Maximum Entropy and Naive Bayes Assignment

Jean Mark Gawron[1]*

Abstract
This document provides a model answer for the Maximum Entropy assignment, as well as some general comments on the goals of this assignment.

Keywords
Statistical models — Maximum Entropy — Log linear models — Naive Bayes — Smoothing – Feature Selection

[1] *Department of Linguistics, San Diego State University, San Diego, CA*
***Corresponding author**: gawron@mail.sdsu.edu

Contents

1. Baseline Facts

The annotated sense corpus we used for this exercise and the **senseval format** it is represented in is due to Leacock et al. (1998). The senses we used are those available in a large semantic graph of English called **WordNet**, which is available in NLTK:[1]

```
>>> from nltk.corpus import wordnet as wn
>>> for ss in wn.synsets('hard','a')[0:4]:
        print ss.definition()
        print
```

This prints out the 4 top senses of the adjective *hard*. The 3 we use in this exercise are shown in Table 1.

We started with a baseline system that prunes stopwords like *the* and *of* from the vocabulary and then uses the 100 most frequent words as features, attempting to use the word **context** in which *hard* occurs to determine which sense it is used in. With 100 iterations of training it earns a score of 86.3% on the test.

[1] You should have NLTK installed along with the NLTK corpora.

	Log likelihood	Accuracy
Training	-.28557	.846
Test		.863

2. Better systems

2.1 How to improve the baseline

The kind of max ent system used in this assignment represents a sentence as the set of vocabulary items present in the sentence, where the vocabulary is chosen in advance, and words not in the vocabulary are ignored.

The baseline system can be improved in a vareity of ways, the simplest being **feature engineering**. Adding noun features that strongly correlate with one of three senses helps considerably, particularly since many of these nouns did not appear in the baseline system since they were too infrequent to make the cut. This suggests something that turns out to be a basic principle of Natural Language Processing: Choosing features by frequency is a bad strategy.

The way to add arbitrary word features was simply to add words one by one to *vocab* dictionary in the definition of the `extract_vocab` function in `call_extract_event.py`. A fictional count was necessary for consistency, but this count made no difference. This was explained in the slides providing information on the assignment. For example, to add the noun *cover* (as opposed to the verb *cover*), you added a line containing the following dictionary update:

```
vocab['cover_NN'] = 1000
```

Many of you had considerable success with such features.

If you did nothing more than add such dictionary updates to the code, you were **adding** features to the existing 100 features in the baseline system. If you wanted to omit all 100 baseline features and just use those you selected, you needed to add a line **before** all your dictionary updates to empty the vocab dictionary:

```
vocab = dict()
```

Sense	Freq(%)	Definition
HARD1	79.7	not easy; requiring great physical or mental effort to accomplish or comprehend or endure (*hard subject*, *hard journey*);
HARD2	11.6	dispassionate, very strong or vigorous (amalgam of senses 2 and 4 in WordNet (*hard work*, *hard look*);
HARD3	8.7	resisting weight or pressure (opposite of soft, *hard pillow*, *hard metal*)

Table 1. The three senses in the Senseval data

This sets *vocab* to be an empty dictionary. The numbers reported in the next section are all for systems that only use IG features. So the dictionary was emptied before adding IG features.

2.2 Example systems

The best system shown below used a 200-word vocabulary selected by an **information gain** algorithm. Let's call these **the IG features**. No student was expected to find exactly this set of features, but the IG system shows that feature selection/feature engineering can improve the accuracy of the system considerably.

The table in Table 2 shows that IG features helped, and that number of iterations used in running the IIS algorithm also mattered.

