exist in the Scikic's prototype conversational interface and therefore should be required in the new interface.

From the analysis in this chapter, the requirements will be able to define what needs to be achieved in the implementation of the Scikic's new web interface.

3.1 How does the Scikic visualize data?

This section explores the Scikic's external data sources and what the API's expected inputs/outputs are for its inferences. This is to understand what requirements are achievable within the scope of the project without having to modify the current Scikic's API. The section will then conclude with the most appropriate visualization library from the surveyed available libraries(*Literature Review section 2.1.3*).

3.1.1 Current data sources

The current external data sources used by the Scikic can be acquired through a request to the API's */metadata* endpoint which currently includes:

- The Facebook graph API[28]
- The US Census Bureau[94]
- The Movielens database[64]
- The ONS provide statistics on the distribution of the names of babies in the UK: 1996-2013[7] and 1904-1994[6].

- The UK office of National Statistics[69]
- US zipcode data from Missouri's Census Data Center[59].
- The bandsintown.com API[8]
- The freegeoip.net API[34] which provide geographic locations depending on the IP address given.

This data means that the web interface's visualisations can show the user where the data is coming from to be transparent and gain the user's *trust*(one of the main guidelines in the *Literature Review section 2.2.2*). Since the external data sources are based on the API's endpoint, they can be acquired dynamically even if the external data sources change in the future.

3.1.2 The Scikic's required input data

To request inference on a list of answered questions, the interface sends a POST request containing the answered questions in a JSON object format to the API's */inference* endpoint. The web interface requests inference using a POST request because the list of answered questions is too much data to send through the URL in a GET request.

Each answered question in the list that gets sent to the API has three parameters: the **dataset** that the question is related to(metadata about the dataset can then be retrieved via */metadata*, see *section 3.1.1*); the **data item** which represents a particular feature about the user from that dataset, since each dataset could be more than one data item(e.g. the 'demographic' dataset could be data item age or gender features about a user); the **user's answer** to the question(this parameter does not exist until the user has answered the question in the interface).

Before being able to answer any questions and form the described answered question parameters, the web interface needs to request the API's */question* endpoint which will get sent the following parameters(each of the below parameters start out empty initially):

- **Questions** already asked to stop the Scikic API from sending the same question again.
- **Facts** about the user, retrieved from previously processed questions to stop the API asking similar questions which will not helpful for the Scikic to produce new insights and to correctly make inferences based on these facts.
- **Unprocessed** questions are the answered questions which have not been processed by the Scikic's inference yet. Any new facts are added to the user's facts after the inference if any user features are 100% confirmed.

Once the requested question is answered in the web interface, the same 3 parameters (*Questions*, *Facts* and *Unprocessed*) are populated. *Facts* are never manipulated by the web interface and can be considered as the user's 'state' which enables the API to be stateless. *Questions* are also returned the same apart from the *user's answer* parameter is populated for every question that the user has answered in the interface. Finally, *Unprocessed* has the same structured list as *Questions*, but only questions that were answered since the last inference are included.

The resulting JSON object containing the keys *Questions*, *Facts* and *Unprocessed* is sent to the API's */inference* endpoint to request an inference on that object. While it is possible to allow the user to answer more than one question at a time, since the aim of the project is to show real-time visualizations with every question that is answered, for every question that is answered there will only ever be a list with one answered question in the *Unprocessed* object.

3.1.3 Information about the Scikic's baysian network from returned inference data

The API's */inference* endpoint returns a JSON object which contains information about the inference includes:

- **Facts**, an object representing the new state of the inference for the user. When the user answers a new question these *Facts* replace the original *Facts* object that gets sent to the API's */inference* endpoint(see *section 3.1.2*).
- **Features**, a list of probability distributions inferred for each of the user's features.
- **Insights**, a list of the interesting text insights inferred by using the *Bayesian Network* with *collaborative filtering*. This includes debug and other list information that will have to be filtered to return just the user's insights.
- **Relationships**, a list of parent and child features showing how features are related to one another in the Scikic's BN.

Features returns conditional probabilities that are calculated using collaborative filtering from the data sets which are combined with a Bayesian network. This is done with Python's *pyMC*[72] module. Each data set provides *pyMC* with 'data items' predicting the user's features which are outputted in the form of probability distributions. The Scikic then processes the distributions to come up with personal insights from those features. The personal insights are returned in the *Insights* object of the data returned from the inference.

The probability distributions in *Features* can be visualized to the user of what the Scikic is inferring. The *Relationships* list gives the web interface information about how the features are connected in the Scikic's BN. As the user answers more questions, the visualisations should update the said distributions and represent them with some kind of visual nodes that also update, connecting them based on the *Relationships* object sent by the API's inference. It is important to make sure that for every feature probability distribution, there is a related node in the BN. Otherwise it will confuse the user about the relationships between the probability distributions of their features and the nodes in the BN, having the opposite effect of helping the user understand BNs better.

Using the question's 'data item' parameter, nodes can be identified and using the 'data set' parameter, they can be related to their respective data sources. Displaying each node's data source will build the user's *trust* by being transparent about the origins of the information(one of the guidelines in the *Literature Review section 2.2.2*). Displaying the data set or feature will help take away the 'magic' from machine learning by helping understanding the BN better and as a byproduct, the user is more likely to carry on using the web application because they will want to understand how the answers are affecting their BN.

Since the inference is on a question by question basis, the user should have control over which question's inference is being shown, by giving the user the ability to select previous questions. This allows the user to be able to confirm what has changed since the selected previously answered question. The user can revisit the inference's transition from their selected question. This would be particularly useful if the user missed the change in features accidentally and allows them to replay the transition without having to go through the process again, where there's no guarantee that the Scikic will even ask the same questions.

3.1.4 Most appropriate visualisation library

The *Literature Review*'s *Available Visualisation Libraries*(section 2.1.3) suggests that *Sigma*[86] is the best choice for modelling the Scikic's BN in the new web interface.

It has the strongest focus on creating networks and the most exponential growth in terms of developer user base. The widespread adoption, despite only being released for a year, is a show of quality and support for the library. Most importantly, based on its documentation and its current plug-in availability, *Sigma*[86] has the required functionality to create an interactive Bayesian network compared to the older less popular libraries.

The reason that *Sigma*[86] is not chosen, is that it can **only** create visualisations for a *Bayesian networks*. This is a major problem because despite being flexible enough to be used in conjunction with other libraries, it will add another level of unnecessary complexity which could be avoided by choosing a library that meets all the requirements for the visualizations.

In that case, the most logical library to use is *D3*[19]. This is due to its overwhelming domination of the visualization libraries and even though there is a steeper learning curve, the guarantee that *D3*[19] can meet 100% of the visualization requirements outweigh the longer learning time (compared to the other libraries). It is also tailored to specifically create data-driven visualizations, unlike some of the other surveyed libraries such as *Raphael*[73] which are aimed at creating vector graphics. *D3*[19] has very powerful functionality to transition between different states which would meet the core requirement to update the visualizations per question that's answered by the user.

3.1.5 Conclusion

This section has analysed what the current Scikic's API expects to be inputted, as well as, the data that is outputted from the inference. Based on that analysis the section can establish a number of requirements to create the visualizations in the web interface:

1. The web interface needs to create and manipulate an object containing *Questions*, *Facts* and *Unprocessed* which initially start as an empty list, empty object and empty list respectively.

2. Before requesting the visualization data from the inference, the web interface needs to request a question from the API's */question* endpoint sending the initial object described in 1.

3. Once the user answers a question, the web interface needs to manipulate the object holding the three parameters for the question by populating the user's answer parameter in the object. That object is then added to the *Questions* and *Unprocessed* lists from 1 and is sent to the API's */inference* endpoint.

4. The web interface will then be returned a number of objects including an object containing a list of probability distributions for each of the user's features. Each feature needs to be visualized in some way and also have a node representing that feature. These nodes will be connected using the 'relationships' from the returned data to create the Scikic's BN.

5. The nodes should be visualized with the data item or feature that it represents and also the data sources that it is from. The data sources can be retrieved using the feature's data set from the API's */metadata* endpoint.

6. For each further question that is answered, that question should be added to the *Question* list from 1 and the *Unprocessed* list should only have the new question in it. The *Facts* object should

be replaced by the *Facts* returned in the inference object from 4.

7. The inferences from the consequent questions in 6 should clearly show a transition for both the features' probability distributions and add any new nodes with their relationships as well.

8. The user should be able to select previously answered questions and the visualization should change to the the state that it was in after answering the selected question.

3.2 How the Scikic talks to users

As already mentioned in the *Introduction*'s *current Scikic*(*section 1.1*), the Scikic has a prototype web interface. The purpose of this section is to critically review its features and discuss appropriate improvements to include in the conversational interface's requirements based on the *Literature Review*'s *conversational user experience*(*section 2.2*).

3.2.1 Scikic's prototype conversational interface key features

The prototype web interface is using Python to send across HTML strings which render depending on which URL was requested while also sending/receiving requests from the *Scikics*'s API. The main reason it has been built in Python is because the developer of the prototype was confident in Python and wanted to create something quickly without having to learn modern industry standard web development technologies. Rendering the web application line by line from Python works, however, it makes the code base very difficult to understand, maintain and add new features to.

As mentioned in the *Literature Review*, the project will be built in *JavaScript*(herein JS), the "language of the web"[37]. Modern web development rarely develops web applications using 'vanilla' JS, HTML and CSS. It tends to also involve a 3rd party framework to structure the code more manageable and scalable. Due to a requirement given by *CitizenMe*, a collaborative start-up in this project which plans on providing the Scikic with user data to use for its inference, the conversational interface needs to modular so they can use that module without having to include the visualizations module. There are a number of frameworks to help achieve this, some of which include, *Angular 2.0*[2], *Google Polymer*[36], *RiotJS*[65] and *Facebook's ReactJS*[30] which each have their advantages and disadvantages. These have not been surveyed in the *Literature Review* since the aim of rebuilding the web interface is not to figure out which JS framework is the best, but to implement a new conversational interface.

This project will use *ReactJS*, mainly because of my familiarity and experience with it guaranteeing that the project can get the separation and modularity that is required. Other reasons include: *Angular 2.0* is not production ready yet and therefore is not mature enough for usage; *RiotJS* is a smaller version of *ReactJS*, aimed at less complex web interfaces and therefore is harder to create more complicated features with; I am not familiar enough with *Google Polymer* and think that it would hinder the project's progress by trying to learn it for the project without a good reason.

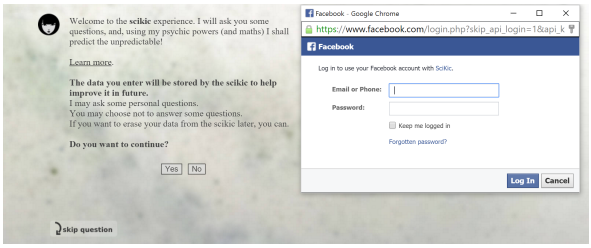


Figure 3.1: Scikic initial screen with option to Facebook login and description of the application.

On opening the prototype interface, it asks the user to login to *Facebook* with the goal of using the data for a more precise prediction(*figure 3.1*). Since this is in the prototype web interface, the *Facebook* login should be a requirement. There are a couple of problems with *Facebook* login popup. Firstly, it asks for the information without explaining why the Scikic needs this information at all, losing some of the user's *trust*. Secondly, the interface asks for this information before needing it, more specifically, it asks the user to login before the user has indicated that they are interested in using the Scikic. These are both criteria described in the *Literature Review*'s *How to help the conversation be successful*(*section 2.2.2*) that the prototype web interface fails. Instead of asking the user obtrusively for the information before explaining why the Scikic would like the information, the new conversational interface should ask if the user would like to login to Facebook to save the Scikic asking some questions, after the user has specified that they are interested in trying the Scikic. This will also follow the model of being less of an interrogative experience(*section 2.2.1*).

The interface then shows information about what the Scikic will do, how data will be stored(if at all) and whether the user wants to continue. According to the criteria described in the *Literature Review*'s *how to help the conversation be successful*(*section 2.2.2*), these aspects are beneficial since it gives the user information about what the prototype web interface is aiming to do. This helps gain the user's *trust* by not pushing questions before the user gives consent and is necessary. All these features should be required of the new interface.

While the interface tells the user its overall aim, there is not any further explanation around the motivation behind the Scikic, nor who created the Scikic. The lack of these could lose the user's *trust* by allowing the possibility that the Scikic was not only made by potentially malicious anonymous users, but also that the motivation is something malicious. To prevent this, the user should be able to easily navigate to the described information.

Once the questions begin, there are two different types of answers that can be given. Firstly, some questions require free flowing text answers such as *figure 3.2*.

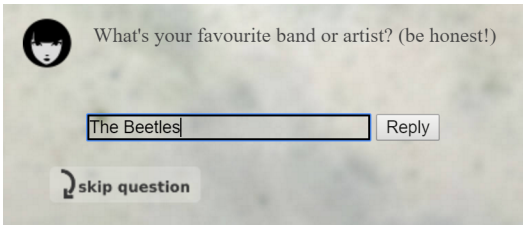


Figure 3.2: Scikic's free flowing answer format.

The other format of answer, depending on the question asked is selection based; There are a number of fixed selections that the user can make such as in *figure 3.3*.

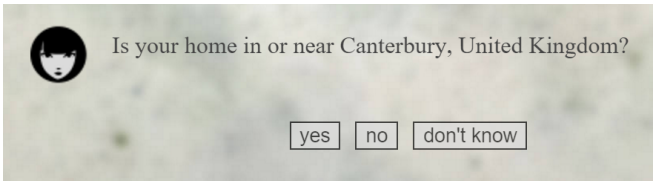


Figure 3.3: Scikic's selection based answer format.

Both of these formats are required as answer types and can be distinguished by the question information sent from the API's */question* endpoint.

There is always an option to skip the question if the user feels that it is too personal or they would rather not answer the question. This meets the criteria of *having a reason* from the *Literature Review*'s *how to help the conversation be successful*(*section 2.2.2*), since the Scikic does not care which questions a user answers. Being able to skip a question is a key requirement for the conversational interface.

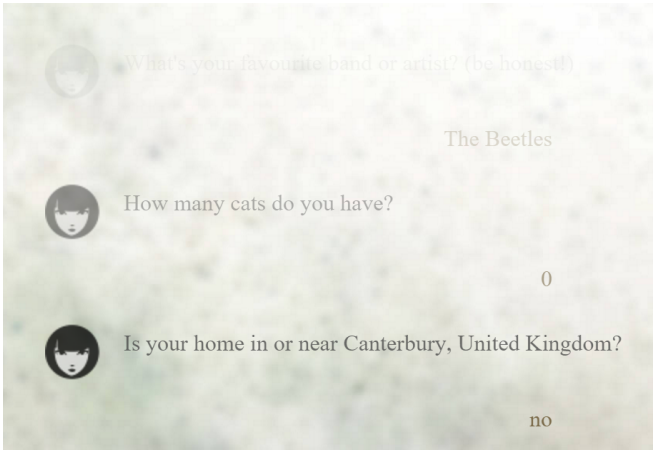


Figure 3.4: Scikic's fading away previous history.

As the conversation goes on, the previous history of the conversation fades away as can be seen in *figure 3.4*. This follows the "useful user experience guidelines"[57] by getting rid of background noise, however, the user loses two capabilities:

- Allowing the user to select and answer previous questions which contradicts the 8th requirement for the visualizations(*section 3.1.5*). This requirement needs to enable the user to select previously answered questions.

- Being able to read the information at the beginning about what the Scikic is doing. Most users do not read information before using web interfaces[68] and so it is important to have that information in case the user decides they do want to read said information. Currently because it is faded away, there is no way to access the initial description of the Scikic after the user answers or skips questions. This will potentially drive users away since they cannot access the information they want.

For those reasons, the new interface will not fade away previous text after the user answers or skips questions.

Since the conversation's text in the prototype web interface fade away, it has not had to try to follow the useful user experience guidelines"[40](*section 2.2.2*). In the new interface, consideration should be taken to add white space between questions indicating natural pauses in a conversation, group questions and answers while at the same time distinguishing them. It is important to make sure there is no background noise by taking a minimalistic design approach to the conversational interface.

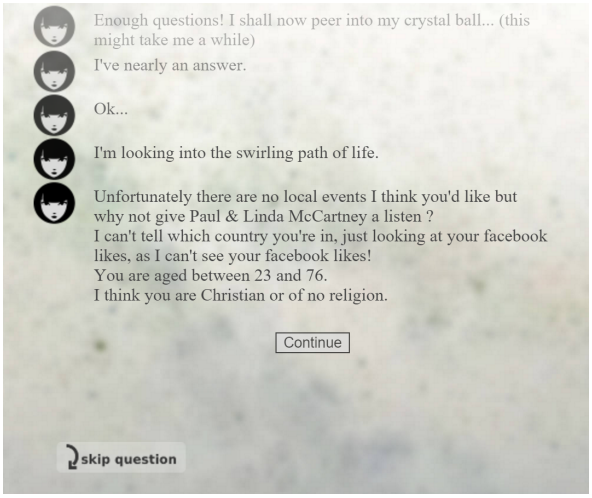


Figure 3.5: Scikic's insights after performing inference.

