An Affordance-Inspired Tool For Automated Web Page Labeling and Classification

by

Karen Anne Sittig

S.B., Massachusetts Institute of Technology (2012)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2013

Author ....
Department of Electrical Engineering and Computer Science
May 24, 2013

Certified by. ....
Dr. Catherine Havasi
Research Scientist
Thesis Supervisor

Certified by.... ....
Dr. Kevin C. Gold
Technical Staff
MIT Lincoln Laboratory
Thesis Supervisor

Accepted by ....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

An Affordance-Inspired Tool For Automated Web Page Labeling and Classification

by

Karen Anne Sittig

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Writing programs that are capable of completing complex tasks on web pages is difficult due to the inconsistent nature of the pages themselves. While there exist best practices for developing naming schemes for page elements, these schemes are not strictly enforced, making it difficult to develop a general-use automated system. Many pages must be hand-labeled if they are to be incorporated into an automated testing framework.

In this thesis, I build an application that assists human users in classifying and labeling web pages. This system uses a gradient boosting classifier from the `scikit-learn` Python package to identify which of four tasks may be performed on a given web page. It also attempts to automatically label the input fields and buttons on the web page using a gradient boosting classifier. It outputs its results in a format that can be easily consumed by the LARIAT system at MIT Lincoln Laboratory, greatly reducing the human labor required to incorporate new web pages into the system.

Thesis Supervisor: Dr. Catherine Havasi
Title: Research Scientist

Thesis Supervisor: Dr. Kevin C. Gold
Title: Technical Staff
MIT Lincoln Laboratory

Acknowledgments

I owe a great deal of thanks to the MIT Lincoln Laboratory Scenario Modeling team, headed by my adviser, Kevin Gold. The GOSMR project inspired this thesis, and your encouragement really helped me through the process. Kevin, you were the best adviser that a student could ask for - supportive, creative, and always willing to explain concepts, no matter how trivial. You have definitely made me a better scientist, and for that I am eternally grateful.

I would like to thank my MIT campus adviser, Catherine Havasi. Catherine was kind and encouraging and helped me to determine the direction of the project. I was truly inspired by your enthusiasm for machine learning, and I am thankful that so much of it rubbed off on me!

I would also like to thank the members of the Cyber System Assessments Group at MIT Lincoln Laboratory, who have supported me during this past year. I greatly enjoyed getting to know everyone, and it was a privilege to be a part of such an intelligent, passionate group of people.

Thank you to Chris Terman, my academic adviser, who has helped me discover my love of computer science over five years of MIT. I never would have written this thesis without your support.

Thanks to my friends, especially Clayton Sims and Catherine Olsson, who provided key thesis-writing emotional support. And finally, thank you to Joseph Colosimo - without our 3pm coffee breaks, I would have slept through this entire thesis!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, I will present a tool that I built to allow users to quickly classify the types of HTML inputs on a given web page, identify which of four tasks can be completed on that web page, and determine which button on the web page is used to submit the information entered. The tool is backed by several machine learning classifiers, whose design and performance I will also discuss. The structure of these classifiers is inspired by the idea of affordances, which I will explain in this section.

1.1 Background

Consider that you are looking around a room that contains many surfaces, including chairs. Your goal is to sit down. As a human, you know that you can sit on a chair - but you also know that you can sit on the ground. In order to sit on a chair, you first approach the chair, turn so that your back is facing it, and slowly lower yourself onto the chair, pivoting at your knees. To sit on the ground, you may lower yourself until you are touching the ground, and then you may fold your legs underneath yourself. Both of these actions have the same endpoint - you are solidly seated - but they have very different paths to this goal.

What if you only knew how to sit by using a chair? If there were no chairs around, you could never sit down, and you would be forced to search endlessly until you collapsed from exhaustion. Silly, but this is analogous to the approach that many automated systems take. Dealing with new situations can be very computationally intensive (and difficult), and many existing systems can only perform well in highly regulated environments.

Instead of learning how to complete a task in a particular way (find a chair to sit in), we will try to learn to identify when a task can be completed, given a set of resources (identify whether a surface can be sat upon). Put another way, we will attempt to learn whether a set of objects "afford" us to complete our goal action.

1.2 The theory of affordances

This theory - looking at objects in terms of the actions that they afford a user - was first mentioned by J. J. Gibson in 1977 [5]. According to Gibson, "the affordance of anything is a specific combination of the properties of its substance and its surfaces taken with reference to an animal" - so flat surfaces afford sitting upon, food items afford eating, clothing and blankets afford making you warmer, etc. etc. Gibson proposes that affordances are learned through experimentation - we can test whether a surface is solid enough for walking-on by prodding it with our feet, or test whether an object is edible by biting into it. On the surface, the theory of affordances seems ripe for implementation in a machine learning context - it is learned by experimentation!

It only took four years for Gibson's affordances to be implemented in robotics by Harri Jäppinen [8]. He suggests using the idea of affordances in a supervised learning context - robots can learn information about objects by attempting to use them to carry out tasks. Then, to the robot, the objects in its environment are characterized by the different tasks that they can be used to complete - the basic idea of affordances in action. Jäppinen even goes a step further and suggests that affordances can be used to determine the relationship between objects. He proposes creating a Logical Taxonomy of Concepts in which objects are organized into a tree, with the actions that they afford as their parents. Then, objects that are siblings may be similar.

Paul Fitzpatrick et al. [4] successfully implemented a system to allow a robot to learn about the affordances of objects by poking them. This robot, Cog, sought to determine whether the objects afforded rolling or sliding. By repeatedly interacting with the objects, the robot was able to build a model of the object's reaction to poking - it learned the affordances by doing. Alexander Stoytchev extended this idea of robots learning by doing to enable a robot to learn about the affordances provided by various tools [17]. Stoytchev's work is particularly interesting, as it was one of the first examples of robots learning new methods of interaction by experimenting (rather than just learning about properties of an object). Stoytchev's robot was even able to use its existing knowledge of tools to cope with failure (a broken tool) - showing just how powerful an affordance-based, exploratory approach can be.

Luis Montesano et al. [9] hypothesized that learning all affordances entirely from scratch, as somewhat suggested by Jäppinen, may be too difficult, and they proposed an alternate bootstrapping infrastructure that could be used for robots instead. Their robot was given some basic actions and perceptions and learned a series of increasingly-difficult tasks and affordances of objects. Like Cog, their robot learned affordances by interacting with objects - but its preprogrammed head start allowed it to learn more efficiently.

The concept of affordances was also highly influential in the field of design, especially graphical and industrial design. Donald Norman's 1988 book "The Psychology of Everyday Things" [10] suggested exploiting affordances to make objects seem more intuitive - putting handles on things that can be opened, for instance. Due to some confusion in the community about what an affordance was, Norman clarified what he meant in a 1999 article [11], proposing the idea of there being two different types of affordances - physical affordances and perceived affordances - that must be taken into account when designing for a virtual environment. Computers, for instance, have a variety of different types of physical affordances - clicking, typing, buttons that can be pressed - but not all of these physical affordances will produce a result in every application. Attempting to type in text on a web page that has no affordance for typing will result in nothing happening, and the user will become frustrated (and maybe even give up). Therefore, it is important for designers to clue users in to the actual affordances of an application, so that only the physical affordances that can actually be used are perceived affordances.

Some argue that the concept of affordances may be overused - and that affordances used in human-computer interactions may be more culture-specific and less universal than Norman may have us believe [12]. Still, affordances are a pervasive design inspiration in both interfaces and robotics, and they have been used to great effect for a number of applications.

1.3 Modern Approaches to Automated Web Testing

There are a variety of frameworks that allow web developers to automate testing of their websites. Automated web testing is important for web applications because it allows developers to ensure that their tests are fast, comprehensive, and repeatable, but it often comes with very high overhead.

