Articles » General Programming » Artificial Intelligence and Machine Learning » Machine Learning and AI Challenge

"Birds of a Feather stick together" To Produce Users-To-Items Recommendations In JavaScript

Arthur V. Ratz, 7 Mar 2018

In this article we'll discuss on how to produce user-to-item recommendations by using SVD++, Pearson Correlation and probability-based similarity computation

Editorial Note

This article is an entry in our Machine Learning and Artificial Intelligence Challenge. Articles in this sub-section are not required to be full articles so care should be taken when voting.

Download u2i_recommend.zip - 847 KB

Introduction

In this article, we'll introduce and demonstrate the using of an approach that allows to produce user-to-item recommendations based on the data on recent users site activities being logged. The approach discussed in this article addresses one of the problems proposed as "Machine Learning And AI Challenge".

Specifically, in this article we'll discuss on how to use various data mining algorithms to solve the problem of finding the probability that a specific user would recommend an article they've viewed in a social media web site, to the population of other users that view and read articles in the following web site. To build a recommendation model we'll use such famous algorithms as SVD++, Pearson correlation, probability-based similarity computation algorithm, etc.

Background

In this section, we'll find out how to use the foundation of various data mining and AI machine learning algorithms to produce efficient User-To-Item recommendations by performing model-based collaborating filtering (CF). Specifically, with the given assignment, our goal is to build an appropriate model that allows to determine the probability of a user to either positively or negatively rate a certain article they've viewed as well as to recommend it to other users based on the "incomplete" data collected and represented as each user's activity logs.

User-To-Item Recommendations Data Model

According to the challenge assignment, we've basically dealt with three datasets containing specific data on either articles, users or log entries that describe an activity of each particular user. The articles dataset includes a set of tags which serve as attributes that describe each particular article. Obviously that, more than one article can have the same values of attributes that makes it possible to combine the most similar articles having a similar description belonging to the same class.

In turn, the dataset of users has a very few, inconsistent data on the personal preferences of each user. That's actually why, to produce recommendations we basically rely on each user's activities factors collected and stored into the third dataset of log entries.

To produce efficient recommendations for each user or a group of users we need to:

1. Determine the probability-based value of relevance between particular articles and build a similarity matrix;
2. Analyze the log entries and collect statistical data (e.g. the number of views, upvotes and downvotes, ...) on each particular article;
3. Associate each article with one or multiple users based on specific log entries being analyzed;
4. Encode and normalize the statistical data being collected and produce factorization vectors for each article entry;
5. Use SVD++ algorithm [2] to train the data model to predict the similarity between a pair of given articles based on the data being previously collected and analyzed;

Since the particular users and articles data are not actually interrelated, we'll determine the similarity between specific tuple of a user and article based on the hypothesis that an article A could be recommended to specific users who have been interested in the article B if and only if those two articles are the most similar. Thus, by determining the similarity of specific articles, we're actually finding a likelihood between particular users based on their preferences and interest to read the same articles. The data model in which the either users or articles data has been organized is shown on Fig. 1, below:

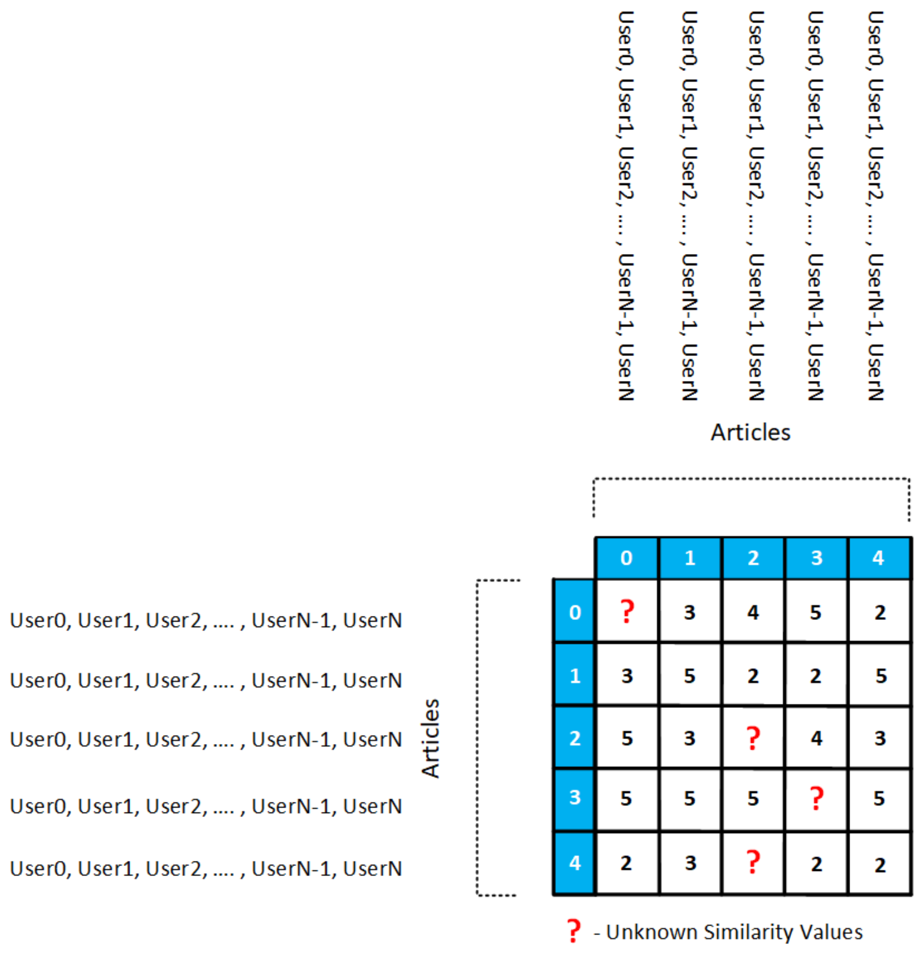
User0, User1, User2, ... , UserN-1, UserN
Articles
0
1
2
3
4
?
3
4
5
2
3
5
2
2
5
5
3
?
4
3
5
5
5
?
5
2
3
?
2
2
? - Unknown Similarity Values

