Automatic Coding of Individual Turns in Negotiations
CS 773 Final Project

Raul Guerra *Wu, Ke* *Abhishek Kumar*
{*rguerra,wuke,abhishek*}*@cs.umd.edu*

Abstract

In this article, we report our experiments on the negotiations corpora for task of automatic coding of individual speaker turns. We outline different approaches that we tried and report experimental results in terms of k-best accuracy. The best result for 1-best accuracy that we obtain is 59.99% with using n-gram and collocation features with maximum entropy classifier.

1 Introduction

The corpus contains the text of negotiations between two persons (possibly from different cultures and backgrounds). Each person takes turns and says something to the other person. There is a label associated with each turn that captures the essence of that particular turn. There are 37 such labels. The task is to automatically assign labels to individual turns in the test corpus by learning a suitable model from the training data.

2 Preprocessing and Experiment Setup

Our preprocessing was mostly about cleaning up the data format of the raw corpus. Though distributed in a CSV file, the raw data has many segmentation errors — many thought units are mistakenly put in the same line. Also, there are some non-unified spellings of codes. We fixed those problems mainly by writing a script[1] with several cleaning up heuristics handling segmentation correction and code correction.

After this, the corpus was lower-cased and fed into the PennTreeBank English word tokenizer shipped in NLTK; we also performed some punctuation tweaks[2].

We also experimented both unstemmed and stemmed[3] n-grams features and the comparison is described in Section **??**.

For the ease of experiment, we randomly divided the data into training (6071 thought units; 38 dyads), dev (2132 thought units; 14 dyads) and test (1867 thought units; 11 dyads) data. In spite of the small size of the data set, coding performance measures turned out to be consistent when we compared with a 10-fold cross-validation in our final phase (done by hand; no code).

3 Baselines

First of all, we tried the simplest baseline that always predicts the most frequent code (SBR). This gives 437 / 1867 = 23.40% accuracy on test data.

Next, we used MaxEnt (using megam) and linear-chain CRF (using crfsuite) models with bag-of-words features of current, previous and next thought units plus current speakers role. Regularization parameters were tuned on dev data. We obtain the following accuracy results on test data:

megam (L2 regularization) 1049 / 1867 = 56.19%

[1]`preprocess/fields.py`
[2]`preprocess/tokenize.py`
[3]Using the Porter stemmer from NLTK.

crfsuite (L2 regularization) 1017 / 1867 = 54.47%

crfsuite (L1 regularization) 1000 / 1867 = 53.56%

Also, we evaluated the resubstitution accuracy on training data to see how much each model underestimate of the true error rate as a result of overfitting:

megam (L2 regularization) 5531 / 6071 = 91.11%

crfsuite (L2 regularization) 5939 / 6071 = 97.83%

crfsuite (L1 regularization) 6024 / 6071 = 99.23%

From these results we find CRF is severely overfitting the training data (achieving 97% (L2) and 99% (L1) accuracy!) while MaxEnt is slightly better in the sense that it performs somewhat worse on training data. Given that the key difference of two models is that CRF also introduces a sequence of dependent latent variables (states), we suspect this sequence dependency is the source of CRF's especially severe overfitting problem.

Given above observations and the relatively small size of available data, we decided to concentrate on working on improving coding accuracy of individual turns first.

4 Feature Engineering

We experimented two categories of features — general lexical features generated directly from the raw text; task-specific features that made use of domain knowledge and external resources (e.g. lexicons provided by the professor).

4.1 More on the Baseline

We created a baseline to compare features against and see if we get improved accuracy or not. This baseline was created using only the tokens of the thought units along with a nifty trick. To increase the number of features and to take into consideration the position of the thought unit with respect to the rest of the thought units, we consider a window of plus and minus one thought units for each thought unit. In other words, the tokens of the thought unit$_{i-1}$ and thought $unit_{i+1}$ are present as features in thought unit$_i$ with 'BACK' and 'FORWARD' annotations. When we trained a Maximum Entropy model (MEGAM) and tested on held out data we got an Overall accuracy of: 1181 / 2132 = 55.39% This nifty trick improved performance compared to not using the thought unit context. Any window bigger than plus and minus one gave a bad performance.

