A Data Science Approach to Murder Mysteries

Clemence Brival

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

7th of September 2016

Abstract

abstract goes here

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

Name: _____ Signature: _____

Acknowledgements

acknowledgements go here

Contents

Chapter 1

Introduction

1.1 Problem Motivation

Chapter 2

Background Survey

2.1 Detective Stories

2.2 Machine Learning

"**Machine learning** is a field of study that gives computers the ability to learn without being explicitly programmed." - Arthur Samuel (1959). [Simon, 2013] Using algorithmic models and mathematics, it consists in iteratively analysing a set of data, in order to make predictions about similar data. Applications for machine learning are varied: speech and handwriting recognition, medical diagnosis, recommendation systems, advertisements, fraud detection, video games, self-driven cars, robot locomotion, ...

A machine learning algorithm is fed a dataset, the **training set**, made up of one or several **instances**, which represent the type of data the prediction will be based on. Each instance contains actual values for **attributes** (or features) the algorithm can analyse and learn from. For example, to predict the gender of the murderer, possible attributes could be the murder weapon, the gender of the victim, the name of the detective and the year of publication of the book. An instance could therefore be "Poison, Female, Hercule Poirot, 1926". The attribute to be predicted is the **concept**, and the predicted value for this attribute is the **concept description**.

The type of machine learning used for this project is **supervised learning**. This technique involves building the training set and providing it with the correct value for the attribute to be predicted[1]. The algorithm can therefore base its observations on cases for which it knows the outcome, learn to produce the desired output, and apply the results of its training on instances for which it doesn't know the right value.

Supervised learning is divided into two categories of problems, based on the type of attribute to be predicted. In this project's case, the app will attempt to predict discrete valued attributes, for instance the gender of the murderer (male or female): this is a **classification** problem.

[1] http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning

2.3 Related Work

To celebrate the 125th anniversary of Agatha Christie's birth, a team composed of Dr Dominique Jeannerod (researcher at Queen's University Modern Languages department in Belfast[2]), Brett Jacob (data analyst) and Dr James Bernthal (researcher in English at the University of Exeter) conducted a study of the author's work. [Siddique, 2015] Commissioned by the UKTV Drama channel, this research is very similar to this project, as it resulted in a mathematical formula that would help readers of Agatha Christie's novels to guess the identity of the killer, before reaching the end of the story.

Based on a selection of 27 books, the team analysed the differences in terms of story details, in order to pick up on patterns that might help predicting attributes of the culprit. Their investigation allowed them to select a set of features, later used as parameters in their mathematical formula, such as the relationship to the victim, the main means of transportation of the story, the cause of death of the victim, the detective in charge of the investigation, and the number of the chapter in which the killer was introduced. For instance, according to the analysts, if the story setting is a country house, or if the narrator mentions land vehicles, then the killer is most likely to be female. However, there is a higher probability that the killer is male if the novel involves boats or planes, or if the victim is strangled.

The researchers also mentioned other clues, although not explicitly used in the formula, like the tone in which Agatha Christie described the killer (generally more neutral or positive when the killer is male), or the way the detective conducted his/her investigation (if based on a domestic item that was discovered, instead of information and logic, the culprit is often a woman).

Even though this study was handled from a mathematical view, and only contains little references to computing science, it is nonetheless a very interesting piece of work. It gives another perspective to the project, as it considers the actual writing, and not just story details independently. However, one negative side to it is that it can only handle two detectives (Hercule Poirot and Miss Marple), probably because of the small selection of novels the research is based on.

2.4 Existing apps

There doesn't seem to be any app that can be used on the side, while reading a detective novel. Nor is there any app that can be fed key elements of a story, which can help it solve a murder. However, it is possible to find apps related to detective stories, and apps that make predictions.

2.4.1 Detective apps

The Google Play store contains a few apps, in which the user is presented a story, a crime scene and few potential culprits. Impersonating a detective, the player then has to examine the crime scene and interrogate the suspects, so as to identify the killer. Here is a description of two such apps.

[2] https://daro.qub.ac.uk/agatha-christie-whodunnit

"Who Is The Killer" [3] was developed by Dmitry Glaznev, Irina Nekrasova, Elena Koroleva and Grigory Chekmasov. The game was first released on iOS in 2013, and now counts four episodes. Playing as Mr. K, the user has to arrest the murderer within seven days, before all the innocent characters are killed, seeing as a new corpse is discovered every day. In order to proceed to the investigation, Mr K. has to solve small puzzles, which allow him to earn cups of coffee. The more coffee the detective has, the more clues he can find during the day, and the more suspects he can interrogate.