Fig. 1. Matrix of Articles Similarity

Similarity Computation

As we’ve already mentioned above, each article is associated with a set of tags that describe an article. Unfortunately, those tags are string values that cannot be represented as numerical values. That’s why to compute the similarity of a pair of articles we use a probability-based method, according to which we’re aiming to find the number of distinct tags existing in both sets of attributes for the first and second article respectively. After that by using a trivial formula from probability axiom we need to divide that number by the overall amount of tags in both sets:

$$S = \frac{N_s}{N}$$

, where Ns – the number of similar tags that describe the article i and article j;

N – the overall number of distinct tags for both article i and article j;

S – the measure of similarity between article i and article j;

Specifically, to build a similarity matrix we’re using an algorithm that can be formulated as follows. Normally, according to this algorithm we need to iterate through the dataset of articles and for each article perform a search to find a similarity of a current article i and each article j in the same dataset. Since the similarity has been computed, we’re assigning the following value to element R[i][j] of the similarity matrix.

Collecting Statistics

To associate each article with one or multiple users who had an either positive or negative activity on a specific article, as well as compute the values of partial probabilities and build specific factorization vectors, we’ll parse the activity logs dataset by using the algorithm according to which we’re iterating through the set of articles and for each article i we’re performing a search in the activity logs dataset to find those users who recently had an activity on the current article i. Then we simply map each user matching the following criteria to the current article in the articles dataset.

Also, during this process, we’re obtaining such statistical data as the number of upvotes/downvotes, downloads or views for the current article i. These data is used to compute the partial probability values used to predict the interest and popularity of the given article.

Simultaneously, we’ll use the statistical data being collected to build factorization vectors for the model learning process.

Encoding Data

To build factorization vectors that further will be used during the model learning process, we’ll use the statistical values of the number of views, upvotes, downvotes and downloads as the components of each factorization vector:

$$\overline{V_f} = \{V_{views}, V_{upvotes}, V_{downvotes}, V_{downloads}\}$$

Prior to using these values, we need to normalize each of them so that they belong to the interval [0;1] by using the following formula:

$$V_{norm} = \frac{|V - V_{min}|}{|V_{max} - V_{min}|}$$

Using SVD++ Algorithm To Train Prediction Model

To train the prediction model being discussed we'll use the SVD++ algorithm that was introduced and formulated in [1]. According to the following algorithm we'll need to adjust base-line predictors Bi and Bj for each tuple of similar articles i and j based on using the stochastic gradient descend (SGD) and ordinary least squares (OLS) methods.

The SVD++ algorithm has the following steps and can be formulated as follows:

1. Compute the "so-far" estimated value of rating $\widehat{r}_{i,j}$ by using formula (2): $\widehat{r}_{i,j} = \mu + b_i^U + b_j^I + \overline{u_i v_j^T}$;
2. Find the error value by subtracting the value of the estimated rating $\widehat{r}_{i,j}$ obtained at the previous step from the value of the existing rating as follows: $\varepsilon_{i,j} = r_{i,j} - \hat{r}_{i,j}$;
3. Compute the square of the error value $\varepsilon_{i,j}^2$ obtained at the previous step and add this value to the sum of error squares;
4. Perform an update of the current value of the average rating: $\mu = \mu + \eta(\varepsilon_{i,j} - \lambda\mu)$
5. Perform an update of the current value of baseline predictor of user i: $b_i^U = b_i^U + \eta(\varepsilon_{i,j} - \lambda b_i^U)$;
6. Perform an update of the current value of the baseline predictor of item j: $b_j^I = b_j^I + \eta(\varepsilon_{i,j} - \lambda b_j^I)$;
7. Perform an update the value of each latent factor in the factorization vector of user i: $u_i = u_i + \eta(\varepsilon_{i,j} v_j^T - \lambda u_i)$;
8. Perform an update the value of each latent factor in the factorization vector of item j: $v_j^T = v_j^T + \eta(\varepsilon_{i,j} u_i - \lambda v_j^T)$

The prediction model training process is performed within multiple "epochs". During each epoch we're aiming to adjust all those coefficients to minimize least-mean-square error value. We're proceeding with the following process until it has converged with the desired error precision value.

