Chatty Stochastic Multi-Armed Bandits

Akshay Kumar

February 18, 2014

Abstract

This thesis uses a variant of the classic stochastic multi-armed bandit framework to attempt to improve the average conversation quality of an online anonymous chat web application. This novel variant of the algorithm attempts to provide new conversation starters each time while still tracking optimality. MORE BLAH.

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 What is Tigers Anonymous? . . . 2
- 1.3 The Multi-Armed Bandit Problem and POM . . . 2

2 Methods **3**
- 2.1 UCB1 Algorithm . . . 3
- 2.2 Matching Heuristics . . . 3
- 2.3 POM Workflow . . . 3

3 Data **4**

4 POM Implementation **4**
- 4.1 Chatroom Back-end . . . 4
- 4.2 Web Server Back-end . . . 6
- 4.3 POM Front-end . . . 7

5 Conclusions **9**

Bibliography **9**

1 Introduction

1.1 Motivation

I believe that Tigers Anonymous (TA) satisfies a crucial need at Princeton (and potentially at other college campuses). As students become more settled within the community, it becomes increasingly harder to branch out and meet people outside one's immediate social graph. This is where TA comes in. By providing a way to anonymously be matched with, chat with, and potentially meet fellow classmates, TA allows students to make new connections and shake up their social network. Additionally, I wanted my thesis to be a tangible application of the multi-armed bandit problem and a step towards making Princeton more "hacker-friendly", both of which are satisfied by TA.

1.2 What is Tigers Anonymous?

Tigers Anonymous (TA) is the working title for a chat application that allows any Princeton student to be matched with another Princeton student. After being matched, the students will be taken to an anonymous chatroom where they can start a conversation. Once both participants have exchanged a pre-determined number of messages, a drop-down menu appears containing a button (see Figure 1)Additionally, there will be a button that, if both participants click, will authenticate both users via Facebook and reveal each users' identities to the other to facilitate communication outside of TA. The algorithm used to select matching heuristics is the main focus of this thesis and the success of the algorithm will be measured by whether users click on the "Reveal My Identity" button. For more information on how POM is implemented, see the "POM Implementation" section.

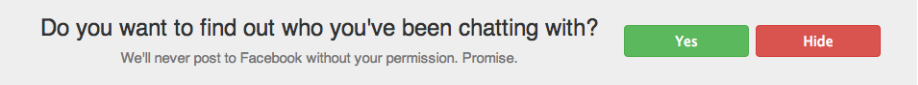

Figure 1: TA Drop-Down Menu

1.3 The Multi-Armed Bandit Problem and POM

This section will contain:

- Description of the multi-armed bandit problem and its common variants, using Bubeck and Cesa-Bianchi (2012) as a starting point
- Description of the bandit problem formulation used for POM and a justification for why it's used
- Description of the algorithms that can be used to solve the POM bandit problem and a justification of the UCB algorithm was chosen

2 Methods

2.1 UCB1 Algorithm

The tentative multi-armed bandit algorithm will be the UCB1 algorithm proposed by (Auer et al., 2002) and re-summarized below.

The UCB1 algorithm is initialized by playing each arm once and creating a vector $\bar{X} = [\bar{x}_1, \bar{x}_2, ..., \bar{x}_K]$ where K is the number of matching heuristics to be used and $\bar{x}_i$ is the empirical estimate of the probability that "Reveal My Identity" will be clicked given the use of matching heuristic i to pair the users. Then, for each subsequent matching, we pick the matching heuristic i using the following rule:

$$\underset{i \in K}{\operatorname{argmax}} \bar{x}_i + \sqrt{\frac{2 \ln n}{n_i}}$$

In this equation n_i is the number of times that machine i has been played and n is the total number of matches that have been completed.

2.2 Matching Heuristics

Let $\bar{t}_j$ be the average number of times that user j has clicked the "Reveal My Identity" button (i.e. the empirical estimate of the probability that they will click the button) and let $\triangle_j$ be the time since user j has last visited the site (with $\triangle_j = M$ if user j is a new visitor, and M is an arbitrarily large number). Let U be the set of users available to be matched and let i be the index of the user that needs to be matched. The following are the matching heuristics that will serve as the set of K arms being used in the UCB1 algorithm.