"Criminal Case" [4] is a similar app created by a company called Pretty Simple. First released in 2012 on Facebook, the developers team later launched an iOS version in 2014. The Android version came out about a year after that. In this game, the user becomes a member of a police unit, and takes part of the investigation of several crimes. Each activity requires energy that can be earned by completing the tasks as fast as possible.

The inconvenient with this kind of games is that the creators implement a story, that they made up, into the app, which then just leads the user through clues, and lets them accuse the right suspect. The app itself doesn't have any guessing or analysis to do. Moreover, although entertaining, the puzzles sometimes tend to be slightly too easy, and the story not captivating enough to remain hooked to the game.

2.4.2 Predictive apps

Most Android predictive apps are sport-related. Analysing statistics from several games (or years even), those apps aim to forecast the outcome of matches. They do not require input from the user, as the data is collected from online sources, so there is no real interaction between the app and the user (there is no feeling of contributing to making the predictions better).

[3] http://www.glaznev.com/#!wtk/c1dmp
[4] http://www.prettysimplegames.com/games/

Chapter 3

Requirements

For this project, two main user roles were defined: readers and writers of detective stories. The requirements of the system were gathered during a meeting with the project supervisor, who acted as the main stakeholder, and a proxy for the different types of users. In accordance to the agile methodology used throughout the whole development process, coming up are the functional and nonfunctional requirements of the system, as well as user stories and constraints created based on those requirements.

3.1 Functional Requirements

As a reader of a detective story, I would like to get clues about the identity of the murderer. Incidentally, I would also like to guess the identity of the murderer, and see the probability that my guess is correct.

As a writer of detective stories, I would like to get ideas for a story plot. More specifically, I would like to provide elements of a story, and see attributes of the predicted murderer for that story.

3.2 Nonfunctional Requirements

As a reader, I would like to be able to use the system while I am reading the novel. The system should therefore be light and easily accessible. I also want it to be fast and easy to use, so that I can go back to the book as soon as possible.

3.3 Additional Requirements

The following requirements were added later in the project, after taking the machine learning side of the system into consideration. Users should be able to leave out details they don't know about or don't want to input. A user should be able to invalidate an incorrect prediction and provide the

right answer to the system. All users should be able to share the details of the stories they read, if they want to.

3.4 User Stories

User stories describe small tasks that a user might want to undertake when using the system. [Cohn, 2004] Here are the user stories corresponding to the functional requirements, and the constraints corresponding to the nonfunctional requirements.

A user can...

1. input details about a story.
2. receive a prediction based on the input details.
3. input details about a character.
4. receive a percentage corresponding to the probability for the described character to be the culprit.
5. select the features they want to input.
6. specify whether a prediction is correct or not.
7. input the the correct value in case of a wrong prediction.
8. send the rectified prediction to a server.
9. download the rectified predictions added to the server by other users.

The system should...

1. be implemented for hand-held devices.
2. require at most 32 Mb on the device.
3. output the prediction within 5 seconds.
4. send and download data to/from the server within 10 seconds.

3.5 Project Development Structure

The first few weeks of the project were spent researching Machine Learning and getting used to Weka, then applying the theory on the murder mystery problem, which consisted in choosing the features to analyse, building the training set, and selecting the machine algorithm to be used to make the predictions. (This will be detailed in Chapter 5.)

A week was then spent on creating wireframes for the UI and getting feedback from users on the different interfaces (see Chapter 6).

The implementation section of the project was divided into three iterations. The first iteration aimed at creating a fixed predictor: the user inputs data, and receives a prediction (user stories 1, 2 and 5). The objective of second iteration was to retrain the classifier on the app (user stories 6 and 7). For the last iteration, crowdsourced data gathering and training were implemented (user stories 8 and 9).

Finally, during the last few weeks of the project, a more formal evaluation of the final system took place, which will also be discussed in Chapter 6).

