# Natural Language Processing [CS4120]
# Assignment 3

Instructor: Professor Lu Wang

## Deadline: April 1st, 2020 at 11:59pm on Blackboard

For the programming questions, you can use Python (preferred), Java, or C/C++. Please include a README file with detailed instructions on how to run your code. Failure to provide a README file will result in **deduction of points** (5 to 10 points per problem). Your final deliverable for this homework should consist of one zipped folder which contains three folders inside (one folder per problem), and each folder must contain the following things:
i) Text files with your answers for questions requiring textual answers (name each file as instructed for each question)
ii) README describing how to run your code for each programming problem.
iii) Code scripts for programming questions. It is recommended to comment your code at important steps to avoid any ambiguity while grading.

**This assignment has to be done individually**. If you discuss the solution with others, you should indicate their names in your submission. If you use ideas/code from any online forums, you should cite them in your solutions. **Violation of the academic integrity policy is strictly prohibited, and any plausible case will be reported once found. No exceptions will be made.**

# 1 Summarization [50 points]

A summary is usually a brief overview of a longer document. A good summary is supposed to be grammatically correct, non-redundant, and coherent. We define these three properties below.

**Grammaticality.** A grammatically correct summary should have no datelines, system-internal formatting, capitalization errors or obviously ungrammatical sentences (e.g., fragments, missing components) that make the text difficult to read. For this problem, grammaticality score can range from -1 [grammatically poor] to 1 [grammatically correct].

**Non-redundancy.** A non-redundant summary should have no unnecessary repetition in the summary, which might take the form of whole sentences that are repeated, or repeated facts, or the repeated use of a noun or noun phrase (e.g., "Bill Clinton") when a pronoun ("he") would suffice. For this problem, non-redundancy scores can range from -1 [highly redundant] and 1 [no redun-

dancy].

**Coherence.** A coherent summary should be well-structured and well-organized. The summary should not just be a heap of related information, but should build from sentence to sentence to a coherent body of information about a topic or entity. For this problem, coherence score can range from -1 [not coherent] to 1 [highly coherent].

**Dataset:** In this question, you will design classifiers to evaluate the summary quality based on aforementioned qualities. You will be given a training set and a test set, both in csv files (you will load them as dataframes). Here's the link to the data: http://bit.ly/2uQS1P3.
The link contains two csv files that are explained below:

- *training_data.csv*: This file contains the entire training data (1737 entries) and each entry includes the summary, the grammaticality score, the non-redundancy score, and the coherence score of the summary. You will load this data as Dataframes (use Python pandas) and extract relevant columns from the dataframe. [You may want to further divide the training set into training and validation sets for training classifiers to avoid overfitting on the test set].

- *testing_data.csv*: This file contains the entire testing data (193 entries) and each entry includes the summary, the grammaticality score, the non-redundancy score and the coherence score of the summary. Again, you will load this data as Dataframes (use pandas) and extract relevant columns from the Dataframe.

Your task is to build logistic regression classifiers on the training dataset to predict the grammaticality, non-redundancy, and coherence of the summaries in the test dataset. You can use any off-the-shelf tools without implementing the logistic regression model (e.g. scikit-learn provides a nice interface).

In addition to a file with the output and results for each question (e.g. *Q1.1.txt*, *Q1.2.txt*, *Q1.3.txt*), please submit your code along with a README file explaining how to run your code.

## 1.1 Building Grammaticality Scorer

### 1.1.1 [12 points]

Implement the following three features and train your classifier with all of the three features on the training data, with summary as input and "grammaticality score" as the gold-standard label, and report the performance of your classifier on the test data.

For evaluation, please report Mean Squared Error (MSE) and Pearson correlation, both calculated between your predicted labels and gold-standard labels in the test data. Before implementation, please read the **Hints** at the end of problem 1.