- Random: $j \in U$
- Hi-Lo: $\underset{j \in U}{\operatorname{argmax}} |\bar{t}_j - \bar{t}_i|$
- Hi-Hi: $\underset{j \in U}{\operatorname{argmin}} |\bar{t}_j - \bar{t}_i|$
- Far-Near: $\underset{j \in U}{\operatorname{argmax}} |\triangle_j - \triangle_i|$
- Far-Far: $\underset{j \in U}{\operatorname{argmin}} |\triangle_j - \triangle_i|$
- More to come!

2.3 POM Workflow

The basic workflow of using the upper-confidence bound algorithm for multi-armed stochastic bandits is as follows:

1. New user connects to POM.

2. If the queue size is below the threshold, place the new user into the queue.
3. If the queue size is above the threshold, insert the new user into the queue, remove the user from the head of the queue and use the UCB1 algorithm to pick the matching algorithm to match the selected user with one of the users in the queue.
4. Observe whether or not the "Reveal My Identity" button was clicked and update the $\bar{x}_i$ and n_i for the chosen arm i.
5. Record the 6-tuple representing the data point of this chat session (see Data section below for more details).

3 Data

The data that will be collected can be represented by the vector of 6-tuples $(a_i, b_i, k_i, l_i, t_i, p_i)$ where a_i and b_i represent the user ids of the two participants in the chat, k represents the index of the matching heuristic used, l represents the length of the chat conversation, t represents the time that the chat was initiated, $p \in (0, 1)$ is a discrete variable with 0 representing failure to click "Reveal My Identity" button and 1 representing success, and i representing the unique ID of the chat conversation.

The efficacy of the UCB1 algorithm will be assessed by testing the average value of l and p over time to see whether the optimized matching heuristics cause people to chat longer and/or be more likely to click "Reveal My Identity". Additionally, the average values of l and p for each matching heuristic k will be analyzed to see if there are statistically significant differences between the matching heuristics.

4 POM Implementation

The following pieces of code implement the back-end and front-end functionality of the basic POM prototype.

4.1 Chatroom Back-end

This module is used to facilitate multiple anonymous chatrooms, as well as having the code to implement the matching heuristics. The code below is written in a modular way so that multiple heuristics can be implemented using the UCB1 algorithm in the next stage of development (i.e. the getItem method and the SocketWrapper object to manage client-side user data). User history data will be stored as a cookie on the client-side, thus eliminating the need to manage a large database of user history data on the server-side. The only database that needs to be maintained on the server-side is the vector of the historical performance of each of the matching heuristics.

```
var all_sockets = null;
var threshold = 0;
var queue = new Queue();
var myName = 'Me';
var theirName = 'Anonymous Tiger';

exports.set_sockets = function (sockets) {
  all_sockets = sockets;
};

exports.connect_chatter = function (current_socket) {

    var partner;
    var currentSocketWrapper = {socket: current_socket, userdata: null};

    current_socket.emit('entrance', {message: 'Welcome to the chat
        room!'});
    if (queue.length() <= threshold) {
        queue.addItem(currentSocketWrapper);
        current_socket.emit('waiting', {message: 'Waiting for partner to
            join.'});
        current_socket.on('disconnect', function() {
                queue.removeItem(currentSocketWrapper);
            });
    }

    else {
        partner = queue.getItem(currentSocketWrapper, 0);
        current_socket.emit('ready', {message: 'Connected! Go ahead and
            start chatting.'});
        partner.socket.emit('ready', {message: 'Connected! Go ahead and
            start chatting.'});

        current_socket.on('disconnect', function() {
                partner.socket.emit('exit', {message: theirName + ' has
                    disconnected. Refresh the page to start another
                    chat!'});
            });
        partner.socket.on('disconnect', function() {
                current_socket.emit('exit', {message: theirName + ' has
                    disconnected. Refresh the page to start another
                    chat!'});
            });

        current_socket.on('chat', function(data) {
                current_socket.emit('chat', {message: myName + ': ' +
                    data.message});
                partner.socket.emit('chat', {message: theirName + ': ' +
                    data.message});
```

```
            });
        partner.socket.on('chat', function(data) {
                current_socket.emit('chat', {message: theirName + ': ' +
                    data.message});
                partner.socket.emit('chat', {message: myName + ': ' +
                    data.message});
            });

    }
};

exports.failure = function (socket) {
    socket.emit('error', {message: 'Please log in to the chatroom.'});
};

function Queue ()
{
    this.array = new Array();
    this.addItem = function(item) {
        this.array.push(item);
    }
    this.getItem = function(current_socket, matching_algorithm) {
        return this.array.shift();
    }
    this.removeItem = function(item) {
        var location = this.array.indexOf(item);
        if (location !== -1) {
            this.array.splice(location, 1);
        }
    }
    this.length = function() {
        return this.array.length;
    }
};
```