Notably, the Selenium framework [14] allows for the development of a variety of different tests, from automated browser actuation to recording and playing back tests. It is open source and extensively used both by hobbyists and professional developers in industry. One failing of automated web testing is stated directly in the introduction to the Selenium documentation - "It is not always advantageous to automate test cases. There are times when manual testing may be more appropriate. For instance, if the applications user interface will change considerably in the near future, then any automation might need to be rewritten anyway."

The reason for this is the way that Selenium (and many other automated testing frameworks) are developed - they tie directly into the source code of the web page, and if this code changes substantially, the tests will no longer work. While automated testing frameworks are very useful in a variety of cases, they require substantial developer effort to integrate into new versions of sites.

It is possible that automated web testing could be easier if the different components on web pages could be automatically labeled instead of needing to be hand-labeled. Chusho et al. attempted to automatically fill out web forms using simple matching expressions, with some success [2]. However, this approach does not generalize well to new examples (and isn't a great predictive model). Heß and Kushmerick developed machine learning models to semantically label the fields on web pages and the tasks that those web pages could be used to complete. However, their classification model required an extremely large data set, which can be infeasible. Automatically labeling web pages remains a different challenge that has yet to be solved.

1.4 Motivations for an affordance-based approach

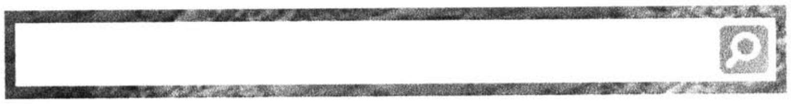
Web pages are almost tailor-made for an affordance-based approach. Different websites may have different requirements for carrying out different tasks - one website may require just an e-mail address to sign up for an e-mail news letter, but another may require a name, e-mail address, birth date, or various other pieces of information. While both websites afford the same basic action - signing up for an e-mailed newsletter - if you are just looking for a form that requires you to enter an e-mail address, you would miss the second page entirely.

Using affordances provides a new strategy for determining the field type of an HTML input. There exist various ways to enter a birth date, for instance - you can use one date field or multiple date fields. Naming conventions are also not consistent - figure 1-1 shows several examples of search boxes paired with the HTML that specifies each one. An affordance-based approach allows us to look past simple issues of whether an HTML element is named something that resembles "search box" and instead looks for objects that will allow us to enter in phrases and click a button to hopefully get a result.

In fact, applying the theory of affordances to web navigation is not a new concept. Hochmair and Frank [7] proposed using an affordance-based approach to organize a corpus of web pages for browsing tasks. They propose that any user-intended action (not just browsing) can be expressed as an affordance on a web page, an idea which I will explore in this thesis.

```
<input id="gbqfq" class="gbqfif" name="q" type="text"
autocomplete="off" value="" style="border: none;
padding: 0px; margin: -0.0625em 0px 0px; height: 1.25em;
width: 100%; background-image: url(data:image/gif;base64,
R0lGODlhAQABAID/AMDAwAAAACH5BAEAAAAALAAAAAABAAEAAAICRAEAOw%3D%3D);
background-color: transparent; position: absolute;
z-index: 6; left: 0px; outline: none;
background-position: initial initial;
background-repeat: initial initial;"
x-webkit-speech="" x-webkit-grammar="builtin:search"
lang="en" dir="ltr" spellcheck="false">
```

(a) A search box from Google.



```
<input class="sw_qbox" id="sb_form_q" maxlength="1000"
name="q" title="Enter your search term" type="text"
value="" autocomplete="off" style="color: rgb(0, 0, 0);
background-color: rgb(255, 255, 255);">
```

(b) A search box from Bing.

```
<input style="" id="p_13838465-p"
class="input-query input-long med-large
compact-input-enabled" name="p" type="text"
title="Search" value="" autocomplete="off" autofocus="">
```

(c) A search box from Yahoo.

Figure 1-1: Search boxes and their HTML.

There are non-affordance based approaches to web navigation as well. One such model is ShopBot, developed by Doorenbos et al. [3]. ShopBot attempts to assist users with comparison shopping by learning about different vendors and then searching for a product description and price on each vendor's page. ShopBot learns which forms on a web page are search forms by using domain-specific heuristics and by using regular expressions to determine the information that should be entered in each field. It then gauges its success or failure by looking at the resulting page. ShopBot then uses this information about the vendor pages when comparison shopping. However, the ShopBot model is very specific to the particular task that it is trying to complete, and its architecture does not seen adaptable to other web tasks. While these sorts of approaches can be successful in certain cases, I think that an affordance-approach will be easier to generalize, which is necessary given the evolving nature of the web.

Chapter 2

Technical Background

2.1 Introduction

This section contains an overview of three important technical components related to my thesis project. I will describe two key projects at MIT Lincoln Laboratory, LARIAT and GOSMR, that will benefit from the work done in my thesis. I will also give background on Selenium, the package that GOSMR uses to interact with web pages, in order to motivate the design of the output of the classification tool presented in Chapter 5.

2.2 LARIAT

The Lincoln Adaptable Real-time Information Assurance Testbed (LARIAT) is a state-of-the-art virtual testbed developed by MIT's Lincoln Laboratory [16]. It allows users to spin up virtual machines and run network-wide simulations using real-world software. The ability to interact with actual software is extremely valuable - not only does it make the the simulations more realistic, it allows real-world threats to be discovered. LARIAT additionally enables users to run real, repeatable experiments on simulated networks, enabling a more scientific approach to automated testing [18].

Currently, incorporating new applications into LARIAT requires substantial developer effort. Each new application must be individually incorporated into the framework, including new application versions. LARIAT simulations can only use an offline snapshot of the web, as even small changes in web pages can completely break the existing implementations of LARIAT functionalities.

The current LARIAT user models also lack realism. They are implemented as Markov chains, developed from real user data. While these user models can generate realistic traffic, they tend to have overly simplistic behavior. The models don't plan out their events, and so they can't develop alternate plans to accomplish their goals. All actions must also be scripted out in chunks, and there is no notion of achieving goals - users only have behaviors (like checking e-mail, browsing the web, typing documents, etc. etc.). All of these considerations have caused the LARIAT team to seek out a better user model.

2.3 GOSMR

The purpose of the Goal Oriented Scenario Modeling Robot (GOSMR) project was to develop more sophisticated user models ('agents') to interact with the LARIAT framework. GOSMR includes a planner, which allows the agents to take actions in order to accomplish goals. This means that GOSMR models should ideally be able to interact with new applications that they haven't seen before and to determine whether these applications can be used to carry out different tasks.

The GOSMR architecture is driven by the idea of affordances. Agent actions are broad categories that describe behavior (like LoginAction), and the agents are given a high-level goal to work towards and use experimentation to determine how to accomplish that goal. For instance, telling an agent to send a message to another agent may require that agent to navigate to a new page and log in before it can send a message - but the agent learns these requirements rather than having them specified.

One goal for GOSMR agents is to be able to interact with new web pages and determine what tasks can be completed. Unlike LARIAT models, GOSMR models should not need pages to be labeled in order to determine whether tasks can be completed and carry them out. This will make GOSMR models more resilient to change and will greatly reduce the necessary developer effort to integrate new applications. Eventually, GOSMR agents should be able to interact with a live version of the web - a totally unachievable goal for the existing LARIAT model architecture.

2.4 Web Page Actuation

GOSMR uses Selenium [14] to actuate web pages. It interacts with the Google Chrome web browser through Selenium's `ChromeDriver`.

Selenium is particularly useful because it allows easy retrieval of HTML tags via `By` statements. These statements allow for searching by tag ids, names, classes, partial link text, and more. Selenium also allows for text entry, clicking buttons, and clicking links, fully allowing GOSMR agents to browse and interact with web pages.

Each GOSMR agent stores specific knowledge about itself and about web pages that it has visited. For instance, each GOSMR agent can be programmed with its own name, e-mail address, etc. It can also store different username / password combinations for web pages.

