**Text Summarizer Menggunakan Metode TextRank**

**Nur Cholis Majid 160535611843**

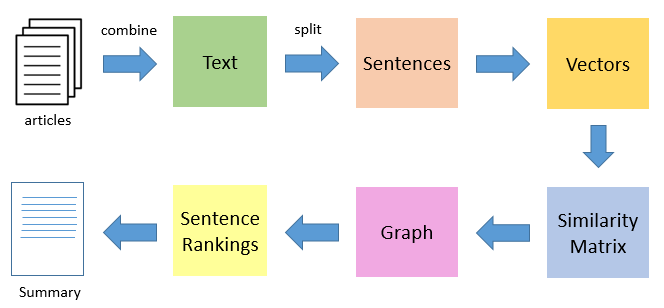
**Sabri Sangjaya 160535611819**

1. **Topik**

Meringkas dokumen secara manual oleh manusia membutuhkan banyak biaya dan waktu apabila dokumen tersebut banyak dan panjang sehingga diperlukan sistem peringkas otomatis (automatic summarization) untuk mengatasi banyaknya biaya dan waktu tersebut . Sistem peringkas otomatis yang dirancang harus efisien terhadap waktu dan efektif terhadap ketepatan penyajian informasi. Ada beberapa metoda dan algortima yang dapat menghasilkan sistem tersebut. Pada penelitian ini akan dibuat sistem peringkas dokumen otomatis yang menggunakan algoritma TextRank sebagai metodenya dan juga dibuat berbasis web menggunakan javascript sehingga dapat digunakan oleh siapa saja yang memerlukannya. TextRank merupakan graph-based ranking algorithm (graf dengan model pemeringkatan) untuk pemrosesan teks. TextRank yang digunakan pada sistem ini adalah metode yang menghasilkan ekstraksi berupa kalimat (TextRank for Sentence Extraction). TextRank sangatlah fleksibel karena dapat digunakan pada berbagai bahasa tanpa mengubah algoritmanya. Hal ini dikarenakan TextRank tidak memerlukan data training untuk proses pengelolahan dokumen

1. **Metode**

Flowchart:



Penjelasan:

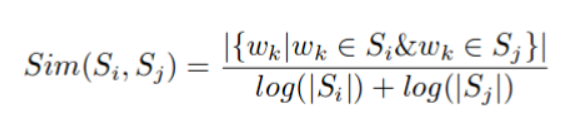
1. Preprocessing

terdapat dua tahap pada preprocessing yaitu segmentasi dan tokenisasi. Pada tahap segmentasi, kalimat kalimat dalam dokumen dibagi menjadi kalimat-kalimat tunggal. Pembagian dokumen ini dilakukan dengan menggunakan splitter, yaitu berupa tanda baca titik (“.”), tanda seru (“!”), tanda tanya (“?”), dan newline (“\”). Dalam proses segmentasi ini, gelar dan suatu singkatan harus dapat dikenali, misalnya gelar pada nama seorang mentri pendidikan di Indonesia “Nadiem Anwar Makarim, B.A., M.B.A” dan pada nama spesies bahasa latin “*E .coli*”. Kemudian pada tahap tokenisasi, tiap kalimat akan dibagi menjadi kata-kata/frasa yang berdiri sendiri dan terpisah oleh spasi.

1. TextRank dan Content Overlap Similiarity

Terdapat dua jenis pengelolahan bahasa dalam TextRank, yaitu TextRank for keyword extraction (ekstraksi kata kunci) dan TextRank for sentence extraction (ekstraksi kalimat). Pada TextRank for sentence extraction akan dibangun sebuah graf yang berisi hubungan antar kalimat dalam dokumen. Vertex di dalam graf ini direpresentasikan sebagai unit satuan yang akan diberikan peringkat. Vertex ini mempunyai similiarity yang dihubungkan oleh edges. Jenis similiarty yang digunakan adalah content overlap. Similiarity disini juga dapat ditentukan dengan menggunakan cosine similarity, tergantung dari kebutuhan sistem yang akan dibangun

Content overlap antara dua kalimat didefinisikan sebagai jumlah kata yang sama (word overlap) antara kedua kata dan dinormalisasi dengan membagi jumlah word overlap dengan panjang tiap kalimat. Pembobotan tidak dilakukan karena nilai kesamaan antar kalimat langsung dihitung berdasarkan banyaknya kata yang sama antar kalimat. Rumusnya adalah sebagai berikut:



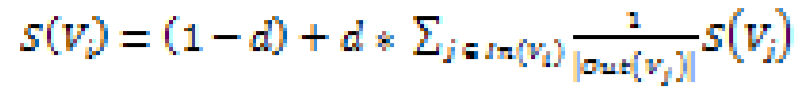
Keterangan:

Wk = Jumlah kata (term) yang sama antara kalimat Si dan Sj.