Feats	**Iters**		**Log Lkly**	**Acc**
Base	100	Training	-.28557	.846
		Test		.863
200	100	Training	-.30445	.831
		Test		.859
	200	Training	-.24869	.880
		Test		.902
	300	Training	-.22139	.896
		Test		**.910**
	400	Training	-.20568	.898
		Test		.907
100	100	Training	-.26166	.873
		Test		.900
	200	Training	-.21813	.889
		Test		.905
	300	Training	-.20128	.893
		Test		.902

Table 2. System comparisons (best in bold)

Summing up: The best system used 200 IG features and ran for 300 iterations. In the best system, accuracy improved to 91.0% on the test. In the 100 feature system, increasing from 200 to 300 iterations consistently reduced the test score. In general, more iterations guarantees higher likelihood, but does not guarantee a better test score. Neverthless, more iterations can be very helpful, as is particularly evident with the 200 feature system. The poor (lower than baseline) performance of the 200 IG feature system at 100 iterations is due to the fact that the incremental improvement in likelihood is likely to be slower with more features, and 100 iterations is not enough to achieve a reasonable likelihood score with 200 features (notice the training set likelihood for that system is worse than the baseline). The improvement with 200 and 300 iterations is considerable.

Looking at the Precision/Recall numbers for each class gives some insight as to how the IG systems achieved their improvement. This is shown in

Feats	Iters	Label	Precision	Recall
Baseline	100	HARD1	.864	.997
		HARD2	.941	.485
		HARD3	.500	.050
200	100			
		HARD1	.855	1.000
		HARD2	.667	.061
		HARD3	1.000	.325
	200			
		HARD1	.903	.994
		HARD2	.842	.485
		HARD3	.950	.475
	300			
		HARD1	.903	.994
		HARD2	.842	.485
		HARD3	.950	.475
	400			
		HARD1	.915	.985
		HARD2	.783	.545
		HARD3	.917	.550
100	100			
		HARD1	.898	.994
		HARD2	.889	.485
		HARD3	.947	.450
	200			
		HARD1	.908	.991
		HARD2	.857	.545
		HARD3	.905	.475
	300			
		HARD1	.907	.988
		HARD2	.810	.545
		HARD3	.905	.475

Table 3. Precision and Recall scores for each of the classes

Comparing the baseline system at the top of Table 3 to the winner, the 200-feat 300-iteration system, we see the most

dramatic improvement comes from improved performance on HARD3, the sense that is the opposite of *soft*. Improving precision on HARD1 is also important. This is done by guessing HARD1 (the most frequent sense) more reliably, because of developing better indicators of SENSE1, and also by guessing SENSE1 less often, because of relying on indicators of the other two senses. We will try to understand that improved performance below.

2.3 The features

Table 4 shows the list of 100 words used in the 100 IG features system, a strict subset of the 200-feature system. Bear in mind as you look that this isn't the actual list of features being used by the Max Ent system. The actual features are pairings of each word with a class (you'll see this in the discussion of most informative features).[2]

The list of words in Table 4. is intended to convey an idea of what the feature selection algorithm can discover. By and large the features are a higgledyiggledy assortment, but there are some features that the feature selection algorithm which have a clear link to the improved performance of the 200 IG feature system. See if you can identify them, You might focus first on looking for words that are good indicators of sense HARD3.

Eyeballing the features in Table 4 gives us some insight as to how improved performance on sense HARD3 was achieved: Quite naturally, there are a number of Noun features; the noun *hard* modifies often disambiguates the sense (*hard winter*[HARD1], *hard feeling(s)*[HARD2], and *hard water*[HARD3]). Among these there a number of relatively low count nouns denoting physical objects which will identify HARD3: *wall (10 tokens), wood (11 tokens), cheese (16 tokens), surface (37 tokens), shell (16 tokens), seat (18 tokens), cover (29 tokens), plastic (15 tokens), ice (10 tokens)*. Since the cutoff point for making the list of the top 100 most frequent words was a frequency of 73, none of these words were features in the baseline system. We will see further evidence of the significance of such features when we look at the most informative features.

The lesson here is that choosing features by frequency is a bad strategy in a classification setting. A lot of the high information words are low frequency.

3. Most informative Features

Table 5 shows the most informative features for both the Max Ent and NB systems. We will restrict our attention to the 200 iteration, 100 IG feature system. They were very close in performance, and there is very little change in informative features between these systems (because the 100 features are the top half of the 200 features, when scored by information gain).

The maximum entropy list includes both positive and negative feature coefficients, corrsponding to whether a feature has a strong positive or negative correlation with a class. The Naive Bayes feature list shows the features that most increased the odds of one class versus another.