Once enough questions have been answered, the inferences from those answered questions produce insights to the user and display them based on the how likely the probability distributions are for particular features as shown in *figure 3.5*.

It is a requirement to show the text insights made by the Scikic either at the end of asking some questions or as soon as they become available in the JSON object returned by the API's */inference* endpoint.

The user can then choose to continue after the insights have been shown. There are a series of short questions which check the accuracy of the insights made(*figure 3.6*).

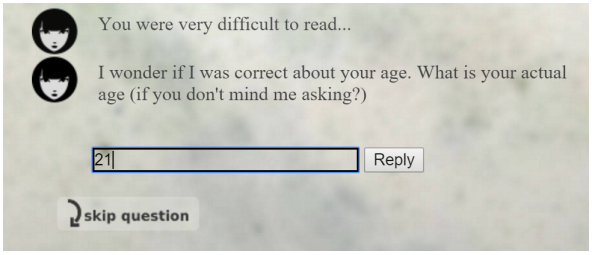


Figure 3.6: Scikic checks the accuracy of its insights to improve its accuracy for future users.

Two requirements can be identified from this. Firstly, to cycle through the user's insights and check their accuracy. Secondly, to create suitable storage for that information which can be evaluated at a later date that is potentially used by the Scikic. This is so in the future, the stored data could be used to improve the Scikic's accuracy.

The final requirement is to ask users to choose how the data that they have inputted to be handled, in terms of storage(*figure 2.7*). Following the prototype's current interface, it can either be deleted within 7 days, used to only improve the Scikic's future readings or the data can be anonymised and used for research in the field of personal medicine.

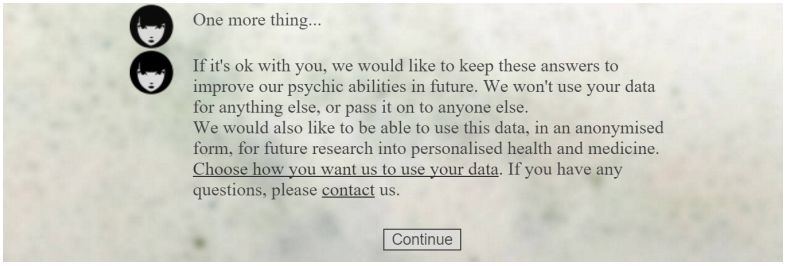


Figure 3.7: Scikic asks users to choose what to do with their data.

3.2.2 Comparison to other conversational interfaces

This section will critically review the conversational interface on whether the previously researched industry leaders' user experience techniques(*Literature Review section 2.2.3*) are appropriate to include in the new web interface as requirements.

Comparing the features from solutions in the industry, there are some improvements that can be made to the prototype's web interface. The first is a clearer chat typing awareness indicator, similar to *Facebook*'s(figure 2.8). There are some small loading bubbles that appear at the bottom of the screen while the Scikic is waiting to respond. It can be is easily missed though because they are not in line with the conversational interface and may be perceived by the user as unrelated.

While the initial description(*figure 3.1*) does tell the user at the beginning what the Scikic intends to do, this is quickly faded away from the user. Something like *Skype*'s user profile section(*figure 2.10*) being persistently shown in the conversational interface would be beneficial.

It is not completely clear which responses are the Scikic's and which are the user's. They are aligned on opposite sides of the conversation, however, it may be worth trying to encapsulate each question of the conversation and differentiating the Scikic's question from the user's response with some kind of coloured boxing(see figure 2.9).

The Scikic does a good job of using the picture of the Scikic as an embodied agent(*figure 2.11*), correlating it to the Scikic's parts in the conversation, there is little room for improvement there.

Due to the nature of how long it takes to perform the inference calculations, there is already processing time emulating human thought like in *figure 2.13*. It may be worth for the future, adding in a minimum processing time before showing a reply from the Scikic, in case the API does become more efficient.

Lastly, the conversation like experience from having a typing cursor which outputs words or letters individually(see *figure 2.12*) is debatable. People are likely to think this must be a robot because there is no other conversational interface in the industry that implements this functionality. On the other hand, the goal of the project is not to fool the user into thinking this is a human they are engaging, the goal is to provide a more conversational like experience. For that reason, this will be added as a requirement.

3.2.3 Conclusion

The implementation of the prototype's web interface has been critically reviewed which has justified the need to rebuild the conversational interface, due to having many clear requirements that the protoype's interface does not meet.

The section has outlined some possible technologies to replace the Python prototype web interface in order to meet *CitizenMe*'s requirements of modularity and decided that ReactJS was the most logical choice.

The rest of the analysis looked at understanding the current conversational flow of the prototype's interface and identifying requirements using the researched user experience guidelines as well as the established industry leaders' conversation user experience techniques from the *Literature Review*. The following requirements have been identified to be part of the new web interface's conversational module:

1. There needs to be a persistent initial description and explanation about what the Scikic is, similar to the *Skype* example(*figure 2.10*)

2. Keep the prototype's *embodied agent*.

3. The initial description should not fade away as in the prototype's conversational interface, to allow the user to revisit what the Scikic is doing.

4. Extra information should be easily found by the user if they wish to find out the motivation and creators of the Scikic.

5. The new interface will follow the prototype's initial question of whether they wish to continue which will follow the "this is a conversation, not an interrogation" model(*section 2.2.1*).

6. Instead of asking the user to log into *Facebook* before the user choosing to use the Scikic, only do so after the user has specified they wish to continue from 5. There also needs to be a clear justification for why it is beneficial to the user to log in with their Facebook account to follow the recommended criteria of a successful conversation(*section 2.2.2*).

7. Each question should type out letter by letter to make it seem more conversation like(*figure 2.12*).

8. There needs to be two types of 'answer types': free-flowing text and options for the user to choose. The type of answer the question requires can be retrieved from the question object that's returned after making a request to the API's */question* endpoint.

9. It is key to retain the ability to skip a question if the user does not wish to answer a question for whatever reason. If a question is skipped, the question should be clearly marked as skipped.

10. Previously answered/skipped questions should not fade away as in the prototype. This not only fades away the initial description making it impossible for the user to go back and read it but more importantly, it contradicts the 8th visualization requirement(*section 3.1.5*) which requires the user to be able to select previous questions.

11. Create a clearer 'typing indicator' inspired by *Facebook*'s example(*figure 2.8*).

12. Add a minimum processing time in case the API's inference becomes more efficient in the future to emulate human thought(*figure 2.13*).

13. Question and answers should be grouped together but clearly distinguished from each other, perhaps using some clear colouring like in the *Whatsapp* example(*figure 2.9*).

14. Each grouped question and answer should have white space between them indicating natural pauses in the conversation.

15. Have a minimalistic design for the conversational interface to ensure there is minimum background noise stopping the user from getting distracted.

16. Show text insights about the user at some point during the conversation with the Scikic.

17. At the end of the questions, verify how accurate the insights from the Scikic were by asking the user.

18. Store the accuracy of the insights as told by the user from requirement 17.

19. Allow the user to decide how long, if at all, their data is stored.

3.3 Final requirements

The overall aim is to implement two different modules into an extendable fundamental application. The project realistically wont be able to meet every identified requirement in the available timescale, however, the code base needs to be able to prove that it can be easily extended in order to meet any of the remaining requirements in the future. The final implementation will combine the two modules in the new web interface without hindering each of their requirements.

3.3.1 Visualization module

The module is based on information from the Scikic's inferences and inputs from the conversational interface which will be displayed to the user real time and transitional updates when new questions are answered. This is to show the user how the Scikic works with the aim of helping the user understand how *Bayesian Networks* and *collaborative filtering* work.

The measurable requirements have already been outlined for the visualizations module in *section 3.1.5*.

3.3.2 Rebuilding the prototype conversational interface module

Based on the analysis of the prototype's web interface, rebuild Scikic's prototype interface to an independent conversation module which ask users questions from the API's */question* endpoint and sends answers to the API's */inference* endpoint so the interface can eventually receive personal text insights to display.

The measurable requirements have already been outlined for the conversational interface module in *section 3.2.3*.

3.4 Evaluating the Scikic's web interface

The success of the project will be evaluated based on the number of the requirements that were met(*Final Requirements*) and proving the remaining unmet requirements can be easily added to the code base in the future. Since this is an implementation project, the aim is not whether the user understands BNs better after using the Scikic but to create a strong foundation to eventually be able to meet that goal in the future.

Automated unit testing within the code will be able to verify any of the requirements that manipulate data or require objects to be in certain forms. An example requirement that can be met with unit testing is creating and manipulating the objects: *Questions*, *Facts* and *Unprocessed*(requirements 3 and 6 in the Visualizations module *section 3.1.5*)·

The second testing method will ask questions a small group of independent users about their experience using the new web interface. This will make sure that visualizations and conversational interface are meeting the requirements that cannot be met with unit testing alone. This could also be achieved by manual testing using before and after screenshots for each requirement which involves rendering objects to the browser. Manual testing can easily spoofed though, especially with the wide availability of image editing software. Testing with independent users from the project is more effective because the results are harder to fake and they can also potentially give ideas for possible future improvements from their feedback. The exact questions that will be asked will be developed after the implementation since they will depend on how many of the requirements this project is able to meet in the available timescale.

4 Design

This chapter is a similar format to the *Requirements and Analysis* chapter. All the explored alternative designs will be shown using simplistic wire frames created using `https://moqups.com/`. They are meant to clearly envisage the structure and content of the new web interface. The content within the designs will indicate what they would represent using the Scikic's data, however, the designs will not be using real data as this would make the design process unnecessarily long; having to retrieve and organise the Scikic's data into a 3rd party application that will not contribute to the new web interface would be a misplaced use of time. The goal of the web interface is to use a final design that is data driven and so it should not matter what data is used within the designs.

The first two sections will present alternative designs for the visualizations module and the conversational interface module. Comparing alternative designs on how well they meet the identified requirements from *section 3.3*. The final design will be justified for each module to be implemented in the new interface based on each design's compatibility with their requirements.

An overall structure of the web interface will be designed, making sure that none of the module's individual requirements are hindered.

Finally, a Unified Modeling Language(herein, UML) activity diagram will describe the possible user interactions with the Scikic's new web interface to be able to correctly implement the required actions within the new web interface.

4.1 Visualization module design

The analysis has shed light on the fact that the visualizations will largely be based on the Scikic's Bayesian Network(herein BN) which represents the relationships between the user's 'features'. A feature being the user's age or religion for example.

The analysis of the returned inference data from the API(*section 3.1.3*) shows that there are two relevant objects to display the BN in the visualizations module. These are the *Relationships*, which show how the user's features are related, and the *Features* probability distributions, that are a result of the inference's relationships between the user's features.

Unfortunately, this means that there is currently no available data to visualize the collaborative filtering process and so despite understanding what collaborative filtering is(*Literature Review section 2.1.2*), it will not be incorporated into the visualization module's design.

The current data that the web interface receives from the Scikic means that the design will be limited to showing the user's BN. This will show the user what features have been inferred but is not able to show how the user's features were inferred specifically from their given data sources(for example, against other 'users'). Knowing that BNs are a set standard(*Literature Review section 2.1.1*) in terms of their visualization, the visualization's design will be a BN be at its core(*figure 4.1*). This core BN meets the 4th requirement from *section 3.1.5* to have some kind of node representation which are connected based on their *Relationships* returned from the API.

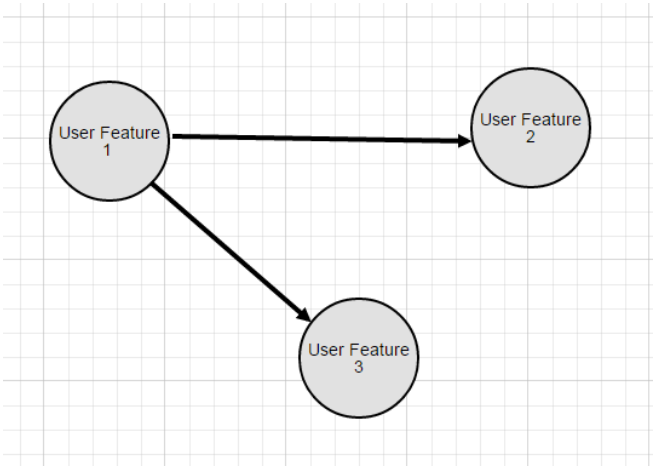


Figure 4.1: A simple BN which the alternative designs will be based on.

There is not much room for modifying the BN visualization, however, the *Feature* probability distributions received from the API does have the flexibility to create alternative designs with. The alternative designs explore how to integrate the feature probability distributions into the BN while meeting the requirements for the visualization's module.

The *Feature* probability distributions represent the discrete possibilities that any of the user's feature could be. A user's gender could be male or female and so the probability distribution will have values for each of these possibilities as categories. The *Features* could be shown using a number of graph types:

- Bar chart
- Scatter plot
- Line graph
- Pie chart

There are many other graph types that could be explored but given *D3*'s main capabilities and the timescale available, these graph types would be the most feasible for the visualizations.

Since the *Feature* probability distributions are discrete data, even in the case of age since the Scikic will only return a subset of age probabilities between 1-100, the data should be visualized using a type of graph which does not suggest any kind of influences between feature categories. Thus, the suitable graph types are either a bar chart or a pie chart to display the *Feature* probability distributions. The alternative designs will use one of these graph types while building on the simplified core BN design from *figure 4.1*. Each of the alternative designs will discuss how well they can meet requirements 5 and 7 from *section 3.1.5*.

Alternative design 1:

The design(*figure 4.2*) uses a bar chart to represent the feature distributions that are connected to each of the feature nodes. While a connection can be made between the *Feature* probability distribution and its node, it is a lot of information floating about in an unstructured way. This can potentially cause confusion to the user.

A problem with having the bar charts follow the nodes is that it is not trivial to implement as the bar chart has to animate to follow the nodes. That can have consequences on performance since the bar charts are another moving part, but also, on the complexity of the implementation since the bar charts

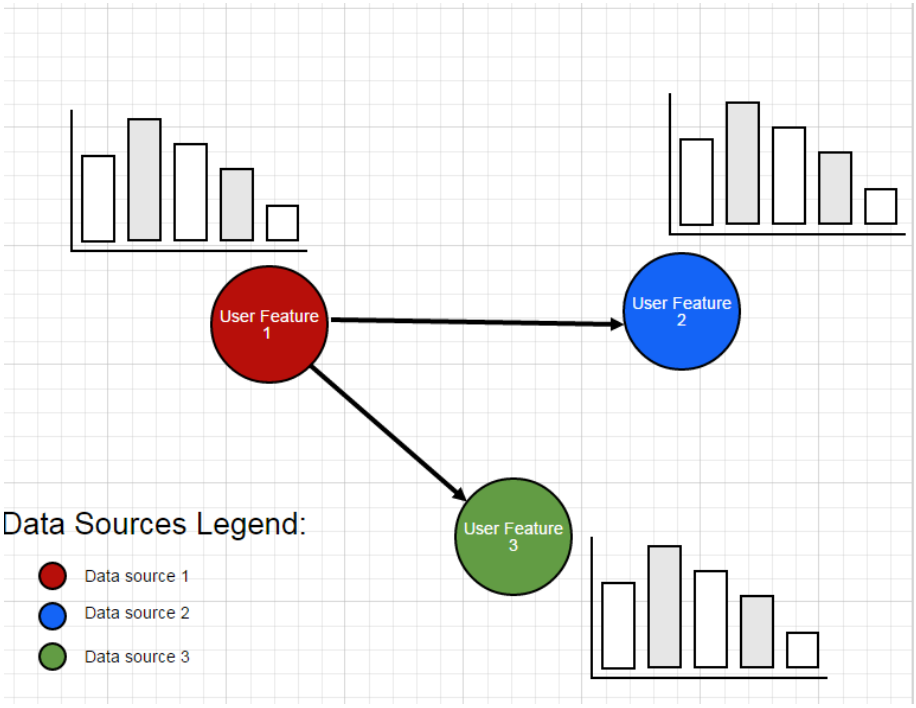


Figure 4.2: Visualizations module alternative design 1

and nodes can not be encapsulated and need to share dynamic positional information.

The bar charts being connected to each of the nodes does mean that there is one less visual characteristic(such as differently shaped nodes) needed to associate them. This means that the nodes can be colour coded by data source used for inference without causing conflict with another type of distinguishing characteristic for its associated *Feature* probability distribution.

When a transition occurs from answering a new question(requirement 7) the bar charts will update and any new bar charts and nodes will be added. Due to the nodes and bar charts being intimately coupled on the screen, even in this small example, it's easy to see that this visualization could quickly become cluttered. There will also have to be extra consideration in the implementation to make sure that any newly added *Features* do not accidentally overlap and cover up any other existing bar charts on entering the visualization, otherwise, this could cause the user to miss the transition of the existing *Features*.

