Aditya Agashe (ava9)
Rohit Biswas (rb625)
John Hughes (jlh457)

Kaggle: Hairy Macho Men (HMM)

Due: October 22, 2016

Project 2 Report: Finding Uncertainty with Sequence Tagging

Introduction

For this project, we had to detect uncertain sentences and pinpoint the exact fragment of the sentence that triggers the uncertainty using a sequence tagging model.

Implementation

Before we trained our model, we first wrote our own code to preprocess the training data to add various tags including BILOU (Beginning, Inside, Last, Outside, Unit-Length), inDict, In our proposal, we hypothesized that closely neighboring words would be more useful than an entire sequence at detecting uncertainty. Therefore, we chose to create a CRF (Conditional Random Field) model as it is much less reliant upon an entire sequence than an HMM model and also allows for a great deal of customization and optimization.

We then chose various sets of features (more on this in ***Experiments***) that we hypothesized would be good for detecting uncertainty and recorded the results of how each performed. Finally, we took the newly processed and formatted data along with our defined features and used a library called CRFSuite to create the model and tag the test data.

Pre/post-processing

While tedious and confusing at times, most of the pre-processing is logically simple because of the nature of our model. As mentioned previously, we took the CUE tags given in the training data and converted those tags to BILOU tags. BILOU stands for beginning, inside, last, outside, unit. The problem with using CUE tags is that our model should almost surely not care about whether a phrase is the first, second, third, etc uncertain phrase in a file. Instead, we want uncertain phrases in the training data all to be treated equally. However, we cannot simply tag words in uncertain phrases with something like a “Unc” tag as consecutive uncertain phrases would appear to run together. BILOU tags allow us to treat each uncertain phrase fundamentally the same as the others while also distinguishing separate, but consecutive uncertain phrases.

Other than adding a BILOU tag to each word in the training data, the rest of our preprocessing was simply parsing the data from the .txt files and putting it in the data structures that our library requires.

After we make the library produce tags on the test data, we again simply take the output data structures and format it as the submissions requires.

Experiments

Because we used a CRF model for the main part of our experiment, the bulk of our testing comes from using various combinations of the data features. In general, our model features followed a similar pattern: a tuple of pairs of the type (feature, offset). For example, we may

have a model feature ((pos,-1),(w,0)) which would represent the combination of the previous part of speech and current word. The precision, recall, and F1 scores obtained on validation (80% train data, 20% validation) for various combinations of model features are given in ***Table 1***.

The CRF library that we used also gave the option to use various algorithms to train on the features we created. The results in ***Table 1*** all use the LBFGS algorithm. In ***Table 2***, we show how the different available algorithms perform for a common set of features.

Results

Table 1*: CRF Performance for Various Model Features*

Features	Precision	Recall	F1
w:(-1,0) **pos:**0	0.570775	0.264982	0.3013961
w:(-1,0),0 **pos:**0	0.611086	0.299794	0.352168
w:(-1,0),0 **pos**,0,(-1,0)	0.618998	0.319696	0.375936
w:(-1,0),0,(0,1) **pos**,0,(-1,0),(0,1)	0.676623	0.364579	0.436437
w:(-1,0),0,(0,1),1,-1 **pos**,0,(-1,0),(0,1),1,-1	0.709035	0.406953	0.490501
w:(-1,0),0,(0,1),1,-1,-2,2 **pos:**0,(-1,0),(0,1),1,-1,-2,2	0.686103	0.440442	0.519559
w:(-1,0),0,(0,1),1,-1,-2,2,(-2,-1),(1,2) **pos**,0,(-1,0),(0,1),1,-1,-2,2	0.678055	0.439332	0.516610
w:(-1,0),0,(0,1),1,-1,-2,2 **pos**,0,(-1,0),(0,1),1,-1,-2,2,(-2,-1),(1,2)	0.686193	0.446086	0.524053
w:(-1,0),0,(0,1),1,-1,-2,2,(-2,-1),(1,2) **pos:**0,(-1,0),(0,1),1,-	0.683191	0.445128	0.522655

1,-2,2,(-2,-1),(1,2) **Combo:**(pos0,w0)			
w:(-1,0),0,(0,1),1,-1,-2,2,(-2,-1),(1,2) **pos:**0,(-1,0),(0,1),1,-1,-2,2,(-2,-1),(1,2) **Combo:**(pos-1,w-1),(pos1,w1)	0.691479	0.447661	0.526691

The list of features is represented as follows: In the **w** category, we give features only containing words. Each value in the list are the offsets we consider. For example, the entry "0" in **w** refers to a feature that only considers the current word. -1 only considers the previous word. (-1,0) considers the pair of the previous word and the current word. **Pos** follows similarly. The **Combo** section has values that correspond to the feature with the offset right next to it. For example (pos-1,w0) would consider the pair of the previous part of speech with the current word.

We can see that as more features are added to the model, the model clearly improves in performance. However, as we start to add more obscure combinations of features, the improvement becomes more marginal. This could suggest that any gains made by adding a potential indicator of uncertainty could be canceled out by the potential overfitting of our train data. Especially as the F1 score approaches ~0.51-0.52 adding more features has almost no impact on the performance. This suggests that the features in the first 6 entries of the table are likely more helpful in predicting uncertainty.