We see significant overlap in which words provide the most information among two feature sets, despite the fact that they're different kinds of features (a Max Ent feature includes a specific class, because it is used in modeling the joint probability of a feature and class).

> wheat_NN, cover_NN, get_VB, plastic_NN, surface_NN, work_NN, cheese_NN, material_NN, eye_NN, find_VB look_NN, shell_NN, wall_NN

As expected, we see a number of physical object nouns in both sets, consistent with our thoughts about improved performance with SENSE3. We will see in the next section that Naive Bayes classification also benefited with this feature set.

[2]The 200 feature IG system actually has only 189 distinct words, since there were 11 words that made the cut when paired with more than one class.

	Test Acc
Baseline MaxEnt	.863
Baseline NB	.878
100 IG Feats NB	.902
200 IG Feats NB	.902

Table 6. Naive Bayes results

4. Naive Bayes

Estimating a Naive Bayes model is way more computationally efficient than than estimating a Max Ent model, and the bottom line is that with the right feature set, and in situations where there is a very limited amount of data (this one), Naive Bayes can be quite competitive with Maximum Entropy. It is always worth trying, because it usually provides insight into the structure of the problem, and because it does not cost a lot of computing time.

Table 6 shows that the baseline Naive Bayes test scores were significantly better than the baseline Maximum Entropy test scores. The baseline NB scores improved with the 100 feature IG system, as one might expect, but the interesting surprise is that there was no change at all in accuracy with 200 feature system.

The lack of improvement with 200 features is probably because those new features did not add a lot of discriminating power to the classifier. This is supported by the fact that there was very little change in the most informative features with the 200 feature system. This suggests that information gain was doing a good job of feature ranking for this task.

Table 7 shows the Baseline, 100 and 200 feat Precision and Recall breakdowns for Naive Bayes.

The 100 and 200 IG Feat numbers are better than the baseline, especially in HARD3, but very similar to each other. Comparing the 100 IG feat system to the 200 IG feat system, various trade-offs have been made resulting in no improvement in overall accuracy. For example the 200 feature system has better Recall for HARD3, but pays for it with worse Precision. Although performance for both Precision and Recall

's_VBZ	extremely_RB	like_VB	school_NN	war_NN
't_NN	eye_NN	line_NN	seat_NN	water_NN
,_,	face_NN	look_NN	see_VB	wheat_NN
25_CD	feeling_NN	look_VBP	sense_NN	willing_JJ
3_CD	find_VB	material_NN	shell_NN	wine_NN
4_CD	finding_VBG	needed_VBN	show_NN	winter_NN
album_NN	five_CD	next_JJ	six_CD	wood_NN
already_RB	fresh_JJ	north_NNP	small_JJ	work_NN
answer_NN	generally_RB	other_NN	soft_JJ	world_NN
believe_VB	get_VB	parent_NN	story_NN	
character_NN	girl_NN	place_NN	success_NN	
cheese_NN	great_JJ	plastic_NN	sunday_NNP	
choice_NN	group_NN	police_NN	surface_NN	
clothe_NN	harder_JJ	possible_JJ	team_NN	
comfortable_JJ	hardest_JJ	pretty_RB	time_NN	
companie_NN	hot_JJ	put_VB	to_TO	
cover_NN	ice_NN	record_NN	used_VBN	
cream_NN	industry_NN	red_JJ	very_RB	
democrats_NNPS	joe_NNP	rock_NN	voice_NN	
doing_VBG	kid_NN	say_VB	wall_NN	
economic_JJ	learned_VBD	say_VBP		
especially_RB	lesson_NN			
evidence_NN				

Table 4. Top 100 features

Feats	Label	Precision	Recall
Base	HARD1	.908	.964
	HARD2	.818	.545
	HARD3	.567	.425
200	HARD1	.919	.973
	HARD2	.720	.545
	HARD3	.857	.600
100	HARD1	.924	.978
	HARD2	.692	.545
	HARD3	.833	.625

Table 7. Precision and recall scores for each class with Naive Bayes

is better with HARD1, it is compensated for by a loss of Precision with HARD2.

Like the Max Ent model, the NB model improves because of massive improvement in recognizing HARD3. As we saw in the last section, the source of this improvement can be inferred from inspecting the most informative features, which have changed considerably from the NB baseline, because of a number of low frequency nouns that help disambiguate senses.

