11-791/693 Design and Engineering of Intelligent Information System Fall 2013

Question Answering for Machine Reading Task (QA4MRE)

Important dates
Hand out: 28th November, 2013
Initial Writeup: 6th November, 2013
Milestone 1: 11-15 November, 2013
Milestone 2: 25-29 November, 2013
Milestone 3: 2-6 December, 2013

Project Progress Reporting Guidelines

Team 1 – 6 : Mondays [During above given weeks]
Team 7 – 12: Wednesdays
Team 13 – 15 : Mondays [In 12 noon class]

You should use the other day of week (Monday/Wednesday) for your team meetings.

Initial Writeup
- 1 page overall plan/design write-up. [*You don't have to make any presentation in class on this]

The write-up/presentation might include

1) initial pipeline/workflow design

2) initial type-system design

3) baseline methods to implement

4) initial division of work among team-members

Commit this report/ppts in your github repository. We will check and provide feedback on Github itself.

Please send us the URL of your project repository page (e.g., https://github.com/ID/hw5-teamXX). Here ID can be one of the team members ID whom team has identified as lead.

Milestone 1 (M1)

- You are required to submit all your Java source codes and UIMA descriptors for your components, and you DON'T need to submit any other documents. Your main XML descriptor should include a required descriptors for annotators.

- We will look into your Issues page and Wiki page, and expect you have created milestones and issues in the Issues system, and reported the results of your QA4MRE system evaluated, your team meeting minutes and probably your project goals or other any important things in your Wiki page.

- Class presentation on your 1st Baseline system [On your team's day] . The presentation include the baseline system's

1) pipeline/components structure
2) type-system
3) baseline methods
4) performance

Teams can present modification of their plan/design as well. Teams will receive feedback from TAs

Milestone 2 (M2)

– You are required to submit all your Java source codes and XML descriptors for your components, and you DON'T need to submit any other documents. You main XML should include a complete pipeline. You can pick the best component for each phase based on your own cross-option evaluation.

– We will again look into your Issues page and Wiki page, and expect you properly solved your previous created milestones and issues, and probably created new milestones and issues, and reported the evaluation results of your M2 on your own, your team meeting minutes and probably your revised project goals.

- Class presentation on error analysis [On team's day]. The presentation might include

1) error analysis on their 1st baseline system improvements made
2) proposed methods for next iteration
3) revision of their plan/design

Milestone 3 (M3)

– You are required to submit all your Java source codes and XML descriptors for your components. You main XML should include a complete pipeline. You can pick the best component for each phase based on your own cross-option evaluation with CSE.

– Name your presentation slides as hw5-teamXX.ppt and put it in the src/main/resources/docs

– We expect you properly solved all the issues and accomplish all milestones, and reported the evaluation results of your final system, your team meeting minutes and a final summary.

- Final Working notes due. Final presentation, final report and wiki.

1. Getting Started

Task 1.1 Team coding with GitHub

You will again use GitHub to host your project code. But different from previous homeworks, all the team members need to collaborate on the project, and commit the code changes to the same repository, which means it is good to learn how to more about git and GitHub in this subtask.

Creating a repository

1. The team leader needs to create a repository with his/her GitHub acccount, and name your project as hw5-teamXX, where XX is your team number, which is a two-digit number ranging from 01 to 15. If you are not sure how to create a GitHub repository, please refer to Homework 0.

At this point, all the team members are able to checkout the empty project and start working on their own. But this is NOT recommended! We recommend the team leader could checkout the project from GitHub repository, go through the entire Maven project building process (we will come to this in the next subtask) until you can ran a simple QA4MRE pipeline. Then, the team leader again, from his/her laptop, commits and pushes everything to the repository before the team members clone the project.

2. Now, everyone has the read permission to your project repository (since it is a public repository). To help your team members gain read/write permissions, you, the team leader, should add them as collaborators. You need to click the Admin tab on your project homepage, and click the Collaborators menu on the left. Put in your teammates' GitHub IDs one by one, and click Add button.