The actions of GOSMR agents are completely abstracted away from the identities of the web pages themselves. A GOSMR agent can be directed to log on to a web page, for example, and it will use site-specific knowledge to determine where to enter its username and password and to determine which username and password to enter. So, given something as simple as the id of an HTML input and the type of text to enter, a GOSMR agent can enter the text specific to that site using Selenium - a fact that I will take advantage of in Chapter 5.

Chapter 3

Experimental Overview

3.1 Introduction

Knowing whether something can be done is often different from knowing how to do it, and this is definitely the case in web navigation. When web pages contain multiple inputs and buttons, figuring out which inputs to fill out and which buttons to click can be far more difficult than determining whether a given task can be completed. This problem gets even more difficult if the buttons and inputs are poorly labeled.

For each web page, I sought to solve two problems - determining whether the page can be used to carry out a task, and then determining where to enter information, what information should be entered, and what button needs to be clicked in order to actually complete that task. Inspired by the theory of affordances, I classified the training set of web pages by first looking at what actions they afforded and then by using those actions as signals to see what larger goals might be accomplished.

This section will outline the various classification tasks in detail and will present the overall architecture of the labeling tool that will be presented in section 5. It will also detail some statistics on the training data set and give an overview of the classification algorithm used.

3.2 Experimental Approach

3.2.1 Classification Tasks

To correspond to the existing GOSMR architecture, I split my machine learning models into two categories - those that dealt with learning whether tasks could be completed, and those that identified fields.

Tasks

Tasks are high-level descriptors of general actions that a GOSMR agent can carry out - the goal actions in the affordance model. Sending a message would be an example of a task, but typing a name into a text box would not be.

I developed models to determine whether four tasks could be completed on a given web page:

1. `search` - whether this page allowed an agent to enter in a search query and receive information related to that query in response

2. `login` - whether this page allowed an agent to authenticate by providing some information (typically a username and password)

3. `digest` - whether this page allowed an agent to sign up an address for an e-mailed digest or newsletter

4. `rewards` - whether this page allowed an agent to sign up for a rewards program, either for the site itself or for the site's parent company

As the tasks are high-level descriptors of an action to be carried out, there is no guarantee that two pages that afford completion of the same task look the same. For instance, one digest page may ask for just an e-mail address, while another may also require a name, phone number, and birth date. Additionally, the naming scheme between the task pages themselves is not consistent - pages contained many different keywords that were not necessarily shared between pages that afforded completion of the same task. This diversity made it difficult for models to learn to classify pages by task

Fields

Fields are specific areas where information can be entered by a GOSMR agent. There were 23 different types of fields present in my dataset, which are enumerated in Appendix A. Fields were used as intermediate signals to the task classifiers, as fields that required the same type of information did tend to be very similar. Fields were very good clues about the type of tasks that could be completed on each page - pages that contained search boxes, for instance, always afforded searching. Statistics specific to the classification of fields can be found in section 4.4.1.

Buttons

A button is the particular part of the web page that must be actuated in order to submit the information that was entered in they fields. Classifying buttons ended up being very similar to classifying fields - the buttons that were used to complete each task tended to be similar to one another.

3.2.2 Classifier Architecture

Separate classifiers were trained to identify HTML inputs as being examples of each field type. These classifiers were used as intermediate features and were input into a classifer that attempted to detect whether a given task could be completed on each page. Then, the button classifier for that particular task was run on the HTML buttons on the page to determine which one would submit the field information entered. A diagram of this architecture can be found in figure 3-1.

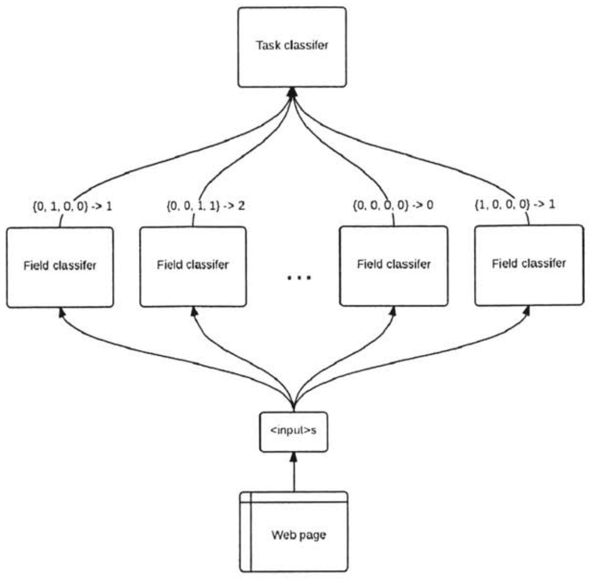


Figure 3-1: The basic architecture of the page classifier.

search	91
login	53
digest	48
rewards	44

Table 3.1: The number of labeled training examples of each task.

3.3 Overview of data set

3.3.1 Pages

My data set consists of 157 web pages, each which could be used to complete one or more of the four tasks. Each of the web pages was hand-labeled with the tasks that could be completed on that page. The number of examples of each task can be found in table 3.1.

3.3.2 Buttons

Each web page that could be used to complete a task contained a variety of input fields where users could enter relevant information for that task and a button that users

search	95
login	54
digest	51
rewards	45

Table 3.2: The number of labeled training examples of each type of button.

needed to click in order to submit that information. Since each button corresponded to the submission of information related to a given task, the buttons were also labeled with the task names. The number of training examples of each type of button can be found in table 3.2.

Some of the pages contained multiple ways of completing each task with multiple buttons (i.e. sign-up forms for different digests on the same page, all with different buttons), which resulted in there being more buttons than examples of each task.

3.3.3 Fields

In addition, the input fields and buttons on each page were also hand-labeled with one of 23 different field types. Field types that had fewer than 10 examples were labeled as "other". Descriptions of the field types and the number of each can be found in 3.3.

search box	98	email	129	password	124
secret question	20	username	62	last name	69
city	46	company	11	gender	10
account number	11	phone number	70	zip code	58
title	18	middle name	17	country	26
state	50	birth year	14	birth month	19
other	87	first name	68	secret answer	11
birth day	18	address	93		

Table 3.3: The number of training examples of each field type.

3.4 Gradient boosting classification

Gradient boosting is a machine learning technique for regression that allows multiple weak classifiers to be combined into one combined, stronger classifier. I transformed my classification task into numerous binary classification tasks to allow the use of this model.

My gradient boosting classifier was supported by decision trees with depth 5. As Hastie et al. [6] note, the results of the classifier work generally well with depths between 4 and 8 and are insensitive to changes within this range. I did validate this assertion on my data set, the results of which can be found in appendix B, and chose a depth of 5 to balance the runtime and performance of my system.

I chose the gradient boosting classifier because I suspected that many of my feature classifiers would not have great performance, and boosting classifiers allow multiple weak classifiers to be combined into one ensemble classifier that outperforms the individual weak classifiers. I chose to use decision trees as my weak classifiers because I wanted to allow the features to interact in the weak classifiers. This would allow for more complex, conditional rules in my classifiers - maybe an e-mail address is only an indication of a rewards sign-up page if it asks for additional information, like a first and last name, and otherwise it is an indication of a digest sign-up page. Simple binary classifiers, like 'does the page contain an e-mail field', would not be able to express this rule.

Chapter 4

Experiments and Results

4.1 Overview

In order to complete a task, GOSMR models need to know what information needs to be entered in each HTML input field and what button to click to submit this information - and so, it is important to be able to classify the types of each input field and button. Additionally, I hypothesized that the types of the input fields themselves may be good features to use in determining what tasks can be completed on each page.

In this chapter, I will present statistics from the machine learning models that make up the page labeling tool presented in chapter 5. I ran experiments to establish a baseline and experiments to assess the performance of my machine learning classifiers. I also will present several different full page classification experiments that will provide justification for the selection of my overall classifier design presented in section 3.2.2.

