COMP90042 Project: Question Answering

Due date: 5pm, Thursday May 26

You'll be working in groups to build a Question Answering system. You will start by implementing a starter system according to our specifications, and then you will need to add two major enhancements, and write a report about the choices you made for your system, and justify your results.

We hope that you will enjoy the project. To make it more engaging we will run this task as a kaggle in-class competition. Kaggle is one of the most popular online platforms for running data science competitions. You will be competing with other teams in the class. The following sections give more details on data format, the use of kaggle, and marking scheme. Your assessment will be based on your team's performance in the competition, your code, and your report.

Submission materials: Python code (.py or .ipynb), project report (.pdf), individual contribution report (.pdf), compressed into a zip file named `team.zip`, where `team` is the name of your Kaggle In Class team.

Submission method: Each group should choose one member to submit the main project file via the LMS. If mutiple files with code are included, please make it clear which should be run. Do not submit any of the json datafiles, your code should assume the data is in the same directory. Each member of the group should also write a short (half page) summary detailing their contribution to the project (include your Kaggle user name and team name), and submit that individually on the LMS. You must also submit at least two entries (as a team) to the Kaggle In Class competition.

Late submissions: -10% per day

Marks: 30% of mark for class

Materials: See the main class LMS page for information on the basic setup required for this class, including an iPython notebook viewer and the python packages NLTK, Numpy, Scipy, Matplotlib, Scikit-Learn, and Gemsim. For this project, you will need to use the Stanford NER tagger via the NLTK interface (and you may use other tools such as the parser and POS tagger for your extensions), the main Stanford tools are installed on the lab computers, see here (https://github.com/nltk/nltk/wiki/Installing-Third-Party-Software) for information about how to set those up on your own system. You are being provided training, dev, and test sets, see the main instructions for information on their format and usage. Any other package, tool, or corpus is not allowed except by special permission of the instructors: if there's something else you want to use, e-mail us.

Evaluation: Your code should, when run, do all necessary setup for your QA system (including model training) and then give predictions for the test data. This entire process should not take more than 1 hour on a lab computer; systems that take longer that this will be marked down. You will be evaluated based on several critera: the correctness of your basic system, the originality and appropriateness of your enhancements, the performance of your final system, and the clarity and comprehensiveness of your final report. In most cases, everyone in the same group will receive the same mark, however, we may make exceptions based on the individual contribution reports.

Updates: Any major changes to the project will be announced via LMS. Minor changes and clarifications will be announced in the forum on LMS, we recommend you check the forum regularly.

Academic Misconduct: Though this is a group project and therefore exchange of code within groups is allowed, reuse of code across gropus or other instances of clear influence will be considered cheating. We will be checking submissions for originality and will invoke the University's Academic Misconduct policy (http://academichonesty.unimelb.edu.au/policy.html) where inappropriate levels of collusion or plagiarism are deemed to have taken place.

Use of Kaggle in Class:

Please do the following within the first week after receiving this assignment:

1. Setup an account on Kaggle with username and email both being your unimelb student email, using the link https://inclass.kaggle.com/c/comp90042-wsta (https://inclass.kaggle.com/c/comp90042-wsta). Note that only unimelb emails can access to the competition;
2. Form your team of student peers and e-mail your the unimelb ids of the team members to Julian; if you do not form one by May 3rd, you will be assigned to one randomly.
3. Connect with your team mates on Kaggle and form a Kaggle team.

You should only make submissions using the team name, individual submissions are not allowed and may attract penalties. Note that teams will be limited to 5 submissions per day.

You should submit your kaggle results as a .csv file. Your outputs should be output, one entry per line, in a CSV file using the following format:

id,answer
1,answer1
2,answer2
3,answer3
...

where the answer strings can be words or sequences of words, and the ids are increasing integers and denote the question in the test dataset. See here (https://trevorcohn.github.io/comp90042/project/sample.csv) for a full sample (with garbage answers). Note that any double quotes (") must be removed and commas (,) replaced with -*COMMA*- as these are both special characters in CSV format. Be sure to include the header line as given above.