Overall while this design has a good way of differentiating the *Features* and their respective data sources, the costs are code complexity, potential performance impact and potentially more bugs related to the transitions when the user answers new questions.

Alternative design 2:

The second design(*figure 4.3*) also uses bar charts for the feature distributions. They are related to their nodes via a colour coding and headers.

There is a clear separation between the *Feature* probability distributions and the nodes, meaning that they can be easily extended or changed to other graph types in the future without causing unintended side effects.

Differentiating the nodes' data source using different shapes and its *Feature* probability distribution using color coding meets Requirement 5 successfully. There could be a limitation where if any of the nodes has more than one data source, it will be more difficult to display that. A potential solution is to have inner shapes, so a node could be a circle and then have a diamond inner shape to show it is from

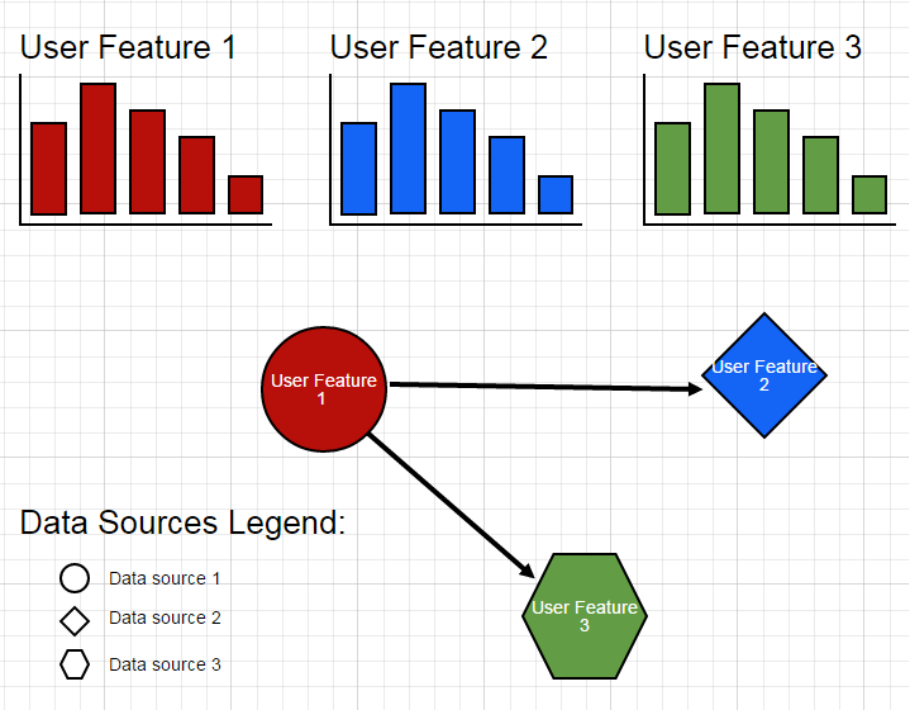


Figure 4.3: Visualizations module alternative design 2

two different data sources.

Needing two types of differentiators is a trade-off for decoupling the *Feature* probability distributions from the nodes. This does make it slightly harder for the user to initially understand what the node's properties(feature and data source) are.

The transitions would be clear because there is allocated space for bar charts to be added on the top row and there is enough space for nodes to be added without considering where available space is. This means that there is less chance that either the nodes or the bar charts blocking one another causes the user to miss the transition.

The overall visualization has a more consistent structure and is more space efficient due to decoupling the nodes from the feature distributions. This comes at the cost of being able to associate the node's data properties with a single visual characteristic(such as colour). As a result, it is a more achievable design within the bounds of *D3*'s capabilities. This design would help the code base be more manageable and flexible to changing the visualizations compared to the other designs.

Alternative design 3:

Instead of using bar charts, this design(*figure 4.4*) uses pie charts to represent the feature distributions as nodes. They are identified with features using headers above the nodes. In combination with changing the border colour, requirement 5 to associate data sources and the *Feature* probability distribution has been met.

The *Feature* probability distribution and nodes are one and the same with this design. The problem of being able to identify a node with multiple data sources still exists. Similarly to design 2, this could be solved by having a multi-layered coloured border(for example Feature 1 could have a red and blue border to represent it has been inferred from two different data sources).

Using a pie chart instead of a bar chart is beneficial in this design because it creates a suitable shape to combine them with the nodes. The data is also compatible with pie charts because the probabilities

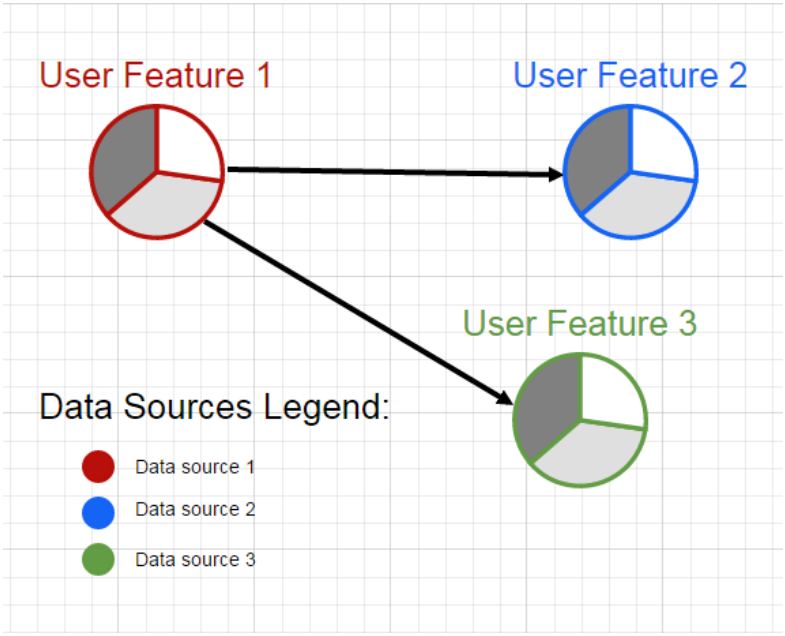


Figure 4.4: Visualizations module alternative design 3

should add up to 1 or be less than 1 since it is not possible for a feature to be any more certain than 1(100%). Problems could arise with features such as age where it is possible to have up to 100 different categories meaning that there could be up to 100 slices all of which may only add up to less than 5%. In that case it would be very difficult for the user to actually be able to the difference in the inference's results about the user. If the data was displayed in a bar chart, that chart could dynamically scale to the data.

Transitions could also pose a problem, since a slice covering a larger surface area is less noticeable than a bar increasing or decreasing in height. This is especially true when the data is not scaled and add up to say 5% as previously discussed. On the positive side, there is almost no concern about newly added nodes colliding since there is a lot of space for them to be added into a free position. This is because there is no extra space being taken up by related *Feature* probability distributions in the visualization.

4.1.1 Choosing a final design

More alternatives were considered while developing the designs for the visualizations module. These three designs have been included and are compared in this section because they have the fundamental variations from all the other considered designs.

All three alternative designs meet requirements 5 and 7 successfully but vary on how well they each meet these requirements. To be able to decide on the most appropriate design, their trade-offs will be compared. The most important trade-offs that have been identified include:

- How easy it is to see the transition after each new question is answered(related to requirement 7).
- Limitations with the clarity of each node's properties(the feature it is identifying and its data sources) which is a factor of requirement 5.
- The level of coupling between the feature distributions and the nodes, since this contributes to the overall technical aim of having a manageable and flexible code base.

After considering design 3's use of a pie chart instead of a bar chart, it is a major problem when the

inferred categories of the feature probability distribution's values are much lower than 1. As discussed in *alternative design 3*, the user may not be able to see the slices representing each category for some of the features and during transitions between questions, the changes may not be noticeable enough compared to a bar chart. Design 3 will be ruled out as the final design since the main purpose of the visualizations is for the user to be able to see the inference values and changes in them during transitions.

That leaves designs 1 and 2. The main trade off between the two designs is that Design 1 is better at clearly associating the node's feature probability distribution and their data sources, where as with design 2, there is a lower risk of blocking updating feature probability distributions accidentally when transitioning between questions because of it's more strict structure.

The core goal of the visualisations is to show the user how answering questions produces a new BN and updates to the feature probability distributions. While it is a requirement to be able to clearly show the data source of each feature to be transparent with the user and gain their *trust*, it is not as high a priority as the core goal of the visualizations. Design 2 still meets requirement 5 successfully, just not quite as clearly as design 1. The structure allows the visualization to update in a consistently anticipatory way(where new features are less likely to accidentally block each other) and gives the flexibility to change the bar charts to different graph types because they are completely independent from the feature nodes in design 2, outweigh the increased difficulty for the user to quickly distinguish the nodes' data source in design 1.

The final design for the visualizations module will be design 2.

4.2 Conversational interface module design

The conversational interface's requirements have been derived by analysing the Scikic's Prototype interface(*section 3.2.1*) and using the industry leader's user experience techniques(*section 2.2.3*) as inspiration for features to add to the new interface. This has made the requirements very detailed(*section 3.2.3*) with little room for drastically different designs. This section will design each of the requirements to mosaic a final design, discussing alternative designs where appropriate.

The first 3 requirements are to do with the initial description, using the Skype example as inspiration(*figure 2.10*), the Scikic's equivalent could look like *figure 4.5* depending on the layout of the available screen space(*discussed in section 4.5*).

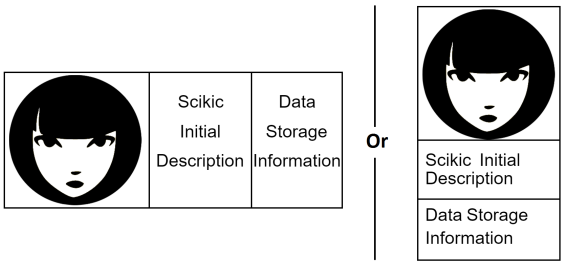


Figure 4.5: Scikic's Skype like profile.

The initial description would show above a question list and be persistently on the screen without fading. This meets all 3 of the first conversational interface requirements(*section 3.2.3*).

Similar to the prototype's interface, before the Scikic asks questions, it should allow the user to choose whether they are interested in using the Scikic(requirement 5). The new design could have the initial

question as part of the question list, where once the user answers, the conversation carries on. However, there is no need for the user to be able to see the initial question ever again because it does not contribute to the visualizations or any other aspect of the web interface. It makes sense to remove the initial question from the interface after the user has shown their interest. This will help to ensure there is as little background noise as possible(requirement 15) in the conversational interface that could distract the user.

If the user is not interested, type out a message saying "Thank you in your interest, the Scikic hopes to get another chance to convince you in the future.". Also to make it easy for the user to still be able to use the Scikic, there will be a "changed my mind" button which would allow the user to start using the Scikic as if the user had chosen to be initially interested.

The core feature of the conversational interface is the question list when the user shows their interest to use the Scikic. The first question that will be asked, is whether the user would like to login to *Facebook* to get more accurate results(*figure 4.6*). This could have just been a pop up like in the prototype but the alternative of letting the user choose whether they want to be shown that pop up would meet requirement 6 better.

Figure 4.6: Proposed Facebook login after a user has shown interest in using the Scikic.

Each question will be typed out character by character meeting requirement 7. There will be 3 answer types(*figure 4.7*) to meet requirement 8. One to input free flowing text, a second to click a button when there are 4 or less choices and finally a dropdown selection for when there are more than 4 options. The reason that the design will use a dropdown instead a large number of buttons is to save on space. Showing 10 different buttons(for example for each country) will quickly fill up the screen, making it more difficult for the user to choose an option hindering requirement 15 to minimize background noise.

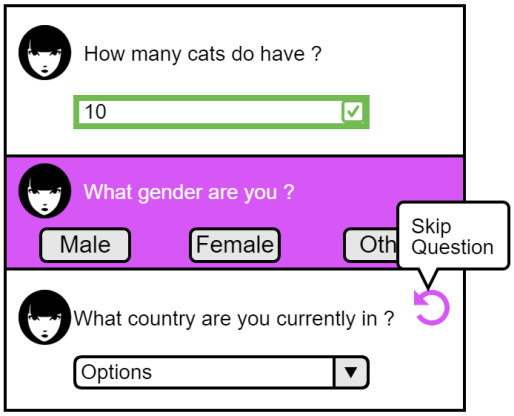


Figure 4.7: Scikic's Skype like profile.

Question states will be easily distinguishable such as in *figure 4.7*. Answered questions will be clearly checked and green. Skipped questions will be highlighted in purple to make it clear to the user that they can always go back and answer the skipped question.

To skip a question(requirement 9), there will be a button on the right hand side of the question that has a tooltip to explain that it skips a question on hover over. This is again to minimize the space used for requirement 15. After exploring alternatives, it was more beneficial for the interface to not have the words "skip question" directly on each question than the alternative of taking up more space.

The rest of this section will compare two alternative designs for the question list's structure to be able

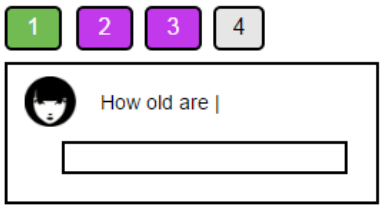
to navigate the questions. A final design will be chosen, based on meeting requirements 10, 11 and 13-15.

Alternative design 1:
The first design is using a simple list structure like in *figure 4.7*. Having a never ending list going down the page would make it inconvenient for the user to be able answer questions and view the visualizations since the length of the questions is likely to overflow past the end of the visualizations. To resolve that issue, there would be no difficulty in adding a vertical scrollbar specifically for the question list, so only a certain number of questions are shown at any given time.

To show the user clearly which question is selected, the questions that are not selected would be slightly less prominent by lowering their opacity to a constant. While this means that the questions are faded which contradicts requirement 10, the questions will only faded at a constant, rather than being completely faded. This still allows the user to select previous questions since no question will be fade away completely, therefore meeting the intention of requirement 10.

Questions have enough whitespace to indicate natural pauses in the conversation(requirement 14) and can easily be styled for the question and answer to be grouped together for each question(requirement 13). The design gives the user more information by displaying all the questions in the list, perhaps making it less minimalistic, however, this allows the user to be able to select previous questions with confidence that they are selecting the question that they wanted to.

Alternative design 2:



Rather than showing every question in a list, this design only shows one question at a time. This meets the requirements since the user still has the ability to select previous questions by selecting the numbered buttons above the shown question. Skipped and answers questions are still clearly distinguishable using colouring the same way the other proposed design does. If the user skips a lot of questions or there are many questions asked, there would be similar problem to the first design because the question selectors could widen the conversational interface. This can be tackled in a similar fashion by having a horizontal scroll bar for the question selectors.

Requirements 13 and 14 are met by default since the other questions are not shown. This means there is no need to add whitespace to indicate pauses in the conversation or to group the question and answer since those are the only two elements shown at any given time.

4.2.1 Final design

Design 2 takes a minimalistic approach to the extreme, making the user completely focus on the current question, while still allowing the user to be able to select previous questions. Design 2 clearly leans more towards minimizing the amount of background noise(requirement 15) more than design 1 and other requirements.

Design 1 on the other hand deliberately tries to show as much information as possible about each question. This approach detracts attention from the current question waiting to be answered, however,

the interface is meant to be heavily orientated around exploring how each question answered affects the visualizations. Design 1 is superior in letting the user select previously answered or skipped questions with the confidence that the user is selecting the question they had wanted. A user selecting a previous question in design 2 has to remember what question was asked and what answer they gave.

Both designs meet all the requirements to some extent, however, they each focus more heavily on a different requirement but neither choice would be wrong. The final choice is Design 1 because while minimizing the amount of noise is important(requirement 15), there is no requirement favouring a more mobile friendly design. There are a number of requirements focused on giving users the ability to switch between questions easily. Design 1 gives all the information about questions without having to remember them like in design 2.

4.3 Combining the visualizations and conversational interface modules

Final designs have been proposed for the visualization module(*section 4.1.1*) and the conversational interface module(*section 4.2.1*). The project's overall aim is to create a single web interface with both these modules clearly shown.

This section will present two alternative layout designs with the goal of choosing a final layout design to implement as the new web interface. Both of these designs will incorporate the menu(*figure 4.8*).