Here are the three features:

1. *Total number of repetitive unigrams*: count how many unigrams were repeated consecutively in a given summary. For instance, for a summary "**the the** article **talks talks** about language understanding", the feature value should be 2 (because "the" and "talks" are both repeated once, so 1+1=2).

2. *Total number of repetitive bigrams*: count how many bigrams were repeated consecutively in a given summary. For instance, for a summary "**the article the article** talks about language understanding", the feature value should be 1. If the summary is like "**the article the article the** talks about language understanding", the feature value should be 2 (because "the article" and "article the" are both repeated once).

3. *Minimum Flesch reading-ease score*: use tool from https://pypi.org/project/readability/ to get readability score for each sentence, and use the minimum value as the feature.

#### 1.1.2 [8 points]

Design **two** new features for this task, and explain your intuition behind choosing these features in the text file (just a couple of sentences for each feature as to why you think this feature might be important).

Add each feature to the classifier built in 1.1.1, and report MSE and Pearson correlation. At least one of your proposed features should get better MSE or Pearson. Take a look at the training samples and explain why your features can improve the classifier's performance.

### 1.2 Building Non-Redundancy Scorer

#### 1.2.1 [12 points]

Implement the following three features and train your classifier with all of the three features on the training data, with summary as input and "non-redundancy score" as the gold-standard label, and report the performance of your classifier on the test data. For evaluation, please report Mean Squared Error (MSE) and Pearson correlation.

Here are the three features:

1. *Maximum frequency of unigrams*: calculate the frequencies of all the unigrams (after removing stop words), and use the maximum value as the feature value.

2. *Maximum frequency of bigrams*: calculate the frequencies of all the bigrams (do not remove stop words), and use the maximum value as the feature value.

3. *Maximum sentence similarity*: represent each sentence as average of its word embeddings, then compute cosine similarity between pairwise sentences, use the maximum similarity as the feature value. Use word embeddings GoogleNews-vectors-negative300.bin.gz from Word2Vec : https://code.google.com/archive/p/word2vec/ as input for each word. Words in a summary that are not covered by Word2Vec should be discarded.

#### 1.2.2 [8 points]

Again, design **two** new features for this task and explain your thought process behind choosing these features in the text file.

Add each feature to the classifier built in 1.2.1, and report MSE and Pearson correlation. At least one of your proposed features should get better MSE or Pearson. Take a look at the training samples and explain why your features can improve the classifier's performance. Please **do not** repeat the features from 1.1.2.

## 1.3 Building Coherence Scorer

### 1.3.1 [10 points]

Implement the following two features and train your classifier with all of the two features on the training data, with summary as input and "coherence score" as the gold-standard label, and report the performance of your classifier on the test data. For evaluation, please report Mean Squared Error (MSE) and Pearson correlation.

Here are the two features:

1. *Total number of repetitive nouns*: count how many **nouns** are repeated in different sentences in a given summary. For example, a summary "**Bill** will come. **Bill** will bring the **key**. **Bill** will use the **key** to open the box." The feature value will be 2+1=3 ("Bill" is repeated in the second and third sentences, and "key" is repeated once.) You can use spacy to extract the POS tags.

2. *Total number of coreferred entities*: count how many entities or textual units are coreferred in a given summary.
   Coreference occurs when two or more expressions in a text refer to the same person or thing; they have the same referent, e.g. "Bill said he would come"; the proper noun "Bill" and the pronoun "he" refer to the same person, namely to "Bill". When two expressions are coreferential, one is usually a full form (the antecedent) and the other is an abbreviated form (a proform or anaphor). Coreference resolution is the task of correctly matching the antecedent with its referent. For sentence, "My sister has a dog. She loves him.", here is the result of spacy's coreference resolution: https://bit.ly/2nm8RBG [Try it here https://bit.ly/2meo8UV if the link doesn't work.] You can use NeuralCoref (https://github.com/huggingface/neuralcoref) to do coreference resolution and get the total number of coreferred entities for constructing your feature.