4.2 Lexical Features

4.2.1 n-grams

It is well known that unigram features do not provide adequate contextual information and therefore we experiment n-gram features with larger ns (namely, up to 3). However, as n increases, the classifier faces a more and more severe problem of data sparsity, which might lead to overfitting and hurt coding accuracy. In our experiment, we found using both unigrams and bigrams yielded the highest accuracy. Also, stemming is also able to slightly increase coding accuracy. Actual experiment results will be discussed in Section **??**.

4.2.2 Collocations

In addition to n-grams, we are also interested to see if longer subparts of a thought unit are useful in predicting codes. Since naive extraction of all long n-grams leads to nothing but tremendous data sparsity, we used an independent routine from previous work[4] for this task. In short, our method is based on the idea of minimal description length. We defined simple length measures for both text and dictionary and then run a hill-climbing search process to find collocations that compress the overall description length. In spite of the suboptimal search

[4]See `mdl/mdl.pdf`.

Temporal “8:30”, “7:30”, “6:30”, “in the morning”, “in the evening”, “in the summer”

Topical “hours of operation”, “grand opening date”, “renovation cost”, “a grocery store”

Discourse particle “you know”, “I mean”

Question/suggestion/compromise “do you want to”, “maybe we can”, “what do you think”, “what do you mean”, “how do you feel”, “why don’t we”, “I’ll give you”

Preference “be willing to”, “I can’t”, “I have to”, “for both of us”, “I would prefer”, “that makes sense”, “I just can’t”, “I’d be willing to”, “important to me”, “as early as possible”, “as soon as possible”, “the earlier the better”, “later the better”

Conversational “let’s see”, “we’re done”, “nice to meet you”

Figure 1: Collocations extracted from the training data

process, this method is able to find linguistically meaningful or data-specific interesting collocations longer than 2. Figure **??** lists some of the collocations extracted from the training data. In total, we extracted 126 collocations longer than 2 words from our training data.

Originally, we planned to try out methods based on collocational LDAs using the adaptor grammar. However, a single iteration with 5 topics on the training data took nearly 8 hours and we were thus unable to finish and obtain a stable sample before the deadline.

4.3 Task-Specific Features

4.3.1 Term Usage

Replacing all of the digits with @ symbols. Doing this facilitated the identification of references to time, money, percentages, temperature, and months. The motivation behind identifying this references comes from the negotiation experiment. In the negotiation each of the participants assumes the role of the grocery owner or the wine shop owner. In the negotiation the grocery and wine owner have five specific issues that they have to resolve.

1. Hours of Operation
2. Renovation Costs
3. Floor Space
4. Temperature
5. Grand Opening Date

Reference to time could be associated with Hours of Operation, references to money could be associated with Renovation Costs, references to temperature could be associated with the Store’s Temperature, references to specific months could be associated with the Grand Opening Date.

4.3.2 Stop Words

One of the simplest things that we tried was to remove stop words or function words. The list of function words used was the one shipped with the NLTK. This however hurt our performance. The reason was that a lot of the tokens in the thought unit were function words, so removing them reduced significantly the already scant data.

4.3.3 POS Tag Usage

Another feature was Part of Speech [POS] tags for the tokens in each thought unit. To get the POS tags we utilized the NLTKs Bigram tagger that utilizes the current word together with the POS of the previous word. The

Bigram tagger backs-off to a unigram tagger. The unigram tagger also backs-off to a tagger that tags everything as NN. All taggers were trained with the NPS Chat corpus that ships with the NLTK. This corpus consists of instant messaging chat sessions, originally collected by the Naval Postgraduate School. The corpus contains 10,000 anonymized posts. We thought that this would be the closest to the transcription text of the negotiations.

POS tagging was necessary to try out Lemmatization as part of the preprocessing **??**. Also we considered POS tags useful because the use of these in a thought unit could help the classification. For example, the use of conjunctions could mark the difference between a multi or single issue being discussed.