4.2 Statistical Measures Used

I used four statistical measures to evaluate the performance of my classifiers.

4.2.1 Accuracy

The accuracy of a system describes the closeness of the predicted value to the true value. The formula for accuracy is:

$$\text{accuracy} = \frac{\text{true positive} + \text{true negative}}{\text{total samples}}$$

Systems with high accuracy are very good at accurately predicting the value of a sample.

4.2.2 Precision

The precision of a system describes its ability to accurately predict positive samples. The formula for precision is:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

A system with high precision means that samples that the system predicts to be positive examples are very likely to be true positive examples, though it may miss many true positive examples.

4.2.3 Recall

The recall of a given system measures how good it is at picking out relevant positive examples. The formula for recall is:

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

A system with high recall is unlikely to miss a positive example, though it may additionally return many false positive examples.

4.2.4 F1

The F1 score is another measure of system accuracy that is often used to assess information retrieval systems. The formula for the F1 score is:

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Systems with high F_1 scores have both high precision and high recall, which means that they are good at both correctly identifying positive examples and correctly classifying all positive examples as positive.

Simply looking at the accuracy of my classifiers is misleading. The classifiers tended to have low recall, as many classification tasks had a small number of positive training examples. This led me to use the F1 score as my primary evaluation metric, as it was more representative of the true performance of the classifiers. The accuracy of each classifier is provided here for information purposes only.

4.3 Baseline

The baseline experiments were extremely simple - given some target phrase (either the field name or the task name), the baseline classifier returned 1 if the target phrase was present in the code sample and 0 if it was not.

4.3.1 Page Classification Baseline

For whole page task classification, the baseline classifier checked to see if the target task name ("search", "login", "digest", or "rewards") was present in the HTML source of the page. I ran this classifier on the labeled pages from the training corpus and calculated the F1 score for the baseline classifier for each page type. Results from running this experiment can be found in table 4.1.

The baseline classifiers did surprisingly well for each task. The high false positive rate for the "login" task seems reasonable given that many pages advertised their separate login page with a link labeled "login" in their standard page header. The

Task	True Positive	False Positive	False Negative	True Negative	F1
search	90	27	1	39	0.8654
login	50	51	3	53	0.6496
digest	31	13	17	96	0.6739
rewards	41	10	3	103	0.8631

Table 4.1: Results from the baseline page classification experiment.

high false positive rate for the "search" task can be partially attributed to the many hotel and flight pages, which allowed for searching for hotel rooms and flights but which did not afford for the keyword search task that we defined "search" to mean.

4.3.2 Field Classification Baseline

For field classification, the baseline classifier checked to see whether the name of the field was present in the HTML tag that defined the field. I ran this classifier on the labeled fields from the training corpus and calculated the F1 score for the baseline classifier for each field type. Results of this experiment can be found in table 4.2.

As expected, this baseline classifier had poor performance. While many of the field names described the sort of information that was supposed to be entered into the input field, they were often not present in the actual HTML source of the inputs. For instance, while search boxes were likely to contain the keyword "search", none of them contained the full phrase "search box". Other fields were likely to have different names depending on the particular page that they were found on, like "company" or "birth month". The exception to this trend was "password", as passwords were unlikely to be referred to by any other keyword on these pages.

Eight out of the 23 field classifiers had an F1 score of 0, and the remaining classifiers did not have excellent performance. In order to be useful in gradient boosting classification, classifiers must be able to correctly classify some samples, and so it was necessary to train classifiers specifically for the field classification task.

Field Type	True Positive	False Positive	False Negative	True Negative	F1
search box	0	0	106	1054	0.0000
secret question	0	0	20	1140	0.0000
city	28	4	19	1109	0.7089
account number	0	0	11	1149	0.0000
title	9	49	10	1092	0.2338
state	33	0	18	1109	0.7857
other	3	3	86	1068	0.0632
birth day	0	0	18	1142	0.0000
email	91	16	48	1005	0.7398
username	15	0	47	1098	0.3896
company	3	0	8	1149	0.4286
phone number	1	0	71	1088	0.0274
middle name	0	0	17	1143	0.0000
birth year	0	0	14	1146	0.0000
first name	1	1	70	1088	0.0274
address	44	26	49	1041	0.5399
password	121	5	3	1031	0.9680
last name	1	1	70	1088	0.2739
gender	6	0	4	1150	0.7500
zip code	1	0	58	1101	0.0333
country	20	0	6	1134	0.8696
birth month	0	0	19	1141	0.0000
secret answer	0	0	11	1149	0.0000

Table 4.2: Results from the baseline field classifier.

Button Type	True Positive	False Positive	False Negative	True Negative	F1
search	43	0	37	141	0.6525
login	17	0	33	171	0.4789
digest	1	1	47	172	0.0377
rewards	3	1	41	176	0.1224

Table 4.3: Results from the baseline button classification experiment.

4.3.3 Button Classification Baseline

For button classification, the baseline classifier checked to see whether the target task was present in the HTML that defined each button. I looked at the hand-labeled button examples and calculated the F1 score for each task. Results from this experiment can be found in table 4.3.

The baseline classifier had very few false positive results for the various button types, which was interesting. It appears that, while "digest" and "rewards" are reasonable keywords for the baseline page classifier, they do not often appear in the HTML defining buttons on pages. While the baseline button classifiers did correctly identify at least one button in all four cases, it was still necessary to train machine learning classifiers to classify the buttons given the high false negative rate.

4.4 Machine Learning Classifiers

Given the poor performance of the baseline button and field classifiers, it was necessary to train machine learning classifiers in order to better classify new buttons and inputs. While some field types still proved difficult to classify, the field classifiers outperformed the baseline classifiers in most cases. The button classifiers were able to achieve very good performance.

4.4.1 Field Classifiers

While gradient boosting does not require the weak classifiers to perform well, it does require the weak classifiers to be able to correctly classify some of the examples. I

Field	K	Accuracy	Precision	Recall	F1
search box	10	0.9702	0.9826	0.8400	0.8998
secret question	10	0.8667	0.3000	0.2000	0.2333
city	10	0.9397	1.0000	0.6350	0.7678
account number	5	0.8187	0.1667	0.1667	0.1600
title	9	0.8511	0.3611	0.2407	0.2741
state	10	0.9017	1.000	0.4133	0.5663
other	10	0.8614	0.7500	0.2153	0.3270
birth day	9	0.8333	0.000	0.000	0.000
email	10	0.9149	0.9023	0.5538	0.6831
username	10	0.8900	0.7700	0.4904	0.5911
company	5	0.8341	0.2000	0.1000	0.1333
phone number	10	0.8611	0.7475	0.3446	0.4447
middle name	8	0.8630	0.5625	0.4167	0.4375
birth year	7	0.7976	0.0000	0.0000	0.0000
first name	10	0.9179	0.9490	0.5500	0.6786
address	10	0.8816	0.8488	0.3856	0.5141
password	10	0.9946	0.9923	0.9756	0.9833
last name	10	0.9153	0.9417	0.5214	0.6536
gender	5	0.8333	0.2000	0.1000	0.1333
zip code	10	0.9209	0.8183	0.7133	0.7358
country	10	0.9296	0.9000	0.5667	0.6633
birth month	9	0.8504	0.2222	0.1111	0.1481
secret answer	5	0.8648	0.4000	0.2000	0.2667

Table 4.4: Results from the gradient boosting field classifiers.

evaluated each of my 23 field classifiers by k-fold cross-validation. For classifiers where I had more than 20 positive examples, I ran 10-fold cross-validation. For examples where I had fewer than 20 positive examples, I ran $\lfloor \frac{n}{2} \rfloor$-fold cross-validation. Both the gradient boosting classifier implementation and the cross-validation implementation were from the `scikit-learn` Python package [13].

For each classifier, I used 5 times the number of negative examples as positive examples.