Chapter 4

Design

4.1 System Architecture

4.2 User Interface Design

Chapter 5

Implementation

5.1 Technologies Used

5.1.1 Android

Difficulties Encountered

Creating the Android interface itself was not particularly hard. It did require a significant amount of time to browse through all the widgets available in the different Android standard libraries (seven Android packages contain widgets[1]), then to decide which ones were the most appropriate for the app, and to learn how to customise and control them programmatically. Compatibility was also an issue, for some UI elements or attributes were introduced in more recent Android versions than the app's target version. It was therefore necessary to find alternative solutions.

However, none of those obstacles were a surprise; as a matter of fact, they are an integral part of the learning process, and overcoming such difficulties can only have a positive impact on my overall programming experience.

The hardest part of the Android development was to try to make the app as reliable and efficient as possible. This includes facing all the possible causes for the app to crash unexpectedly, which involves handling the activity lifecycle of the app[2] (what to do when the app is paused, resumed, stopped), the potential memory leaks, loading images efficiently, properly managing the different activities and fragments, and the way they communicate. It also includes monitoring the amount of memory and battery use of the app. Unfortunately, although this was one of the main points of concern, not all of these elements were fully handled.

Another difficulty arose on a code-level, when trying to reduce the amount of redundancy from one activity or fragment to another, without making the overall code harder to read. Because of the way projects are structured in Android[3], it is necessary to jump from one file to another in order to

[1] https://developer.android.com/reference/packages.html
[2] https://developer.android.com/training/basics/activity-lifecycle/index.html
[3] https://developer.android.com/studio/projects/index.html

follow the flow of the system. This becomes worse when creating extra classes to store the common code shared by multiple elements.

5.1.2 Weka

To make predictions, this app referenced machine learning algorithms implemented in the Weka Java library [Hall Eibe Frank et al., 2009], created by Eibe Frank, Mark Hall, Peter Reutemann, and Len Trigg. For consistency, the key-terms used throughout the description of this project will be the same as the ones used in the Data Mining book [Witten et al., 2011] written by the first two members of the Weka team, in collaboration with Ian Witten.

Integrating External Code

As mentioned previously, the machine learning side of this project was handled using the Weka package. Thus, a large part of the time was spent researching the different functionalities available in the Weka explorer[4], and understanding how to programmatically get the same information as the one produced using the GUI. As a result, a utility class was created, containing methods to create instances (from user input or from a file), to filter those instances, to create a classifier and train it on the instances, to evaluate the classifier, to classify one or more instances and retrain the classifier on new data.

This preliminary work was difficult, mainly because it was hard to find explanations in the documentation, for the different exceptions that were thrown when executing some instructions. The exception messages were not descriptive enough, and it took a lot of reading of problems faced by other users of the library to eventually find the causes of the errors.

Merging the functionalities implemented in the utility class with the Android code happened without almost any obstacle. The .jar file containing the source code of the library was added as a dependency in the project, and all the needed classes were imported in the standard way: "`import weka.`" followed by the package containing the class to be used, and the name of the class. The only encountered issue came from the fact that, in order to train a classifier, one of the Weka classes uses the Java Graphics class from the awt package[5], which is not available in Android. It was therefore not possible to train the classifier on the app. This was fixed by training it externally, outputting the trained object to a .model file, including that file as an asset of the project, and recreating the corresponding Classifier object from the .model file when launching the app.

[4] http://www.cs.waikato.ac.nz/ ml/weka/gui_explorer.html
[5] ??

5.1.3 Django

5.1.4 Scrapy

5.2 Machine Learning Algorithm

5.3 Attributes Selection

5.4 Story Details Input

5.5 Prediction Output

5.6 Communication with the Server

In the earliest version of the system during the third iteration, the training

5.7 Testing

Chapter 6

Evaluation

6.1 User Interface

6.2 Final App

Chapter 7

Conclusion

7.1 Limitations and Future Work

7.1.1 Additional functionalities

Mostly due to a lack of time, two functionalities were not implemented.

The first one corresponds to the user stories number 3 and 4 mentioned in the Requirements section, to allow the users to make a guess on the identity of the killer, and receive the likelihood of their supposition to be correct. The users would have to input details about the character they suspect, and the app would compare those elements to its own prediction. Based on the number of matching attributes, the app could output a percentage notifying the users how close they are of finding the culprit.

Another potential functionality is related to the context of the user-interaction with the system. The idea is for detective novels fans to be able to use the app, while reading a book. If they get a clue on the murderer before the end of the story, they cannot know whether or not the prediction is right, as the identity of the culprit has not been revealed yet. It could therefore be made possible for the users to save the prediction they received, and go back to it later to confirm it or input the correct data if it was wrong.