4.3.4 Lexicon Resources

The Linguistic Inquiry and Word Count resource is a dictionary composed of almost 4,500 words and word stems. This dictionary has been used to study various emotional, cognitive, structural, and process components present in individuals' verbal and written speech samples. Each of the words in the dictionary is annotated with one or more word categories. The idea of this resource is to keep a count of these word categories such that if a word within a category is found in a text, the count of this category is incremented. The usage of this categories is related to the individual's different characteristics mentioned before. We utilized this resource to do word look-ups and to try to capture the use of these word categories. To do this we used the different methods mentioned in **??**.

The other lexical resource used was the Subjective Clues. This corpus has been used in sentiment analysis. In this corpus, the words have two dimensions type and polarity. The type specifies how subjective a word is and the polarity whether the word is positive, negative, both, or neutral. The corpus has 8,221 words, some of them stemmed. Like with the LIWC2007 resource we tried to capture the usage of subjective words and of words with different polarity to help us classify the thought units.

5 Coarse-to-Fine Classification

5.1 Intuitions from Coarser Groups

There are 37 codes in the coding scheme, and they are highly fine-grained. For example, there are 3 different codes for "substantiation" (arguing for oneself or against the other) — rational (SBR), emotional (SBE) and factual (SF). Since they are all substantiation, one might expect them sharing some characteristics among a coarser group. Figure **??** lists the grouping scheme from the coding manual. To model this probabilistically, we let h be a random variable for this "coarser grouping" and break the conditional probability down in the following way:

$$P(Y|X) = \sum_h P(Y|X,h)P(h|X)$$

If we use the grouping scheme from the coding manual, one group is disjoint from each other and therefore for any code Y there will be only one h such that $P(Y|X,h) \neq 0$. Let $h(Y)$ be such an h, the probability is now

$$P(Y|X) = P(Y|X,h(Y))P(h(Y)|X)$$

Under this assumption, we could break prediction into two phases: one predicting the coarser grouping h and the other predicting the actual code given the coarser grouping. To estimate parameters for this model, we put codes into 9 groups according to the coding manual and trained a classifier to predict the coarser group of one thought unit and another 9 to predict the actual code within a group.

Intuitively, if codes under the same group do share something in common, this should improve the coding accuracy. However, in our experiments, this was not observed. For more details and analysis on the experiment, see Section **??**.

5.2 A Latent Model

For most of our classification tasks, we used Maximum Entropy (MaxEnt) based classifiers. As disscussed in class, MaxEnt models are widely used in NLP and has shown plausible performance on various tasks. MaxEnt

Offer OS, OM

Provide information IP, IR, IB

Substantiation SBR, SBE, SF

Questions QP, QR, QB, QS, QM

Summarizing INN, INP, MU, IDN, IDP, IS, GD

Threats/power/rights TH, PW, RT

Reactions RP, RN

Procedural comments P1, PC, PP, PM, PX, PO, PT

Miscellaneous/other CS, MIN, MIC, MINEG, OT

Figure 2: Grouping of codes in the coding manual

models the probability of an observation (X, Y), where $\Phi(X, Y)$ is the feature vector and w is the weight vector, as follows,

$$P(X, Y) \propto \exp\{w^T \Phi(X, Y)\}$$

The conditional probability $P(Y|X)$ is thus

$$P(Y|X) = \frac{\exp\{w^T \Phi(X, Y)\}}{\sum_y \exp\{w^T \Phi(X, y)\}}$$

Usually the training objective of a MaxEnt model is to maximize the conditional likelihood of training data and L2 regularization is often used for smoothing.

Based on our intuitions from coarse-to-fine classification in Section **??**, we introduced an additional latent variable "H" and extended MaxEnt to a slightly more sophisticated model with a latent variable (called "latent model" in this write-up). We model the probability of (X, Y, H) as

$$P(X, Y, H) \propto \exp\{\Psi(X, Y, H)\}$$

where $\Psi(X, Y, H)$ is a user specified potential function (discussed below). Now the conditional probability $P(Y|X)$ becomes

$$\begin{aligned} P(Y|X) &= \sum_h P(Y, h|X) \\ &= \frac{\sum_h P(X, Y, h)}{\sum_{y,h} P(X, y, h)} \\ &= \frac{\sum_h \exp\{\Psi(X, Y, h)\}}{\sum_{y,h} \exp\{\Psi(X, y, h)\}} \end{aligned}$$