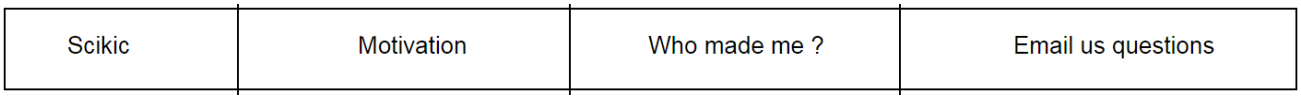
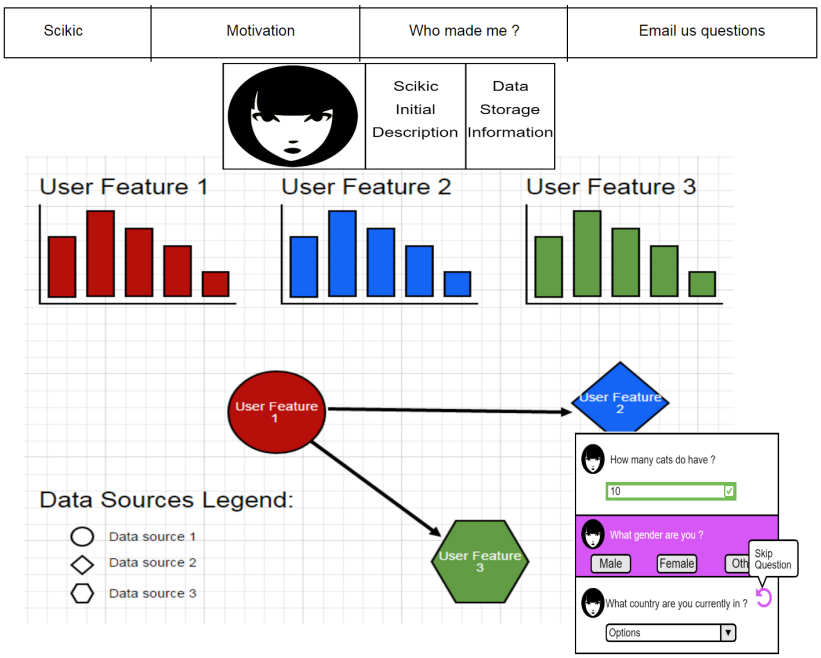


Figure 4.8: Basic menu design for the Scikic.

The menu is to allow users to easily be able to navigate to information about the motivation and creators behind the Scikic(*section 3.2.3 requirement 4*).

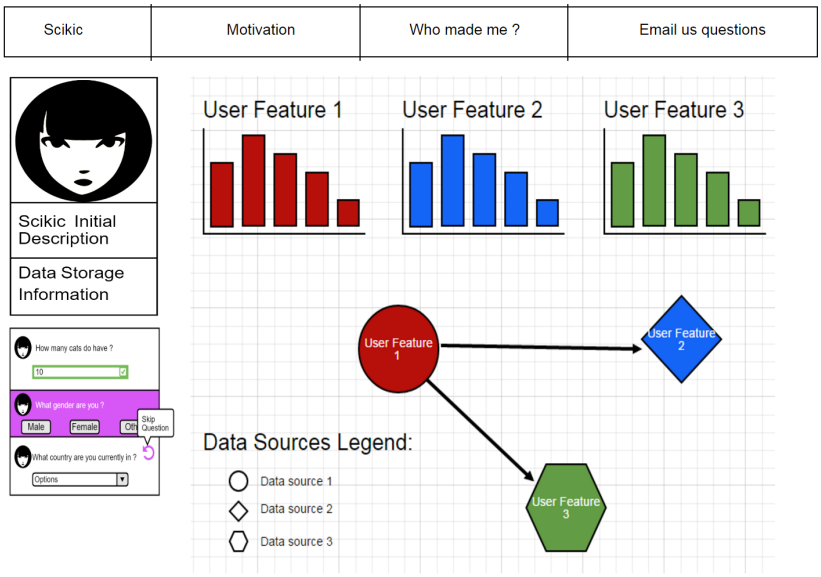
Alternative design 1



This design's layout maximizes the spaces that is used for the visualization module by using the whole

width of the screen. The conversational interface overlays the visualization which can be hidden by the user. There is no space for the Scikic's description to be part of the question list and so, as a result, it is displayed at the top of the interface instead.

Alternative design 2



This layout takes a more structured approach by keeping the Scikic's initial description and the question list together. This comes at the cost of the visualization module being slightly smaller than it could be if it took up the whole width of the screen.

4.3.1 Final layout design

Both of the proposed designs combine the two modules that will be included in the web interface successfully. Design 1 has a layout that fully utilizes the screen space to display the visualization module while design 2 takes a more structured approach by clearly giving part of the screen space to the conversational interface module.

Even in the module design, the overlay conversational interface shows that it could unintentionally cover up part of some of the node's visibility. Design 2 does not have to deal with that issue and its structure means that the interface's elements are more spaced out making it easier on the eye.

While design 2 is a more 'chat like' layout similiar to chat applications such *Facebook* uses, there is no need for the conversational interface to overlay the visualization module since there will only ever be a single conversation at any given time. That means that the rest of the bottom horizontal space would be used inefficiently.

There currently is no requirement to make the new web interface mobile responsive but with the exponential growth of mobile usage to the internet, design 2 would be the easier layout to make responsive in the future.

Overall Design 2's more structured approach is a better design and so is the proposed final layout for the new web interface.

4.4 Designing the user's possible interactions with the new web interface

To show all the possible user interactions with the interface, this section uses a UML activity diagram. The activity diagram is appropriate since it captures the dynamic behaviour of the new web interface. It clearly shows the sequence of events and interactions, as well as the necessary decisions the user has to make to finish using the Scikic(*figure 4.9*).

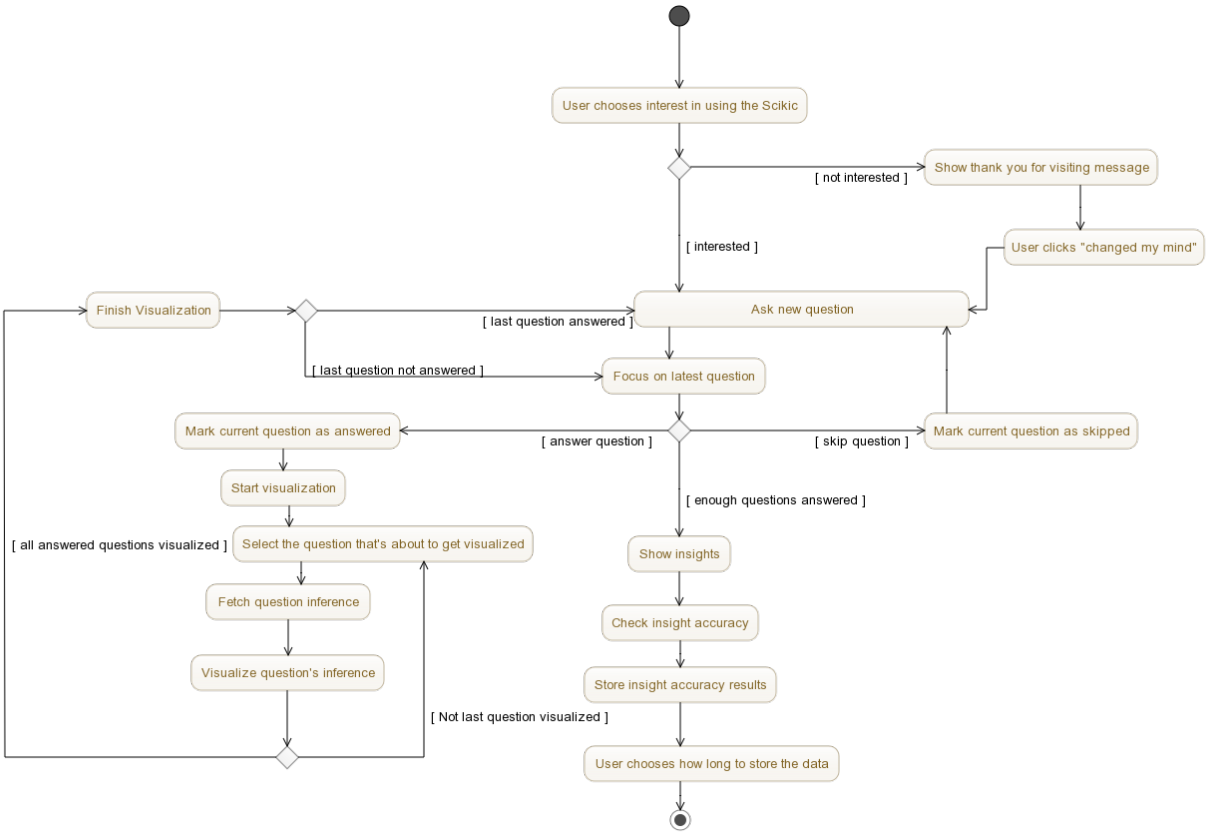


Figure 4.9: UML diagram showing the possible user interactions from entering the interface to showing the user's insights

The user starts by choosing whether they are interested in using the Scikic. Even if the user chooses they are not interested they still get the option to use the Scikic in the case they change their mind without having to refresh the page(requirement 5 *section 3.2.3*).

The user will get asked the Facebook login question only after the user has indicated they are interested in using the Scikic(requirement 6 *section 3.2.3*). The *Facebook* login question is represented by 'Ask a new question' activity in *figure 4.5* since there's no reason to distinguish it from other further questions. Consequent questions get asked from the API's */question* endpoint(requirement 2 *section 3.1.5*). The user is given the option to skip the question and if they do, the question will be marked as skipped(requirement 9 *section 3.2.3*). If the user chooses to answer the question, the web interface will mark then question as answered and be notified to *Start visualization*. the user will not be able to answer or skip any other questions until the visualization has finished. This is to avoid the visualizations trying to visualize a different question than originally requested which could potentially lead to unintended side effects and conflict with the visualization's module requirement 7(*section 3.1.5* of clearly showing the question's inference transitions.

Once the visualization has started, the interface will make a POST request to the */inference* end-

point(shown with the 'Fetch question inference' activity in *figure 4.5*). This meets requirement 3 in the visualization's module(*section 3.1.5*). Once the inference has been returned, the data will be visualized with the proposed final design for the visualization's module(*section 4.1.1*).

After the originally answered question by the user has been visualized, the web interface will then decide if that the latest question that was answered ? If it was not the latest question answered(latest being the highest numbered question), then repeat the same process of fetching and visualizing the later questions. This is because if an earlier skipped question was answered or one of the previously answered questions was edited, then the inference's would be changed for any later questions. To make it clear which question is currently being visualized there will be a label in the visualizations module and the question will be selected by having full opacity(described in *section 4.2, alternative design 1*). Visualizing the latest question answered will trigger the 'finish visualization' activity in *figure 4.5*.

The web interface then checks if the last question displayed has been answered. If it has been answered then go back to 'Ask a new question' otherwise focus on the last question.

Finally, if by a flexible condition which indicates enough questions have been answered, show the text insights to meet requirement 16 in the conversational module's requirement(*section 3.2.3*). Ask how accurate the insights were to the user(requirement 17 *section 3.2.3*) and store the insight's accuracy(requirement 18 *section 3.2.3*). As the last activity, the user can choose how long to store their data for(requirement 19 *section 3.2.3*).

5 Implementation and Testing

The final implementation for this project has been completed and the code can be found on *Github*(`https://github.com/VasilyShelkov/scikic-app.git`). It has been open sourced to allow users of the Scikic's new web interface to view how the project works technically. There are no intellectual property or confidentiality concerns because the rest of the Scikic project is also open sourced on *Github*(`https://github.com/scikic`).

This chapter will explain on a high level how the final designs(*Design* chapter 4) have been implemented using the chosen JS libraries(*Requirements*, *sections 3.1.4 and 3.2.1*).

The first section shows how *ReactJS* has been used to implement the overal goals in modularized 'components' and how they were combined according to the final layout design(*section 4.3.1*).

The next section then goes on to explain why ReduxJS was appropriate to implement the UML Activity Diagram's(*figure 4.6*) possible user interactions in the web interface. Specifically, how the interactions have been decoupled from the components in the first section of this chapter which has made it easy to unit test the interactions and ultimately fulfil the requirements related to the interactions.

Since one of the main motivations(*section 1.2*) of rebuilding the Scikic's prototype conversational interface was to make the new interface's code flexible and easily extensible, the third section will briefly go through some of the key technologies used to ensure a high quality code base during the development process.

The implementation of the new web interface has been deployed to http://scikic.org, replacing the prototype web interface. The steps taken for deployment will not be discussed in further detail since it is not relevant to the report.

The testing part of this chapter will go through the two different testing methods(described in *Requirements section 3.4*) used to prove that the Scikic's implementation meets the requirements and works:

- Automated unit testing which verifies the implemented requirements related to any kind of data manipulation.
- User testing which analyzes a set of 10 questions that were asked to users to verify that the requirements which include rendering to the browser, have been met successfully.

5.1 Implementing the final design with components using ReactJS

ReactJS is *Facebook*'s Javascript framework. By using ReactJS to build the new web interface, it has greatly improved the developer experience and the flexibility of modifying the Scikic's new web interface.

This section will show how the final designs in *Chapter 4* have been converted to encapsulated independent 'components' that can be easily changed to a different layout if necessary. Having the core modules encapsulated in 'components' has also meant that it has been easier to focus on their particular

requirements, as well as, making it easy for any unfinished requirements to be added to those modules in the future.

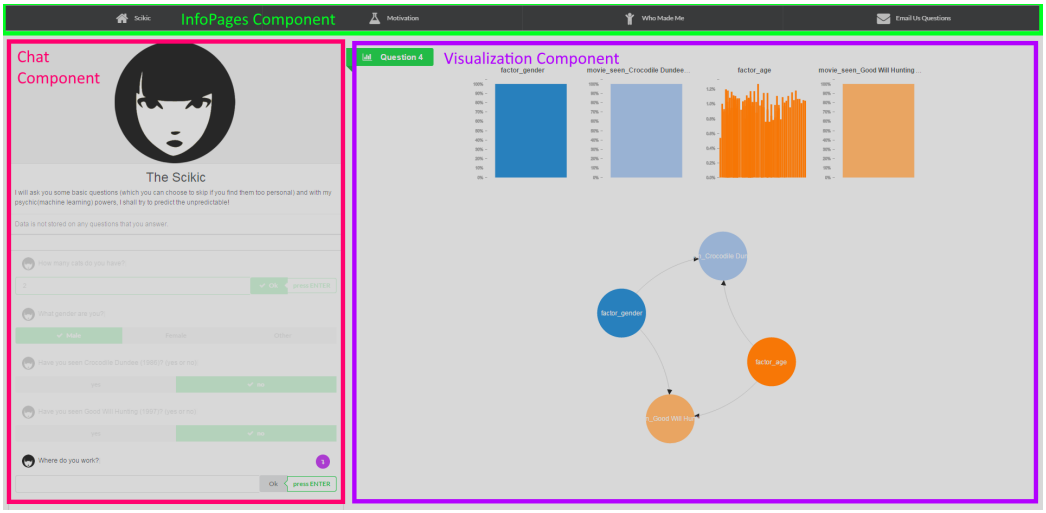


Figure 5.1: The new interface overlayed showing what components it includes

The final layout as shown in *figure 5.1* has been broken up into 3 different components:

- InfoPages - Navigation to the extra information related to the Scikic.
- Chat - This is the conversational interface module's component(*Design section 4.2.1*).
- Visualization - This is the visualization module's component(*Design section 4.1.1*).

The rest of the section will show more depth about how these 3 components have been broken down further to implement their respective final designs. All the labels for the border highlighted components correlate to their implemented components in the code.

5.1.1 InfoPages Component

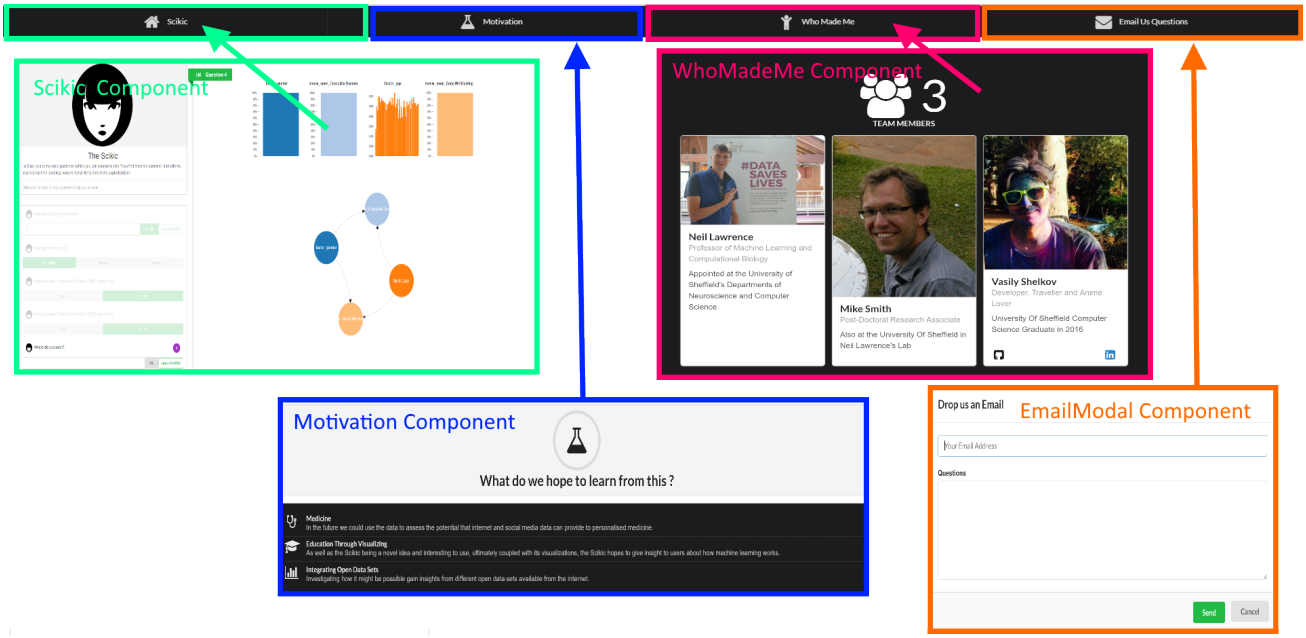


Figure 5.2: All the different info pages which contain extra information about the Scikic.