7.1.2 Accuracy

The accuracy of the predictions could be increased by improving the dataset, so that the algorithm can handle much more details about the story. It would therefore be necessary to try and find extra attributes that have an impact on the prediction. Such additional features could be the profession of the murderer, the relationship between the murderer and the victim or the chapter in which the murderer is first introduced.

Once such attributes were found, the interface of the app could then be updated, so that it can handle more concepts, other than the murder weapon and the gender of the murderer. Thus any attribute directly related to the murderer could be the possible target of the predictions. It could also be

useful to consider features related to the title of the book: for example checking if the title contains a reference to a location, to a person, to a mean of transport, or to a weapon, and whether such reference helps to infer anything about the murderer.

7.1.3 Scalability

Another part of the system which could be improved is the server. At the moment, the server uses a .txt file to store shared data between the different clients. This is not optimal in terms of scalability: if several thousands of users were to use the app, and the training set updated regularly, that file would quickly become very large, and having all the users to download it every time they launch the app would not be practical.

A first alternative would be to store each entry of the dataset in a database, and to make the client-apps fetch the new entries. A new problem would then arise: how to check if an entry is new to a given client? A possible solution could be to timestamp every entry with the date and time they were sent to the server. Each client would also keep the time it last downloaded new data, so that the server can send all the entries that were added after the client's last update.

A second way to increase the scalability of the system could be to remodel it, so that the training is handled by the server and not the client. The server would not even need to store any data, it would just receive it, retrain the classifier, and send a .model file containing the output classifier, which could then be loaded by the client to retrieve the Classifier object.

This solution would then require using the Weka library with python, which is only possible to a certain extent, given that none of the different packages and libraries which can be used for this purpose seems to implement all the functionalities found in the Java version of Weka[1]. It might even be necessary to use a completely different library, to be exclusively used with Python[2].

Another approach could be to handle all the machine learning side of the system on the server: the mobile app would just serve as an interface for the users to input data and view the predictions, but the data would be sent to and processed on the server, which would take care of the training, the classification, and the classifier update (as described for the second solution).

This solution has several advantages. The first one is that it reduces the client-server communications to the minimum, that is, when the users request a prediction, and when they submit the correct answer after the system made a wrong conjecture. The data sent in those cases would be up to a few hundreds of bytes[3], which means that the requests would be sent in a few seconds at most[4].

Another advantage is that the Android app would not have to store the .jar file containing the Weka source code, or any of the classes which handle the machine learning, and would therefore take much less space in the devices' memory. It would also use less memory, given that it would have less processes to handle.

Moreover, it would substantially increase the reusability and portability of the system. Indeed, this structure would allow a better separation of concerns, making the system responsible for the

[1] https://weka.wikispaces.com/Can+I+use+WEKA+from+Python%3F
[2] ??
[3] ??
[4] ??

backend, and the device for the frontend. As a result, it would be much easier for another developer to implement the system for other Operating Systems, by only having to modify the code for the interface.

The drawbacks, however, include the restriction caused by using Weka in Python, as well as enforcing the users' devices to be connected to the network for the app to output any prediction. This is not the case with the current version, as the users are asked permission before downloading the updated dataset, or before sending the correct answer to the server. Should they choose to refuse, they would miss out on additional data to make the predictions more accurate, yet their classifier would still be locally updated.

7.2 Personal Reflection on the Project

Chapter 8

References

Bibliography

[Cohn, 2004] Cohn, M. (2004). *User stories applied : for agile software development*. Addison-Wesley.

[Hall Eibe Frank et al., 2009] Hall Eibe Frank, M., Holmes, G., Pfahringer Peter Reutemann, B., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explorations*, 11(1).

[Siddique, 2015] Siddique, H. (2015). How to spot whodunnit: academics crack Agatha Christie's code — Books — The Guardian.

[Simon, 2013] Simon, P. (2013). *Too big to ignore : the business case for big data*. John Wiley & Sons.

[Witten et al., 2011] Witten, I. H. I. H., Frank, E., and Hall, M. A. M. A. (2011). *Data mining practical machine learning tools and techniques*. Morgan Kaufmann.

Appendix A

First appendix

Appendix B

Second appendix