Pearson Correlation Formula

To compute the most adequate similarity values and to speed up the prediction model learning process, we'll also use the famous Pearson correlation formula that is closely related to the formula used for computing an angle between two vectors in n – dimensional space:

$$r_{xy} = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2 \sum_{i=1}^{m}(y_i - \bar{y})^2}} = \frac{cov(x,y)}{\sqrt{s_x^2 s_y^2}},$$

During the model training process, we simply divide the scalar product of two factorization vectors by value of Pearson correlation coefficient.

Prediction

Since, we've trained our prediction model, now we can predict an interest of a user to a specific article. To predict the probability that a user would recommend an article to one or more other users we need to iterate through the articles dataset and for each article verify if the given user recently viewed, upvoted or downvoted this article. If so, we're computing the value of probability by using the following formula:

$$\widehat{r}_{i,j} = \mu + b_i^U + b_j^I + \overline{u_i v_j^T};$$

Using the code

```
<!DOCTYPE html>
<html>
 <head>
  <title>User-To-Item Recommender Engine v.1.0a</title>
 </head>
 <body>
  <table border="1" style="width: 1200px;">
   <tr>
    <td align="center"><p style="font-size:30px;"><b>User-To-Item Recommender Engine v.1.0a<b></p></td>
   </tr>
   <tr>
    <td>
     <form>
```

```
      <div>
       <label for="datafile_upload">
        <strong>Upload Data File (*.txt):</strong>
       </label>
       <input type="file" id="datafile_upload" accept=".txt" onchange="loadData();">
       </div>
     </form>
    </td>
   </tr>
   <tr>
    <td>
     <table border="1">
      <tr>
       <td>
        <table>
         <tr>
          <td><button onclick="renderArticles();">Articles</button></td>
          <td><button onclick="renderUsers();">Users</button></td>
          <td><button onclick="renderFactors();">Logs</button></td>
          <td><button onclick="renderStats();">Statistics</button></td>
          <td><button onclick="renderResults();">Results</button></td>
         </tr>
        </table>
       </td>
      </tr>
      <tr>
       <td>
        <div id="train_set" style="width: 1200px; height: 500px; overflow-y: scroll;"></div>
       </td>
      </tr>
     </table>
    </td>
   </tr>
   <tr><td><span id="status"></span></td></tr>
   <tr>
    <td>
     <table>
      <tr>
       <td>
        User: <input type="text" id="user" value="" size=200><br>
        Article: <input type="text" id="article" value="" size=200><br>
        <button onclick="predict();">Predict</button>
       </td>
      </tr>
      <tr>
       <td>
        <table>
         <tr><td><b>Recommended:</b><span id="rc_p"></span>%</td></tr>
         <tr><td><b>Viewed:</b><span id="view_p"></span>%</td></tr>
         <tr><td><b>Upvoted:</b><span id="upvote_p"></span>%</td></tr>
         <tr><td><b>Downvoted:</b><span id="downvote_p"></span>%</td></tr>
         <tr><td><b>Downloaded:</b><span id="download_p"></span>%</td></tr>
        </table>
       </td>
      </tr>
     </table>
    </td>
   </tr>
   <tr>
    <td>

    </td>
   </tr>
   <tr><td align="center"><b>CPOL (C) 2018 by Arthur V. Ratz</b></td></tr>
  </table>
</body>
<script>
 var users_ents    = new Array();
 var factors_ents   = new Array();
 var articles_ents = new Array();
 var rel_table = new Array();
```

```
  var trained = 0;
  var p_avg = 0, rel_table = new Array();
  var alias = [ "# Articles", "# Users", "# User actions" ];
  var max_views = 0, max_upvotes = 0, max_downvotes = 0, max_downloads = 0, max_logs = 0;
  function loadData()
  {
       var file_reader = new FileReader();
       var fp = document.getElementById("datafile_upload");

       file_reader.onload = function() {

           var contents = file_reader.result;
           var lines_array = contents.split("\r");
           var is_article = 0, is_user = 0, is_factor = 0;
           for (var r = 0; r < lines_array.length; r++)
           {
                if (lines_array[r] == "\n" + alias[0]) { is_article = 1; is_user = 0; is_factor = 0; }
                if (lines_array[r] == "\n" + alias[1]) { is_article = 0; is_user = 1; is_factor = 0; }
                if (lines_array[r] == "\n" + alias[2]) { is_article = 0; is_user = 0; is_factor = 1; }

                if (lines_array[r][0] == '\n' && !isNaN(parseInt(lines_array[r][1], 10)))
                {
                    var dataset_raw = lines_array[r].split(",");
                    if (is_article == 1) {
                   var attr_array = dataset_raw.slice(2, dataset_raw.length);
          articles_ents.push({ "id" : dataset_raw[0], "name" : dataset_raw[1],
       "attrs" : attr_array, "stats" : { "views" : 0, "upvoted" : 0, "downvoted" : 0,
    "downloads" : 0, "logs" : 0, "users1" : null, "users2" : null, "bias" : 0.01, "vf" : null } });
                    }

                    else if (is_user == 1) {
                        users_ents.push({ "id" : dataset_raw[0], "name" : dataset_raw[1] });
                    }

                    else if (is_factor == 1) {
                        factors_ents.push({ "day" : dataset_raw[0], "action" : dataset_raw[1],
"user_id" : dataset_raw[2],
                             "user_name" : dataset_raw[3], "article_id" : dataset_raw[4],
"article_name" :  dataset_raw[5]});
                    }
                }
           }

           document.getElementById("status").innerHTML = "Processing...";

           update_stats(articles_ents, factors_ents);
       }

       file_reader.readAsText(fp.files[0], "UTF-8");
  }

function printOut()
{
     var ds = null, index = -1;
     var chunks = -1, tsID = -1;
     self.onmessage = function(e) {
             if (e.data["msg"] == "data_set") {
