Name: Daniel Van Der Maden
Email: dvandermaden0@berkeley.edu

1 Model

I decided to implement the logistic regression model to classify each possible antecedent (as referencing the same mention or not) for each PRP and PRP$ in the given text. In doing so I had to create a function that parses the given data frame into tokens that can be trained on. With inspiration from SHW1, I did so as follows:

Given a data frame *df*, I parsed *df* into all possible pronoun and antecedent tuples (using their index in *df*), call this tuple t. Then, I paired up t with m (creating the tuple: (t, m)) where m indicates if the indexes in t have the same matching_id in *df* (indicated by a 1 or 0). Note that m changes with the training data frame, but it is defaulted to 0 (b/c it is not used) for test data frame. So in the data frame of the prompt, some of the parsed data (called '<filename>_dat' in the code's main) are: $[((15, 0)1), ((15, 1)1), ((15, 6)0), ((15, 7)0), \ldots]$. This essentially describes the function: **create_data**.

The parsed data from above, call it *dat*, is then sent through a processor that is very similar to the Logistic regression processor function found in SHW1. Said processor featurizes each tuple in *dat* using their respective data frame, *df*. (features described in the next section). Then it adds the processed features to a sparse matrix X and the desired output (m in a *dat* tuple) to a vector Y. Then we train the Logistic regression model with X and Y, and evaluate it (though this evaluation isn't 100 accurate b/c their can be multiple valid antecedents).

Once the Linear Regression model is trained, it is then use to generate a prediction on the dev data frame. The prediction are then passed to a method that modifies the respective data frame. This method is called **modify_df_with_preds** in the code. Finally the dev data is then passed into the given **check_accuracy** function.

The test data is processed in a similar way in that the test data frame is parsed as mentioned above, then it uses said trained logistic regression model to generate a prediction vector. The prediction vector and test data frame is then modified using the function mentioned in the paragraph above. Then said modified data frame is written to a csv file.

2 Features

Everything here is in the context of 1 parsed data token: $((i, j), k))$ where i is the index of the pronoun in the respective data frame, j is the index of the proposed antecedent in the respective data frame, and k is a binary value that represents if said antecedent has the same mention_id. Note that k is never used in the features.

Initially I just featurized the part of speech and token and noticed that I got 97% accuracy on the dev data/data frame, which is a clear sign of over fitting. So I decided to implement some of the features described in lecture.

The first feature I implemented (in addition to said pos and token feature) was a lemmatized version of the token. The reason being is that some words may have different tenses or nominations that would normally have their own feature. Having the lemma of the token relates them a little better.

The next feature that I implemented was the token distance between the pronoun and the proposed antecedent. The reason here is that antecedents should have some sort of a preferable distance to its pronoun. Note that I capped the token distance at 100, partially because that's what the naive solution did, but justified because very far tokens are probably not the correct antecedent (though they could be hence why I didn't throw them out). Furthermore, I binned the distances in factors of 4 because they would all create their own features if they weren't binned. Also, this binning is justified because small variations in distances are fine, since we are only concerned about significantly large distances steps.

The last feature that I implemented was a binary number agreement. This simply checks if the pronoun and proposed antecedent are both singular or both plural (and adds a feature respectively). If they are different, then no feature is added (thus an implicit feature). The justification here is that if both are the same 'number of things' they are more likely to mention the same thing.

All of these features dropped the dev accuracy to $\approx 61\%$.

3 Mention Filtering

To further improve the dev accuracy, I implemented some of the mention filtering (described in the lecture) in the data frame parsing function (**create_data**). Said filter doesn't add a token (the token that is featurized) if proposed antecedent has digits in the word. The justification for this is that numbed things are probably not the correct antecedent. It also doesn't add a token if the proposed antecedent is 'it' (since it is so ubiquitous). Lastly it doesn't add a token if the proposed antecedent is a stop word (for fairly obvious reasons).

After doing this mention filtering, the dev accuracy increased to 70.5%

4 Last Note

Training the data doesn't take too long after it is parsed. However, creating the parsed the data takes a considerable amount of time. Furthermore doing the substitution of the data frame (after the prediction is make) takes a considerable amount of time. So this method isn't too efficient if the data needs to be parsed and 'cleaned'.