CS839 Project Stage 1: Information Extraction
Date: 3/3/2018 Team name: Big HIT
Team members: Hoai Nguyen, Isaac Sung, Trang Vu

Summary: This report describes how we created an information extractor to extract person names from various news articles for project stage 1. Table 1 below summarizes the minimal information required for the project report. We provide a longer description of the project in sections following this table.

Table 1: Project summary

Entity type	**Person name**					
Examples of person name mentioned in text documents	Ex1: A White House official on Thursday anonymously attacked Sen. Lindsey Graham (R-S.C.) for his role in the immigration debate in Congress.		Ex2: Andrew Gillis and Bhreagh MacNeil play Blaise and Vanessa, two addicts on a methadone program.		Ex3: Given Portman's active involvement in the Time's Up movement, her signing of the petition came from a misplaced sense of empathy.	
Data statistics	**# of total docs**	**# of total mentions**	**# of docs in set I**	**# of mentions in set I**	**# of docs in set J**	**# of mentions in set J**
	319	5242	210	3490	109	1752
Classifier Statistics	**Precision**		**Recall**		**F1**	
M - After cross validation: Random Forest	0.69		0.51		0.59	
X - Before post processing: Random Forest	**0.92**		**0.68**		**0.79**	
Y - After post	**0.92**		**0.68**		**0.79**	

processing: Random Forest			
Y - Applied to test set J	**0.86**	**0.63**	**0.73**
Examples of post processing rules	Keep only supersets of positive labels: e.g. if from the same sentence, “Will Smith is an actor.” both “Will” and “Will Smith” are marked as positive, we only keep “Will Smith” and mark “Will” as negative		
Reasons for not achieving required P and R	Achieved required P and R for set I, not for set J. The classifier was probably over-fitted for set I		
Things can we do to improve P and R	Go back and define more precise rules for markups and re-markup each document; extend pruning rules to handle special cases; generate more useful features using the surrounding context, perhaps using more part-of-speech tagging		

1. Entity Type and Text Documents

We selected person names as our entity to be extracted from news articles simply because we can easily find many text documents with person names. We scraped articles from various news outlets such as The Hill, The Atlantic, Vulture, The Washington Post, The New York Times, The Guardian, etc. for a number of different topics like politics, movies, entertainments, etc. We downloaded 319 documents and for each document, we marked up person names using a start and end tags followed this format <pname>person name</pname>. A few examples of person name are given in Table 1 above. For cases in which a person name is used as an adjective like “Trump administration”, or “Trump era” we decided not to mark “Trump” as person name because we think it does not referred to the person named Trump but rather an administration or an era. Using this definition, we got 5242 mentions of person names from all the 319 documents. We split the 319 documents into 210 documents for the development set I and 109 documents for the test set J.

2. Example and Feature Generation

For each document, we first split the text by space. We further split each string by punctuations except for a few special cases such as the initials “R.” as in “George R. R. Martin’ or the title ‘Mr.’ as in ‘Mr. Smith’. These cases will not be split in to “R”, and “.” or “Mr” and “.” Respectively (see code “feature_data_generator.py” under folder “Code for this Project” for more detail.) The reasons we split strings by punctuations because

splitting string by space would not be enough to produce single words. For example, “Dominika (Jennifer Lawrence)” will first be split to “Dominika”, “(Jennifer”, and ”Lawrence). “(Jennifer” would not be considered a person name by our definition so we need to separate “Jennifer” from the left parenthesis.
Once the text is split, we created strings of 1, 2, 3, and 4 words (in this situation, a word is not necessarily made up of all alphabetic characters, i.e. it could be just a number, or a parenthesis). Each string is given the attributes listed in Table 2 below.