                 ds = e.data["value"];
             }
             else if (e.data["msg"] == "index") {
                 index = e.data["value"];
             }
             else if (e.data["msg"] == "chunks") {
                 chunks = e.data["value"];
             }
             else if (e.data["msg"] == "tsID") {
                 tsID = e.data["value"];
             }
             else if (e.data["msg"] == "invoke") {
                 console.log(ds.length);
```

```
                  var chunk_size = Math.ceil(ds.length / chunks);
           var start = (index * chunk_size) < ds.length ?
                (index * chunk_size) : ds.length;
                  var end = ((index + 1) * chunk_size) < ds.length ?
               ((index + 1) * chunk_size) : ds.length;

                 var ts_s = "";
                 for (var s = start; s < end; s++)
                 {
                      if(tsID == 0) {
                      ts_s += "<tr><td>" + ds[s]["name"] +
              "</td><td>" + ds[s]["attrs"].toString() + "</td></tr>\n";
                      }

                      else if(tsID == 1) {
                          ts_s += "<tr><td>" + ds[s]["name"] + "</td></tr>\n";
                      }
                      else if(tsID == 2) {
                          ts_s += "<tr><td>" + ds[s]["day"] + "</td>" +
                             "<td>" + ds[s]["action"] + "</td>" +
                             "<td>" + ds[s]["user_id"] + "</td>" +
                             "<td>" + ds[s]["user_name"] + "</td>" +
                             "<td>" + ds[s]["article_id"] + "</td>" +
                             "<td>" + ds[s]["article_name"] + "</td></tr>\n";
                      }
                      else if(tsID == 3) {
                          ts_s += "<tr><td><center>" + ds[s]["name"] +
        "</center></td><td><center>" + ds[s]["stats"]["upvoted"].toString() + "</center></td>" +
                        "<td><center>" + ds[s]["stats"]["downvoted"].toString() + "</center></td>" +
                             "<td><center>" + ds[s]["stats"]["downloads"].toString() + "</center></td>"
+
                             "<td><center>" + ds[s]["stats"]["logs"].toString() + "</center>
</td></tr>\n";
                      }
                      else if(tsID == 4) {
                          ts_s += "<tr><td><center>" + ds[s]["name"] +
                   "</center></td><td><center>" + ds[s]["stats"]["users1"].toString() + "</center></td>"
+
                             "<td><center>" + ds[s]["stats"]["users2"].toString() + "</center></td>" +
                             "<td><center>" + ds[s]["stats"]["bias"].toString() + "</center></td>" +
                             "<td><center>" + ds[s]["stats"]["vf"].toString() + "</center></td></tr>\n";
                      }
                 }

                 self.postMessage(JSON.stringify({ "ts_buf" : ts_s, "index" : index }, null, 3));
              }
      }
}

function renderData(tsID, ds)
{
       var thw_count = 0;
       var ts = new Array();

       var ts_buf = "";

       if (tsID == 0)
       {
           ts_buf = "<table border=\"1\" style=\"table-layout: fixed; overflow-x:auto; width: 100%;" +
         "word-wrap: break-word;\"><thead><th width=\"25%\">Article</th><th width=\"25%\">Tags</th>";

           ts_buf += "</thead><tbody>\n";
       }

       else if (tsID == 1)
       {
    ts_buf = "<table border=\"1\" style=\"table-layout: fixed; overflow-x:auto; width: 100%;" +
       "word-wrap: break-word;\"><thead><th width=\"25%\">User</th>";

           ts_buf += "</thead><tbody>\n";
       }
       else if (tsID == 2)
       {
       ts_buf = "<table border=\"1\" style=\"table-layout: fixed; overflow-x:auto; width: 100%;" +
              "word-wrap: break-word;\"><thead><th width=\"25%\">Day</th>";

           ts_buf += "<th width=\"25%\">Action</th>";
           ts_buf += "<th width=\"25%\">UserID</th>";
           ts_buf += "<th width=\"25%\">UserName</th>";
           ts_buf += "<th width=\"25%\">ArticleID</th>";
           ts_buf += "<th width=\"25%\">ArticleName</th>";

           ts_buf += "</thead><tbody>\n";
       }

       else if (tsID == 3)
       {
    ts_buf = "<table border=\"1\" style=\"table-layout: fixed; overflow-x:auto; width: 100%;" +
        "word-wrap: break-word;\"><thead><th width=\"25%\">Article</th>";

    ts_buf += "<th>Upvoted</th>";
    ts_buf += "<th>Downvoted</th>";
    ts_buf += "<th>Downloads</th>";
    ts_buf += "<th>Logs</th>";

           ts_buf += "</thead><tbody>\n";
       }

       else if (tsID == 4)
       {
    ts_buf = "<table border=\"1\" style=\"table-layout: fixed; overflow-x:auto; width: 100%;" +
        "word-wrap: break-word;\"><thead><th width=\"25%\">Article</th>";

    ts_buf += "<th>Positive</th>";
    ts_buf += "<th>Negative</th>";
    ts_buf += "<th>Bias</th>";
    ts_buf += "<th>Factorization Vector</th>";

           ts_buf += "</thead><tbody>\n";
       }

       var chunks = 50;
       if (ds.length / chunks < 1)
           chunks = 1;

       document.getElementById("train_set").innerHTML = "";

       for (var i = 0; i < chunks; i++)
       {
            var code = printOut.toString();
            code = code.substring(code.indexOf("{")+1, code.lastIndexOf("}"));
            var blob = new Blob([code], {type: "application/javascript"});

            var w = new Worker(URL.createObjectURL(blob));
            w.onmessage = function(e) {

                var json_obj = JSON.parse(e.data);

                if (thw_count == chunks - 1) {
                    for (var t = 0; t < ts.length; t++) {
                         ts_buf += ts[t];
                    }

                    document.getElementById("train_set").innerHTML = ts_buf + "</tbody></table>";
                }

                if (e.data != null) {
