Converting In-N-Out Orders into a Structured Form

Michael Gummelt, David Brody

June 3, 2009

1 Abstract

In this project, we hope to convert written (text) fast food orders from In-N-Out into a normal, structured form. Our approach is to build off of PA3, using a Maximum Entropy Classifier for Named Entity Recognition, and then apply a PCFG Parser to convert the order into a normalized, tree form. We also developed an additional stage consisting of rule-based conversion into a custom intermediate language existing between the NER stage and the parser stage in order to help with parsing.

We found that we can acheive good results with a relatively small corpus of orders. Orders are made up a small vocabulary, thus a small corpus can effectively train a classifier. However, there are ambiguities in a subset of the orders which can result in incorrect conversion. Further work in automatically asking the customer questions that can resolve these ambiguities could help.

2 Introduction

Our approach to converting In-N-Out orders into a structured form involved three main steps:

1. **Maximum Entropy Markov Model for Named Entity Recognition**
2. **Conversion into an intermediate language to assiste the parser**
3. **PCFG Parser for converting a flat list of tokens into a tree order**

However, a huge hurdle in approaching this project was first acquiring a sufficient corpus of In-N-Out orders. As far as we know, no such corpus previously existed, thus we performed our own data collection as described below. We decided that doing voice recognition to extract the text of the order was too ambitious, so the initial input to our program is a text-based English order. Luckily, In-N-Out orders tend to be made up of a small vocabulary, thus we can acheive decent results with our final corpus of 204 orders. However, these orders can often be ambiguous, and we have found that careful feature extraction in the Maximum Entropy Classifier is very important for the success of the entire program.

3 Data Collection

No previous corpus of fast food orders existed so we set out to collect one. Our first idea was to approach In-N-Out, and ask them if we could affix a voice recorder to their drive through box in order to collect the orders they receive throughout the day. Of course, this didn't go over too well since they were worried about legal implications and all that stuff companies have to be worried about.

We approached McDonald's and received a positive response, but ultimately decided against using their menu due to its complexity. In-N-Out's menu is much simpler, and thus our classifier can be trained with much fewer orders.

Our solution was to set up a mock In-N-Out register outside of dining halls on Stanford's campus and ask students for their typical In-N-Out orders. We recorded their orders with a voice recorder, and later manually transcribed their orders, keeping all filler words such as "Uhm", "Yea", etc. We had an In-N-Out menu printed out and on display so that students could order anything. Students were, though confused at first, very willing to help out, and we ultimately received 204 orders after visiting 3 different dining halls.

We then manually created gold results for each order, for all three stages of conversion (tagging, intermediate language, parsing). We also defined a custom F-Score function for the intermediate language so we could track our progress on each stage of conversion as we were developing our program.

An example of the raw input and the final tree is given below.

4 Overview of Process

4.1 Summary

Conversion of an order goes through the following stages:

1. **Original Order**

   Can I have two double doubles with no onions no tomatos and a strawberry shake

2. **NER tagging**

   o o o o quantity object object o sub sub sub sub and quantity type object

3. **Intermediate Language**

   2 BURGER 2_MEAT 2_CHEESE TOMATO_NONE ONIONS_NONE AND 1 STRAWBERRY SHAKE

4. **Parsed Tree**

   (
   (OBJ

```
  (QUANTITY 2)
  (TYPE BURGER)
  (ATTR_MEAT 2_MEAT)
  (ATTR_CHEESE 2_CHEESE)
  (ATTR_TOMATO TOMATO_NONE)
  (ATTR_ONIONS ONIONS_NONE)
 )
 (SEP AND)
 (OBJ
  (QUANTITY 1)
  (ATTR_TYPE STRAWBERRY)
  (TYPE SHAKE)
 )
)
```

4.2 Reasoning

Maximum Entropy Markov Models and PCFG Parsers are often used for part of speech tagging and sentence tree parsing. Fast food parsing has many similarities to this problem which led us to choose to use a MEMM and a PCFG parser, however, there are major differences between Part-Of-Speech parsing and fast food parsing.