Si = Panjang kalimat Si.

Sj = Panjang kalimat Sj.

Rumus TextRank merupakan metode yang berasal dari PageRank. Rumus pada metode PageRank ini telah diubah/dimodifikasi untuk kebutuhan meringkas suatu dokumen. Rumus dari PageRank adalah sebagai berikut:



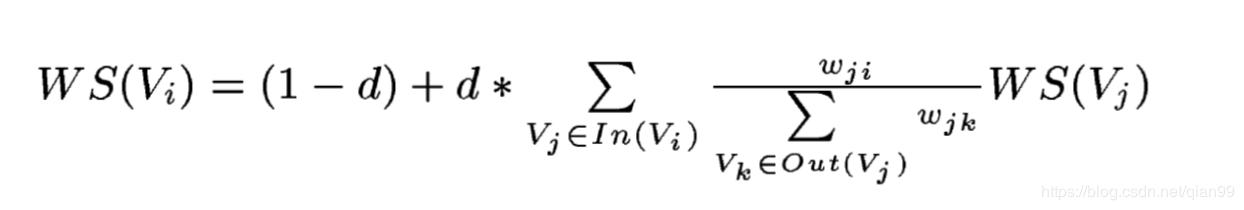
Keterangan:

Vi = vertex yang dihitung skor-nya.

Vj = vertex yang bertetanggaan dengan Vi.

d = damping factor yang nilainya antara 0 dan 1

TextRank akan menghasilkan graf yang tidak memiliki arah (undirected) dan berbobot (weighted) atau disebut juga undirected weighted graph. Rumus dari TextRank adalah sebagai berikut:



Keterangan:

WS = Weight Sentence (Bobot Kalimat).

w = nilai content overlap similarity.

Vi = vertex yang dihitung skor-nya.

Vj = vertex yang bertetanggaan dengan Vi.

Vk = vertex yang bertetanggaan dengan Vj.

d = damping factor yang nilainya antara 0 dan 1

Dari penjelasan diatas. Peringkasan dokumen menggunaka metode TextRank dapat dilakukan dengan langkah langkah berikut:

1. Lakukan ektraksi kalimat dengan menjadikan seluruh kalimat sebagai vertex dalam graf.
2. Identifikasi hubungan antarkalimat dengan membuat edges antara vertex-vertex. Dapat digunakan content overlap similarity untuk mengidentasikannya.
3. Beri skor awal vertex untuk menentukan iterasi.
4. Lakukan iterasi algoritma TextRank sampai error rate tiap vertex konvergen di bawah threshold. Error Rate disini adalah perbedaan antar dua skor vertex yang dihitung pada iterasi yang berurutan dengan rumus:

Sk+i(Vi) – Sk(Vi)

Keterangan :

Vi = Vertex yang dihitung skornya

k = pada iterasi ke-k

1. Setelah graf terbentuk, kemudia vertex diurutkan berdasarkan skor akhirnya. Lalu diambil top-rank sebagai hasil ekstraksi ringkasannya.
2. **Program**

**Source code Lampiran 1**

1. **Kelebihan dan Kekurangan**
2. Kelebihan
3. TextRank tidak memerlukan data training untuk proses pengelolaan dokumen
4. TextRank dapat digunakan dalam bahasa apapun karena TextRank tidak perlu memahami makna dari isi dokumen.
5. Kekurangan
6. TextRank tidak bisa memahami makna dari rangkuman atau dokumen yang akan diringkas
7. **Percobaan**

Paragraf

|  |
| --- |
| being a major tennis buff, I always try to keep myself updated with what’s happening in the sport by religiously going through as many online tennis updates as possible. However, this has proven to be a rather difficult job! There are way too many resources and time is a constraint.  Therefore, I decided to design a system that could prepare a bullet-point summary for me by scanning through multiple articles. How to go about doing this? That’s what I’ll show you in this tutorial. We will apply the TextRank algorithm on a dataset of scraped articles with the aim of creating a nice and concise summary. |

Dipecah menjadi kalimat

1. sentence: being a major tennis buff, I always try to keep myself updated with what’s happening in the sport by religiously going through as many online tennis updates as possible
2. sentence: However, this has proven to be a rather difficult job
3. sentence: There are way too many resources and time is a constraint
4. sentence: Therefore, I decided to design a system that could prepare a bullet-point summary for me by scanning through multiple articles
5. sentence: How to go about doing this
6. sentence: That’s what I’ll show you in this tutorial
7. sentence: We will apply the TextRank algorithm on a dataset of scraped articles with the aim of creating a nice and concise summary

Kemudian antar kalimat dihitung nilai kemiripannya. Setelah menghitung kemiripan maka dibuat graph antar kalimat. Kalimat 1 dan lainnya sebagai nodenya dan nilai kemiripan sebagai costnya/bobotnya.