Though now the log conditional probability becomes non-convex due to summing over the hidden variable, its gradient over the weight vector is still easily computable

$$\begin{aligned} \nabla \log P(Y|X) &= \nabla \log \sum_h \exp\{\Psi(X, Y, h)\} - \nabla \log \sum_{y,h} \exp\{\Psi(X, y, h)\} \\ &= \frac{\sum_h \exp\{\Psi(X, Y, h)\} \nabla \Psi(X, Y, h)}{\sum_h \exp\{\Psi(X, Y, h)\}} - \frac{\sum_{y,h} \exp\{\Psi(X, y, h)\} \nabla \Psi(X, y, h)}{\sum_{y,h} \exp\{\Psi(X, y, h)\}} \end{aligned}$$

Thanks to this, we still can use standard non-constraint optimization routines such as LBFGS to optimize the objective[5]. Also, adding L2 regularization is trivial and thus omitted here.

This model is in fact a simple conditional random field with 3 variables (X, Y, H). Therefore, any combination of these variables can be used to structure the potential function $\Psi(X, Y, H)$. In our experiment, we set $\Psi(X, Y, H) = w_H^T \Phi(X) + w_{(Y,H)}^T \Phi(X)$, where $\Phi(X)$ is the feature vector, which corresponds to the "coarser grouping" intuition. Another choice we need to make is the number of values the latent variable can take, here we set this to 5.

6 Feature Extraction Methods

We were motivated to try feature extraction methods after getting severe overfitting of models to the original sparse high dimensional features. The assumption was that having less number of model parameters to estimate will result in more robust estimates and so better generalization to unseen data. We tried following methods.

6.1 Latent Semantic Analysis

Latent Semantic Analysis (LSA) effectively performs a low rank approximation of the term-document matrix. If we have total T number of terms, the original documents are interpreted as vectors in this T-dimensional space. Instead of this, LSA tries to get a low dimensional embedding of these original features. Each dimension obtained by LSA can be treated as a *concept* or *topic* that represents similar words (or terms). LSA is performed by constructing a term-document matrix of size $T \times N$ and then doing the Singular Value Decomposition (SVD) of this matrix. The right singular vectors will give a *topic* based representation of the documents. In our task, we took text of each turn as a document (dimension ≈ 10000 in original space) and applied LSA on the whole corpus to get a representation based on concepts. LSA is a unsupervised method so we did not need the labels.

6.2 Probabilistic Latent Semantic Analysis

There are some shortcomings of LSA that can be overcome by probabilistic LSA. First, polysemy (multiple meanings of same words) can present a problem in LSA. Finding a low rank representation by throwing away low singular values has an inherent Gaussian assumption. However, often words are generated by a multinomial distribution. PLSA assumes existence of a latent variable z (denotes the concept) given which the terms t and documents d are independent. In other words, Z, T and D form a Markov chain $D \to Z \to T$. PLSA decomposes the probability of a term-document co-occurrence as product of conditional probability of terms (given concept), probability of concept and conditional probability of documents (given concept). This product is then marginalized over all concepts (latent variable z). We take the probabilities $P(z|d)$ as our new feature vectors that denote the probabilities of concepts given a document d. PLSA is solved using EM algorithm.

6.3 Principal Component Analysis

We also tried PCA on the data to reduce the dimensions. PCA extracts dimensions of highest variance from the data and represents it along those dimensions.

7 Experiments and Analysis

7.1 Effects of Various Features

7.1.1 n-grams and collocations

In Table **??**, we list k-best accuracy on 4 lexical feature sets from Sections **??** and **??** , namely

UN Unstemmed n-grams with $n = 1, 2$;

[5]We used liblbfgs (`http://www.chokkan.org/software/liblbfgs`) to implement the model.