One similarity between the two problems is that given a list of tokens, we have to create a tree. However, where in the case of sentence parsing we can simply feed in the original sentence or possibly the chunked sentence into the parser, the elements in our fast food tree are normalized across several different types of orders. That is, orders such as "cheeseburger", "burger", and "double double" are all instances of a BURGER, and thus we would want to pass in "BURGER N_MEAT M_CHEESE" to our parser, where N and M specify the type of burger. This way, customers can phrase their orders in several different ways, and the result is a normalized, machine readable format. See the Appendix for a lexicon conatining a complete list of tokens that can be passed into the parser, based on the In-N-Out menu.

Thus, we created an intermediate language out of these tokens that has to be generated before we can send input into the parser. The raw input is used to generate this intermediate language, but first we have to group similar words so they can be used to determine the intermediate language. For example, the words "hold the tomatos" must be grouped together so they can create the intermedate token NO_TOMATOS. We use a MEMM to tag these words, and then all adjacent words with an identical tag are chunked together and analyzed to create the intermediate language. A complete list of all types of tags is given below. The intermediate language is structured enough to ge completely rule based, and it is described below.

5 Maximum Entropy Classifier for Named Entity Recognition

The Maximu Entropy Classifier is used to tag the raw input into discrete classes, much like POS tagging, or the medical tagging done in PA3. However, the tags we use are meant to chunk similar

words together so the chunk can later used to translate the sentence into an intermediate language.

5.1 Tags

The following is the set of tags we used on the raw input order, along with examples of how they can match words:

add	- "grilled onions", "onions", "tomatos", etc.
and	- "with", "and", etc.
with	- "with", "and", etc.
edit	- "instead of the drink", "on the burger", etc.
object	- "cheeseburger", "fries", "milkshake", "double double", etc.
quantity	- "a", "an", "three", "twenty seven", etc.
size	- "small", "medium", "large", "extra large", etc.
style	- "animal", "protein style", etc.
sub	- "without pickles", "no tomatos", etc.
type	- "chocolate", "vanilla", etc.
o	- "Can I get", "can I have", "yea", etc.

1. **Can I get three double doubles with no tomato and a large root beer**
2. **I'll have a double double animal style with no pickles and an animal style cheeseburger with no pickles**
3. **We want three triple triples and an extra large coke**
4. **I'll have a double double animal style with grilled onions and french fries medium chocolate shake and ice water**
5. **Can I get a four by four.**
6. **Can I get a number one combo only with a medium chocolate shake instead of the drink and onions on the burger**

Note that the word "with" can be tagged as "and", and the word "and" can be tagged as "with" depending on the context. Any word used to separate objects in an order is tagged as "and", and any word that modifies a previous object is tagged as "with".

5.2 Feature Extraction

The F-Score algorithm we use for evaluating the success of the tagging stage of the program is the exact same algorithm as used in the NER evaluation in PA3.

Most features only make for small, incremental improvements. However, the improvements become larger when the size of the training corpus is smaller.

All testing was done on a test set of 40 sentences, containing a total of 557 words.

The following features were added incrementally:

1. **Use the word as a feature** - This is our baseline test, with only the word itself being used as a feature. We get pretty good results with just this. This is most likely due to the fact that most words are only observed to be part of a single class ("milkshake" is always an object): F=81.52

2. **Previous Label** - Previous label is now a feature F=85.38

3. **Previous Word** - Previous word as a feature. This didn't help much at all which we found as surprising since many word pairs are often found together. However, it is already conditioning on the previous label, so this might not be providing much more information. F=85.93

4. **Next Word** - Next word is a feature. This is a big jump (5pairs appear together, so the adjacent word is a good indicator of the current word's type. F=91.01

5. **Matches Quantity Words** - At this point there were several ad-hoc quantity words that were not being tagged correctly because they were not seen in the training data (ex. "twenty seven"). So we created a binary feature that is true when the word represents a number. This resulted in a pretty decent boost. F=93.3

6. **Sentence Position** - We tried adding relative sentence position (beginning, middle, end) as a feature, but the F-Score actually dipped down a bit, possibly from overfitting. F=92.44

7. **Between two objects** - Our connectors ("with" and "and") were not always being tagged correctly, so we added a binary feature that is true when there are objects to the left and right of the word. This would have been an indicator of an "and". Surprisingly, this didn't help. F=92.44