4.2 Web Server Back-end

This module runs the web-server that POM is running on. IP address filtering (to Princeton internal network IP addresses) and Facebook integration will be implemented in this module.

```
var io = require('socket.io'),
  connect = require('connect'),
  chatter = require('chatter');
var port = process.env.PORT || 3000;
var app = connect().use(connect.static('public')).listen(port);
\\ Implement IP address filtering here
var chat_room = io.listen(app);
```

```
chatter.set_sockets(chat_room.sockets);
chat_room.sockets.on('connection', function (socket) {
  chatter.connect_chatter(socket);
});
```

4.3 POM Front-end

This HTML page provides the basic user interface for POM. It will be updated to use Angular.js so that POM looks more visually appealing. The picture below is the extremely bare-bones version of the prototype that is currently being hosted on Heroku for testing.

```
<html>
<head>
<script
    src="http://akshays-chatroom.herokuapp.com/socket.io/socket.io.js"></script>
<script src="jquery-1.7.2.min.js"></script>
<script type="text/javascript" charset="utf-8">
jQuery(document).ready(function () {
       var log_chat_message = function (message, type) {
              var li = jQuery('<li />').text(message);

              if (type === 'system') {
                     li.css({'font-weight': 'bold'});
              } else if (type === 'leave' || type === 'error') {
                     li.css({'font-weight': 'bold', 'color': '#F00'});
              }

              jQuery('#chat_log').append(li);
       };

       var socket = io.connect('http://akshays-chatroom.herokuapp.com');

       socket.on('entrance', function (data) {
              log_chat_message(data.message, 'system');
       });

       socket.on('waiting', function(data) {
              log_chat_message(data.message, 'system');
       });

       socket.on('ready', function(data) {
              log_chat_message(data.message, 'system');
       });

       socket.on('exit', function (data) {
              log_chat_message(data.message, 'leave');
       });
```

```
        socket.on('chat', function (data) {
                log_chat_message(data.message, 'normal');
        });

        socket.on('error', function (data) {
                log_chat_message(data.message, 'error');
        });

        jQuery('#chat_box').keypress(function (event) {
                if (event.which == 13) {
                        socket.emit('chat', {message:
                            jQuery('#chat_box').val()});
                        jQuery('#chat_box').val('');
                }
        });
});

</script>
<style type="text/css" media="screen">
        div#chatroom {
                display: block;
                height: 300px;
                border: 1px solid #999;
                overflow: auto;
                width: 100%;
                margin-bottom: 10px;
                position: relative;
        }

        ul#chat_log {
                list-style: none;
                position: absolute;
                bottom: 0px;
        }

        input#chat_box {
                width: 99%;
        }
</style>
</head>
<body>

        <div id="chatroom">
                <ul id="chat_log">
                </ul>
        </div>

        <input type="text" name="chat_box" value="" id="chat_box"
            placeholder="type to chat..." />
```

```
</body>
</html>
```

5 Conclusions

Analysis of data gathered and conclusions drawn go here.

References

Peter Auer, Nicoló Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256, 2002.

Sébastien Bubeck and Nicoló Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.