ALIGNMENT TREES

Vishesh Gupta (vishesh) Assisted by Professor Martin Kay

ABSTRACT

Alignment Trees, based on DeKai Wu's work on Inversion Transduction Grammars, offer a new way to better choose alignments in a traditional translation problem. Here we see that a classifier can be made to detect differences in languages with F1 score of ~93% solely based on features extracted from the trees, rather than a more usual tree-grammar style approach taken by previous authors. When this classifier is used as a feature in a phrasal translation model, it shows a 2 point increase in BLEU score from a baseline of 15.000 to a final score of 17.011.

MOTIVATION

The primary problems in translation have always been selecting the appropriate words to translate to (an issue of dictionary), and then rearranging them in the most fluent way possible (the issue of alignment). Of the two, Alignment Trees deal with the second.

Trivially, the ideal rearrangement is one of the possible permutations of the sentence. However, considering all of the permutations quickly becomes unwieldy, mostly because there are too many variations of sentences and too many permutations to handle.

However, DeKai Wu and other researchers have noted that "related information stays together" in previous research. For example, if there is "red car" in english, it might become "autombile roja" in spanish, but the word for red and the word for car would never be very far from each other. What this suggests (and follows from our own intuition of grammar) is that there is some tree-like structure to natural language, and that some of those subtrees have the same leaves, but in different orders. In that case, it should be possible to map one grammar tree to another grammar tree (or the corresponding subtrees) and learn when those trees are similar or different. This is generally called tree-based translation, and there are many algorithms for it on the moses tree-based translation page

DeKai Wu et all concieved of a tree of operations that map the order of the source language to that of the target language. His intuition was that these mappings could be produced by simply reordering each pair of words, with two operations - N and R, where N preserves order and R reverses the order.

As an example, consider an alignment problem with the sentences "I went home" and "Mein ghar gaya" (phoneticized Hindi, translated to I home went). Now, the appropriate tree here would be `(N I (R went home))` to transform the original ordering of the words to the new ordering in hindi. There are still issues with this approach, such as spurious words, and phrase alignments among other problems, and those are discussed below.

HOW TO

Alignment trees are made by taking a parallel alignment problem such as

```
english: The Government will help develop customer satisfaction indices ,  and encourage their inclusion in existing and future benchmarking s
hindi: सरकार सेवाओं से उपभोक्ताओं की संतुष्टी नाँपने के पैमाने विकसित करने और इनको  वर्तमान तथा भविष्य की बैंचमार्किंग स्कीम्ज में शामिल करने के लिए मदद देगी  ।
alignment: 1-0 2-24 2-25 3-24 3-25 4-9 4-10 5-3 5-4 5-5 6-3 6-4 6-5 7-6 7-7 7-8 9-11 11-12 12-20 12-21 13-19 14-13 15-14 16-15 17-17 18-18 19-
```

And produce a tree like

```
(:N
 (:N
  (:N
   (:N
    (:R
     (:R
      (:R
       (:N 10 11)
       (:N 8 9))
      7)
     (:N
      (:N
       (:N
        (:N
         (:N
          (:N 0 1)
          2)
         3)
        4)
       5)
      6))
    12)
   13)
  (:R
   (:R
    (:R
     (:R
      (:R 23 22)
      (:N 20 21))
     (:N 18 19))
    17)
   (:N
    (:N 14 15)
    16)))
 24)
```

Here's the flattened (simplified) version of this tree:

```
(:N
 (:N
  (:R
   (:R
    (:N 10 11)
    (:N 8 9)
    7)
   (:N 0 1 2 3 4 5 6))
  12
  13)
 (:R
  (:R
   (:R
    (:R 23 22)
    (:N 20 21))
   (:N 18 19)
   17)
  (:N 14 15 16))
 24)
```

This is clojure syntax (edn format) for the tree where :N is the normal operation and :R is the reverse operation. The best way to think about how this tree is generated is to see it as a series of operations applied in order to reduce the tree in passes. First, we reduce all "normal" sequences - any node in the tree that is contiguous is joined with the adjacent ones. When that is done, all the remaining nodes are processed as other "special" operations. Mostly, this looks like a 'R' operation, but as shown below, there is occasionally a need for higher order operations. The idea behind the tree is to represent the operations it takes to make the scrambled word order transform back to normal.

It may be easier to see this as a operation tree as an actual tree online. As you can see, taking the leaf nodes in their original order (as given on the right of the picture) and then applying the operations as you go up the tree would take the out of order leaf nodes and put them back in order 0-whatever. So this tree represents a series of operations that can reorder a tree from one language to another.

It's much easier to visually understand if you go to the website. Pressing right/left moves you around the different sentences, and you can choose between french, hindi and swedish.