The real answers for the test data are hidden from you, but were made available to kaggle. Each time a submission is made, half of the predictions will be used to compute your public accuracy score and determine your rank in public leaderboard. This information will become available from the competition page almost immediately. At the same time, the other half of predictions is used to compute a private accuracy and rank in private leaderboard, and this information will be hidden from you. At the end of the competition, only private scores and private ranks will be used for assessment. This type of scoring is a common practice and was introduced to discourage overfitting to public leaderboard. A good model should generalize and work well on new data, which in this case is represented by the portion of data with the hidden accuracy.

The evaluation score used in this competition is the accuracy over all classes, defined as the number of instances that are predicted correctly as a fraction of the total number of instances. For a prediction to be judged correct, it must be string identical to the gold answer. Before the end of the competition each team will need to choose their best submissions for scoring (this could be your basic QA system or, more likely, one of your extensions.) These do not have to be the latest submission. Kaggle will compute a private accuracy for the chosen submissions only, and the best of these used to compute the private leaderboard ranking, which will be used for marking.

Datasets:

You are being provide with 3 datasets for use on this project.

- training data (https://trevorcohn.github.io/comp90042/project/QA_train.json)
- dev data (https://trevorcohn.github.io/comp90042/project/QA_dev.json)
- test data (https://trevorcohn.github.io/comp90042/project/QA_test.json)

Each of datafiles is a json list, with each element of the list corresponding to a text whose "sentence" key provides an ordered list of the sentences of a Wikipedia article, and whose "qa" key is a list of questions whose answers can be found in the article. For the training and dev data, each "qa" dictionary consists of the "question" (a string) as well as the "answer" (a string) and "answer_sentence" (an integer), which is the index of the sentence where the answer can be found: you can assume there is an exact string match of the answer in the answer sentence. For the test data, the answer information is missing, but an id number is included which should be used when creating the Kaggle submission. You should use the Python json library to load these files.

Each of this datasets has a different purpose. The training data should be used for building the supervised model you implement in the enhancements, and it is also the only set which you are allowed to manually inspect. The training set is large, and you should not feel you do not have to use all of the data if it is not feasible to do so. You will use the test set, as discussed above, to participate in the Kaggle competition. The dev set (which is identical to the training set data in format) is included to help you make major implementation decisions, and should also be used for detailed analysis of your system in the report. To avoid overfitting, you may want to separate a portion of the training set as an additional dev set for low-level tunning and/or error analysis (you can also use crossvalidation). You should not *at any time* manually inspect any of part of either the dev or the test dataset; any sign that you have done so will result in loss of marks.

Basic QA System (10 marks)

Sentence Retreival

The first part of your basic QA system will use a bag-of-words (BOW) vector space model to identify the sentence in the Wikipedia article which is most likely to contain the answer to a question, using standard information retrieval techniques. Here the "query" is the question, the "documents" are actually sentences, and each Wikipedia article should be viewed as separate "document collection". You should apply various preprocessing steps appropriate to this situation, including term weighting; if you are at all uncertain about what choices to make, you should evaluate them using the dev data, and use the results to justify your choice in your report.

Entity Extraction

The second main part of your basic QA system is an NER system. In this initial system you should have at least four answer types: PERSON, LOCATION, NUMBER, and OTHER. You should run the Stanford NER system over your sentences to extract people and location entities (Hint: make use of the "tag_sents" method in the NLTK interface to do this efficiently for multiple sentences in a single call, otherwise this will be very slow; you may also want to cache the entity information during development of your system, rather than calling Stanford NER for each run). Note that contiguous words tagged as the same type should be considered part of the same entity. ORGANIZATION entities extracted by the NER system should be considered OTHER. You should also extract and treat as OTHER any other non-sentence initial sequence of capitalized words not tagged by Stanford NER. Finally, you should label all numbers as NUMBER. In this process, you might notice errors related to your preprocessing (e.g. tokenization), errors which can be easily corrected should be addressed at this stage.

Answer Ranking

After you have extracted a set of entities from your sentence, you will rank them to choose the best answer. The ranking should be based on three factors. First, answers whose content words all appear in the question should be ranked lowest. Second, answers which match the question type should be ranked higher than those that don't; for this, you should build a simple rule-based question type classifier based on key words (e.g. questions which contain "who" are people). Third, among entities of the same type, the prefered entity should be the one which is closer in the sentence to a closed-class word from the question.

Error Analysis (3 marks)

Run your system over the test data and upload the results to Kaggle. If you have a reasonably good basic system, your results should be comparable to the baseline provided on Kaggle (which is a system build we have built following the instructions). If your results are significantly worse, you should look for bugs in your code.