The results of this experiment can be found in 4.4.

The gradient boosting classifiers were also unable to correctly classify examples of the "birth year" and "birth day" fields, although they were able to correctly classify examples of the "birth month" fields. Generally, the gradient boosting classifiers

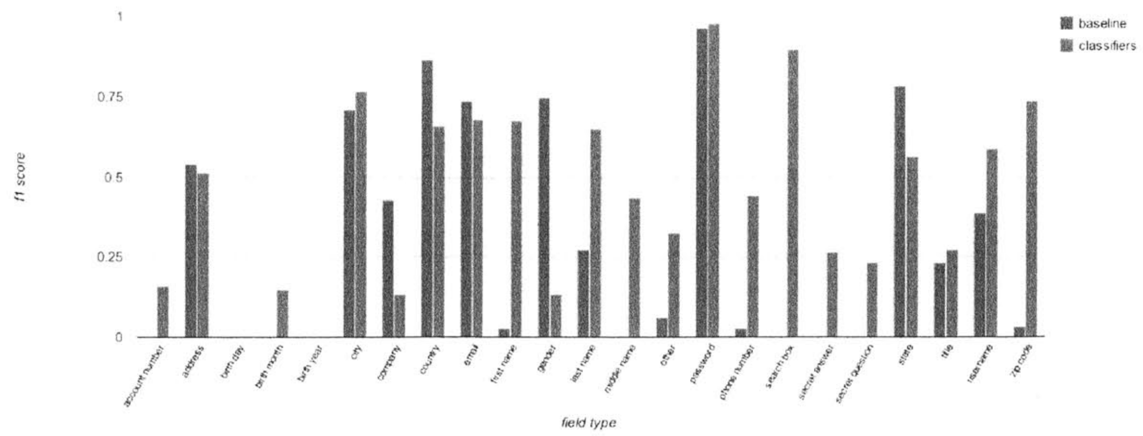


Figure 4-1: A comparison of the baseline and gradient boosting classifiers.

were able to do relatively well, but there were some interesting exceptions to this rule. A comparison of the gradient boosting classifiers and the baseline classifiers can be found in 4-1.

The key difference between the baseline classifiers and gradient boosting classifiers is highlighted in the cases where the baseline classifier outperforms the trained classifier. The features of the input are the full values of the different HTML tag attributes - so, a "country" field labeled `id = home-country` would be correctly labeled by the baseline, but it may be missed by the trained classifier if no other "country" fields contained the phrase "home-country". The baseline classifiers tended to outperform the trained classifiers in the cases where the HTML inputs tended to contain the target phrase in conjunction with other words that made that particular phrasing unique. This occurred in the cases of the "address", "company", "country", "gender", and "state" classifiers.

One potential way to boost the performance of these trained classifiers is to add 23 additional features, one for each target phrase, that are 1 if the input contain the phrase and 0 if it does not - basically augmenting the trained classifiers with the baseline classifiers. Given that these classifiers were designed for a system that includes human verification of each classification, I thought that this was unnecessary. In the future, if it is important to improve the performance of the field classifiers, this

Field	Accuracy	Precision	Recall	F1
search	0.7713	0.8047	0.8022	0.8012
digest	0.8658	0.8439	0.7500	0.7514
login	0.9813	1.0000	0.9433	0.9659
rewards	0.8796	0.8157	0.8150	0.7842

Table 4.5: Results from the gradient boosting page classifiers, using the field classifiers as features.

may be a good place to start.

4.4.2 Page Classification

I hypothesized that using the presence of different input fields on a web page as features in the overall page classifier would result in better performance than the baseline classifier. I was additionally curious to see whether using the field types of the input fields on each page would give better results than just using the HTML of the input fields themselves. To test these hypotheses, I ran experiments on three different classifiers to determine the tasks that could be completed on each page. All of these experiments used 10-fold cross-validation to produce the statistics.

Simple Page Classification

I first attempted to classify the page based on the attributes of the input fields on that page that were relevant to one of the four tasks. The binary representation of each page was the number of times each text attribute was present in its inputs.

Since I didn't know the types of the inputs, it was impossible to filter the inputs to those that were relevant to the classification task. I did know that each input was relevant to some task, as I only used the inputs on each page that had been hand-labeled with their field type and task type, but including unrelated inputs could have only reduced the performance of the classifier, not improved it. Results from this experiment can be found in table 4.5.

In the "digest" and "login" cases, the simple page classifier outperformed the baseline, and in the "search" and "rewards" cases, the baseline marginally outperformed

Field	Accuracy	Precision	Recall	F1
search	0.6000	1.0000	0.9556	0.9750
digest	0.2857	0.2000	0.0500	0.0800
login	0.3333	0.9000	0.7600	0.8099
rewards	0.2667	1.000	0.8000	0.8762

Table 4.6: Results from the gradient boosting page classifiers, using the field classifiers as features.

the page classifier. We saw in the baseline case that the word "digest" was not a great predictor of whether the digest task could be completed on the page, and it appears that using the input fields instead gives a better result. We also saw the high false positive rate in the baseline "login" case, which was much improved by only looking at the input fields. It also appears that some of the words used in each field on the login pages are relatively consistent - this is confirmed by the very high f1 score for the "password" classifier.

The underperformance of the classifier in the "search" and "rewards" cases does not appear to be particularly interesting - we already knew that the keywords "search" and "rewards" being present in a page's HTML were relatively good predictors for whether that task could be completed on that page. The "search" and "rewards" classifiers still show that the attributes of the HTML input fields can still serve as decent classifier features in these cases.

Page Classification via Field Classifiers

I then sought to determine whether using the field types of the input fields on each page as features for the overall page classifier would be more successful. The binary representation of each page was a list containing the predicted number of each input type on the page. The field classifiers were almost all able to correctly classify some of the input fields, and it seems like the gradient boosting classifier that these features fed into would be able to learn which results were more and less reliable and would be able to improve on the performance of the field classifiers. Results from this experiment can be found in table 4.6.

Field	Accuracy	Precision	Recall	F1
search	0.9808	0.9900	0.9778	0.9830
digest	0.9367	0.9000	0.9100	0.8967
login	0.9621	0.9381	0.9633	0.9465
rewards	0.9108	0.8250	0.8900	0.8477

Table 4.7: Results from the gradient boosting page classifiers, using the field classifiers as features.

In three cases, this was true - this is especially evident in the "search" case, whose positive example pages all contained "search box" fields, which had a very reliable field type classifier. The "login" classifier did slightly worse than the very reliable simple "login" page classifier, which seemed reasonable, as the "username" field classifier was not very reliable. The "rewards" classifier had surprisingly good performance - given the diversity of the field types present on the "rewards" pages, I was surprised that there was a consistent enough pattern to get good classifier performance.

The surprising result was that of the "digest" classifier, which had terrible performance. I believe that this is due to the diversity of the different examples of "digest" pages in addition to the unreliability of the field classifiers. We saw in the "login" page case that the unreliability of the field type classifiers can affect the performance of the overall page classifier, even when the contents of the page are very consistent (all "login" pages contained a username field and a password field). In the "rewards" case, I believe that this effect was mitigated by an overall consistent pattern between the pages that was not present in the "digest" case.

Page Classification via True Fields

In order to eliminate the variable of the reliability of the field classifiers, I ran a new experiment that used the true field types of each input field on the page. The binary representation of the page was a list of the actual number of each type of field present on that page. Results from this experiment can be found in table 4.7.