             ts[json_obj["index"]] = json_obj["ts_buf"];
                }

                thw_count++;
```

```
            }

            w.postMessage({ "msg" : "data_set", "value" : ds });
            w.postMessage({ "msg" : "index", "value" : i });
            w.postMessage({ "msg" : "chunks", "value" : chunks });
            w.postMessage({ "msg" : "tsID", "value" : tsID });

            w.postMessage({ "msg" : "invoke" });
        }
}

function renderArticles()
{
    if (trained == 0) {
        alert('Train the model first...');
        return;
    }

    renderData(0, articles_ents);
}

function renderUsers()
{
    if (trained == 0) {
        alert('Train the model first...');
        return;
    }

    renderData(1, users_ents);
}

function renderFactors()
{
    if (trained == 0) {
        alert('Train the model first...');
        return;
    }

    renderData(2, factors_ents);
}

function renderStats()
{
    if (trained == 0) {
        alert('Train the model first...');
        return;
    }

    renderData(3, articles_ents);
}

function renderResults()
{
    if (trained == 0) {
        alert('Train the model first...');
        return;
    }

    renderData(4, articles_ents);
}

function update_worker()
{
     var chunks = -1;
     var articles = null, factors = null, index = -1;
     self.onmessage = function(e) {

             if (e.data["msg"] == "articles") {
                 articles = e.data["value"];
             }
             if (e.data["msg"] == "factors") {
                 factors = e.data["value"];
             }
             else if (e.data["msg"] == "index") {
                 index = e.data["value"];
             }
             else if (e.data["msg"] == "chunks") {
                 chunks = e.data["value"];
             }
             else if (e.data["msg"] == "invoke") {

                 var chunk_size = Math.ceil(articles.length / chunks);
          var start = (index * chunk_size) < articles.length ?
               (index * chunk_size) : articles.length;
                 var end = ((index + 1) * chunk_size) < articles.length ?
              ((index + 1) * chunk_size) : articles.length;

                 max_views = 0; max_upvotes = 0;
                 max_downvotes = 0; max_downloads = 0; max_logs = 0;

                 for (var i = start; i < end; i++)
                 {
                      var logs_count = 0;
                      var users1 = new Array();
                      var users2 = new Array();
                      var views = 0, upvotes = 0;
                      var downvotes = 0, downloads = 0;
                      for (var j = 0; j < factors.length; j++) {
                           if (factors[j]["article_name"] == articles[i]["name"])
                           {
                               views     = (factors[j]["action"] == "View")     ? views + 1     :
views;
                               upvotes   = (factors[j]["action"] == "UpVote")   ? upvotes + 1   :
upvotes;
                               downvotes = (factors[j]["action"] == "DownVote") ? downvotes + 1 :
downvotes;
                               downloads = (factors[j]["action"] == "Download") ? downloads + 1 :
downloads;

                               if (factors[j]["action"] == "View" ||
                                   factors[j]["action"] == "UpVote" ||
                                   factors[j]["action"] == "Download") {
                                   users1.push(factors[j]["user_name"]);
                               }

                       else {
                                      users2.push(factors[j]["user_name"]);
                   }

                               if (views > max_views || max_views == 0)
                                   max_views = views;
                               if (upvotes > max_upvotes || max_upvotes == 0)
                                   max_upvotes = upvotes;
                               if (downvotes > max_downvotes || max_downvotes == 0)
                                   max_downvotes = downvotes;
                               if (downloads > max_downloads || max_downloads == 0)
                                   max_downloads = downloads;
                               if (logs_count > max_logs || max_logs == 0)
                                   max_logs = logs_count;

                               logs_count++;
                          }
                     }

                     articles[i]["stats"]["logs"] = logs_count;

                     articles[i]["stats"]["views"]     = views;
                     articles[i]["stats"]["users1"]    = users1;
                     articles[i]["stats"]["users2"]    = users2;
                     articles[i]["stats"]["upvoted"]   = upvotes;
                     articles[i]["stats"]["downvoted"] = downvotes;
                     articles[i]["stats"]["downloads"] = downloads;
                     var views_norm      = Math.abs(0.01 - articles[i]["stats"]["views"]) /
Math.abs(0.01 - max_views) / 10;
                     var upvotes_norm    = Math.abs(0.01 - articles[i]["stats"]["upvoted"]) /
Math.abs(0.01 - max_upvotes) / 10;
                     var downvotes_norm  = Math.abs(0.01 - articles[i]["stats"]["downvoted"]) /
Math.abs(0.01 - max_downvotes) / 10;
                     var download_norm   = Math.abs(0.01 - articles[i]["stats"]["downloads"]) /
Math.abs(0.01 - max_downloads) / 10;
                     var logs_count_norm = Math.abs(0.01 - articles[i]["stats"]["logs"]) /
Math.abs(0.01 - max_logs) / 10;

                     articles[i]["stats"]["vf"] = [ views_norm, upvotes_norm,
                  downvotes_norm, download_norm, logs_count_norm ];
                 }

                 self.postMessage(JSON.stringify({ "result" : articles, "start" : start, "end" : end },