**Hints for Question 1**

1. Tokenize using spacy https://spacy.io/usage/linguistic-features.

2. Lowercase words (Exception: For Coherence scorer since you require to run a parser on your summaries, you will need to use non-lowercased version of your summaries.)

3. Remove extra spaces

4. If you want to use a stop word list, spacy has one.

5. Mean squared error is the error of the results obtained from your classifier compared to the true values of the labels.
   The output ranges from 0 to 1 where

   (a) 0 indicates that your predicted values match the gold-standards perfectly.

   (b) 1 indicates that your prediction are not very accurate.

   You can use scikit-learn package to compute the mean squared error: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

6. Pearson Correlation Coefficient calculates the correlation between the predicted values and the gold-standards.
   The result ranges from -1 to 1 where

   (a) 1 indicates that your predicted values positively correlate with the gold-standards perfectly.

   (b) 0 indicates that there is no correlation between predictions and the gold-standards.

   (c) -1 indicates that your predicted values negatively correlate with the gold-standards perfectly.

   You can use scikit-learn package to compute the pearson coefficient: https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.pearsonr.html

# 2 Question Answering [40 points]

**Background**: Question Answering involves information retrieval and natural language processing, which focuses on building systems that automatically answer questions posed by human in natural language.
**Dataset**: For this problem, you will work with the popular SQuAD corpus. The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of factoid questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text from the corresponding reading passage.
We have processed the dataset for you, and made it available at: http://bit.ly/3bAeSPE

The link contains two files for training and test separately. We have organized them in a simplified structure for your convenience. The .json format is explained below:

- *title*: The title of the paragraph from which questions will be asked.
- *paragraph*: The paragraph which is a list of sentences.
- *question*: The question which is being asked based on this paragraph.
- *answer*: The correct answer to the question.

- *correct answer index*: The index of the sentence from which the answer is taken.

In this question, you will study the SQuAD dataset by exploring the distributions of the types of questions asked, the types of answers, and then finally build your own answer detection system.

## 2.1 Analyze Question Types [5 points]

In this question, you will explore the different types of questions that exist in the SQuAD training set (squad_v1.1_dataset_training.json file). Your task is to count the number of times a particular type of question appears in the dataset, and then create a frequency distribution of these question types.

Concretely, you can identify the questions into different types by looking at the starting word of each question. The different starting words you will use to classify questions are as follows:

- What
- When
- Where
- Which
- Why
- How
- In
- On
- all other words (i.e. any word other than the ones mentioned above)

Once you have the frequency for each type of question, you need to visualize this data using an appropriate Visualization tool. You can use any library of your choice for visualizing the data such as Matplotlib, Seaborn, Plotly, etc. Label and scale each axis properly. Embed your final picture in the zip file you are submitting for this assignment.

## 2.2 Analyze Answer Types [5 points]

Next, you will explore the different types of answers to the questions in the SQuAD training data. Your task is to count the number of times a particular type of answer appears in the dataset, and then create a frequency distribution of the occurrence of these answer types. You can identify the answers into different types by looking at their NER tags.

For example, if the answer to the question "Who won the NFL this year?" is Denver Blancos, then the NER tag of the answer Denver Blancos will be ORG which suggests that it is an organization (team). Similarly, group all the answers according to their NER tags. You will use

spacy https://spacy.io/usage/linguistic-features to get the NER tags for all answers.

Once you have the counts for the number of times each type of answer appears, again you need to visualize this data. You can use any library of your choice for visualizing the data. Label and scale each axis properly to be able to make inference from the data. Embed your final picture in the zip file you are submitting for this assignment.

## 2.3 Building an Answer Detection System [30 points]

In this question, we will build an Answer Detection system to predict which sentence in the given paragraph contains the correct answer. For this case, the candidate sentences are the list of sentences in the paragraph from which the question is asked. Overall, you need to compare the question with each sentence in the paragraph and pick the sentence that is most likely to contain the answer. We will first explore two approaches and then you will build your own classification model for this problem.