Creating milestones, issues, and Wiki pages

To create milestones for your development will help better organize your team members, know your collaborators' progress, and help users/customers know what they could expect from a future release. Issues can be detailed action items team members would like to contribute to each milestone. Action can be bug fix, feature enhancement, etc. If you are still unclear what a milestone is or what an issue is, or you want to understand how milestone and issue tracking system can help software development, you can search on Google and find out many interesting blogs. You are recommended to read about the features of GitHub issue tracking system at https://github.com/blog/831-issues-2-0-the-next-generation and https://github.com/features/projects/issues.

We recommend all the team members have a discussion on how to set the milestones, and what your first several initial issues will be (i.e., what they first couple of things you want to do at the beginning.) And the team leader does the following steps.

Remember that you are unlikely to submit all the issues to the tracking system at once, you may create new issues for bugs to fix or new features to implement, change the owner (assignee) of the issue, close implemented issues, or mark some as wontfix. All your team members are responsible to maintain your milestones and issues.

1. To create a milestone, you can navigate to the Issues tab of your project home page, then click the Milestones below the main menu bar. Now you are able to see the button Create a new milestone, click it.

2. Type your title for the milestone, e.g., “M1” for the first milestone (you can also make it more meaningful and specific). Write a few descriptions for this milestone, like the goal and the brief descriptions of features will be enhanced in this milestone. Finally, select a “Due Data” for the milestone. Click the “Create Milestone” to finalize creating your milestone.

3. Do the previous step once again until all your milestones are created.

4. Now, you need to submit your first several issues. Go back to the Issues tab on your project homepage, and click New Issue. Type a title, assign the task to a particular team member, link this issue to a previously created milestone.

Finally write down detailed comments to the issue.

You may find when you type “@” followed by your collaborator’s ID in the comment box, you are able to mention your collaborator as you mention your friend in a tweet on Twitter.

You can learn how to write in GitHub Flavored Markdown language, by clicking the link above the textbox.

You can also attach labels to each issue by selecting them from the right panel. As we mentioned earlier, you can change the milestone assignment, in particular, if you find you couldn’t finish it by M1, then you can change it to M2 later, or you can also relabel it as wontfix.

Now, you can create a Wiki page for your team meeting minutes and other important items.

1. Click the “Wiki” tab at the top of your project homepage, and click Edit Page to start editing it.
Once you reach this point, remember to send us an email to report the URL of your project repository page (e.g., https://github.com/ID/hw5-teamXX).

Team coding with git-branch, git-merge, git-rebase
Collaboration is important for this homework. All the team members start their individual development after the team leader gets the framework ready, and pushes all his/her local commits to the repository. Once a team member finalizes his/her development, he/she might take responsibility on another task, or

want to check the integrity or compatibility with features implemented by collaborators. After all the individual developments are done, team leader is responsible to merge all the newly developed components into the same codebase and test the integrity.

Git branching is a good tool to help the team manage parallel development and distributed codebase. In fact, the branching mechanism is widely adopted by not only git, but many other version control systems, e.g., SVN or Mercurial (Hg). But one of the most important reasons that people love git branching over SVN or Mercurial (Hg) is its light-weight nature, which allow switching between development branches within the same clone of a repository.

Normally, the team leader is in charge of the master branch (the one you probably used for homework 0 and 1), which usually corresponds to a codebase for the most recent stable release, while team members should create branches for each individual task assignment, e.g., bug fixes, feature enhancement with git branch command.

You can also do it within Eclipse by right-clicking the project name, and select Team→ Switch To → New Branch. . . .

To merge multiple development branches back to master branch (or in Eclipse Team → Switch To → master), you should switch back to master, and then git merge the branch you want to be merged (or Team → Merge. . .). Sometimes, you may also need git rebase when you later realize your development should depend on another feature that was also being developed by your collaborator, which hadn't been integrated in the master branch by the time you created your development branch.

To better understand git branching, you should read the "Git Branching" chapter of Pro Git Book (http://git-scm.com/book/en/Git-Branching). You can also find a more sophisticated branching model at http://nvie.com/posts/a-successful-git-branching-model/, which may be too complicated for this homework, but it can inspire how you want to manage your branches.

If you are looking for a Eclipse plug-in that can bring the GitHub issue tracking system to your workspace, e.g., create/close/comment issues, label them, assign to a person within eclipse, or automatically get notified by a message bubble if an issue is assigned to you, then you can try to investigate Mylyn plug-in. In the "Tips and Tricks using Eclipse with Github" post7 , you will find the basic idea how Mylyn GitHub connector works. By default, Eclipse Juno for Java developers comes with EGit, Mylyn, and Mylyn Github connector, and the Task View is on the top-right corner of the Java perspective by default.