5. Additional questions

1. You were asked about the *pdist* object associated with one example, and what prob it returned when applied to a class (such as HARD1):

```
>>> (feats, label) = test[0]
>>> pdist =
    nb_classifier.prob_classify(feats)
>>> pdist.prob('HARD3')
```

If you applied *pdist* to all three classes as suggested in the hint, you saw the three probabilities added up to 1.0, strongly suggesting that what this *pdist* object represented was the probability of the class given the feature set *pdist* was made from (P(class | feats)).

2. A question you were asked on the assignment is what features were responsible for the probability of one class being so high on `test[7]`.

   First let's use what we learned above to see how `test[7]` is classified:

```
>>> (feats, label) = test[7]
>>> pdist =
     nb_classifier.prob_classify(feats)
>>> pdist.prob('HARD1')
0.9911 ...
>>> pdist.prob('HARD3')
0.0012 ...
>>> pdist.prob('HARD2')
0.0076 ...
```

So this is very confidently classified as HARD1. Let's see why: Here's how to find out what features it had in the baseline system (the answer may change in your various systems). You just look at the feature dictionary,

Max Ent			Naive Bayes		
Weight	Word	Class	Word	Classes	Odds
2.071	north_NNP	HARD3	cover_NN	HARD3 : HARD1	72.5 : 1.0
1.987	plastic_NN	HARD3	wheat_NN	HARD3 : HARD1	71.2 : 1.0
1.940	look_VBP	HARD2	look_NN	HARD2 : HARD1	62.6 : 1.0
1.870	shell_NN	HARD3	surface_NN	HARD3 : HARD1	53.2 : 1.0
1.863	surface_NN	HARD3	cheese_NN	HARD3 : HARD1	52.6 : 1.0
1.843	feeling_NN	HARD2	shell_NN	HARD3 : HARD1	46.5 : 1.0
1.840	comfortable_JJ	HARD3	rock_NN	HARD3 : HARD1	42.2 : 1.0
1.833	hot_JJ	HARD3	wall_NN	HARD3 : HARD1	40.3 : 1.0
1.806	democrats_NNPS	HARD2	plastic_NN	HARD3 : HARD1	39.0 : 1.0
1.792	evidence_NN	HARD2	soft_JJ	HARD3 : HARD1	38.5 : 1.0
1.733	wall_NN	HARD3	ice_NN	HARD3 : HARD1	34.1 : 1.0
1.725	line_NN	HARD2	work_NN	HARD2 : HARD3	22.4 : 1.0
1.713	cover_NN	HARD3	red_JJ	HARD3 : HARD1	19.3 : 1.0
1.703	25_CD	HARD2	wood_NN	HARD3 : HARD1	17.3 : 1.0
1.656	like_VB	HARD2	cream_NN	HARD3 : HARD1	17.3 : 1.0
1.645	red_JJ	HARD3	feeling_NN	HARD2 : HARD1	16.2 : 1.0
1.626	answer_NN	HARD2	material_NN	HARD3 : HARD1	15.5 : 1.0
1.618	work_NN	HARD2	find_VB	HARD1 : HARD2	14.1 : 1.0
1.585	material_NN	HARD3	get_VB	HARD1 : HARD3	12.5 : 1.0
1.567	look_NN	HARD2	eye_NN	HARD2 : HARD1	11.8 : 1.0
-3.850	look_NN	HARD1			
-3.676	cover_NN	HARD1			
-3.422	find_VB	HARD2			
-2.644	get_VB	HARD3			
-2.629	look_NN	HARD3			
-2.598	harder_JJ	HARD3			
-2.496	s_VBZ	HARD3			
-2.484	work_NN	HARD3			
-2.479	cheese_NN	HARD1			
-2.453	rock_NN	HARD1			
-2.398	surface_NN	HARD1			
-2.330	wall_NN	HARD1			
-2.318	shell_NN	HARD1			
-2.286	soft_JJ	HARD1			
-2.216	find_VB	HARD3			
-2.210	wheat_NN	HARD1			
-2.179	feeling_NN	HARD3			
-2.157	plastic_NN	HARD1			
-2.123	rock_NN	HARD2			
-2.092	work_NN	HARD1			