Setelah itu dihitung bobot kalimat terhadap kalimat lainnya hasilnya perhitungannya adalah Kemiripan / Jumlah Seluruh kalimat

1. sentence: being a major tennis buff, I always try to keep myself updated with what’s happening in the sport by religiously going through as many online tennis updates as possible, PR: 0.20960571464830943
2. sentence: However, this has proven to be a rather difficult job, PR: 0.17435546771371316
3. sentence: There are way too many resources and time is a constraint, PR: 0.10585000755471552
4. sentence: Therefore, I decided to design a system that could prepare a bullet-point summary for me by scanning through multiple articles, PR: 0.18023815917512126
5. sentence: How to go about doing this, PR: 0.11632348558704961
6. sentence: That’s what I’ll show you in this tutorial, PR: 0.07559587086979169
7. sentence: We will apply the TextRank algorithm on a dataset of scraped articles with the aim of creating a nice and concise summary, PR: 0.13803129445129939

Setelah keluar bobot setiap kalimat maka bobot diurutkan mulai dari terbesar hingga terkecil. Kemudian ditentukan jumlah kalimat yang ingin ditampilkan misal separuh dari jumlah total kalimat maka contoh diatas akan menghasilkan 3 kalimat yaitu

|  |
| --- |
| being a major tennis buff, I always try to keep myself updated with what’s happening in the sport by religiously going through as many online tennis updates as possible. Therefore, I decided to design a system that could prepare a bullet-point summary for me by scanning through multiple articles. |

1. **Source Code Program**

|  |
| --- |
| summarizer.js |
| var Summarizer = {};  Summarizer.Utility = {};  // Get text from an HTML document.  Summarizer.Utility.getTextFromHtml = function (someHtmlDoc) {  var tmp = document.createElement("DIV");  tmp.innerHTML = someHtmlDoc;  return tmp.textContent || tmp.innerText;  }  // Get sentences from text.  Summarizer.Utility.getSentences = function (text) {  var sentences = text.split(/\. |\.|\?|!|\n/g);  $(sentences).each(function (idx) {  sentences[idx] = $.trim(sentences[idx]);  });  sentences = $(sentences).filter(function (idx) {  return sentences[idx].length > 0;  });  return sentences;  }  // Calculate similarity between 2 sentences.  Summarizer.Utility.calculateSimilarity = function (sentence1, sentence2) {  var words1 = sentence1.split(" ");  var words2 = sentence2.split(" ");  var intersection = \_.intersection(words1, words2);  var sumOfLengths = Math.log(words1.length) + Math.log(words2.length);  if (sumOfLengths == 0) {  return 0;  } else {  return intersection.length / sumOfLengths; // JS uses floating point arithmetic by default.  }  }  // Make directed graph.  Summarizer.Utility.makeGraph = function (sentences) {  var graph = {};  for (var idx1 = 0; idx1 < sentences.length; ++idx1) {  for (var idx2 = idx1 + 1; idx2 < sentences.length; ++idx2) {  if (graph[idx1] == undefined) {  graph[idx1] = [];  }  if (graph[idx2] == undefined) {  graph[idx2] = [];  }  var similarityScore = Summarizer.Utility.calculateSimilarity(  sentences[idx1], sentences[idx2]);  graph[idx1].push({  "node": idx2, "weight": similarityScore  });  graph[idx2].push({  "node": idx1, "weight": similarityScore  });  }  }  // Inculde a lookup from the sentenceId to the actual sentence.  graph.sentenceIdLookup = sentences;  return graph;  }  // Page Rank calculation driver.  Summarizer.Utility.calculatePageRank = function (graph, maxIterations,  dampingFactor, delta) {  var pageRankStruct = {};  var totalWeight = {};  var totalNumNodes = graph.sentenceIdLookup.length; // Number of nodes.  for (var idx = 0; idx < totalNumNodes; ++idx) {  pageRankStruct[idx] = {  "oldPR": 1.0, "newPR": 0.0  };  totalWeight[idx] = 0.0;  }  for (var idx = 0; idx < totalNumNodes; ++idx) {  var adjacencyList = graph[idx];  if (adjacencyList == undefined) {  continue;  }  // The adjacency list is an array containg objects that contain the neighbours' index as  // key and similarity score as the weight.  \_.each(adjacencyList, function (item) {  totalWeight[idx] += item["weight"];  });  }  var converged = false;  for (var iter = 0; iter < maxIterations; ++iter) {  maxPRChange = Summarizer.Utility.runPageRankOnce(graph, pageRankStruct,  totalWeight, totalNumNodes, dampingFactor);  if (maxPRChange <= (delta / totalNumNodes)) {  converged = true;  break;  }  }  var pageRankResults = {};  for (var idx = 0; idx < totalNumNodes; ++idx) {  pageRankResults[idx] = {  "PR": pageRankStruct[idx]["oldPR"] / totalNumNodes,  "sentence": graph.sentenceIdLookup[idx]  };  }  return pageRankResults;  }  // Single iteration of Page Rank.  Summarizer.Utility.runPageRankOnce = function (graph, pageRankStruct,  totalWeight, totalNumNodes, dampingFactor) {  var sinkContrib = 0.0;  for (var idx = 0; idx < totalNumNodes; ++idx) {  if (graph[idx] == undefined || graph[idx].length == 0) {  // Sink.  sinkContrib += pageRankStruct[idx]["oldPR"];  continue;  }  var wt = 0.0;  // Now iterate over all the nodes that are pointing to this node.  \_.each(graph[idx], function (adjNode) {  var node = adjNode["node"];  // Get the total weight shared by this adjacent node and its neighbours.  var sharedWt = totalWeight[node];  if (sharedWt != 0) { // To prevent NaN  wt += (adjNode["weight"] / sharedWt) \* pageRankStruct[node]["oldPR"];  }  });  wt \*= dampingFactor;  wt += (1 - dampingFactor);  // Update the structure w/ the new PR.  pageRankStruct[idx]["newPR"] = wt;  }  // Apply the sink contrib overall.  sinkContrib /= totalNumNodes;  var max\_pr\_change = 0.0;  for (var idx = 0; idx < totalNumNodes; ++idx) {  pageRankStruct[idx]["newPR"] += sinkContrib;  // Report back the max PR change.  var change = Math.abs(pageRankStruct[idx]["newPR"] - pageRankStruct[idx][  "oldPR"  ]);  if (change > max\_pr\_change) {  max\_pr\_change = change;  }  // Set old PR to new PR for next iteration.  pageRankStruct[idx]["oldPR"] = pageRankStruct[idx]["newPR"];  pageRankStruct[idx]["newPR"] = 0.0;  }  return max\_pr\_change;  } |