Strictly speaking the navigation bar that was highlighted as the InfoPages component in *figure 5.1* was not accurate, it is actually the Navigation component. The Navigation component consistently stays on the top of the interface and only the content below it changes when one of the *Links* are clicked. There is an InfoPages folder which contains the components for each of the implemented InfoPages(*figure 5.2*).

To navigate to each of the conponents, A JS library called *React Router*[76] has been used. Each of the buttons in the Navigation component *Link* to one each of the 4 different InfoPages. When a *Link* is clicked, the particular InfoPage that it is related to is displayed below. The URL also changes to the particular component's route. For example, when the Motivation component is navigated to, it changes the URL from http://scikic.org to http://scikic.org/**motivation**.

The default component that is shown upon entering the website is the main Scikic interface(top left *figure 5.2*). This page contains the core of the project and its components are discussed further in later sections.

The Motivation component accessed at */motivation* displays extra information about why the Scikic exists.

The WhoMadeMe component accessed at */who-made-me* shows the user who were the creators of the Scikic project.

Both the Motivation and and the WhoMadeMe components are extra information part of the requirements(*section 3.2.3 requirement 4*).

Finally the EmailModal component allows the user to send an email to the creators for any questions or queries the user might have. While this was not a strict requirement, this is an improvement on the prototype conversational interface which just displayed an email address to contact them.

5.1.2 Chat Component

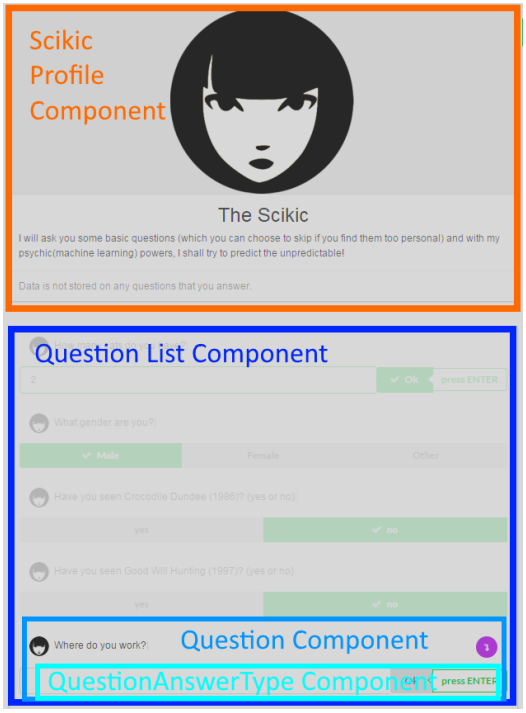


Figure 5.3: The components in the implemented conversational interface otherwise known as the Chat Component.

The Chat component is the implementation of the conversational interface's final design(*Design section 4.2.1*). Like the design, the Chat module is made up of two subcomponents.

The Scikic Profile Component is the description of what the Scikic is doing as well as showing how data will be stored. *CitizenMe*'s only requirement in this project was to make the Chat component completely modularized so it could be used in their own web interface. To fulfil the requirement, the Chat component can be passed in a 'Profile' component. In the Scikic's case, that is the Scikic Profile component. *CitizenMe* is unlikely to want to use the Scikic's Profile which is why it is not included in the Chat component.

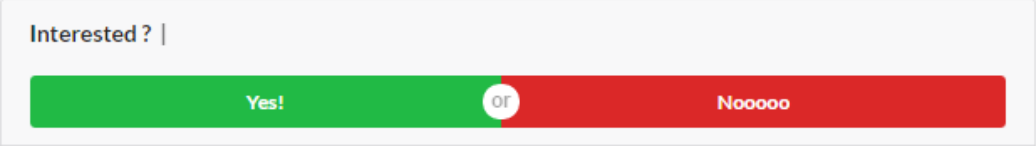


Figure 5.4: The InitialQuestion component rendered in the QuestionList component asking the user "Interested?".

The questions themselves are then displayed in the QuestionList component. The QuestionList will render one of three different sub-components. Initially when the user starts the Scikic, they get asked the InitialQuestion component(*figure 5.4*). If the user chooses they are "interested" then the QuestionList component will render a list similar to the one in *figure 5.3*. Otherwise, they will be shown a "thank you for using the Scikic" message(*figure 5.5*).

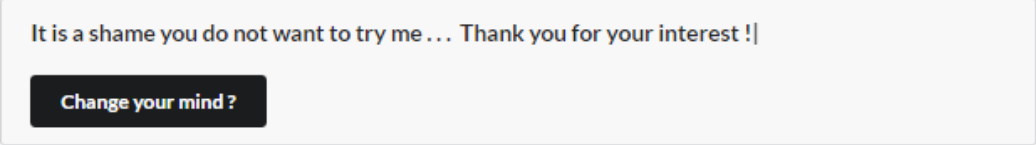


Figure 5.5: The component that is rendered if the user chooses they are not interested in using the Scikic.

Each Question component is mapped into the QuestionList component which takes the question text to display and also contains a sub-component called QuestionAnswerType.

The question text is typed out letter by letter using a ReactJS specific library called *React-Typist*[77] which provides a ReactJS component to type out the letters as soon as it is mounted onto the screen(*section 3.2.3 requirement 7*).

The QuestionAnswerType component will render differently depending on whether the passed down question data's type is 'text' or 'options'. If the type is 'text', then a text input box will be rendered. Otherwise, it checks how many different options there are for the question answer. When there are more than 4 options, the component renders a dropdown with the available options; 4 or less options, then render them as 4 buttons that can be selected. These 3 different answer types follows the expected design(*section 4.2*).

5.1.3 Visualizations Component

The Visualizations component will initially have a placeholder with a graph logo before any questions have returned any inference data to show the user that the empty space will be used and is not being wasted.

Once the user answers some questions, they will have something similar to *figure 5.6*. The Probability-Chart Components each gets the feature probability distributions passed down to them and *D3* renders

Figure 5.6: The components in the implemented Visualizations Component.

them as bar charts with the name of the feature above it and scaled to the data. For example, the 'factor_age' feature in *figure 5.6* only has a y-scale up to 1.2%. The FeatureNetwork is passed down the nodes and their 'relationships' from the inference object for that particular question. D3 then renders a FeatureNetwork similar to the one in *figure 5.6*. Some of the nodes do not fit their node names properly and so overflow out of the node. This is due to the user's features being named too long on the Scikic API.

5.1.4 Using D3 in React for the Visualization component

To understand how D3 was integrated into React, first there needs to be an understanding of what React Component's lifecycle is and how they can cause problems for D3.

React has a number of lifecycle methods to allow developers to manipulate the components in certain ways at certain points in the component's 'life'.

The key method of a React component is the 'render' methods. This is the method that tells ReactJS, "this is what I want shown in the browser". The 'render' method is invoked every time that the component's properties are changed and returns a Java Serialization to XML(JSX) object. This object is a HTML structure which can include other React components. The reason the 'render' method uses JSX is to process any included React components and HTML elements into pure javascript[29].

The problem with using D3 in ReactJS is that both these libraries try to manipulate the browser's Document Object Model(DOM which is where the HTML elements are rendered). This means that if D3's <svg> and other inner elements are directly added to the render method, there is no way to use D3's transition API which was one of the main reasons the new web interface chose to use D3(*Requirements and Analysis section 3.1.4*). The transitions do not work because every time the properties update(when new feature probability distributions are added/changed to the ProbabilityChart component or new nodes are added to the FeatureNetwork component), the render method re-renders all of the D3 elements, replacing all the old D3 elements meaning they can not be selected and therefore cannot be transitioned in the DOM.

The component goes through a cycle where methods are executed at certain points where the 'render' method is the pivotal method. The lifecycle methods needed for the solution to integrate D3 with ReactJS are:

- **componentDidMount** - invoked only once after the initial 'render' of the component.
- **componentDidUpdate** - invoked every time after the 'render' method has executed apart from the initial 'render'.

There are other ReactJS component lifecycle methods[74], however, for this solution they are not relevant.

To resolve the integration issue between D3 and ReactJS, the D3 components(FeatureNetwork and ProbabilityChart, see *section 5.1.3*) are not added directly to the 'render' method as expected. Instead, in the render method, there is a single element which has some kind of identifier to be able to select the element node. The described 'componentDidMount' lifecycle method is then used to select and attach the necessary base D3 <svg> element using D3's API on the selected div element as a mountpoint. Since 'componentDidMount' is only invoked once, the <svg> that is attached stays rendered until the end of that components 'life'. The reason that there's just a element used as a mountpoint in the 'render' method is because when React checks for differences in the virtual DOM after the component receives new properties, it will check the identified element node and realise that there's nothing new to render, leaving the element alone in the DOM. This allows D3 to manipulate the created D3 object using its API in the componentDidUpdate method which is how the component specifies the transitions between two different sets of data.

To create the FeatureNetwork, the component uses D3's Force Directed Graph Layout[15] which expects a list of nodes and a list of the relationships between the nodes.

The ProbabilityCharts create vertical rectangles with D3's API to represent each feature's category for the user which can be seen in *feature 5.6*. When new data is received, D3 uses a function to check the maximum value in the dataset and scales the y-axis. At the same time, D3 transitions each rectangle depending on if the data value correlating to that rectangle has changed by increasing or decreasing the rectangle's height.

5.2 Implementing the user interactions with ReduxJS

This section will explain how the designed possible user interactions(*section 4.4*) have been implemented using ReduxJS. ReduxJS is a Javascript library which implements an architecture that was discovered as a result of ReactJS called the Flux architecture.

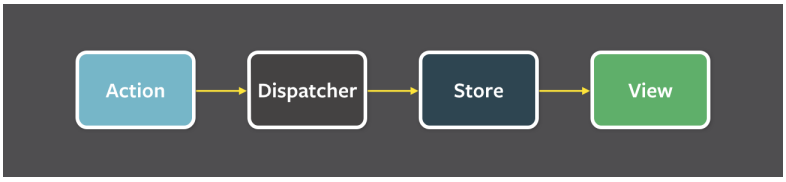


Figure 5.7: Data in the Flux architecture flows in a single direction.[33]

The main principle of the Flux architecture is that there is a single directional data flow(*figure 5.7*). This allows 'Views' or React components to be decoupled from its data by not caring where the data is coming from and not knowing where the data is sent. Component's being decoupled from the state's data is important because it means when the data's structure changes in the future(which it most likely will), the only part of the code base that needs to be changed are the non-view parts(ie React

components do not need to be changed). This further fulfils the main motivation for to the prototype web interface's rebuild to replace it with a flexible and extensible code base(*Introduction section 1.2*).

React components are connected to 'Stores' which can be thought of as a container for all the interface's state. The component does not know what the data is or where it comes from, it only has properties that need to be passed down to it in order to render the component. The 'Store' sends the component the required properties when it is connected to the component and any subsequent updates to that data are pushed as properties to the component causing the component to rerender with the new data.

As well as the properties that are passed down to the component from the 'Store', the component may also be passed functions. These functions are callbacks which command the 'Dispatcher' to send a specific 'Action'. These 'Actions' are the possible interactions with the user interface. When an 'Action' is 'Dispatched', it manipulates the data in the store to achieve the intended change in the interface. These 'Actions' are how the possible user interactions described in the UML Activity Diagram(*figure 4.6*) that have been implemented.

ReduxJS is a convenience library that implements the Flux Architecture by having its own 'Dispatcher'. A small evolution from the Flux Architecture that occurs in ReduxJS is that the 'Store' part is represented as a single immutable global state tree that stores all the data for the interface. To implement ReduxJS' global state tree, something called 'Reducers' are used which returns a javascript switch block[45]. A 'Reducer' recieves 2 parameters, the current state of the global state tree and an 'Action'. The switch statement then takes the current state tree and returns a completely new state to replace the current state by running the case in the switch statement which correlates to the 'Action' triggered. The fact that state is replaced in the global state tree instead of manipulated is what makes it immutable that comes with performance and other benefits[40].

Multiple 'Reducers' can be combined to encapsulate state to certain parts of the state tree, making the global state tree more manageable and only worry about the parts of the state tree that is relevant to certain parts of the interface. In the Scikic, the global state tree is broken down into two core 'Reducers' which are related to each of the two modules in the new interface. 'Actions' have been implemented in a file for each module called "[module name]Actions.js" and the same goes for each module's 'Reducer' which has been implemented in a file called "[module name]Reducer.js".

The rest of the section will explain each module's actions and reducers which together make up the user's possible interactions with the new web interface(*Design section 4.4*).

5.2.1 Chat module's Actions and how they affect its Reducer

When the user enters the application, they are presented with a decision as to whether they are interested in using the Scikic. Whichever decision the user makes, an 'INTERESTED' action is dispatched with their choice. The reducer then changes the state tree to tell the interface what decision the user made, causing the QuestionList to render a new component(*section 5.1.2*) that correlates to their interested decision. By having this 'INTERESTED' action, the button that says the user has "changed their mind" if they initially said they were not interested(*figure 5.5*) can be re-used.

Once the user has shown they are interested, a new multi action fetching a new question is triggered which dispatches a sequence of actions one after the other using an *ECMAScript 6*[42] *Promise*[44]. A promise allows functions to be invoked only when the previous promise has finished. *Promises* also allow functions to be invoked depending on whether they succeeded or failed. The first action that is dispatched to fetch a new question is 'REQUEST_NEXT_QUESTION' which simply changes the global state tree to say that a question is being requested. The reason for this flag is that it allows React components to show some kind of loading indicator. Then, as described in *Requirements and Analysis section 3.1.2 requirements 1 and 2*, a request is sent to the Scikic API's */question* endpoint. Finally,

to finish fetching a question a 'RECEIVE_NEXT_QUESTION' action is dispatched adding the new question to the Chat reducer which is connected to the QuestionList component so it can render the new question that was fetched.

The user then makes a decision with the question. The user either skips or answers the question. Based on the decision, an action correlating to the user's decision('SKIP_QUESTION' or the answer question multi action) is dispatched.

'SKIP_QUESTION' simply flags that particular question in the global state tree as skipped which then allows it to render the question differently in the user interface marking it is skipped. After a question is skipped, a new question is retrieved from the Scikic API using the multi action to fetch a new question that has been already described.

Answering a question triggers another sequence of actions like how fetch a new question works. The first action sets a flag that to say the question has been answered. Then a the global state tree is sent an action called 'START_VISUALIZATION' is dispatched to tell the Chat component to stop any other Chat related actions from triggering that could affect the current visualization until it has finished. If a Chat action is triggered before the visualization finishes, an action will be dispatched to show the user an error so the user is aware they have to wait for the visualizations to end. Finally, an action to display the visualization from the Visualizations module(*section 5.2.2*) is triggered.

A Visualization action will signify the end of the visualization at which point the next new question is fetched and the user interactions then continue in the Chat module as described in this section.

5.2.2 Visualization module's Actions and how they affect its Reducer

Picking up where *section 5.2.1* ended, 'do visualization' multi action, similarly to fetching a new question(*section 5.2.1*) multi action starts with 'REQUEST_INFERENCE'. This sets a flag to say that the particular question's inference is happening. The Visualizations component(*section 5.1.3*) shows a loader until the inference has been received from the Scikic API. The inference is then fetched from the Scikic API's */inference* endpoint(as described in *Requirements and Analysis section 3.1.2 requirements 3*). Upon receiving the question's inference, it is added to the list of other question inferences in the global state tree(*Requirements and Analysis section 3.1.2 requirements 4*). By adding it to a list of question inferences, it allows the user to select previous question inferences without having to make the same request again and the user is then able to select previously answered questions transitioning the visualizations(*Requirements and Analysis section 3.1.2 requirements 8*). The new question's inference is selected and another multi action finishing the question visualization is triggered.

Finishing the question visualization multi action checks if the question visualized was the last question that was answered by the user. Every answered question after the initially visualized question's inference will be affected by the new answer. This could happen if the user decided to answer a previously skipped question or edited the answer to a previous question. If either of those were the case, then the interface loops through the rest of the answered questions in order, using the do visualization action just described to update their inference's. Once all the answered questions' inferences that were required are complete, check the last question in the asked question's list to see if it has been answered. dispatch the 'FINISH_VISUALIZATION' action which allows the Chat module's user interactions to continue. If the last question in the question list was not been answered then focus it to let the user know that's the question they can next answer. Otherwise trigger the Chat module's fetch new question action(*section 5.2.1*).