Feature set	1-best	2-best	3-best	4-best	5-best
UN	59.13%	76.00%	82.54%	86.13%	89.50%
ST	59.35%	76.11%	82.86%	86.72%	89.66%
C+UN	58.92%	76.11%	82.70%	86.61%	**90.09%**
C+ST	**59.99%**	**76.27%**	**83.40%**	**86.98%**	89.93%

Table 1: k-best accuracy of various lexical feature sets

Feature set	1-best	2-best	3-best	4-best	5-best
TUF + LIWC2007 Dictionary	56.29%	72.70 %	81.24 %	85.46 %	88.84 %
TUF + MPQA Sentiment Lexicon	56.33%	72.56 %	80.35 %	85.83 %	88.46 %
TUF+ Term Usage	54.55%	72.05 %	80.68 %	85.74 %	88.37 %
TUF + POS tag usage	55.49%	72.51 %	80.21 %	85.04 %	87.95 %
Feat. Extraction Baseline	55.39%	72.47 %	80.44 %	85.27 %	88.13 %

Table 2: Best accuracy using different Lexical features

ST Stemmed n-grams with $n = 1, 2$;

C+UN Collocations and unstemmed n-grams;

C+ST Collocations and stemmed n-grams.

There are several points worth noting from the table:

Stemmed n-grams yields slightly better then unstemmed ones. There are 23465 features in the unstemmed set and 21729 features in the stemmed one. This decrease in feature dimension explains the slight increase, in terms that the unstemmed set is less prone to overfitting while still being able to preserve most of the information.

Collocations are helpful in improving coding accuracy. At first, collocations didn't help when used unstemmed n-grams. However, when used with stemmed n-grams, the combination out-performed the other 3. We suspect this is because the classifier was overfitting the training data in C+UN because of the large size of bigram features. Especially notice there were only 126 collocation features. The increase is even more significant when the small size of these additional features are taken into account[6] .

7.1.2 Task-specific features

As mentioned in **??**, for the feature engineering baseline, using windows of plus or minus one thought units only helped when we were using unigram features. It hurt even when we included bigrams. A reason could be that adding too many features makes it really easy for the model to over-fit.

We tried different approaches to generate the features and to express these to the classifier. First we used the features as word tags. For a given token$_i$ we used its features to obtain "token$_i$_FEATURES". We tried this approach with different features and it failed each time to improve accuracy. The second approach was to use the features of all the tokens in a thought unit as features of the thought unit. To do this, we got all of the features from the tokens in the thought unit and append them at the end of the thought unit as individual features. We appended to the thought unit in two ways. One way was to use a Set, data structure, to remove repetition. In this case, we only cared if the feature was present or not. Another way was to allow repetition. In this case, we cared if a feature occurs n times or not. For many of the features there was no difference in improvement between repeating features and not repeating them. For the features there was a difference the difference was small.

Next we tried to better capture the information about the usage of the different features. To do this, we calculated the average frequency of each feature per thoughtUnit. Then we used this average to decide the usage

[6]In fact, we also ran an experiment using only collocation features and it was able to achieve 47.56% 1-best and 80.07% 5-best accuracy, where we also included another 477 extracted bigram collocations and using a total number of only 603 features

Model	1-best	2-best	3-best	4-best	5-best
MaxEnt	**59.99%**	**76.27%**	**83.40%**	**86.98%**	**89.93%**
Coarser grouping	59.35%	74.99%	82.22%	86.23%	89.66%
Latent	58.44%	75.20%	82.49%	85.70%	89.61%

Table 3: k-best accuracy of coarser grouping and our latent model

Group	Accuracy
Offer	144/167 = 86.23%
Provide information	208/227 = 91.63%
Substantiation	437/447 = 97.76%
Questions	146/207 = 70.54%
Summarizing	16/40 = 40.00%
Threats/power/rights	0/6 = 0%
Reactions	82/93 = 88.18%
Procedural comments	109/155 = 70.32%
Miscellaneous/other	329/525 = 74.67%
Across group	1284/1867 = 68.77%

Table 4: In-group and across-group coding accuracy

type of specific features in individual thoughtUnits. If a thoughtUnit did not have a feature, we did nothing. If the thought unit had feature$_k$ n times, then we annotated the feature as feature$_k$_ABOVE, feature$_k$_BELOW, or feature$_k$_AVG depending on whether n was above, below, or equal to the average occurrence of the feature in the thoughUnits. Using the approaches mentioned we present the best results for a given feature in Table **??**.