My hypothesis, that using the field types present on each page as features for the overall page classifier would be successful, appears to be correct - the "digest" classifier

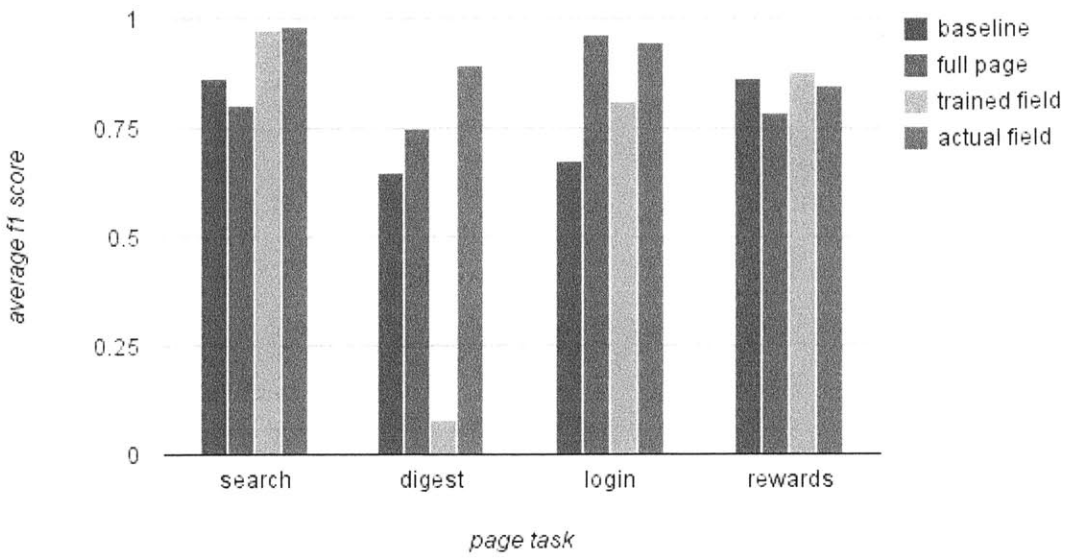


Figure 4-2: A comparison of the f1 scores of each of the four types of full page classifier.

in this case was able to achieve far better performance than all the previous classifiers. The "login" classifier was minimally outperformed by simple page classifier, and the "rewards" classifier was minimally outperformed by the predicted input field type classifier, but these performance deviations were not significant. In all cases, the true input field type classifiers were able to outperform the baseline classifiers and achieve good performance, showing that using the field types of the input fields as features is actually a good approach.

A graphical comparison of the performance of these four classifiers can be found in figure 4-2.

4.4.3 Button Classification

Unlike the trained field classifiers, the trained button classifiers outperformed the baseline classifiers in all cases. Results from this experiment can be found in 4.8.

In fact, the gradient boosting classifiers were able to achieve excellent performance

Field	Accuracy	Precision	Recall	F1
search	0.9603	0.9708	0.5927	0.7310
digest	0.9586	0.9719	0.8532	0.9080
login	0.9922	0.9900	0.9272	0.9567
rewards	0.9336	0.9136	0.9775	0.9443

Table 4.8: Results from the gradient boosting button classifiers.

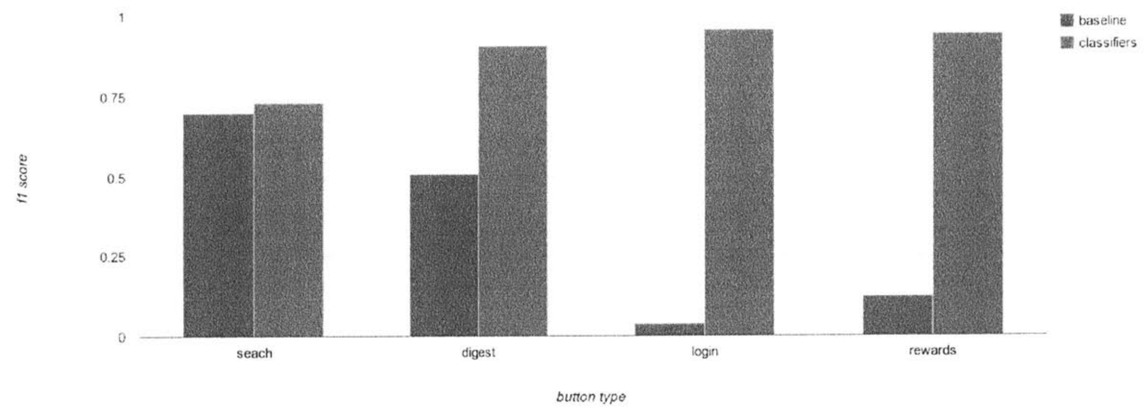


Figure 4-3: A comparison of the baseline and gradient boosting button classifiers.

on the buttons. The "search" button classifier had the worst performance, and its low recall score implies that search buttons tend to use a more diverse set of tag values than the other three button types, as there must be many false negative values. This is unsurprising given the results from the baseline experiment - many "search" buttons don't even use the word "search" at all!

A comparison between the gradient boosting classifiers and the baseline classifiers can be found in figure 4-3.

4.5 Conclusions

In general, the gradient boosting classifiers were able to outperform the baseline classifiers, as expected. In all cases, the button classifiers outperformed the baseline button classifiers, but the field classifiers had some interesting exceptions where the baseline classifiers outperformed. The field classifiers were able to correctly classify

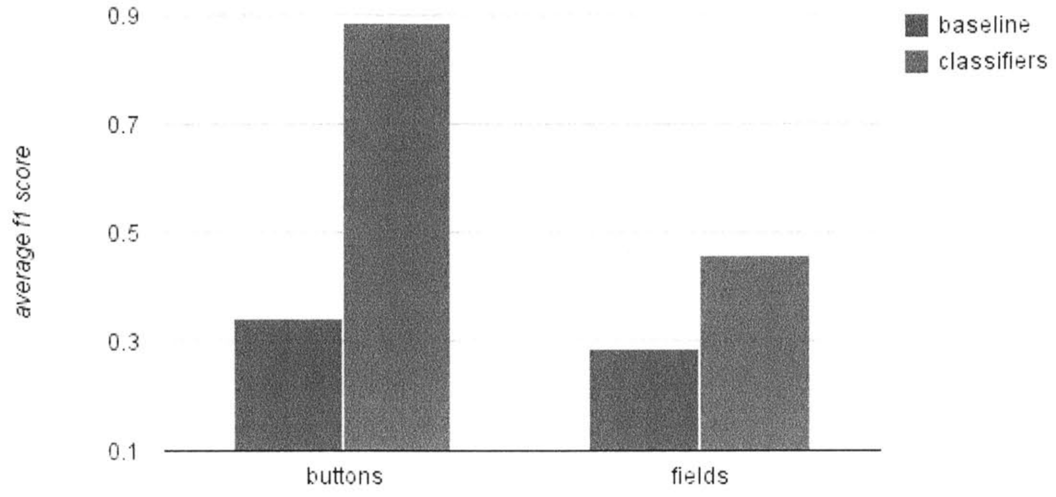


Figure 4-4: A comparison of the f1 scores for the baseline and gradient boosting button and field classifiers.

examples of all but two of the field types.

The baseline page classifiers were surprisingly effective, although they tended towards false positives in the "login" case. It therefore still makes sense to use the field classifiers as feature inputs to the overall page classifiers, as it is useless for our purposes to believe that a task can be completed on a given page without knowing how to complete it.

A graphical comparison of the performance between the baseline classifiers and the trained gradient boosting classifiers can be found in figure 4-4.

Chapter 5

The Affordance-Based Labeler (ABL)

5.1 Technical Goals

While I was able to achieve relatively good performance with classification, the performance of my classifiers was not good enough to trust them to identify fields and tasks on their own. Additionally, given the diversity in the labeling of HTML inputs, it would be unwise to assume that the classifiers would be able to correctly label all new web pages. I therefore chose to incorporate the classifiers into an Affordance-Based Labeler (ABL) so that they could help developers carry out the labeling tasks that are necessary for automated web testing, reducing developer workload while still maintaining accuracy.