|  |
| --- |
| driver.js |
| $(document).ready(initJS);  function initJS() {  // Configure this object for tweaking summarization params.  var configObj = {  "maxIter": 100,  "dampingFactor": 0.85,  "delta": 0.5  };  var summarizeButton = $("#summarizeButton")[0];  var inputText = $("#input")[0];  var outputText = $("#output")[0];  // On focus the default input text should go away.  $(inputText).focus(function () {  console.log("Focus event!");  console.log($(inputText).val());  if ($(inputText).val() == "Insert the text to be summarized here...") {  $(inputText).val("");  }  });  // On blur the default input text should be injected.  $(inputText).blur(function () {  console.log("Blur event!");  console.log($(inputText).val());  if ($(inputText).val() == "") {  $(inputText).val("Insert the text to be summarized here...");  }  });  $(summarizeButton).on("click", function (event) {  var inputToSummarize = $.trim($(inputText).val());  if (inputToSummarize.length == 0) {  $(outputText).val("No text to be summarized...");  } else {  // Invoke the summarizer algo.  var sentences = Summarizer.Utility.getSentences(inputToSummarize);  var graph = Summarizer.Utility.makeGraph(sentences);  var result = Summarizer.Utility.calculatePageRank(graph, configObj.maxIter,  configObj.dampingFactor, configObj.delta);  var arr = [];  var idx = 0;  \_.each(result, function (v, k) {  arr.push({  "sentence": v.sentence,  "PR": v.PR,  "idx": idx++  });  console.log("sentence: " + v.sentence + ", PR: " + v.PR);  });  // Sort in descending order of PR.  arr = arr.sort(function (a, b) {  return b.PR - a.PR;  });  // Just returning half the original number of lines.  var halfNumLines = Math.floor(arr.length / 2);  if (halfNumLines == 0) {  halfNumLines = arr.length;  }  // Collect the half number of lines and sort them according to their occurence in the original text.  arr = arr.splice(0, halfNumLines);  arr = arr.sort(function (a, b) {  return a.idx - b.idx;  });  var finalResult = "";  for (var idx = 0; idx < halfNumLines; ++idx) {  finalResult += arr[idx].sentence + ". ";  }  $(outputText).val(finalResult);  }  });  } |