Table 5. Most Informative Features

and print out just the words in the vocab for which the feature value was *True* (the word was present in the sentence):

```
>>> (f,cls) = test[7]
>>> for (k,v) in f.iteritems():
        if v:
           print k
's_VBZ
said_VBD
to_TO
```

All three of these features can be looked up using the the Naive Bayes most informative features function, and all three lean toward *HARD1*:

's_VBZ = True	HARD1 : HARD3	8.9 : 1.0
to_TO = True	HARD1 : HARD3	2.4 : 1.0
said_VBD = True	HARD1 : HARD3	2.4 : 1.0

Looking at the code at the end of the `call_maxent.py` file shows how to call the most informative features function, and with a little variation on what's there you can look at more features. For example:

nb_classifier.show_most_informative_features(n=200)

6. Takeaways

1. Feature selection is your friend.

2. Information gain is a useful way of discovering features that are useful, Feature selection by intuition is hard (in sense HARD1) and unlikely to be optimal with natural language problems. But you can discover facts about the structure of a classification problem this way, and intuition is best guided by careful error analysis. Many of you discovered physical object features that improved your performance with HARD3. It was in observing how even useful bag-of-word features could mislead us that we saw that multi-word or syntactically informed features might help.

3. More iterations of the IIS algorithm does not necessarily help, but it may be necessary for larger feature sets.

4. Feature selection by frequency is a really bad idea. Very rare features are often the best. The trick is to have enough information in the training data to find this out.

5. Naive Bayes is computationally very cheap.

6. Despite its serious theoretical limtations, Naive Bayes can be extremely effective with the right feature set and the right problem.

Appendix: Limitations

A couple of more advanced points:

1. The fact that improving likelihood in many cases does not improve performance is one piece of evidence that our *objective function* (what we are maximizing) should include some kind of **regularization**. (roughly, smoothing). Finding appropriate ways to do this is a major topic in statistical analysis and machine learning.

   For example, using L_D for likelihood with respect to the data, m for the size of the feature set, and α is a penalty weighting between 0 and 1 controlling how much regularization you do:

   $$\hat{w} = \arg\min_{w} -L_D(w) + \alpha \sum_{j=1}^{m} |w_j|^2$$

   This is a form of smoothing that discourages 0 probabilities, because we penalize feature sets by the length of the weight vector, discouraging high negative values in the log sum, i.e., models that drive probabilities od FEATURE to 0. In a Bayesian setting, this penalty can be interpreted as a Gaussian prior over weight vectors, though we will not pursue that here.

2. A look at the data also gives a sense of the limitation of the method we are using. One of our top 100 IG word features is *feeling_NN* and, as suggested above, its occurrence in the expression *hard feelings* makes it a good indicator of sense HARD2. But our feature extractor takes no note of where in the sentence a word feature occurs; *feeling* is a fairly common noun and there are numerous examples in our data in which it occurs in a sentence, but not in the expression *hard feelings*. For example:

   > we_PRP find_VBP it_PRP hard_JJ to_TO duplicate_VB the_DT feeling_NN of_IN love_NN that_WDT encompasses_VBZ us_PRP when_WRB the_DT odor_NN of_IN french_JJ fries_NNS floats_VBZ through_IN the_DT air_NN ...

   This is sense HARD1 ("difficult"): Relying on the *feeling* feature here would mislead us. The problem, of course, is that our current feature extractor can't represent the fact that we don't have the expression *hard feelings* in this example. Using multiword features or building syntactic information into our features might help.[3]

[3] in this particular case, it might also help to distinguish singular and plural, which our extractor does not do; but doing that across the board would run the risk of impairing feature selection, because, for example, the 29 tokens of *cover* would be split, potentially reducing the information gain of that feature.

Acknowledgments

The author of this model answer would like to thank his brain for all the clever thoughts it supplied in the writing process. Any remaining errors are the sole responsibility of the typesetting program.

References

Leacock, Claudia, George A Miller, and Martin Chodorow. 1998. Using corpus statistics and wordnet relations for sense identification. *Computational Linguistics* 24(1):147–165.