One you have are satisfied that your basic system is working as intended, you should use the training data to do a thorough error analysis, looking for patterns in the kinds of errors your basic system is making. You should consider all three of the steps above, and identify where the most serious problems are occurring. If there are any relatively simple fixes that might have a sizeable impact on performance, you should feel free to note and apply them, but your main goal is to identify opportunities for major enhancements. You should include a summary of this error analysis in your report.

Enhancements (8 marks)

You should implement at least two major enhancements to your QA system. The choice of what you will do is ultimately yours, however we require that:

- They should be potential solutions to problems noted in your error analysis
- At least one of the enhancements should involve creating a supervised model using the provided training data
- At least one of the enhancements should make use of syntactic or semantic information that we have introduced in this class. Examples of this include POS tagging, syntactic parsing, WordNet, or the word2vec vectors included in NLTK (we don't recommend you try to build your own distributional vector representations)

In addition to the error analysis, we recommend you review your QA reading and relevant papers mentioned in it for ideas. Though you can do whatever you like to improve performance of your model, enhancements which rely entirely on small sets of rules or hand-built lexicons would not generally be considered major enhancements. The amount of effort required for a major enhancement should be at least comparable to the effort required to create the basic QA system. Though we would expect most students to use the basic QA system as a starting point for their enhancement, you may choose to make major changes or even build an entirely new QA system from scratch. If you are using the base system as a starting point, your enhancements should ideally be implemented in a way such that they can be turned on and off as an option, but if you are modifying the basic QA system in a such way that this isn't possible (or creating an entirely new one), you should keep a separate copy of the basic system and submit it as a python script (.py) or ipynb file (e.g. basic.py or basic.ipynb)

Expect to spend much of your time and effort on developing a good working system that can outperform your baseline method above. Marks will be awarded for both ambition and the performance of your approach (see below).

Report (5 marks)

Each team will submit a report with the description, analysis, and comparative assessment (where applicable) of methods used. There is no fixed template for the report, but it should start with a very brief introduction of the problem. You should mention any choices you made in implementing your basic QA system along with empirical justification for those choices. Use your error analysis of the basic system to motivate your enhancements, and describe them in enough detail that we could replicate them without looking at your code. Using the dev dataset, you should evaluate whether your enhancements increased performance as compared to the basic system, and also report your relative performance on the Kaggle leaderboard. Finally, discuss what steps you might take next if you were to continue development of your system (since you don't actually have to do it, feel free to be ambitious!).

For the evaluation, you should generally avoid reporting numbers in the text: include at least one table, and at least one chart. Using the dev set, you should report a breakdown of performance for the sentence retrevial and answer extraction/ranking portions of the task (for the latter, assume you have already identified the correct sentences). In addition to exact match accuarcy (which is used on Kaggle), for full credit you should propose an appropriate, more forgiving evaluation metric for the end-to-end task which gives partial credit for partial matches or for answers which are ranked lower than first; evaluate using it as well.

Your description of your enhancements should be clear and concise. You should write it at a level that a postgraduate student can read and understand without difficulty. If you use any existing algorithms, you do not have to rewrite the complete description, but must provide a summary that shows your understanding and references to the relevant literature. In the report, we will be very interested in seeing evidence of your thought processes and reasoning for choosing one approach over another.

The report should be submitted as a PDF, and be no more than four A4 pages of content, including all plots, tables and references. Please use a single column, a font size of 11 or more and margins at least 1 cm. You do not need to include a cover page, but please ensure that all usernames are clearly stated on the first page as is the Kaggle team name. If a report is longer than four pages in length, we will only read and assess the report up to page four and ignore further pages.

Reports will be assessed for critical analysis, clarity and structure. Note that although only 5 marks are assigned to the report, your presentation here will be used as the primary means of assessing the earlier three parts: the QA system, error analysis and enhancements. The code is only secondary, so please ensure that you report the your basic and enhanced approaches used, error analysis and results clearly in your report.

Evaluation (4 marks)

This component of your assessment is based on the accuracy results achieved by your method, both for the baseline system and for your final submission to the Kaggle competition. We will consider both your ranking against the rest of the class on the private leaderboard, as well as your performance relative to the baseline system.