null, 3));

             }
     }
}

  function update_stats(articles, factors)
  {
       var chunks = 10;
       if (articles.length / chunks < 1)
           chunks = 1;

       var thw_count = 0; var workers = [];
       for (var i = 0; i < chunks; i++)
       {
            var code = update_worker.toString();
            code = code.substring(code.indexOf("{")+1, code.lastIndexOf("}"));
            var blob = new Blob([code], {type: "application/javascript"});

            workers.push(new Worker(URL.createObjectURL(blob)));
            workers[i].onmessage = function(e) {

  if (thw_count == chunks - 1) {
                    a2a_table();
                    for (var q = 0; q < workers.length; q++) {
                         workers[q].terminate();
                    }
                }

                //else {
                    var json_obj = JSON.parse(e.data);
                    for (var t = json_obj["start"]; t < json_obj["end"]; t++) {
                         articles_ents[t] = json_obj["result"][t];
                    }

                    thw_count++;
                //}
            }

            workers[i].postMessage({ "msg" : "articles", "value" : articles_ents });
            workers[i].postMessage({ "msg" : "factors", "value" : factors });
            workers[i].postMessage({ "msg" : "index", "value" : i });
            workers[i].postMessage({ "msg" : "chunks", "value" : chunks });

            workers[i].postMessage({ "msg" : "invoke", "value" : null });
       }
  }

  function count_if(attrs_s, tag_name)
  {
       attrs_s  = attrs_s + ''; tag_name = tag_name + '';
       tag_name = tag_name.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
       return (attrs_s.match(new RegExp(tag_name, 'gi')) || []).length;
  }
```

```
function count_unique(attributes)
{
     var count = 0, tags = new Array();
     var attrs_text = attributes.toString();
     for (var i = 0; i < attributes.length; i++) {
          if (count_if(attrs_text, attributes[i]) == 1) {
              tags.push(attributes[i]); count++;
          }
     }

     return new Object({ "tags" : tags, "count_unique" : count });
}

function normalize(value, min, max)
{
     return Math.abs(min - value) / Math.abs(min - max);
}

function a2a_worker()
{
     function similarity(article_p1, article_p2)
     {
         var unique_attrs1 = article_p1["attrs"];
         var unique_attrs2 = article_p2["attrs"];

         unique_attrs1 = unique_attrs1.filter(function(tag_name, pos, tags)
  { return tags.indexOf(tag_name) == pos; });

         unique_attrs2 = unique_attrs2.filter(function(tag_name, pos, tags)
{ return tags.indexOf(tag_name) == pos; });

         var count_unique = 0;
         for (var i = 0; i < unique_attrs1.length; i++) {
              count_unique += (unique_attrs2.indexOf(unique_attrs1[i]) >= 0) ? 1 : 0;
         }

         return count_unique / (unique_attrs1.length + unique_attrs2.length);
     }

     var sum = 0, count = 0;
     var rl_value = 0, r = new Array();
     self.onmessage = function(e) {

           if (e.data["msg"] == "articles") {
               articles = e.data["value"];
           }
           else if (e.data["msg"] == "rel_table") {
               rel_table = e.data["value"];
           }
           else if (e.data["msg"] == "index") {
               index = e.data["value"];
           }
           else if (e.data["msg"] == "chunks") {
               chunks = e.data["value"];
           }
           else if (e.data["msg"] == "invoke") {
               var chunk_size = Math.ceil(articles.length / chunks);
        var start = (index * chunk_size) < articles.length ?
             (index * chunk_size) : articles.length;
               var end = ((index + 1) * chunk_size) < articles.length ?
             ((index + 1) * chunk_size) : articles.length;

               for (var i = start; i < end; i++) {
                    r[i] = new Array();
                    for (var j = i + 1; j < end; j++) {
                         r[i][j] = ((rl_value = similarity(articles[i], articles[j])) > 0) ? rl_value
: 0.01;
                    }

                    sum += r[i][j]; count++;
```

```
                  }

                  p_avg = sum / count;

                  self.postMessage(JSON.stringify({ "result" : r, "start" : start, "end" : end, "p_avg"
: p_avg }, null, 3));
              }
        }
   }

   function a2a_table()
   {
      var chunks = 10;
        if (articles_ents.length / chunks < 1)
            chunks = 1;

        for (var i = 0; i < articles_ents.length; i++) {
             rel_table[i] = new Array();
             for (var j = 0; j < articles_ents.length; j++) {
                  rel_table[i][j] = 0;
             }
        }

        var thw_count = 0; var workers = [];
        for (var i = 0; i < chunks; i++)
        {
             var code = a2a_worker.toString();
             code = code.substring(code.indexOf("{")+1, code.lastIndexOf("}"));
             var blob = new Blob([code], {type: "application/javascript"});

             workers.push(new Worker(URL.createObjectURL(blob)));
             workers[i].onmessage = function(e) {
                 var json_obj = JSON.parse(e.data);
   if (thw_count == chunks - 1) {
                    learn(rel_table); renderData(0, articles_ents);
                    for (var q = 0; q < workers.length; q++) {
                         workers[q].terminate();
                    }
                 }

                 //else {

                    for (var i1 = json_obj["start"]; i1 < json_obj["end"]; i1++) {
                         for (var i2 = i1 + 1; i2 < json_obj["end"]; i2++) {
                              rel_table[i1][i2] = json_obj["result"][i1][i2];
                         }
                    }

                    p_avg += json_obj["p_avg"] / chunks;