5.2.3 Conclusion

This section has explained how ReduxJS's implementation of the Flux Architecture implements the design of all the possible user interactions. The global state tree is broken down by the Scikic's Chat and Visualization modules.

ReduxJS is being used to deal with the data flow which means the possible user interactions are completely decoupled from the React components contributing towards making the code base more flexible. The data manipulation and fetching requirements(*section 3.1.5 requirements 1-4 and 8*) are completely separate from the requirements which have some kind of rendering to the browser. This helps achieve one of the main motivations for this project; to make the code base extensible and easy to improve its features in the future(*Introduction section 1.2*).

5.3 Keeping the code quality high during development

The previous sections in this chapter have explained how the code base has been made flexible and easily changeable to meet new needs. Another important aspect of keeping a code base manageable is to keep it 'clean'. Clean in the sense of readable and minimizing the amount of repeated/unnecessary code while still meeting the project's requirements. There have been a number of technologies used during development to keep the code quality at a high standard. This section will briefly cover some of the most important libraries that have helped with that.

The biggest contributors to a higher quality of code is the use of *ECMAScript 6*[42](herein, *ES6*) which one of its features has already been mentioned(*Promises* in *section 5.2.1*). It has allowed the code to be more concise and use features which are included in Javascript's next specification. Naturally, being the next specification, it has many improvements in terms of code style(e.g. not needing the function keyword, also known as *Lambda*s). Since this is the next specification of Javascript, many browsers still are not compatible with *ES6* by default. To tackle this issue, a library called *Babel*[5] is used to compile the code into Javascript's *ECMAScript 5* which all browsers are compatible with.

To ensure consistent code styling throughout the project, a library called *Eslint*[27] has been used to fix these issues automatically or highlight issues in compatible code editors. A number of *Eslint* plugins have been used to make it compatible with *ES6* and React's JSX files. Finally, *Eslint* is highly configurable which allows individual code style rules to be enabled/disabled and to be used with different preset configurations. This project has used *Eslint-config-Airbnb*[1] since this is one of the technology industry's leading Javascript styling standards.

5.4 Testing

This section will identify which requirements have been successfully met using certain testing methods.

5.4.1 Verifying the data's is in the correct format using automated unit tests

As part of the implementation of the new web interface, unit tests have been developed in order to verify the requirements that have specified how the interface's state data should be structured and manipulated to communicate with the Scikic API(*Requirements and Analysis sections 3.1.2 and 3.1.3*).

Mocha's[61] test runner is used to run the web interface's unit tests. Any other test runner could have

been used instead of *Mocha*, it was arbitrarily chosen due to the developer having previous experience in it which saved time in learning any other test runner. The focus of the project is not to test using the best test runner which is why others have not been surveyed before picking *Mocha*.

Due to the decoupling between React components and all the data related user interactions(*section 5.2*), the implemented unit tests are only related to the user interaction code which means that only the *ReduxJS* related code implementing the user interactions have needed to be unit tested. All the unit tests can be found under the same directories as the *ReduxJS* user interaction implementations and can be consistently indentified as "[ReduxJS user interaction implementation file name].spec.js" instead of just having the ".js" extension. This is because the unit tests can be seen as "specs" or "specifications" to their actual implementations and this way a developer can easily visit a function's unit test to understand what the user interaction function is supposed to be doing.

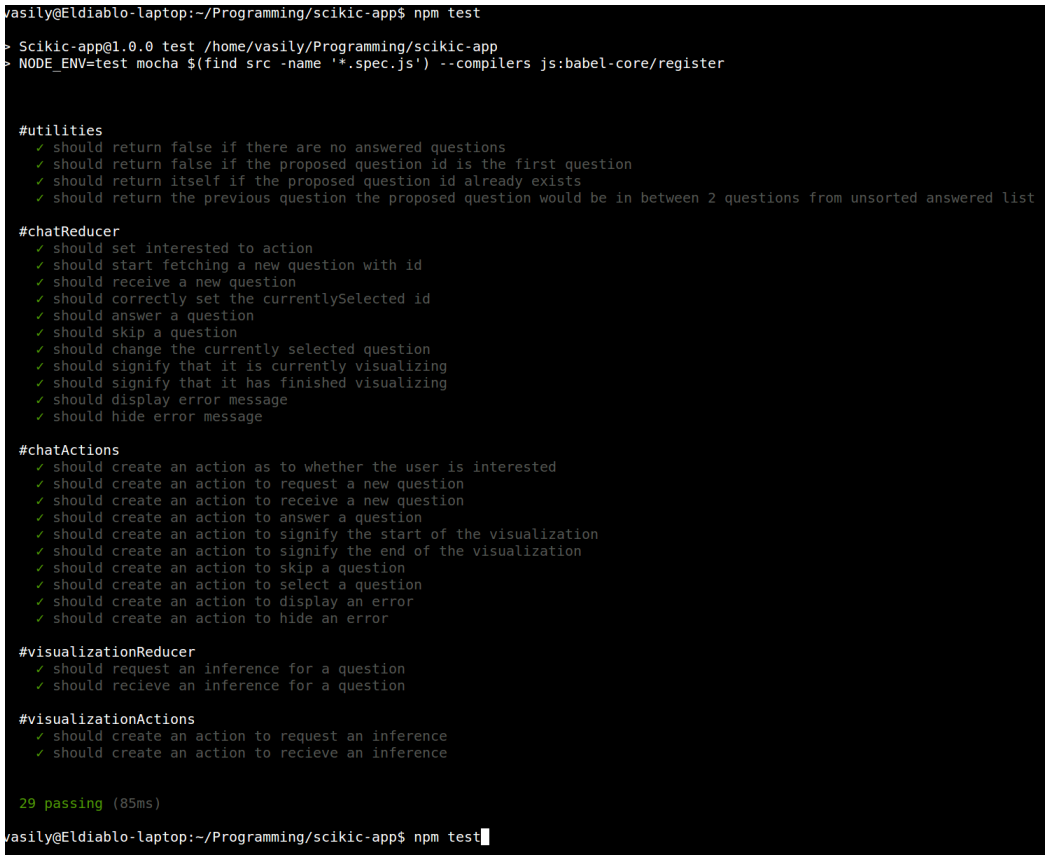


Figure 5.8: Unit tests in the web interface run by Mocha pass successfully.

The specific requirements that have been met by using the unit tests are all related to the Visualization module(*section 3.1.5* and fulfil requirements 1-4 and requirement 8). Since the unit tests prove that these requirements are met, there is no need to do any kind of manual or user testing to verify them.

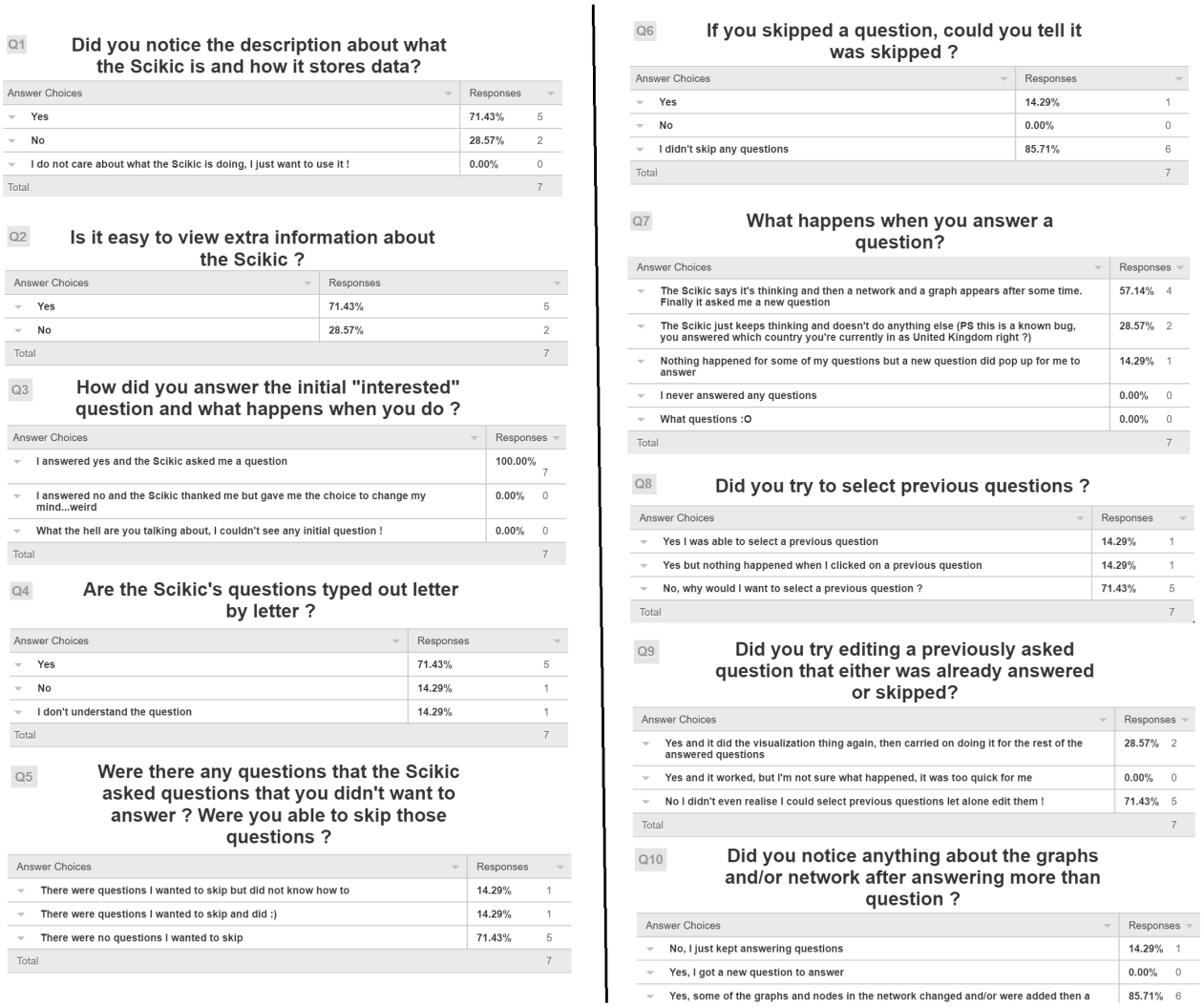
To run the unit tests the developer has to set up the new web interface's project and run a single command('npm test'). Results will then appear about which tests passed that can be identified by their given description when they were originally implemented. *Figure 5.8* shows how all the implemented unit tests successfully pass, verifying that the requirements discussed in this section currently have been met.

The unit tests give the added benefit of making these requirements robust. When the expected input/output by the Scikic API inevitably changes, the new requirements can be easily met by modifying the unit tests and then fixing their related user interaction implementations. This is known as *Test Driven Development*[47] and is another contributing factor to the overall aim of improving the prototype web interface's code base(*Introduction section 1.3, aim 1*).

5.4.2 Analysis of the user testing

A survey of 10 questions were asked to a small independent user group using `https://www.surveymonkey.com`. The test was conducted ethically by clearly stating that the survey data is completely anonymous and the purpose is to use the answers from the survey in this report to help verify that the requirements for the project have been met.

The 10 questions that were asked are based on the requirements that were actually met by the implementation rather than all the requirements that were outlined in the *Requirement and Analysis sections 3.1.5 and 3.2.3*. This section will analyze each question's results in terms of which requirements do they show were successful or failures and do the results suggest possible improvements that could be made to the new web interface(*figure 5.9*). Each question will be referred to from *figure 5.9* by its number that can be found to the left of each question.



Q1 Did you notice the description about what the Scikic is and how it stores data?

Answer Choices	Responses	
Yes	71.43%	5
No	28.57%	2
I do not care about what the Scikic is doing, I just want to use it !	0.00%	0
Total		7

Q2 Is it easy to view extra information about the Scikic ?

Answer Choices	Responses	
Yes	71.43%	5
No	28.57%	2
Total		7

Q3 How did you answer the initial "interested" question and what happens when you do ?

Answer Choices	Responses	
I answered yes and the Scikic asked me a question	100.00%	7
I answered no and the Scikic thanked me but gave me the choice to change my mind...weird	0.00%	0
What the hell are you talking about, I couldn't see any initial question !	0.00%	0
Total		7

Q4 Are the Scikic's questions typed out letter by letter ?

Answer Choices	Responses	
Yes	71.43%	5
No	14.29%	1
I don't understand the question	14.29%	1
Total		7

Q5 Were there any questions that the Scikic asked questions that you didn't want to answer ? Were you able to skip those questions ?

Answer Choices	Responses	
There were questions I wanted to skip but did not know how to	14.29%	1
There were questions I wanted to skip and did :)	14.29%	1
There were no questions I wanted to skip	71.43%	5
Total		7

Q6 If you skipped a question, could you tell it was skipped ?

Answer Choices	Responses	
Yes	14.29%	1
No	0.00%	0
I didn't skip any questions	85.71%	6
Total		7

Q7 What happens when you answer a question?

Answer Choices	Responses	
The Scikic says it's thinking and then a network and a graph appears after some time. Finally it asked me a new question	57.14%	4
The Scikic just keeps thinking and doesn't do anything else (PS this is a known bug, you answered which country you're currently in as United Kingdom right ?)	28.57%	2
Nothing happened for some of my questions but a new question did pop up for me to answer	14.29%	1
I never answered any questions	0.00%	0
What questions :O	0.00%	0
Total		7

Q8 Did you try to select previous questions ?

Answer Choices	Responses	
Yes I was able to select a previous question	14.29%	1
Yes but nothing happened when I clicked on a previous question	14.29%	1
No, why would I want to select a previous question ?	71.43%	5
Total		7

Q9 Did you try editing a previously asked question that either was already answered or skipped?

Answer Choices	Responses	
Yes and it did the visualization thing again, then carried on doing it for the rest of the answered questions	28.57%	2
Yes and it worked, but I'm not sure what happened, it was too quick for me	0.00%	0
No I didn't even realise I could select previous questions let alone edit them !	71.43%	5
Total		7

Q10 Did you notice anything about the graphs and/or network after answering more than question ?

Answer Choices	Responses	
No, I just kept answering questions	14.29%	1
Yes, I got a new question to answer	0.00%	0
Yes, some of the graphs and nodes in the network changed and/or were added then a	85.71%	6

Figure 5.9: The 10 questions and answer results from the user testing using `https://www.surveymonkey.com`.

1. Aims at making sure that the description of what the Scikic is and how it stores data is clear and visible to the users(*section 3.2.3* requirements 1-3). The results show that, while not everyone noticed the description, the majority(72%) of the users answered they did, which shows that the description related requirements were met successfully.

2. Verifies that users can easily navigate to extra information if they so wish(*section 3.2.3* requirement 4). The question specifically was worded to not indicate to the user how to navigate to the extra

information. Since the majority of user's(72%) answered "yes" without being told how to view the extra information, the related requirement can be considered a success.

3. Checks that whichever answer the user gives to the initial question, the web interface does not break(*section 3.2.3* requirement 5). All the results(100%) said that users answered "yes" and the Scikic asked a new question(*section 3.1.5* requirement 2). While it is positive that every user answered "Yes" showing that users are interested in trying the Scikic, there are no results verifying that when a user answers "No" to the Scikic's initial interested question, a thank you message appears with the option to change their minds. Despite that, the important takeaway from this question is there were no results suggesting that the initial question was not working since none of the user tests answered with "I couldn't see any initial question".

4. This question makes sure that in the Chat module, questions are typed out letter by letter(*section 3.2.3* requirement 7). The majority of users answered with either "Yes"(72%), verifying that the typing requirement is met, or "I don't understand the question" (14%) which suggests that the survey's question was unclear to those users, rather than indicating a problem with the interface. There was a single "No" answer in the survey but since it is the only "No" answer, the likely hood is that this was an anomaly and perhaps an issue with the browser used when accessing http://scikic.org.

5. Due to the majority of users saying that "there were no question they wanted to skip"(72%), the results are inconclusive, however, there was at least 1 user which said that "they wanted to skip a question and did" which at least proves that the skip question button(*section 3.2.3* requirement 9) works.

6. Similarly to Q5, the majority of users did not skip any questions and so could not check if the question skipped was marked clearly in the user interface(*section 3.2.3* requirement 9 and 10). The one user that did skip a question that they could see it clearly marked which is a positive result, however, further testing should be done to confirm this.

7. The main goal of this question was to verify that the Visualization module successfully displayed to the user the Scikic BN and feature probability distributions(*section 3.1.5* requirements 5-7), as well as, that a new question is asked at the end of the visualization and they can carry on using the new web interface. The results show that for the majority of users(58%), the Visualization module met the described requirements. The rest of the results show either that the question did not affect the Scikic's inference which explains why one user answered answered "nothing happens but a new question is asked" or 28% of users answered "the Scikic just keeps thinking..." which as described in the answer is a known bug of the Scikic based on a specific answer to one of the questions. Overall this shows that the requirements were successfully met, especially since none of the results suggested that new questions were not being asked.