The goal of the labeling tool (ABL) is not to accurately label every field on every page. Rather, it identifies and exposes potentially relevant fields to the human users, allowing them to more quickly label pages. It also learns from this human-generated input by incorporating newly-labeled examples into its training corpus. Finally, it outputs the classification results in a format that is easily parsed by GOSMR so that it is easy to introduce new pages.

Without this tool, developers need to search through the HTML source of the page, identify each relevant input field, and write out GOSMR affordances for each task that can be completed, which is a time-intensive (and dull) process. With this tool, developers can rapidly and easily add new web pages to GOSMR.

5.2 Specifications

The tool uses the BeautifulSoup [15] and LXML [1] Python packages for parsing HTML. It uses the native Python `webbrowser` package to display web pages.

The tool has several key methods:

5.2.1 `label(url)`

The `label` method takes a `url` as an argument. It displays the web page to the user in a web browser and uses its trained classifiers to label all of the visible HTML inputs - defined as inputs that are not HTML type "submit" and do not contain the word "hidden". It then iterates through these inputs one by one and asks the user to confirm (or correct) its labeling decisions or to provide a label if it was unable to classify the input. It then uses the output of this user classification as input to the page classifier and requests the user to confirm (or correct) its classifications. Finally, it attempts to classify all buttons on the page, using the button classifiers from the identified task, and it outputs information to be input into GOSMR in the following format:

```
GOSMRAction
Selenium 'by'input text type GOSMR info
Selenium 'by'input text type GOSMR info
...
Selenium 'by'button text click
```

One output section is generated per labeled button on the page. The `GOSMRAction` is the particular name for the action to be completed from GOSMR (`LoginAction`, `SearchAction` etc.). The `Selenium 'by'` is a valid input to Selenium's `By` action[1] -

[1]http://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/By.html

currently implemented as the input or button's `Id` if one is available, then `Name`, then `LinkText`. The `GOSMR info` is the information that each GOSMR model should enter in this field - `first name`, `email`, page-specific `username` etc. The keyword "text" denotes that this information should be typed into the input field, and the keyword "click" denotes that the button should be clicked.

One thing to note is that the labeling tool instructs the GOSMR agent to fill out all of the input fields on a web page rather than only filling out the fields that apply to the specific action that the agent is carrying out. Currently, the tool does not attempt to determine whether filling out an input field is necessary to completing a task, but this information exists in the labeled database of samples and could be incorporated at a later date if desired.

5.2.2 `retrain`

Each page and its inputs that are labeled using the labeling tool are incorporated into the database of training examples for the classifiers. Calling the `retrain` method causes all of the field, page, and button classifiers to be refit to the data corpus.

`retrain` uses the existing database of text attributes to be used to convert the samples to binary inputs. If a user wants to incorporate text attributes from new samples, he or she should call the `relabel` method before calling `retrain`. These methods are provided separately because the `relabel` method gets substantially slower as the size of the corpus increases, while the `retrain` method's runtime remains relatively constant, and so users may want to retrain their models without waiting to relabel each of the training examples.

5.2.3 `relabel`

Relabeling affects the way that each input example and button are converted to their binary representations. A list of text attributes is saved and used to convert each example. These text attributes are the values of the different HTML fields - an example HTML snippet and its attributes can be found in figure 5-1. A list of all

```
<input class="sw_qbox"                   sw_qbox
id="sb_form_q"                           sb_form_q
maxlength="1000"                         1000
name="q"                                 q
title="Enter your search term"           enter your search term
type="text"                              text
value=""
autocomplete="off">                      off
```

(a) HTML that specifies a search box. (b) Text attributes of the search box.

Figure 5-1: HTML and text attributes of a search box.

of the text attributes in all of the samples is stored, and for each sample, the binary representation of that sample is the list of text attributes, with each value set to "1" if that value is present in the sample and "0" if it is not.

Relabeling the samples causes a new text attribute list to be generated and each sample to be re-converted to its binary representation. Classifiers must be retrained after samples are relabeled, so calling `relabel` also causes `retrain` to be called.

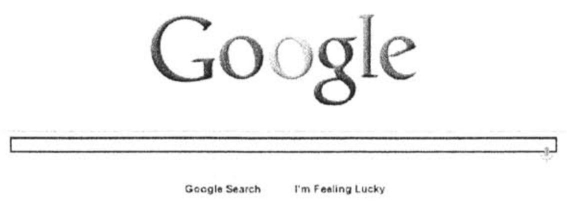
5.3 Performance

ABL has two components, a console component that takes in user input and a browser component that displays the web page that is currently being classified. It highlights the input or button that it is currently asking the user to classify.

Sample console output and web output can be seen in figure 5-2.

In this case, the ABL field classifiers were unable to classify the input, requiring help from the user. In other cases, ABL provides its classification and asks the user to confirm - this console output can be seen in figure 5-3.

Once the user has assisted ABL in classifying the fields on the page, ABL runs through the output of the page classifiers and asks the user to confirm each one - in the case of the two examples shown, ABL would ask "Can this page be used for search?" Once the user has identified the tasks that can be completed on each page, ABL asks the user about the buttons on the page that it has identified as being



(a) ABL highlights inputs and asks the user to classify them

```
<input id="gbqfq" class="gbqfif" name="q"
type="text" autocomplete="off" value=""
style="border: none; padding: 0px; margin: 0px;
height: auto; width: 100%; background-image:
url(data:image/gif;base64,R0lGODlhAQABAID/AMDAw
AAAACH5BAEAAAAALAAAAAABAAEAAAICRAEAOw%3D%3D);
background-color: transparent; position: absolute;
z-index: 6; left: 0px; outline: none;
background-position: initial initial;
background-repeat: initial initial;"
x-webkit-speech="" x-webkit-grammar="builtin:search"
lang="en" dir="ltr" spellcheck="false">

What is this?
```

(b) ABL displays input HTML to help the user classify the inputs

Figure 5-2: ABL helping a user classify `google.com`

```
<input class="sw_qbox" id="sb_form_q" maxlength="1000" name="q"
title="Enter your search term" type="text" value="" autocomplete="off">

Is this a search box?
```

Figure 5-3: ABL asks the user to confirm its classifications

potentially useful.

Currently, in order to display the highlighting to the user, ABL writes the modified HTML output to a separate file that the user must refresh each time ABL attempts to classify a new input. This is suboptimal, and in the future, I hope to modify ABL to be completely web-based for a better user experience. However, the current implementation is very lightweight, and depending on the HTML text attributes of the buttons and inputs, it is possible that the user will not need to look at the web interface at all.

Chapter 6

Conclusions

In this thesis, I have designed and implemented ABL, a tool to allow for better classification of web pages. ABL enables users to quickly and easily classify the different types of input fields that are present on a web page, the different tasks that can be completed on that web page, and the buttons that must be clicked in order to complete that task. It also outputs this information in a format that is easy to import into GOSMR, which will allow developers to easily incorporate new web pages into the existing GOSMR and LARIAT frameworks to use in large-scale tests.

ABL is enabled by several machine learning classifiers that determine the relevant inputs on each page, attempt to classify them, and then use this information to classify the type of page itself (and determine which button needs to be clicked). The design of the two-tiered system of classifiers was inspired by Gibson's theory of affordances [5] and justified by experimental results, which I also presented.

I generally found that simply using the trained classifiers to classify the input fields and the page type was not enough. While some of the field type classifiers were reliable, when used as features to the overall page classifier, their individual errors tended to compound and detract from the overall performance of the classifier. Instead, I found that it was more reliable to use human input to check the results of the individual field classifiers and use these results as input to the page classifier. It is possible that, with more data, the classifiers will become more reliable, but for now, using human input not only allows for better performance, it makes it easy to incorporate new, reliable training examples into the corpus.

The output of the labeling tool can be easily fed into GOSMR, and it could be modified to not require human input at all. Instead, the tool could output its classifications of each input field and GOSMR models could use these classifications to attempt to submit the form. Given the huge, parallel nature of the LARIAT infrastructure, it would be easy to run many models at once, which would allow for fast, automatic labeling of many web pages.