                    thw_count++;
                 //}
             }

             workers[i].postMessage({ "msg" : "articles", "value" : articles_ents });
             workers[i].postMessage({ "msg" : "rel_table", "value" : rel_table });
             workers[i].postMessage({ "msg" : "index", "value" : i });
             workers[i].postMessage({ "msg" : "chunks", "value" : chunks });

             workers[i].postMessage({ "msg" : "invoke", "value" : null });
        }
   }

   function vf_product(article_p1, article_p2)
   {
        var vf1 = article_p1["vf"];
        var vf2 = article_p2["vf"];

        var product = 0;
        for (var i = 0; i < vf1.length; i++) {
             product += vf1[i] * vf2[i];
```

```
     }

     return product;
}

function vf_sum(vf)
{
     var sum = 0;
     for (var i = 0; i < vf.length; i++) {
          sum += vf[i];
     }

     return sum;
}

function vf_average(vf)
{
     return vf_sum(vf) / vf.length;
}

function correlation(article_p1, article_p2)
{
     var vf1 = article_p1["vf"];
     var vf2 = article_p2["vf"];

     var vf_avg1 = vf_average(vf1);
     var vf_avg2 = vf_average(vf2);

     var vf_sum1 = 0, vf_sum2 = 0, vf_sum3 = 0;
     for (var i = 0; i < vf1.length; i++) {
         vf_sum1 += (vf1[i] - vf_avg1) * (vf2[i] - vf_avg2);
     }

     for (var i = 0; i < vf1.length; i++) {
         vf_sum2 += Math.pow(vf1[i] - vf_avg1, 2);
     }

     for (var i = 0; i < vf2.length; i++) {
         vf_sum3 += Math.pow(vf2[i] - vf_avg2, 2);
     }

     return vf_sum1 / (vf_sum2 * vf_sum3);
}

function compute_p(article_p1, article_p2)
{
     var article1 = article_p1["stats"];
     var article2 = article_p2["stats"];

     return p_avg + article1["bias"] + article2["bias"] +
  (vf_product(article1, article2) / correlation(article1, article2));
}

function learn(rl_table)
{
     var ts = 0.5;//0.025;       // The training speed
     var rc1 = 0.0005;      // Regularization coefficient lambda1
     var rc2 = 0.0025;      // Regularization coefficient lambda2
     var eps = 0.00001;    // Error precision accuracy coefficient
     var threshold = 0.01; // Threshold coefficient

     var is_done = 0;
     var RMSE = 0, RMSE_New = 0;

     do
     {
              RMSE = RMSE_New, RMSE_New = 1;
              for (var i = 0; i < rl_table.length; i++)
              {
                   for (var j = i + 1; j < rl_table[0].length; j++)
                   {
```

```
                            if (rl_table[i][j] > 0 && rl_table[i][j] != 0.01)
                            {
                                var error = rl_table[i][j] - compute_p(articles_ents[i],
articles_ents[j]);

                                RMSE_New = RMSE_New + Math.pow(error, 2);

                                p_avg += ts * (error - rc1 * p_avg);

                                articles_ents[i]["stats"]["bias"] += ts * (error - rc1 * articles_ents[i]
["stats"]["bias"]);
                                articles_ents[j]["stats"]["bias"] += ts * (error - rc1 * articles_ents[j]
["stats"]["bias"]);

                                for (var t = 0; t < articles_ents[i]["stats"]["vf"].length; t++) {
                                     articles_ents[i]["stats"]["vf"][t] += ts * (error * articles_ents[i]
["stats"]["vf"][t] + rc2 * articles_ents[j]["stats"]["vf"][t]);
                                     articles_ents[j]["stats"]["vf"][t] += ts * (error * articles_ents[j]
["stats"]["vf"][t] + rc2 * articles_ents[i]["stats"]["vf"][t]);
                                }
                            }
                      }
                  }

                  RMSE_New = Math.sqrt(RMSE_New / (rl_table.length * rl_table[0].length));

              if (RMSE_New > RMSE - threshold) {
                   ts *= 0.66; threshold *= 0.5;
              }

         } while (Math.abs(RMSE - RMSE_New) > eps);

         trained = 1;

         document.getElementById("status").innerHTML = "Completed...";
    }

    function predict()
    {
         if (trained == 0) {
             alert('Train the model first...');
             return;
         }

         var user_name = document.getElementById("user").value;
         var article_name = document.getElementById("article").value;

         var article_index = -1;
         for (var t = 0; t < articles_ents.length && article_index == -1; t++)
              article_index = (articles_ents[t]["name"] == article_name) ? t : -1;

         if (article_index != -1) {
             var found = 0;
             for (var i = 0; i < articles_ents.length && !found; i++)
             {
                  var users = articles_ents[i]["stats"]["users1"];
                  if (users.find(function (user) { return user == user_name; }) != undefined) {
                      found = 1;
                      var total = articles_ents[i]["stats"]["views"] + articles_ents[i]["stats"]
["upvoted"] +
   articles_ents[i]["stats"]["downvoted"] + articles_ents[i]["stats"]["downloads"];

                      var probability = compute_p(articles_ents[article_index], articles_ents[i]) * 100;
                      document.getElementById("rc_p").innerHTML = Math.round(probability).toString();
                      document.getElementById("view_p").innerHTML = Math.round(articles_ents[i]["stats"]
["views"] / total * 100);
                      document.getElementById("upvote_p").innerHTML = Math.round(articles_ents[i]
["stats"]["upvoted"] / total * 100);
                      document.getElementById("downvote_p").innerHTML = Math.round(articles_ents[i]
["stats"]["downvoted"] / total * 100);
                      document.getElementById("download_p").innerHTML = Math.round(articles_ents[i]
["stats"]["downloads"] / total * 100);
                }
            }
        }
  }
</script>
</html>
```