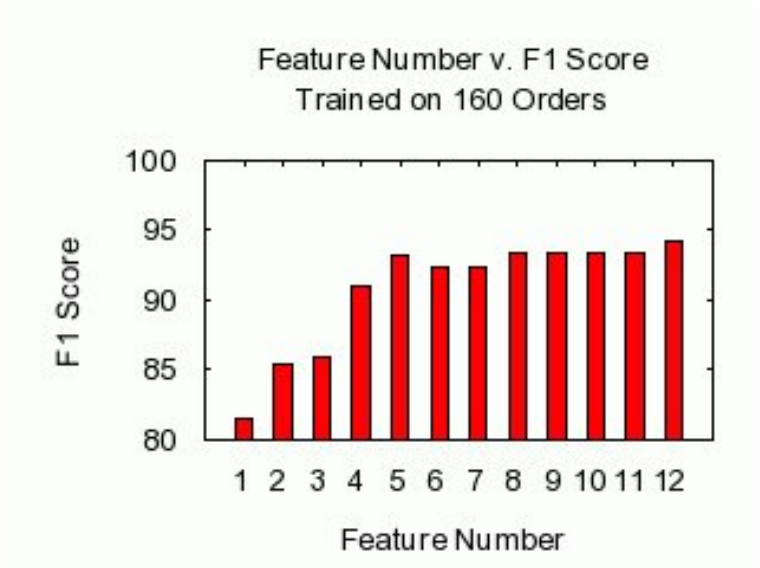
8. **"number" or "combo" appears before it** - Again, to try and tag the "with" class correctly, we added this binary feature because words tagged as "with" almost always come as part of a combo meal order. F=93.88

9. **"number" or "combo" appears after it** - The previous feature helped, so we thought this might. It didn't. F=93.88

10. **"on the burger" appears before it** - This is a common phrase indicitive of an object (burger) that needs to be edited, we we thought this might help tag the "edit" class more closely. It didn't. F=93.88

11. **"on the burger" appears after it** - Similar Reasoning. No improvement. 93.88

12. **"Matches Size Words"** - The size words were not always being tagged correctly, so we added a binary feature that is true when the word matches likely size words (small, medium, large), which helped a bit F=94.17



Our corpus is small, but due to the small vocabulary utilized in fast food orders, even a little bit of data can go a long way as the following two data points show:

training on 80 sentences - F=86.55
training on 160 sentences - F=94.17

5.3 Error Analysis

The Errors which we recognized and made an attempt to fix are described in the previous section on feature extraction. However, there are still existing errors in the tagging phase.

Our poorest performing class is the "add" class with an F-Score of 75.00. It fails on cases such as:

I'll have a double double grilled onions and no tomatos.

Where "grilled onions" should be classified as an "add" but is not. This seems to be because the modifier (grilled onions) comes after the object (double double) with no separator, making it hard to tell if it is an add, subtract, or other. Adding features that check and see if the previous word is likely an object word don't seem to help.

Most other errors are ad-hoc sentences that, if we had significantly more training data, we may be able to recognize, but currently we don't. Examples:

Yea can I have a cheeseburger with nothing on it

We don't have any cases of "with nothing on it", so while this is supposed to be tagged as a "sub", it gets tagged as an "o"

I'll have the double double animal style with grilled onions and french fries medium chocolate shake and ice water

In this case, "water" has been seen before as an object, but "ice water" hasn't. So neither gets tagged as an object because the word "ice" throws off the tagging for the word "water"

Can I have one double double without onions

In this case, "without" does not get tagged as a "sub", which is extremely surprising, because "without" appears multiple times in training, and each time it is tagged as a "sub". Further investigation is needed to find the root cause of this error.

6 Intermediate Language Translation

Now that we have each word of our original input tagged as 1 of the 11 classes we have identified, we must convert the stream of English words into a stream of standard tokens usable by the parser.

We Have:

Um/**o** sure/**o** can/**o** I/**o** get/**o** a/**quantity** hamburger/**object** and/**and** an/**quantity** iced/**object** tea/**object** with/**with** onions/**add** on/**edit** the/**edit** burger/**edit**

We Need:

1 BURGER 1_MEAT 0_CHEESE 1 DRINK ICEDTEA WITH EDIT ONIONS_RAW BURGER

You can reference the Appendix for a lexicon with a full list of all possible tokens in the vocabulary of our intermediate language.