8. Selecting a previous question shows a transition between the current question and the newly selected previous question's inference by the user(*section 3.1.5* requirement 8 and *section 3.2.3* requirement 10). 72% of users had said that it was not clear "why they would want to select a previous question", suggesting that most users do not need to edit a question and based on Q5, most users would not need to answer a question they previously skipped. While one user was able to select a previous question, another one user was not able and so while the requirement has been implemented, the user testing shows that more testing needs to be done to make sure that being able to select a previous question works correctly.

9. This checks that when editing a previously asked question, the visualizations transition updates the later answered questions(*section 3.1.5* requirement 8 and *section 3.2.3* requirement 10). As expected, most of the results(72%, same as Q8) show that there was not a need for users to edit previous questions because they answered them correctly the first time and were not interested in seeing the transition in the visualization if they changed their answers. The other results, however,

contradict the answers given in Q8, which is that the two users that answered Q8, one of which said they could and the other said they could not select a previous question. Both of them were able to edit a previously asked question and the visualizations updated the inference's for each of the later questions successfully. This either suggests that one of the users made a mistake in Q8 of the survey or they may have failed to select a previous question, but then were able to after they answered Q8. Either way, selecting previous questions should be more thoroughly tested to make sure that it has met its requirements.

10. Finally, the main aim of the project was to visualize to the user how the Scikic's BN and feature probability distributions transitioned and changed as the user answered more questions. 86% of the user testing group had said that they noticed the "graphs and nodes in the network changed and/or were added". This shows that the overall aim has been a success, even if the users do not understand what the change means yet, the implementation has clearly demonstrated that the visualizations are dynamic according to the Scikic's inference.

5.4.3 Conclusion

To verify that the requirements have been met for this project, there were two types of testing used. The unit tests have verified that the data related requirements in the Visualization module have been successfully met and are flexible enough to be able to meet new requirements based on how the data from the Scikic API changes in the future. The user testing has been insightful in showing how successful requirements that rendered objects to the browser have been. While there was a clear indication that the majority of requirements were met(*section 6.2* for further details), more testing would be beneficial to verify some of the requirements due to the user test results being unclear such as selecting previous questions.

6 Results and Discussion

This chapter will evaluate the new web interface that has been implemented in terms of whether it has met the overall aims of the project. It will also discuss some interesting findings that have been discovered as a result of this implementation project.

Visualizing the Scikic's inference data to the user has been the main driver for this project and to do that, D3 was used. The first section explores the D3 and ReactJS integration implemented in this project and the different possible approaches that were explored before coming to the final solution which meets the Visualization module's requirements.

Then there is an overview of how well the new web interface has met the overall aims of the project(*Introduction section 1.3*) and in more detail the identified requirements(*Requirement and Analysis*). The success criteria of the requirements will be based on whether the requirements have been implemented(*sections 5.1-5.3*) and whether the different types of testing(*section 5.4*) have verified their functionality.

The chapter then discusses what adding the visualizations to the new web interface has actually achieved. Is it a step forward towards what the project's motivations(*Introduction section 1.2*)?

Finally based on the discussion from the rest of this chapter, some ideas around how the Scikic could be improved in the future are explored.

6.1 Different approaches to integrating D3 and React

Integrating D3 with React is a known problem in both developer communities(described in *section 5.1.4*). This section discusses the suggested solutions that have been previously implemented to solve the integration problems that these two Javascript libraries have. The project took inspiration from these solutions to use D3 for the visualizations module and as a result, implements a good approach which enables both libraries to work as intended.

The main problem is that both libraries try to control the Document Object Model(herein, DOM) which causes them to override each other with unintended side effects to the interface. To create D3's visualizations, the project has tried various methods to find the most appropriate implementation to visualize the Scikic's inference data.

A possible approach involves letting ReactJS render the base elements that D3 uses. This would be considered the expected 'React' implementation. This approach uses React to re-render the elements that D3 uses based on the new data being passed into the React component which then visualizes the new data[17]. The problem with this is that D3 loses control over the elements which means that the powerful animation transitions that D3 comes with, are unusable. The transitions were one of the main requirements for the Scikic in order to show the user how the inference data was changing when they answered new questions about themselves. This approach clearly was not viable to meet the overall goal of the project.

The community was not satisfied with not being able to use the power of D3 by doing things the 'React' way. The community came up with solutions which involved giving D3 complete control over the DOM

by not implementing it with ReactJS at all. React and D3 both need a HTML element as their mount point. Instead of putting D3 into a React component, projects were structuring their web interfaces with a HTML file which would have two mount points, one for ReactJS' application and another for D3 which were treated completely separately. This is clearly limited since data for the visualizations cannot be managed by React which was the whole reason why developers wanted to integrate D3 with React in the first place.

Not using D3 within React was clearly not an option for most peoples' needs, from this, developers explored ways to stop React from controlling the DOM of certain components within its tree. From this came the idea of creating a new DOM within React, a 'fake DOM' which supports enough methods just for D3(and support for other other libraries could be added in the future)[16]. The way it works is that within the render method of a React component, a new component is made in the 'D3' way, using all of D3's API methods as a developer would normally expect to be able to in Javascript. The component then becomes the owner of a newly created DOM. Instead of having to get to grips with the DOM technically and get that component working, there was a Javascript library created to generalize the 'fake DOM' component[75]. This was great, there was now a way to use the power of D3's API in React components, or was it ? While this solved the problem of not being able to use D3 as it was intended to be used with its API in React, there was still the problem of the component being re-rendered entirely when new data was passed down to the 'fake DOM' component and the visualizations should transition.

In the end because of the failures in all these discussed approaches, the solution did not require the use of any extra libraries and instead relies on React's lifecycle methods(*section 5.1.4*). The chosen approach only renders a single mount point element and using the React lifecycle methods to manipulate the DOM using D3s API after React has finished with the DOM in the render method. This approach has been described as using "D3 with React"[18] as opposed to D3 in React like in the previous examples. The main concept is using D3 API's main DOM manipulations(create, update destroy) as part of React component's lifecycle methods(*componentDidMount*, *componentDidUpdate*, *componentWillUnmount*). As a result, React is used as a thin wrapper around D3 which is why D3 is described as being used with React rather than in React. This is the approach that was used in this project because of how it enables the use of D3s transition updates in React's *componentDidUpdate* lifecycle method to meet the requirements of visualizing to the user how their inference data changes after each personal question is answered.

6.2 Did the project meet all of its aims and requirements?

The project has been orientated around creating user specific visualizations using the Scikic's API and showing the user how the machine learning techniques that process the answers to the questions given to the Scikic produce personalized insights from open data sets. To communicate with the Scikic, users could use a prototype conversational interface, however, upon investigating the prototype it was apparent from the beginning of the project that to add visualizations to the interface required a complete rebuild of the prototype.

The end result of this project is a new web interface which not only has attempted to improve the previous prototype conversational interface both in terms of code quality and functionality but also added the visualizations for the user to be able to view. Overall, the project has successfully achieved its main goal of adding visualizations to the Scikic's web interface.

The project may have achieved its goal, but to what extent. To be able to establish the extent of its success, this section will go through which requirements that were identified in *Requirements and Analysis* have been met successfully based on the *Implementation and Testing*.

6.2.1 Visualization module's requirement analysis

Unit tests that have been implemented(*section 5.4.1*) into the project show that all the data related requirements have been fulfilled. Most of the Visualization module's requirements(*section 3.1.5*) are data related(requirements 1-3, 6 and 8).

That leaves three other requirements which are to do with the overall aim of displaying the visualizations to the user. User testing has revealed that the feature graphs and nodes in the Bayesian Network are changed/added when questions are answered(*section 5.4.2*). This verifies that the last three requirements have also been successfully fulfilled(requirements 4, 7 and part of 5).

There is a part of one of the requirements that the Visualization module was not able to meet due to the restricted time scale of the project. This is showing the datasource for each of the user's features by using differently shapes nodes as identifiers(*section 3.1.5* requirement 5).

Overall the Visualization module has been fully implemented and proven to work by other independent users. Even though each feature's node cannot be identified, this does not stop the project from meeting its overall goal of being able to visualize the machine learning technique used by the Scikic.

6.2.2 Conversational interface module's requirement analysis

As predicted in the *Final Requirements*(*section 3.3*), the project was not able to meet every identified requirement that was described(*section 3.2.3*). Unlike the Visualizations module's requirements, none of them were data related and so the only evidence to show that they were fulfilled in this report has been through user testing.

The user testing only asked users about the requirements that had been implemented. The requirements that were not implemented and therefore will not be discussed are:

- Requirement 6 - The *Facebook* login integration
- Requirement 11 - The indicator that the Scikic is typing, although there is a loading symbol while the Visualization module is retrieving the question's inference.
- Requirement 12 - Minimum processing time so that when the Scikic does its inference very quickly, it can emulate human thought.
- Requirement 17 - Verifying how accurate the user's insights were.
- Requirement 18 - Storing requirement 17's results
- Requirement 19 - Allowing the user to choose how long, if at all, their data is stored.

While these requirements have not been implemented into the new web interface, the implementation has shown how modular and flexible the code base is(*sections 5.1-5.3*), making it easy to implement these unmet requirements at a later date.

Most of the requirements that were implemented have been verified through user testing(*section 5.4.2*). There were some unexpected results such as how the majority of users did not think to or need to select previous questions. Due to how few users did use the select previous questions, there is little evidence showing that selecting previous questions has been successfully implemented. The lack of evidence has meant that the user testing was not able to verify that selecting previous questions transitions the user's

inference visualizations(*section 3.1.5 requirement 8*). Fortunately, the unit testing and Q10 of the user testing survey was able to verify that requirement instead. Regardless, selecting previous questions should be further tested in future development of the new web interface.

6.2.3 Conclusion

The analysis of the new web interface meeting the project's requirements has shown that the core aim of building visualizations was almost completely met. As anticipated from the *Requirements and Analysis*, not all the requirements were implemented for the conversational interface rebuild. Despite that, the unmet requirements have not affected the Visualization module and shown that they can easily added to the new web interface in the future because of how modularized the new code base is.

6.3 What has visualizing the Scikic's data achieved ?

As explained in the *Introduction*'s motivation(*section 1.2*), eventually these visualizations and the Scikic hope to be used as an interesting and interactive educational tool to help people understand machine learning techniques better. The result of this project is still a long way from being an effective educational tool, however, the implementation has given a strong foundation to eventually try and meet that goal when the Scikic improves overall.

Apart from creating a strong foundation to meeting the Scikic's goals, being able to visualize the data coupled with the understanding about Bayesian networks(*Literature review 2.1.1*) has meant that the creators of the Scikic are able to review how effective the Scikic itself is. For the first time since the Scikic's creation, it is possible to conveniently and accurately view the specific data that the Scikic is returning. This was not possible from the Scikic's prototype web interface, where all that was displayed to the user were the produced personalized text insights at the end of using the Scikic.

In the short time that the new web interface has been available, a number of bugs have been identified from where answering certain questions has produced unexpected question inference results. Some of the identified bugs with the Scikic include:

- The user's features are not named in the clearest way. For example rather than identifying a feature with the label "age", it is identified as "factor_age". While this could be dealt with in the new web interface by converting all the identifiers to something more appropriate, it would be more logical to change them at their source on the Scikic.

- Some of the inferences are not working correctly. The questions that verified that the inferences have some issues are the "have you seen movie X" questions in combination with the user feature questions either about their age or gender. When the user answers a question about a movie, two of the feature probability distributions representing their age and gender appear as expected, however, if after that the user answers a question about their gender or age, a user would expect their related feature probability distributions to transition to a single bar since it has been made fact. This is not the case and they remain unchanged. Unlike when the user answers a question about what country they are currently in which the feature probability distributions correctly transition their age or gender if the user confirms it in another question.

- There are a number of questions that currently do not contribute to the inference in any way such as "how many guns do you have ?".

- The only question that actually produces insights is the "what country are you currently in?" question.

This list are just a few of the current problems with the Scikic. Without the addition of the Visualizations module, it would have been unlikely to have been possible to identify and confirm these bugs with the Scikic.

Until the newly identifiable bugs are reduced, visualizing the Scikic's question inference unfortunately currently does not achieve anything further to the non-creators of the Scikic.

6.4 The future of the Scikic

This chapter has explained how the new web interface has been successful in meeting the project's overall aims and many of the original identified requirements.

There are still many requirements that the project was not able to meet, specifically with the conversational interface(section *6.2.2*) which naturally are improvements that can be implemented in the future. The highest priority change would likely be to include the ability for the user to log into *Facebook* before any questions from the Scikic API are asked.

There are a number of identified bugs from the Scikic itself which will also be improved in the future. The next big step with the new web interface after the improvements to the Scikic have been implemented is to have a stopping condition for the number of questions that are asked. A stopping condition did not make sense with this project because of the current bugs in the Scikic, where some of the questions would not produce inferences leading to the possibility of the Scikic not returning any interesting insights to the user nor visualizing anything at all.

Once it is possible to flag a suitable end of the Scikic's question, it will also make sense to add the insight verification so the user can indicate how accurate the insights from the Scikic returns are. At that point, there can be a feedback loop using other(potentially machine learning) techniques to improve its predictions.

The Scikic was planning to use data that collaborative filtering could be used to contribute to the user's insights. For various reasons, that improvement was not added to the Scikic in time for this project. This will be a key development for the Scikic to become a more effective educational tool in machine learning in the future, since at the moment the visualizations that are displayed to the user is quite a narrow topic of machine learning.

7 Project Conclusion

This project originally set out to build "Interactive Machine Learning Visualizations" using an existing application called the Scikic. To achieve that, this project ambitiously had to rebuild the prototype conversational interface that was created with the Scikic before the start of this project because it was not possible to add machine learning visualizations that the Scikic was producing in the prototype web interface.

There are many stories that have become a part of this project due to the rebuild of the prototype. A heavy analysis was undertakenm, based on understanding how to rebuild the Scikic's prototype web interface using guidelines that have been previously established and industry leaders' implementations of successful conversational interfaces as inspiration for making the new web interface a far better solution than the prototype. From this report, a clear process can be seen of how to analyze and identify concrete requirements which have led to concise designs and eventually implementations meeting the overall aims of the project.

Bayesian networks and collaborative filtering were the key machine learning techniques that had to be understood, since they are what the Scikic uses to produce its inferences. Without that knowledge, the included technologies could not have been justified in the new web interface. Exploring the currently available libraries that could be integrated in order to succeed in building interactive machine learning visualizations is what led to deciding D3 was the best library to use. D3 was the clear front runner, both in terms of developer popularity and features that met all the visualization requirements.

Modularizing the new web interface and making it extendable to meet *CitizenMe*'s(the collaborative start up with this project) requirement to be able to use the conversational interface in their own product created further challenges for this project. ReactJS was used to achieve that goal and while a more detailed survey of other modularizing solutions could have been undertaken, it would have further detracted from the main purpose of this project which is to create machine learning visualizations.

In theory, D3 was the most appropriate library to achieve the main visualization goal, during implementation, the interesting obstacle of integrating this powerful library in ReactJS, which was chosen to meet *CitizenMe*'s requirement, has allowed the project to try different possible approaches that the community has developed to overcome this obstacle. The solution that was implemented is using D3 with ReactJS as a thin wrapper and the discussion around the justification for this approach as opposed to the others hopefully can be contributed back to the community, to help it further evolve solutions for this technical obstacle.

The project was not expecting to be able to meet all of the outlined requirements from its analysis, however, after evaluating the final implementation of the new web interface using unit tests and user testing, there has been a clear indication of the success in meeting many of the requirements in the new web interface. Especially, the key requirements to be able to visualize the machine learning techniques used by the Scikic, which were all proven to work.

The user testing with the small independent user group has shown almost every single user was interested in trying out the Scikic because as part of the conversational interface, it specifically asks whether they were interested in using the Scikic. There was no prompting or pressure on the user to force any users to click "yes". This is a very positive response and will drive the Scikic to improve itself even more in the future.

This project has explained in detail how the components correlating to the main aims of this project have been modularized and how all the positive user interactions are completely decoupled from the rest of the interface. This has shown that the requirements which the new web interface was not able to fulfil in this project are certainly attainable in future iterations of the Scikic. Therefore, the detailed analysis will not be wasted, since it will contribute to the future of the Scikic's new web interface.

The implementation of this project has successfully attained its main goal of building interactive machine learning visualizations. It has taken a big step forward in creating a useful and unique tool that for the first time that lets users see how the innards of machine learning works, using their own personal answers as input. This has progressed the Scikic's motivation of helping users understand machine learning better, contributing to resolving the public's misconceptions about machine learning.

The future of the Scikic has been given a clear direction with the valuable information that the visualizations in the new web interface creates, which otherwise would unlikely have been realized. The Scikic will continue to grow due to this project by further developing the ways that it uses machine learning and visualizing them to whoever wants to learn more.