Table 2: List of attributes in the data

Attributes	**Description**	**Data type**	**Used in classifiers**
string_id	Id of the string in the list	Integer	No
string	The string itself	String	No
filename	Name of the file from which the string is generated	String	No
documentID	The id of the filename	Integer	Yes
start_index	The position index indicates where the string is in the list of strings generated from the document	Integer	Yes
end_index	Equals to start_index + number of words in the string	Integer	Yes
frequency	Number of occurrences of the string in the documents. For 2, 3, 4-word string, frequency = 1	Integer	Yes
prefixed	Indicator of whether the string has a special prefixes (eg. Mr., Dr., etc)	Binary	Yes
suffixed	Indicator of whether the string is followed by special words (eg. who, writes, said)	Binary	Yes
otherEntity	Indicator of whether the string can potentially be other entity type such as location, organization, sport teams	Binary	Yes

	or movie titles from a list of movie titles [1]		
near_capitalized	Indicator of whether the string is surrounded by other capitalized strings	Binary	Yes
name_suffix	Indicator of whether the string contains a name suffix such as Jr., Sr., VI	Binary	Yes
common_word	Indicator of whether the string contains any common words such as that, which, he, she	Binary	Yes
first_name	Indicator whether the string contains common first names from a list of common first names [2]	Binary	Yes
actor_legislator_name	Indicator whether the string contains actor names from a list of actor names [1] or the string contains name of legislators from a list of legislators [3]	Binary	Yes
in_name_list	Indicates that the candidate name occurs in a list of capitalized words: e.g. Bob, Steve, and Jane or Bob/Steve/Jane, etc.	Binary	Yes
near_parentheses	Indicator whether the string is preceded or succeeded by a '(' or ')'	Binary	Yes
near_verb	Indicator whether the string is preceded or succeeded by a verb (using nltk)	Binary	Yes
near_adj	Indicator whether the string is preceded or succeeded by an adjective (using nltk)	Binary	Yes
is_proper_noun	Indicator whether the string is a proper noun (using nltk)	Binary	Yes
class_label	Indicator whether the string is a person name based on the markup tags	Binary	Yes (used as class attribute)

3. Pruning Rules

We apply the following rules to prune away obviously negative examples. If a string is not capitalized or it contains punctuations, or it is a common word, or it is blacklisted, it will be removed from the data set. We made exceptions to special cases such as name initials in ‘George R. R. Martin’ so that names with initials will pass the rule while instance like ‘Mr. Trump’ will be removed from the data set. Another example of special cases is name with hyphen like ‘Ayman al-Zawahri’. Applying these pruning rules, the data set is reduced by about **97%**. The training data set was reduced from **452,232** instances to only **15,717** instances. Our code kept track of the labels so that if a positive example was accidentally pruned, it would report it as an error. We continued to edit our pruning rules until no positive example was thrown out. The code in “feature_data_generator.py” under “Code for this Project” directory provides detail implementation of the pruning rules.

4. Cross Validation

We performed 5-fold cross validation (in “cross_validation.py” under “Code for this Project” directory) on the data generated from the development set I using built-in classifier packages from scikit-learn including decision tree (DT), random forest (RF), support vector machine (SVM), linear regression (linReg), and logistic regression (logReg). We found random forest classifier to be the best classifier to train the data. The original precision, recall and F1 score of this classifier was **0.69, 0.51, and 0.59** respectively.

Table 3: Precision, recall, and F1 scores of classifiers before major debugging

Classifier	Original Precision	Original Recall	Original F1
decision tree	0.57	0.58	0.58
random forest	**0.69**	**0.51**	**0.59**
SVM	0.18	0.01	0.01
linear regression	0.77	0.19	0.31
logistic regression	0.69	0.49	0.57

5. Debugging Classifier

Based on cross validation results, we selected random forest to be the best classifier based on cross validation, we then split set I to set P and set Q (each set has roughly the same number of instances). We trained random forest classifier on set P, and used

the trained classifier on set Q. At the start of debugging, our classifier was achieving **69% precision and 51% recall**.
One of the first bugs we discovered was with the frequency feature (number of occurrences of the candidate string in the document). We discovered that it was searching the document before we removed the <pname>...</pname> tags, which led to the frequency count being incorrect because the count would include non-parsed strings like "<pname>Bob". Thus, we corrected it to correctly count words excluding the tags <pname>...</pname>. After we fixed this bug, our precision and recall dramatically increased to **80% and 69%** respectively.