You can use spacy or any tool for tokenization.

### 2.3.1 Word Overlap Similarity [5 points]

First, you will compute the similarity between the question and each sentence in the paragraph by counting the number of words (after removing stop words) that are common between the question and the sentence. The sentence with the most overlapping words with the question is returned as the prediction. Then you can compare this sentence's index with the "correct_answer_index" value for each question provided to you in the original dataset.

Do this for all questions and report accuracy on training set and test set separately.

### 2.3.2 Embedding Similarity [5 points]

Next, you will collect embeddings for all words in a sentence and take the average of the embeddings to represent that sentence. Using the same process to create a representation for the question as well. Now find the cosine similarity between the question and sentence vectors, and output the sentence that has the highest similarity with the question as the prediction.

Do this for all questions and report accuracy on training set and test set separately.

Note: Here you can use the pre-trained word embeddings in the "GoogleNews-vectors-negative300.bin.gz" file from Word2vec library in Python. You can download the pre-trained word vectors from: https://github.com/mmihaltz/word2vec-GoogleNews-vectors.

### 2.3.3 Your Answer Detection Classifier [20 points]

In this step, you will train a machine learning classifier of your choice (logistic regression, feedforward neural models, support vector machines, etc) from the training data (squad_v1.1_dataset_training.json) and report accuracy on the test data (squad_v1.1_dataset_testing.json).

Concretely, you will train the classifier with five features, including the two features in 2.3.1 and 2.3.2, and **three new** features designed by you, on the training set. Then run the trained classifier on the test set.

In a text file, explain your intuition behind the new feature design in a text file (just a few sentences for each feature as to why you think this feature might be important) and report accuracy on the test set. You may want to divide the training dataset into training and help-out set for any parameter tuning to avoid overfitting on the test set.

# 3 Chatbots [10 points]

**Background:** Chatbots often act as an online representative that can answer your questions. The best success stories about Chatbots in present times are their usage by financial organizations to address specific client requests without having to call or visit a branch, as well as personal assistants such as Apple's Siri, Microsoft's Cortana etc.

Recently, Microsoft unveiled Tay — a Twitter bot that the company described as an experiment in "conversational understanding." The more you chat with Tay, said Microsoft, the smarter it gets, learning to engage people through "casual and playful conversation." Unfortunately, the conversations didn't stay playful for long. Pretty soon after Tay launched, people started tweeting the bot with all sorts of misogynistic, or racist remarks. And Tay — being essentially a robot parrot with an internet connection — started repeating these sentiments back to users: flaming garbage pile in, flaming garbage pile out.

**Your Task:** You need to play around with any one of your favourite chatbot/personal assistant (For example, Apple's Siri, Google's Assistant, Amazon's Alexa, Mircosoft's Cortana etc. Any is fine). Keep on conversing for a prolonged period of time and try to find any scenarios where the assistant says something morally, ethically, or culturally wrong; or find any offensive types of responses that you feel inappropriate.

If you do not have access to a personal assistant on your phone or for some other reason, you can try out the following free online Chatbot Mitsuku https://www.pandorabots.com/mitsuku/.

Note: The conversation should not be an obvious one or a forced one where you deliberately ask something obvious to the assistant and the assistant immediately replies in an offensive manner. We want your conversations to be well thought of and natural, so as to observe some important patterns in the Chatbot. If you have any doubt regarding how to go about this question, please reach out to the staffs.

**Deliverables:** Submit a screenshot of your conversation (web screenshot or phone screenshot). And then briefly explain your experiment: (1) what you have observed, (2) what was surprising or interesting about this observation, (3) how your observation might be caused by the system design, and (4) how we can avoid this kind of improper chatbot responses, in the a separate text file named "chatbotExperiment.txt".