References

[1] Airbnb. *JavaScript Style Guide*. URL: https://github.com/airbnb/javascript.

[2] *Angular 2.0*. URL: https://angular.io/.

[3] *arbor.js*. URL: http://arborjs.org/.

[4] David Auerbach. *I Built That "So-and-So Is Typing" Feature in Chat*. 2014. URL: http://www.slate.com/articles/technology/bitwise/2014/02/typing{_}indicator{_}in{_}chat{_}i{_}built{_}it{_}and{_}i{_}m{_}not{_}sorry.html.

[5] *Babel*. URL: https://babeljs.io/.

[6] *Baby Names 1904-1994*. URL: http://www.ons.gov.uk/ons/rel/vsob1/baby-names--england-and-wales/1904-1994/top-100-baby-names-historical-data.xls.

[7] *Baby names 1996-2013*. URL: http://www.ons.gov.uk/ons/about-ons/business-transparency/freedom-of-information/what-can-i-request/published-ad-hoc-data/pop/august-2014/baby-names-1996-2013.xls.

[8] *Bandsintown API*. URL: http://www.bandsintown.com/api/overview.

[9] Nathan Barry. *Forms are a Conversation*. 2012. URL: https://uxmag.com/articles/forms-are-a-conversation.

[10] Yaroslav Bulatov. *Trends in Machine Learning according to Google Scholar*. 2005. URL: http://yaroslavvb.blogspot.co.uk/2005/12/trends-in-machine-learning-according.html.

[11] *C3.js*. URL: http://c3js.org/.

[12] Rory Cellan-Jones. *Stephen Hawking warns artificial intelligence could end mankind*. 2014. URL: http://www.bbc.co.uk/news/technology-30290540.

[13] Kuo Chu Chang and Wei Sun. "Comparing probabilistic inference for mixed Bayesian networks". In: 5096 (2003), pp. 346–353. DOI: 10.1117/12.486863. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=764495.

[14] *Cytoscape.js*. URL: http://js.cytoscape.org/.

[15] *D3 Force Layout*. URL: https://github.com/mbostock/d3/wiki/Force-Layout.

[16] *D3 Within React the Right Way*. URL: http://oli.me.uk/2015/09/09/d3-within-react-the-right-way/.

[17] *D3 without D3*. URL: http://blog.scottlogic.com/2015/09/03/d3-without-d3.html.

[18] *D3act*. URL: https://github.com/AnSavvides/d3act.

[19] *D3.js*. URL: http://d3js.org/.

[20] AS Das et al. "Google news personalization: scalable online collaborative filtering". In: *Proceedings of the 16th international conference on* (2007), pp. 271–280. ISSN: 1595936548. DOI: 10.1145/1242572.1242610. URL: http://portal.acm.org/citation.cfm?id=1242610.

[21] *Dominos UK Pizza Delivery*. URL: https://www.dominos.co.uk/store.

[22] Stuart Dredge. *How does Facebook decide what to show in my news feed?* 2014. URL: http://www.theguardian.com/technology/2014/jun/30/facebook-news-feed-filters-emotion-study.

[23] Mohamed A Du, Xiaoyong and Fan, Wenfei and Wang, Jianmin and Peng, Zhiyong and Sharaf. "Web Technologies and Applications: 13th Asia-Pacific Web Conference, APWeb 2011. Proceedings". In: Beijing, Chiina: Springer Science & Business Media, 2011.

[24] D. L. Dutton. "The cold reading technique". In: *Experientia* 44 (1988), pp. 326–332. ISSN: 00144754. DOI: 10.1007/BF01961271.

[25] Gal Elidan, Iftach Nachman, and Nir Friedman. "Ideal Parent Structure Learning for Continuous Variable Bayesian Networks". In: *The Journal of Machine Learning Research* 8 (2007), pp. 1799–1833. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1314498.1314559.

[26] W.A. Awad ELseuofi and S.M. "Machine Learning Methods for Spam E- Mail Classification". In: *Journal of Computer Science* 3.1 (2011), pp. 173–184. URL: http://airccse.org/journal/jcsit/0211ijcsit12.pdf.

[27] *Eslint*. URL: http://eslint.org/.

[28] Facebook. *Facebook's Graph API*. URL: https://developers.facebook.com/docs/graph-api.

[29] Facebook. *JSX Specification*. URL: https://facebook.github.io/jsx/.

[30] Facebook. *ReactJS*. URL: https://facebook.github.io/react/.

[31] *Facebook Messenger*. URL: https://www.messenger.com.

[32] *FamousEngine*. URL: http://famous.org/.

[33] *Flux Architecture*. URL: https://facebook.github.io/flux/docs/overview.html{\#}structure-and-data-flow.

[34] *Freegeoip.net*. 2015. URL: http://freegeoip.net/?q=31.205.82.28.

[35] *Github*. URL: https://github.com/.

[36] Google. *Polymer*.

[37] Scott Hanselman. *JavaScript is Assembly Language for the Web*. 2011. URL: http://www.hanselman.com/blog/JavaScriptisAssemblyLanguagefortheWebPart2MadnessorjustInsanity.aspx.

[38] David Heckerman. "A Tutorial on Learning With Bayesian Networks". In: *Innovations in Bayesian Networks* 1995.November (1996), pp. 33–82. ISSN: 1860949X. DOI: 10.1007/978-3-540-85066-3. URL: http://www.springerlink.com/index/62mv333389016034.pdf.

[39] Jeffrey Heer, Stuart K. Card, and James a. Landay. "Prefuse: a Toolkit for Interactive Information Visualization". In: *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05* (2005), p. 421. ISSN: 02749696. DOI: 10.1145/1054972.1055031. URL: http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=1054972.1055031.

[40] Pat Helland. "Immutability Changes Everything". In: *7th Biennial Conference on Innovative Data Systems Research* (2015). ISSN: 15577317. DOI: 10.1145/2844112.

[41] *Huffduffer Sign Up Form*. URL: https://huffduffer.com/signup.

[42] Ecma International. *ECMA-262 ECMAScript 6th Edition Language Specification*. Tech. rep. 2015, pp. 1–566. URL: http://www.ecma-international.org/ecma-262/6.0/.

[43] Paul Irish. *pointpointwork*. 2012. URL: https://www.quora.com/How-does-http-pointerpointer-com-work.

[44] *Javascript Promise Definition*. URL: https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global{_}Objects/Promise.

[45] *Javascript Switch Statement Definition*. URL: https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Statements/switch.

[46] Jessi Hempel. *Facebook Launches M, Its Bold Answer to Siri and Cortana*. 2015. URL: http://www.wired.com/2015/08/facebook-launches-m-new-kind-virtual-assistant/.

[47] Christian Johansen. *Test-Driven JavaScript Development*. 2011, 1 online resource (xxvii, 497 s.), ill. ISBN: 9780321683915.

[48] Michael I. Jordan and Terrence J. Sejnowski. "Graphical Models: Foundations of Neural Computation". In: *Pattern Analysis & Applications* 5.4 (2002), p. 8. ISSN: 14337541. DOI: 10.1007/s100440200036. URL: http://www.springerlink.com/openurl.asp?genre=article{\&}id=doi:10.1007/s100440200036.

[49] Firas Khatib et al. "Crystal structure of a monomeric retroviral protease solved by protein folding game players". In: *Nature Structural & Molecular Biology* 19.3 (2012), pp. 364–364. ISSN: 1545-9993. DOI: 10.1038/nsmb0312-364b. arXiv: NIHMS150003. URL: http://dx.doi.org/10.1038/nsmb.2119.

[50] Eugene Kim. *1 in 3 Americans say they will never consider a self-driving car, according to a new poll*. 2015. URL: http://uk.businessinsider.com/self-driving-cars-have-a-long-way-to-go-in-the-us-according-to-this-chart-2015-5?r=US{\&}IR=T.

[51] Robert V. Kozinets. "Netnography: Doing ethnographic research online". In: *International Journal of Advertising* 29.2 (2010), pp. 328–330. ISSN: 02650487. DOI: 10.2501/S026504871020118X.

[52] Helge Langseth. "Bayesian Networks for Collaborative Filtering". PhD thesis. Norwegian University of Science and Technology, 2009, pp. 67–78.

[53] P Larrañaga and J A Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Vol. 2. 2002, p. 416. ISBN: 0792374665. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20{\&}path=ASIN/0792374665.

[54] Greg Linden, Brent Smith, and Jeremy York. "Amazon.com recommendations: Item-to-item collaborative filtering". In: *IEEE Internet Computing* 7.1 (2003), pp. 76–80. ISSN: 10897801. DOI: 10.1109/MIC.2003.1167344.

[55] Erwin Marsi and Ferdi Van Rooden. "Expressing Uncertainty with a Talking Head in a Multimodal Question-Answering System ∗". In: *Communication And Cognition* (2007). DOI: citeulike-article-id:2429269.

[56] Microsoft. *Skype*. URL: http://www.skype.com/en/.

[57] Justin Mifsud. "An Extensive Guide To Web Form Usability". In: (2011). URL: http://www.smashingmagazine.com/2011/11/extensive-guide-web-form-usability/.

[58] Mike Bostock. *Introducing d3-shape*. 2015. URL: https://medium.com/@mbostock/introducing-d3-shape-73f8367e6d12{\#}.2bkwnkb0u.

[59] *Missouri Census Data Centre*. 2015.

[60] Koji Miyahara and Michael J Pazzani. "Collaborative Filtering with the Simple Bayesian Classifier". In: *PRICAI 2000 Topics in Artificial Intelligence*. Springer, 2000, pp. 679–689. ISBN: 3-540-67925-1. DOI: 10.1007/3-540-44533-1. URL: https://www.cs.rutgers.edu/{~}pazzani/Publications/koji.pdf.

[61] *MochaJS*. URL: https://mochajs.org/.

[62] Eric Driver Morrell and Darryl. "Implementation of continuous Bayesian networks using sums of weighted Gaussians". PhD thesis. Arizona State University, 1995, pp. 134–140. URL: https://dslpitt.org/uai/papers/95/p134-driver.pdf.

[63] Moshanin. *Collaborative Filtering Wikipedia*. 2013. URL: https://en.wikipedia.org/wiki/Collaborative{_}filtering{\#}/media/File:Collaborative{_}filtering.gif.

[64] *MovieLens Dataset*.

[65] Muut. *RiotJS*. URL: http://riotjs.com/.

[66] Clifford Ivar Nass, Jonathan Steuer, and Ellen R Tauber. "Computers are social actors". In: *Computer-Human Interaction (CHI) Conference: Celebrating Interdependence 1994* JANUARY (1994), pp. 72–78. ISSN: 0897916506. DOI: 10.1145/259963.260288.

[67] Rs Niculescu, Tm Mitchell, and Rb Rao. "Bayesian network learning with parameter constraints". In: *The Journal of Machine Learning ...* 7 (2006), pp. 1357–1383. ISSN: 1533-7928. URL: http://dl.acm.org/citation.cfm?id=1248597.

[68] Jakob Nielsen and John Morkes. "Applying Writing Guidelines to Web Pages". In: *ACM SIGCHI Conference on Human Factors in Computing Systems* April (1998), pp. 321–322. DOI: 10.1145/286498.286792. URL: http://www.sigchi.org/chi98/.

[69] *Office of National Statistics*. 2011. URL: http://www.ons.gov.uk/ons/guide-method/census/2011/census-data/ons-data-explorer--beta-/index.html.

[70] *pointerpointer*. URL: http://www.pointerpointer.com/.

[71] *Prefuse*. URL: http://prefuse.org/.

[72] *pymc*. URL: https://github.com/pymc-devs/pymc.

[73] *Raphael.js*. URL: http://raphaeljs.com/.

[74] *React Component Lifecycle*. URL: https://facebook.github.io/react/docs/component-specs.html{\#}lifecycle-methods.

[75] *react-faux-dom*. URL: https://github.com/Olical/react-faux-dom.

[76] *React Router*. URL: https://github.com/reactjs/react-router.

[77] *React Typist*. URL: https://jstejada.github.io/react-typist/.

[78] Ahmed Rebai. "IBM Knowledge Center - Bayesian Network Nodes". In: (2010), p. 2. URL: https://www-01.ibm.com/support/knowledgecenter/SS3RA7{_}17.0.0/clementine/bayesian{_}networks{_}node{_}general.dita.

[79] Mamamia Rogue. *Facebook's Chat Typing Awareness Indicator*. 2015. URL: http://www.mamamia.com.au/typing-indicator-bubbles/.

[80] Julia Rozwens. "10 Ways to Build Trust on Your Landing Page". In: (2014). URL: http://uxmovement.com/content/10-ways-to-build-trust-on-your-landing-page/.

[81] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, 3rd edition*. 2009, pp. 1 –1132. ISBN: 0136042597. DOI: 10.1017/S0269888900007724. URL: http://portal.acm.org/citation.cfm?id=1671238{\&}coll=DL{\&}dl=GUIDE{\&}CFID=190864501{\&}CFTOKEN=29051579$\backslash$npapers2://publication/uuid/4B787E16-89F6-4FF7-A5E5-E59F3CFEFE88.

[82] Richard M Ryan and Edward L Deci. "Self-Determination Theory and the Facilitation of Instrinsic Motivation, Social Development, and Well-Being". In: *American Psychologist* 55 (2000), pp. 68–78. ISSN: 0003-066X. DOI: 10.1037/0003-066X.55.1.68. arXiv: 0208024 [gr-qc].

[83] Badrul Sarwar et al. "Item-based collaborative filtering recommendation algorithms". In: *Proceedings of the 10th ...* Vol. 1. 2001, pp. 285–295. ISBN: 1581133480. DOI: 10.1145/371920.372071. arXiv: 119. URL: http://portal.acm.org/citation.cfm?doid=371920.372071$\backslash$nhttp://dl.acm.org/citation.cfm?id=372071.

[84] Diana Slade Scott Thornbury. *Conversation: From Description to Pedagogy*. 2006, p. 377. ISBN: 9.78052E+12.

[85] Alexander Serenko, Nick Bontis, and Brian Detlor. "End-user adoption of animated interface agentsin everyday work applications". In: *Behaviour & Information Technology* 26.2 (2007), pp. 119–132. ISSN: 0144-929X. DOI: 10.1080/01449290500260538.

[86] *Sigma.js*. URL: http://sigmajs.org/.

[87] Phil Simon. *Too Big to Ignore: The Business Case for Big Data*. Wiley, 2013, p. 89. ISBN: 978-1-118-63817-0.

[88] Xiaoyuan Su and Taghi M. Khoshgoftaar. "Collaborative filtering for multi-class data using belief nets algorithms". In: *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI* (2006), pp. 497–504. ISSN: 10823409. DOI: 10.1109/ICTAI.2006.41.

[89] Morgan Brown; Dylan La Com; Everette Taylor. *GitHub's Secret Super Powers — The Stuff That Makes GitHub Great (Part 2)*. Tech. rep. 2013, p. 1. URL: https://growthhackers.com/growth-studies/github-part-2{\#}.

[90] TheANTWorks. *Pfizer*. Tech. rep. 2012. URL: http://theantworks.herokuapp.com/industry-report/pfizer-az/.

[91] John Thuma. "Why Machine Learning Is The Next Penicillin". In: (2015), p. 1. URL: http://www.forbes.com/sites/teradata/2015/07/16/why-machine-learning-is-the-next-penicillin/.

[92] Alan M Turing. "Turing. Computing machinery and intelligence". In: *Mind* 59.236 (1950), pp. 433–460. ISSN: 0026-4423. DOI: http://dx.doi.org/10.1007/978-1-4020-6710-5_3. URL: papers2://publication/uuid/E74CAAC6-F3DD-47E7-AEA6-5FB511730877.

[93] *Typeform*. URL: http://www.typeform.com/.

[94] *US Census Bureau*.

[95] *vis.js*. URL: http://visjs.org/.

[96] *vis.js Showcases*. URL: http://visjs.org/showcase/index.html.

[97] Martin Warren. *Features of Naturalness in Conversation*. 152nd ed. John Benjamins Publishing, 2006, p. 8. ISBN: 9789027253958.

[98] Ilana Westerman. "Designing to Build Trust : The factors that matter". In: (2012). URL: http://uxmag.com/articles/designing-to-build-trust-the-factors-that-matter.

[99] *Whatsapp*. URL: https://www.whatsapp.com/.

[100] Darren J. Wilkinson. "Bayesian methods in bioinformatics and computational systems biology". In: *Briefings in Bioinformatics* 8.2 (2007), pp. 109–116. ISSN: 14675463. DOI: 10.1093/bib/bbm007.

[101] Topher Winward. *KittenMouse*. 2013. URL: http://winwardo.co.uk/KittenMouse/.

[102] Ellie Zolfagharifard. *Is the world heading towards a post-human future? Sir Martin Rees warns that super-intelligent robots could wipe out humanity*. 2015. URL: http://www.dailymail.co.uk/sciencetech/article-3091749/Is-world-heading-post-human-future-Sir-Martin-Rees-warns-super-intelligent-robots-wipe-humanity.html.