It is possible that I could have implemented this structure instead - many models, all attempting to complete the same task in a different way - and brute-forced the solutions to the labeling problem rather than attempting to devise a machine learning approach. But, this would be slow and wasteful of resources, and if any web pages changed, the solutions may no longer be applicable. Therefore, I believe that it is still best to use a classification model and to update that model so that it can help generalize to new web pages.

It would also be interesting to see how the existing field models can be used to learn new tasks. It seems that, if these new tasks required much of the same information as the four tasks that the model can currently identify, it would be easy to learn whether new tasks can be completed on a given web page. This approach has great potential to be generalized to a variety of different pages, and it would be exciting to do so.

Ultimately, it would be exciting to see if using the tool and adding a significant amount of additional training data to the corpus would allow the model to operate essentially autonomously - correctly filling in different fields, leaving out fields to see whether they are required information to complete a task, and learning to use entirely new websites simply because they are similar to websites that it has seen before. The tool certainly makes it easy to incorporate new data into the training corpus, so this goal may just be a matter of time. Creating a system that is able to learn from its mistakes would go a long way towards autonomy - and towards eventually interacting with the live web.

Just as many others have before me, I have found approaching a task through the lens of affordances to be useful. The tool that I have created is a foundation for building a truly automated web page labeling system, and I hope to soon see GOSMR models interacting with the live web!

Appendix A

Field descriptions

My data set contained examples of the following 22 fields, plus an additional `other` category for fields that had fewer than 5 examples.

- `search box` - an input where a set of keywords could be entered and information relevant to that query would be returned.
- `secret question` - a field for entering a security question. Could be either user-supplied text or selected from a drop-down menu.
- `city` - a field for entering the name of a city. Generally user-supplied text.
- `account number` - the user's existing account number, often for rewards sign-ups.
- `title` - the formal title for the user (Mr., Mrs. etc.).
- `state` - the user's state, often a drop-down menu.
- `birth day` - the day of the user's birth, often a drop-down menu.
- `email` - the user's e-mail address, when not used as a username.
- `username` - a unique identifier for the user on a particular site. Could be text or an e-mail address.

- `company` - the user's company.
- `phone number` - the user's phone number. Sometimes appeared more than once on a page if split into multiple chunks of digits.
- `middle name` - the user's middle name or middle initial.
- `birth year` - the year of the user's birth.
- `first name` - the user's first name.
- `address` - the user's street address. Does not contain city, state, zip code etc.
- `password` - the user's site-specific password - always a text field.
- `last name` - the user's last name.
- `gender` - the user's gender. Typically a drop-down menu or radio button.
- `zip code` - the user's postal zip code.
- `country` - the user's country. Typically a drop-down menu.
- `birth month` - the user's birth month. Typically a drop-down menu containing either month names or numbers.
- `secret answer` - an answer for the user's security question. Always a text field.

Appendix B

Classifier performance vs. tree depth

Hastie et al. comment that, for most data sets, gradient boosting classifiers typically work poorly with a tree depth ≤ 2 (tree stumps), that they work well with a tree depth of $4 \leq d \leq 8$, and that depths ≥ 10 are typically unnecessary . I found that this was true for my data set as well.

I ran the gradient boosting classifier with depths of 2 to 15 on the `search box` examples. I used the 97 positive examples of search boxes and 485 negative examples of search boxes. The results of varying the tree depth can be found in figure B-1.

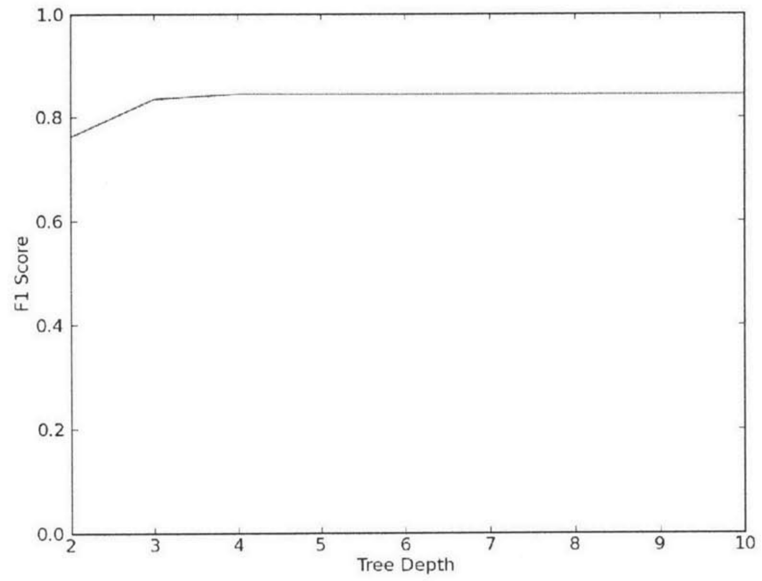


Figure B-1: The relationship between the depth of the decision trees used in the gradient boosting classifiers and the F1 measure, run on the `search box` data set.

Bibliography

[1] Stefan Behnel, Martijn Faassen, and Ian Bicking. lxml - processing xml and html with python. `http://lxml.de/`, 2013.

[2] Takeshi Chusho, Katsuya Fujiwara, and Keiji Minamitani. Automatic filling in a form by an agent for web applications. In *Software Engineering Conference, 2002. Ninth Asia-Pacific*, pages 239–247. IEEE, 2002.

[3] Robert B Doorenbos, Oren Etzioni, and Daniel S Weld. A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the first international conference on Autonomous agents*, pages 39–48. ACM, 1997.

[4] Paul Fitzpatrick, Giorgio Metta, Lorenzo Natale, Sajit Rao, and Giulio Sandini. Learning about objects through action-initial steps towards artificial cognition. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3140–3145. IEEE, 2003.

[5] James J. Gibson. *Perceiving, Acting and Knowing*, chapter 8 - The Theory of Affordances. Lawrence Erlbaum Associates, 1977.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.*, chapter 10 - Boosting and Additive Trees. Springer, 2009.

[7] Hartwig Hochmair and Andrew U Frank. A semantic map as basis for the decision process in the www navigation. In *Spatial Information Theory*, pages 173–188. Springer, 2001.

[8] Harri Jäppinen. Sense-controlled flexible robot behavior. *International Journal of Computer and Information Sciences*, 10(2), 1981.

[9] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and José Santos-Victor. Learning object affordances: From sensory–motor coordination to imitation. *Robotics, IEEE Transactions on*, 24(1):15–26, 2008.

[10] Donald A Norman. *The psychology of everyday things.* Basic Books (AZ), 1988.

[11] Donald A Norman. Affordance, conventions, and design. *interactions*, 6(3):38–43, 1999.

[12] Lidia Oshlyansky, Harold Thimbleby, and Paul Cairns. Breaking affordance: culture as context. In *Proceedings of the third Nordic conference on Human-computer interaction*, pages 81–84. ACM, 2004.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Selenium Project. Selenium - web browser automation. `http://docs.seleniumhq.org/`, 2013.

[15] Leonard Richardson. Beautiful soup: We called him tortoise because he taught us. `http://www.crummy.com/software/BeautifulSoup/`, 2012.

[16] Lee M Rossey, Robert K Cunningham, David J Fried, Jesse C Rabek, Richard P Lippmann, Joshua W Haines, and Marc A Zissman. Lariat: Lincoln adaptable real-time information assurance testbed. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 6, pages 6–2671. IEEE, 2002.

[17] Alexander Stoytchev. Behavior-grounded representation of tool affordances. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3060–3065. IEEE, 2005.

[18] Charles V Wright, Christopher Connelly, Timothy Braje, Jesse C Rabek, Lee M Rossey, and Robert K Cunningham. Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security. In *Recent Advances in Intrusion Detection*, pages 218–237. Springer, 2010.