Task 1.2 Getting QA4MRE codebase in your github repository

We have created baseline system for you to start with at https://github.com/oaqa/qa4mre . You will have to fork this in your repository created earlier. You may want to have look at the steps at https://help.github.com/articles/fork-a-repo for successfully forking the project.

qa4mre contains two modules: i) qa4mre-base and ii) qa4mre-alzheimer-task.

qa4mre-base is parent for qa4mre-alzheimer-task : this implies that you need to first compile and install qa4mre-base with

```
mvn clean install
```

maven goal.
Then, you can compile qa4mre-alzheimer-task.
The code should compile successfully without any errors.

Task 1.3 Running Document Annotators and Question Answering Pipeline

In qa4mre-alzheimer-task, there is a package "edu.cmu.lti.deiis.hw5.runner", which contains two entry classes:

i) SimpleRunCPE.java : Document Annotation pipeline. This should be run every time you change your document annotators. Generally you don't run this frequently.

ii) SimpleQuestionRunCPE.java : Question Answering pipeline. This should be to get the final answers. It will produce different results when you change your candidate sentence selection, answer ranking or answer selection algorithm. You will run it very frequently to see the effects of your algorithm change on performance of the system.

Collection Reader: It should read test document xml from "data/" folder
CAS Consumer: It should write output of document annotations to "XMIs" folder
It should write output of question answering to "results" folder

SolrIndexer: We have provided sample Java code for Solr indexing. This is optional. If you don't want to use it, just ignore. "solr" folder contains sample schema.xml and solrconfig.xml files in case someone wants to run local solr server on his laptop. Given schema.xml will help them as guideline. More information about setting up local Solr server can be found at http://lucene.apache.org/solr/4_4_0/tutorial.html

QuestionCandSentSimilarityMatcher : This baseline candidate sentence finder uses Solr for finding most relevant sentences with respect to question. If you are not indexing your sentences using Solr, you have to change this class and compute the cosine similarity between question and each sentence of the document. [You may want to refer your hw4 about vector space retrieval]

QA4MRE for Biomedical Texts about the Alzheimer Disease

The description of the task can be found at http://celct.fbk.eu/QA4MRE/index.php?page=Pages/biomedicalTask.html

Use of Background corpus (Optional): For your convenience, we have indexed the background corpus of 161K documents at http://peace.isri.cs.cmu.edu:9080/solr/genomics-simple/admin/ . The code for querying this corpus can be found at "AnswerChoiceCandAnsPMIScorer.java".

The PDF of QA4MRE guidelines, our working note paper and presentation is attached with this handout. Read the instructions carefully before you start.

Test document set is provided in "data/" folder.
Document annotations is created in "XMIs/" folder.
Final Result xmls are created in "results" folder.
We will use your **"results/TEAMID-result.xml"** file for grading purpose. Make sure you follow the naming conventions and folder hierachy.

Deliverables

1. Submission: The same way as you did for previous homeworks (set up GitHub repo, create Maven project, write your code, submit to Maven repo), except that the name has changed to hw5-teamXX (XX is a two-digit number from 01 to 15).

2. Your source files: They should be in appropriate packages - follow the baseline code given. You can add additional packages whenever needed.
Your descriptors: They should be in appropriate folders inside src/maion/resources/ - follow the baseline code given. You may add additional folder if required.

3. Comments, Javadocs, and documentations: They are still important if you want us and other users to better understand your code. (Remember: we will become your customer when we run the big experiment.)

4. Working Notes: We have attached our working note for QA4MRE 2013 Biomedical Alzheimer's task. This may provide some guidelines about the structure, we expect. You can see more such working notes at http://www.clef2013.org/index.php?page=Pages/proceedings.php

More Questions

We have tried to cover most of the concerns in this handout. However, you may discover some information not explicitly present in the handout. For any clarification, please use piazza so that entire class is benefited.

For any specific question regarding Homework 5, please send us mails to, Alkesh Patel (alkeshku@andrew.cmu.edu), Di Wang (diwang@cs.cmu.edu), or Zi Yang (ziy@cs.cmu.edu).