We then investigated the false positive instances, and we found the following general patterns:

a. String with a prefix title such as Mr., Dr. was predicted as a person name
b. String that was blacklisted was not pruned away by pruning rules as it should have been.
c. Many names of organization, locations got predicted as person name.

Based on these observations, we tightened the pruning rules, and also extended our "Blacklist" which contains strings, which in our opinion, should never be a person name. We also found that the original implementation of the pruning rules have bugs in the code that are not syntactically incorrect but logically wrong. For example when a string was converted to lower cases in order to compare with other strings in a list, the function '.lower' was used instead of '.lower()'. This typo did not generate any error when the code was compiled but clearly the function did not do what we intended for it to do. After fixing all the pruning rules, we generated the data and performed cross validation again (as in Section 4) on this new data set. Indeed, the modification of the pruning rules improved the predictions significantly with precision, recall and F1 scores are **0.86, 0.75, and 0.80** respectively.

Table 4: Precision, recall, and F1 scores of classifiers after major debugging

Classifier	**Debugged Precision**	**Debugged Recall**	**Debugged F1**
decision tree	0.78	0.77	0.78
random forest	**0.86**	**0.75**	**0.80**
SVM	0.14	0.01	0.01
linear regression	0.82	0.74	0.78
logistic regression	0.85	0.49	0.63

Our classifier at this point was very close to 90% precision, and so we decided to add in a probability threshold of 0.75 using sklearn's predict_proba() function. This increased our random forest classifier to **92% precision.** See Table 5 for the full results.

Table 5: Precision, recall, and F1 scores of random forest after predict_proba()

Classifier	Debugged Precision	Debugged Recall	Debugged F1
random forest	0.92	0.68	0.79

6. Post Processing and Classifier Validation

We ran our random forest classifier trained from set I on set J. The results are listed in Table 6.

Because the precision score does not meet the 90% requirement, we applied a post processing rule such that if multiple strings at the same start_index are classified as positive labels, we would only assign the positive label to the superset of these strings. For example, if "Will" and "Will Smith" are predicted as a person name in sentence "Will Smith congratulates Black Panther cast.", the rule will override the classifier's prediction by assigning a positive label to only "Will Smith". However, this processing rule did not seem to help improve the precision of the classifier both for set I and set J. Therefore, the final precision, recall and F1 score for set I and J are as in Table 5 and Table 6 respectively.

Table 6: Precision, recall, and F1 scores of random forest for set J

Classifier	Precision	Recall	F1
random forest	0.86	0.63	0.73

7. Discussion

Our classifier reached a precision of >90% at around 92% and a recall of 68% for the development set I. However, our classifier only reached a precision of 86% and a recall of 63% on test set J. We could have over fitted the classifier on set I during debugging steps when we used list of other entities (eg. movie titles, locations) to filter out potential non-person name entity. There are still many ways in which we could improve our classifier. One possibility is that there may have been inconsistencies in the data markups, as three individuals were marking up the documents separately. Subjectivity and human error could have potentially incorrectly labeled some of the documents, which would affect the classifier's performance. For example, some group members might have marked superhero names (e.g. Batman or Black Panther) as a person name, while others may have not. Other potential methods to improve our classifier include creating more useful features to describe the context in which a person name is mentioned. Although our table clearly shows we included many interesting and helpful features, they are far from exhaustive, and we could continue to add more useful features in order to push the classifier to the max. Another thing we could try is to extend our strategy in text parsing and string pruning to handle more special cases. For example, we currently have a rule that allows a candidate string to contain a period in it to handle name initials. This rule will not prune away strings with initials like “George R. R. Martin”, however it will also not prune away strings like “self-assured.What do” because we have a rule to preserve names with hyphen like ‘Ayman al-Zawahri’. We believe adding more sophisticated rules to handle special cases like this will also help decrease false positives and improve precision score.

8. References

[1] Movie Titles and actor names: http://www.imdb.com/interfaces/
[2] US census first names: http://deron.meranda.us/data/census-derived-all-first.txt
[3] Congressmen names: https://github.com/unitedstates/congress-legislators