6.1 Method

Due to the relatively small number of possible Intermediate Language tokens as well as the small vocabulary customers use when ordering fast food, we have found that the most effective (and most easily implemented) method for generating the intermediate language is a rule-based approach.

The generation of the Intermediate Language could not be accomplished by a Maximum Entropy Classifier because not all rules are one-to-one. The following are examples of some rules:

Some are one-to-one as in:

"coke" → "COKE"

Some are one-to-many as in:

"cheeseburger" → "BURGER 1_MEAT 1_CHEESE"

Some are many-to-one as in:

"number three" → "NUMBER_3"

and Some are many-to-many as in:

"double triple" → "BURGER 2_MEAT 3_CHEESE"

We do not list all possible replacements here. Refer to the method getOutputTag in MaximumEn-

tropyClassifierTester.java to see a full implementation.

6.2 Chunking

In order to translate an entire sentence into the intermediate language, we first chunk the sentence into adjacent word chunks that have the same tag, and then feed in the list of chunked words along with their common tag as input into a function that returns a list of intermediate language tokens.

So, in the example above, the chunked input along with their resultant output would be:

(Um, sure, can, I, get) - "o" → []
(a) - "a" → [1]
(hamburger) - "object" → [BURGER, 1_MEAT, 0_CHEESE]
(and) - "and" → [AND]
(an) - "quantity" → [1]
(iced, tea) - "object" → [DRINK, ICEDTEA]
(with) - "with" → [WITH, EDIT]
(onions) - "add" → [ONIONS_RAW]
(on, the, burger) - "edit" → [BURGER]

These outputs are then concatenated and fed into the parser for parsing.

6.3 Error Analysis

In order to build the rule-based intermediate translator, we iteratively added rules to match common input. An example of a simple rule is that any chunk labeled as an "object" that contains adjacent numeric words such as "double single" or "triple double" must be translated to BURGER N_MEAT M_CHEESE where N and M match the numeric words.

We developed an F-score method for evaluating the success of the intermediate language translation.

Given two lists (the correct tokens and the guessed tokens), we line up each words with the same index in each sentence with each other.

Precision: $\dfrac{\text{\# Guessed Tokens matching Correct Tokens}}{\text{\# Correct Tokens}}$

Recall: $\dfrac{\text{\# Correct Tokens matching Guessed Tokens}}{\text{\# Guessed Tokens}}$

F-Score: Geometric Mean of Precision and Recall

Our average F-Score for Intermediate Translation on a test set of 40 sentences and a training set of 160 sentences was F=89.75

Errors in the first, tagging stage of the program propogate into errors in intermediate language translation, lowering the F-Score. In fact, these are the only observed errors in the intermediate language translation. That is, if any sentence is normal enough to be correctly tagged by our

Maximum Entropy Classifier, it is normal enough to be recognized by our rule-based intermediate language translation. However, the intermediate translator cannot recover from tagging errors. Consider the following case:

original sentence: Yea can I get a burger with nothing on it
correct tags: (o, o, o, o, quantity, object, o, sub, sub, sub, o)
guessed tags: (o, o, o, o, quantity, object, o, add, add, add, o)

The intermediate translator will correctly translate each chunk except for the incorrectly tagged "add" chunk, which it would not have a rule for.

guessed tokens: 1 BURGER 1_MEAT 0_CHEESE
correct tokens: 1 BURGER 1_MEAT 0_CHEESE SPECIAL_NOTHING

7 PCFG Parser

7.1 Implementation

Once the order is converted into the intermediate language it then proceeds to a PCFG parser which attempts to build the correct order tree. This process takes the input sentence (the intermediate language) as the leaf nodes and attempts to fill the rest of the tree with the most probable parse of such word order. Along with a specialized intermediate language, the menu also has specialized tags:

OBJ	for objects
OBJ_EDIT	for orders which backtrack and edit previous menu items
GROUP	for groups (typically meal numbers)
GROUP_SUB	for referencing a subgroup of multiple objects

Together with the intermediate language, these tags form the node labels of the output tree from the PCFG parser.