***Table 2**: Algorithm Performance for Common Feature Set*

Algorithm	Precision	Recall	F1
LBFGS	0.691479	0.447661	0.526691
L2SGD	0.713051	0.403119	0.487925
AP	0.619284	0.462113	0.520920
PA	0.618075	0.454805	0.515473
AROW	0.563296	0.432602	0.481479

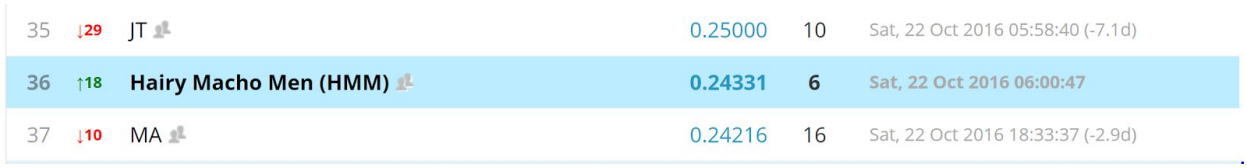
LBFGS =L-BFGS with L1/L2 regularization, L2SGD = SGD with L2-regularization, AP = Averaged Perceptron, PA = Passive Agressive, Arow = Adaptive Regularization of Weights (AROW)

In this previous table we can see that the different algorithms have similar, but slightly different performances. The LBFGS algorithm that we used for various feature testing has the highest F1 score.

Competition Scores

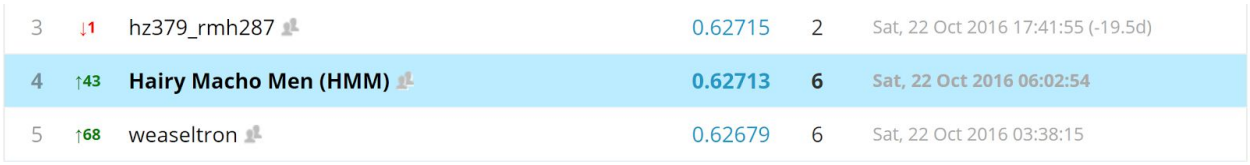
Kaggle Team Name: Hairy Macho Men (HMM)

In the uncertain phrase detection competition, our best submission is currently in 36th place with a score of 0.24331.



35	↓29	JT	0.25000	10	Sat, 22 Oct 2016 05:58:40 (-7.1d)
36	**↑18**	**Hairy Macho Men (HMM)**	**0.24331**	**6**	**Sat, 22 Oct 2016 06:00:47**
37	↓10	MA	0.24216	16	Sat, 22 Oct 2016 18:33:37 (-2.9d)

In the uncertain sentence detection competition, our best submission is currently in 4th place with a score of 0.62713.



3	↓1	hz379_rmh287	0.62715	2	Sat, 22 Oct 2016 17:41:55 (-19.5d)
4	**↑43**	**Hairy Macho Men (HMM)**	**0.62713**	**6**	**Sat, 22 Oct 2016 06:02:54**
5	↑68	weaseltron	0.62679	6	Sat, 22 Oct 2016 03:38:15

Extension

For our first extension, we implemented an extra feature for our CRF model. We chose a set of words that have very high probabilities of being uncertain and simply tagged a word with “true” if the word is in our set or “false” if it is not. The results are shown below, using the same system as in **Table 1** for describing the set of features used.

Features	Precision	Recall	F1
inDict:0	0.193958	0.200000	0.196933
w:(-1,0),0,(0,1),1,-1,-2,2,(-2,-1),(1,2) **pos:**0,(-1,0),(0,1),1,-1,-2,2,(-2,-1),(1,2) **inDict:**0 **Combo:**(pos-1,w-1),(pos1,w1)	0.689284	0.444535	0.522915

The dictionary appears to establish a decent baseline for prediction. However, it does not seem as predictive as some of the other previously mentioned features, at least by itself. This could be due to a poorly chosen dictionary or it could be that having a dictionary of common weasel words is simply not predictive.

As our second extension, we chose to use a different library (Seqlearn) that uses a HMM to tag the data. We wrote the HMM model in hmm.py and were able to create a MultinomialHMM model that fit to the training data. However, when we tried to use the model to run predictions

on the test data, we were not able to obtain good results. The output always contained tags of 1 nature (only “O” tags), and so we were not able to obtain precision, recall, and F1 scores for the model.

In our proposal, we had predicted that the HMM would not perform as well because it is dependent on entire sequences (sentences in our case) as opposed to more closely neighboring words. While we were not able to produce our own HMM results to compare, our model did produce a pretty high score on the Kaggle leaderboards. Since most other groups appear to have implemented an HMM instead of a CRF model, this could suggest that a CRF model is simply a better fit. This certainly requires more testing, however.

Contribution of Team Members

All members of our team worked together on the project, contributing to the design, code, experimentation, and report.