1. WT = feature used as word tag
2. TUF = feature used as thought unit feature
3. USG = feature was used as a thought unit feature but was annotated with its usage.

None of the individual features improved accuracy significantly and different combinations of these yielded worse results. We think this is due to the fact that the codes try to capture behavioral information such as making priority statements across two or more issues (IR) or questioning others priorities across two or more issues (QR). This notion of priority would be hard to capture using the individual features of words.

7.2 Coarse-to-fine classification

Using the best performing "C+ST" lexical features, we ran two experiments using our two-level coarser grouping intuition (Section **??**) and latent model (Section **??**) respectively. See Table **??** for accuracy and a comparison with the plain MaxEnt model. Neither of these performed as good as the plain MaxEnt model.

To see the reason, let's first take a look at the in-group coding accuracy when the coarser group is known[7] (Table **??**. One can see that it was significantly easier to predict a label in a coarser group than it was across the group. And in fact, in-group prediction was relatively easy not because the classifier were able to concentrate on smaller set of instances but mostly due to the highly skewed distribution of codes (for example, of 227 test instances in "provide information" group, 170 are coded as "IP"; of 447 test instances in "substantiation" group, exactly 437 are coded as "SBR"). At the same time, we observe that even within the scope of coarser groups, it is still very difficult to tell thought units apart. Together with skewness of in-group code distributions, these two factors led to the observed result, where there was no improvement at all for the coarser grouping model.

Similarly, in the latent model, it was the model itself to infer the grouping. We may consider this part of the model as clustering the data using some underlying distribution. However, the parameters of this distribution

[7]In the actual coding senario, we never know which coarser group the input is in, therefore this is an overly optimistic estimate.

Features	1-best	2-best	3-best	4-best	5-best
LSA (100-dim)	43.81%	59.80%	68.67%	73.83%	77.06%
PLSA (100-dim)	41.60%	56.52%	65.90%	71.90%	75.89%
PCA (100-dim)	41.74%	57.74%	67.21%	72.65%	77.77%

Table 5: k-best accuracy of various feature extraction methods

was yet to be estimated. As a matter of fact, we were asking the model to come up with not only a reasonable clustering but also the corresponding parameters, which was simply excessively too much for such a simplistic model. In addition to this, due to the skewness of the codes, it might not just be so useful even when the model was able to come up with a reasonable clustering. This actually taught us a lesson about latent variable models. If we take a look back at successful applications of such models, we will see most of them, unlike ours, are trying to split current category of interest (e.g. POS tags or nonterminals) into finer-grained subcategories. One major difference between this splitting approach and our merging approach is there is much structural information about the latent variables known when splitting while there is little during merging.

7.3 Feature Extraction

Table **??** shows the empirical results with features extracted using latent semantic analysis (LSA), probabilistic LSA, and PCA. The 1-best accuracy with the original features of dimension 9908 was 55.60%. We observe that the accuracy with all three methods is worse than with original features. We also tried experimenting with higher dimensional features in latent space but results were not much different. With 400-dimensional LSA features the 1-best accuracy was 44.61%. Hence, we report only the results on 100-dimensional features. There can be a couple of reasons why we get deterioration in performance after dimensionality reduction. First, it can be due to nature of this data. There are large number of classes and there can be words that occur less frequently but are crucial in discriminating some of the classes. These can be treated as noise and could have been filtered out in all methods (LSA, PLSA, PCA). Second, these are all unsupervised methods and are not optimized for class separability. However, the models trained on these features do not overfit as expected. The training accuracy for all these extracted features lie in the range 65%-75% that is a better estimate of generalization error than 90+% training accuracy that we get with models trained with original features.

8 Individual Contributions

We discussed the possible approaches we wanted to try on this problem and divided these among ourselves.

Abhishek focused on trying out probabilistic latent semantic analysis (pLSA) and principal component analysis (PCA) on the data for feature extraction.

Raul focused on feature extraction and alternative preprocessing methods

Ke focused on preprocessing, n-gram and collocation features, and implementation of our latent model.