In this project the PCFG is used in a different fashion than normal. A majority of the rules only have one possible grammar which makes this grammar significantly less ambiguous than normal language. In looking at the parse table with the probabilities of different grammars, a majority of words have only 1 probability that is nonzero making it unambiguous. The outcome of this is a very reliable text parser which spits out almost an XML like tree. However, this also creates a few difficulties. Since everything is almost a 1-1 relation, if a word isnt found in training, it will result in a completely failed parse. Furthermore, if a word isnt seen in training that is used in testing, the grammar will often fail since there is no grammar going from the new tag to another one. However, since our language tags for this PCFG parse is define and finite, we can ensure that this does not happen.

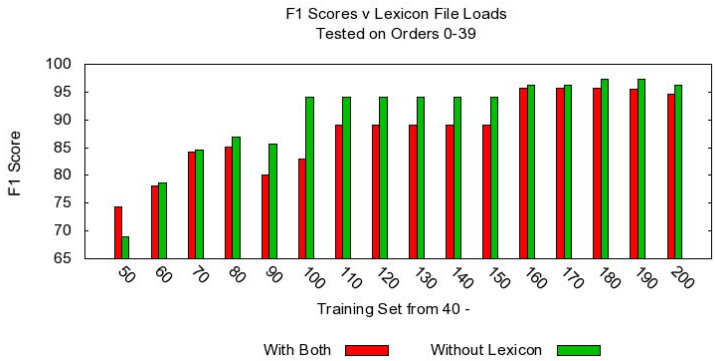
7.1.1 Lexicon

The first problem comes from order which use attributes or menu language that has not been seen in training. This results in it missing from the lexicon and thus very unlikely it will get the correct tag. The solution to this comes from defining a menu file which contains our finite intermediate language. The lexicon first loads this files to create a base probability for each word that is uniform and then through training slowly adjusts the probability to be more accurate. The layout of this menu file (See Appendix) is to allow easy addition and editing of the menu. This file begins with definitions of variables to allow easy updating and maintenance of elaborate menus. Variables defined with DEFINE: and are used by doing *variableName*. After the defines, it then proceeds down the list associating all the words in the list with the word at the beginning of the line. For example:

TYPE: BURGER, FRIES, SHAKE, DRINK

sets the default lexicon to associate BURGER, FRIES, SHAKE, and DRINK with the TYPE tag.

However, after running some tests with the new Lexicon reader in place, we found it actually did better without it. While reading in the lexicon helps if the training set is very small, since not all cases are covered, it starts to hurt performance as the training set increases. Although this seems like it shouldnt be the cause since a type is always one specific tag, loading the Lexicon seems to make performance worse. After some investigation the cause of this is partly caused because the trees are binarized. This creates different tag labels aside from menu items which have separate, non-uniform probabilities. Thus, by setting this probability initially to be uniform, it would take a large amount of training to get it to flatten out. For a more detailed understanding of this result, further investigation is needed.