References

1. "C#.NET: Implementing SVD++ AI Data Mining Algorithm To Produce Recommendations Based On Ratings Prediction" - https://www.codeproject.com/Articles/1166739/Csharp-NET-Implementing-SVDplusplus-AI-Data-Mining

2. "C#.NET Implementation of K-Means Clustering Algorithm to Produce Recommendations" - https://www.codeproject.com/Articles/1123288/Csharp-NET-Implementation-of-K-Means-Clustering-AI

History

- **February 28, 2018** - The final revision of this article was published;

License

About the Author

Arthur V. Ratz

Software Developer (Senior) Engineer DPLKB

Ukraine

Arthur V. Ratz, 35 years old, C++ software developer, system analyst and network engineer graduated from L'viv State Polytechnical University (L'viv, Ukraine) and attained his Computer science and Information technology master's degree in January 2004. Since the middle of 2005, Arthur Ratz is a senior IT-professional. His professional career began as a financial and accounting software developer in DPLKB company's small local branch in L'viv, Ukraine. His professional interests include C/C++ programming, windows platform applications development using Win32API, parallel programming and multithreading, SQL relational database development, PHP and JavaScript web development, algorithms, system analysis, distributed information systems, computers networks design and analyzing, Windows Server and Linux administration, cloud computing, IoT, system security, technical writing and science publications etc. Arthur Ratz published his first article at CodeProject.com in June 2015.

Comments and Discussions

11 messages have been posted for this article Visit **https://www.codeproject.com/Articles/1230304/Birds-of-a-Feather-stick-together-To-Produce-Users** to post and view comments on this article, or click **here** to get a print view with messages.