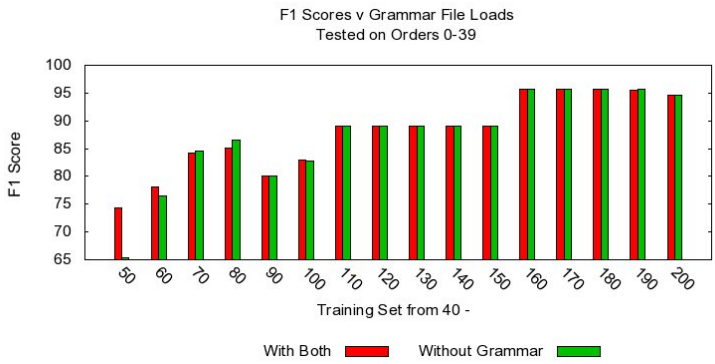


7.1.2 Grammar

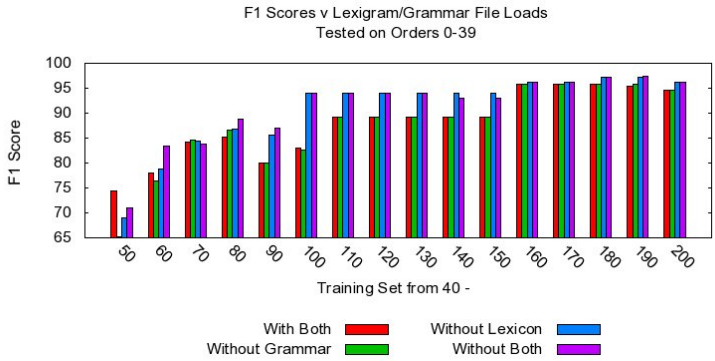
The next problem comes from orders which use different ordering and structures than other training orders. This results in a missing grammar rule in the PCFG parser which usually results in a completely incorrect parse since each tag has very few or only one rule. The solution to this comes from defining a menu grammar file which contains a very limited unary rule implementation. The grammar loads this file to create a uniform base probability for each grammar which then gets modified as the grammar is trained. The layout of this grammar file (See Appendix) is created to

allow easy addition and editing of grammar rules. The file begins with definitions of tag groups and then has a list of unary grammar rules.

Although in theory this sounds like a good idea to set base probabilities, in practice we found the results to be contrary. It showed that after we have a decent amount of training data the preloaded grammar rules seem to hurt the F1 score. Although this seems like it shouldnt be the cause since most tags have one grammar rule, loading the grammar seems to make performance worse. After some investigation the cause of this is partly caused once again because the trees are binarized. This creates different grammar labels aside from tag items which have separate, non-uniform probabilities. Thus, by setting this probability initially to be uniform it takes more training to get it to flatten out. For a more detailed understanding of this result, further investigation is needed.



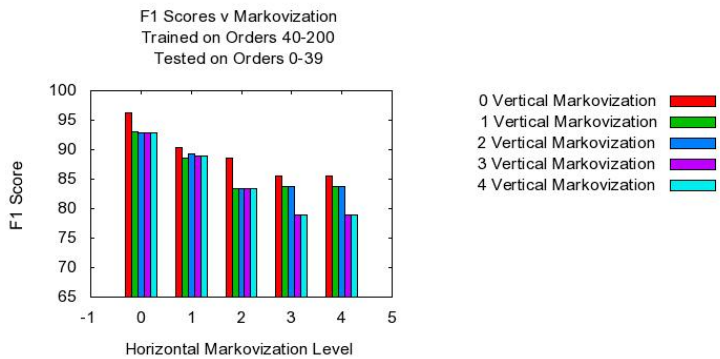
7.1.3 Visualized Effects of Lexicon and Grammar



7.2 Configuring

Once the initial PCFG parser was created, optimized, and specialized for our purposes, we then had to go through some trial and error sequences to see which method and numbers worked the best. Our first test was to see if and how markovization could help our results. We tested our data using various numbers of horizontal and vertical markovization and came to a strong agreement to

use h=0 and v=0. We set v=0 since most order trees are relatively simple with only a dependence of up to one parent and since our grammar covers these cases since it is mostly one-to-one, having v ¿ 0 overfits the data and causes our F1 score to drop. Similarly, we set h = 0 since most order tree items simply depend on their parent and not their sibling. While certain attributes can only be applied with certain type tags, since the tags can be in any order , having an h¿0 overfits the data as well and causes F1 scores to drop.



7.3 Error Analysis

Once our training size got large enough we were able to consistently get 95GROUP_SUB but that is simply because it does not appear in the training set. Otherwise the parser gets an F score of 100 on all sentence with correct input from the sentence labeler.

8 Discussion

8.1 Running the Program

In order to run our program:

./ant
./run

Will train both the MEMM and the PCFG parser on 160 orders, and test on 40 sentences. Output contains the following information for each test sentence:

1. test sentence
2. guessed sentence tags
3. correct sentence tags
4. guessed intermediate tokens

5. P,R, and F score for the intermediate language translation
6. guessed parse tree
7. correct parse tree
8. P,R, and F score for the parse tree

At the end of the output for the individual test sentences, there will be an average P,R, and F score for all parse trees. This is the main indicator for the overall success of the program.

After the test results are displayed, you will be prompted to enter an order to be parsed. Note that written orders can be quite different than spoken orders, so the result might not be too accurate.

Our final program produces the following scores:

P: 88.24
R: 80.65
F: 84.27

8.2 Conclusion

Our final F-Score of 84.27 gives some idea of the success of converting an English order into a normal form tree. However, this evaluation is somewhat inaccurate. The majority of sentences (75.00are parsed completely correctly. However, sentences that are parsed slightly incorrectly (missing a single, small branch) are often assigned very low F-Scores (0-20almost exactly the same, it is evaluated as almost completely wrong. Thus, through these observations, we conclude that parsing English In-N-Out orders into a normal form tree can be done very accurately due to the small vocabulary and repeated structures used in fast food orders.

9 Further Work

9.1 Alternate Methods for Intermediate Language Translation

Our process for developing rules for intermediate language translation was rather ad-hoc. We identify word combinations that often appear in orders, and develop general translation rules for those combinations. For the simple menu used by In-N-Out, this solution works very well, with almost no errors in the intermediate language translation. However, for a more complicated menu, alternative methods could be used.

One method could be developing a normal form, or a configuration file, for defining translation rules. However, the general solution, and what we feel could be the best solution, is implementing a language translation between English and the Intermediate Language. A larger corpus than our 200 order corpus would probably be necessary to develop an accurage language model, but we believe language translation could work very well for translating into the Intermediate Language.

9.2 Resolving Ambiguities Through Intelligent Interrogation

Certain sentences are unavoidably ambiguous:

I'll have a burger animal style fries

Does the customer refer to the burger or the fries when she says "animal style"? This ambiguity could be resolved by an intelligently chosen disambiguation question such as "Would you like the fries or the burger animal style?" (We actually tested this ambiguity by placing this exact order, in monotone, at In-N-Out, and the cashier asked me this exact disambiguation question). The response to to this question could then be used to disambiguate the order.

Thus developing a bank of disambiguation questions which could be used to augment the sentence might increase performance.

We manually implemented this during our data collection to some extent. For some customers, we asked "Would you like onions on that?" or "Would you like grilled onions on the burger?" after there order, and, depending on their response, concatenated "with onions on the burger" or "with grilled onions on the burger" at the end of their order.

10 Appendix

10.1 Lexicon

DEFINE: N = 0-100
DEFINE: M = 1-3
DEFINE: AMOUNTS → NONE, REG, EXTRA

//Define Seperators
SEP: AND,WITH

//Object types
TYPE: BURGER, FRIES, SHAKE, DRINK
QUANTITY: *N*
NUMBER: NUMBER_*M*

//Object sub types
ATTR_TYPE: VANILLA, STRAWBERRY, CHOCOLATE, NEOPOLITAN, COKE, SEVENUP, ROOTBEER, DRPEPPER, LEMONADE, ICEDTEA, MILK, COFFEE, WATER
ATTR_MEAT: *N*_MEAT
ATTR_CHEESE: *N*_CHEESE
ATTR_STYLE: ANIMAL, PROTEIN
ATTR_SIZE: SMALL, MEDIUM, LARGE, X_LARGE
ATTR_ONIONS: ONIONS_*AMOUNTS*, ONIONS_GRILLED, ONIONS_RAW
ATTR_TOMATO: TOMATO_*AMOUNTS*
ATTR_PICKLES: PICKLES_*AMOUNTS*

ATTR_LETTUCE: LETTUCE_*AMOUNTS*
ATTR_SAUCE: SAUCE_*AMOUNTS*, SAUCE_REPLACE, SAUCE_KM, SAUCE_KETCHUP
ATTR_SPECIAL: SPECIAL_NOTHING, SPECIAL_WELLDONE, SPECIAL_EVERYTHING
ACTION: ADD, REM, EDIT, CLEAR

10.2 Grammar

GROUP ATTRS
  TYPE
  ATTR_TYPE
  ATTR_MEAT
  ATTR_CHEESE
  ATTR_STYLE
  ATTR_SIZE
  ATTR_ONIONS
  ATTR_TOMATO
  ATTR_PICKELS
  ATTR_LETTUCE
  ATTR_SAUCE
  ATTR_SPECIAL
END GROUP

ROOT→OBJ=0.7, GROUP=0.25, OBJ_EDIT=0.05

OBJ→GROUP_SUB=0.05, ATTRS=0.95

GROUP_SUB=→ATTRS=1.0

GROUP→OBJ_EDIT=1.0

OBJ_EDIT→ATTRS=1.0