One thing to note is that the only required input to the tree-generation is the alignment string. In other words, this problem can take a string like '0-0 1-10 2-3 4-9 ...' and output a tree of operations. From here on out, we will only showcase those strings and trees, not deal with the original words of the sentences. Part of the point of this project is to be completely agnostic to other grammar structures of the language (eg, part of speech tags, or an actual parsed grammar tree).

Another visual presentation of this is given in these slides Play through slides 5-8 to get a picture of how an alignment produces an alignment tree.

INCONSISTENCIES

As it was stated before, there are some issues with generating these trees. Here's a list of them, and how we got around the problems.

ONE-TO-MANY MAPPING

Here we'd get a string like '1-1 1-2 1-3'. The way to think about this is that some word in one language maps to a phrase in another language. In the situations where there was a contiguous mapping (like in the example given) it was easy to just shrink down the phrase in the target language to one terminal (let's just replace 1,2,3 by 1) and then you'd have '1-1' in the alignment string. This is pretty much the intent of a contiguous one-to-many mapping anyway - the three words it produces are pretty much one entity.

This gets more confusing with things like split infinitives, when you might see an alignment like '1-1 1-3 2-2'. Now it's impossible to "shrink" the mapping since the phrase is split around another word. The solution we came up with is to duplicate the original word (ie., the two 1's are separate entities), which would give you a tree like `(N 1 (R 3 2))`

For the purposes of identifying these numbers differently, we would uniquely number the 1's. So the new string would be '1-1 2-3 3-2', and the corresponding tree would be `(N 1 (R 3 2))` with no hassle. This seems strange, but it's actually a well known tactic in the generative stories of IBM model 3 with its fertility parameter.

VANISHING WORDS (ONE-TO-NO MAPPING)

If the string looks like '1-1 3-2', 2 doesn't go anywhere. The solution we came up with was to copy the previous word's alignment (i.e fold the non-producing word left).

In many languages (english included) it actually makes more sense to fold right (hindi is a good example - heads always come after their modifiers. This is also true of english (i.e, the red car -> lal gadi, but you'd want to fold 'the' right so that it grammatically comes under the clause it's actually part of)). It really depends on which word is being left out of the mix.

We actually don't create a new terminal in our implementation, and just renumber the old terminals. So we get '1-1 2-3' -> '1-1 2-2' (and we pretend that word #2 doesn't exist.) This is an equivalent result of generating an extra terminal and then folding it into the original word without actually doing any of that. The benefits of this strategy are that extraneous null alignments don't clutter the data (ie, there aren't extra fabricated `N` or `R` nodes in our tree).

However, this particular strategy does remove linguistic information about null alignments. Since we were only dealing with the final trees, we felt this was an appropriate choice.

It would be helpful to know that, for example '[a|an|the] noun -> noun' in hindi in almost all cases (hindi doesn't have articles, only specifiers), if this could be implemented as a special kind of operation.

MANY TO ONE MAPPINGS

This is the opposite of one-to-many.

In this case, we duplicate the terminal in the target languge, and get '1-1 2-1 3-2' -> '1-1 2-2 3-3' and it parses to `(N 1 (N 2 3))`. The same tradeoffs are present in this direction with respect to the generative story that's being generated. The only effect this has is that it dilutes other operations in the tree, and makes it seem more heavily trained towards N operations.

MATERIALIZING WORDS

This is the target side version of the vanishing words. Again, we just fold the non-aligned words into the previous terminal. So '1-1 2-3' would become '1-1 1-2 2-3' which would become '1-1 2-2 3-3' per previous rules. We end up with `(N 1 (N 2 3))`.

HIGHER ORDER OPERATIONS.

Try parsing `2 4 1 3` with only N and R operations. You can't. (nothing is contiguous in this case).

The solution to this is to allow higher order operations where needed. They tend to be rare ('2413' and '3142' are the only two length-4 ones, there are 6 length-5, and 46 length-6 ones). This corresponds to the sequence A111111 in the integer sequences dictionary, and it actually has a very beautiful mathematical interpretation as A(n) = # of simple permutations of length n. This makes perfect sense, as one would have to consider the entire sequence's permutation to reverse it back to the original order for each simple permuation.

The next few numbers in the sequence are 338 and 2926. It's really hard to imagine that our minds are dealing with 2926 different permutations of things when we construct sentences, rather than dealing with 46. We cut off the operations at order 6, and anything larger, we attribute to a faulty alignment. This assumption is actually quite generous; most of the time, even seeing a 4-operation on any non-phrase related translation indicates that the alignment string has some problem, and closer inspection of the translation sentence pair generally reveals that there is a more fluent way to organize the translations

PHRASE TRANSLATIONS

Unfortunately, there are lot of shortcuts taken in the data, and one of them are wholesale phrase translations. There are a lot that aren't "minimal", and they tend to contain more words than they need to. Here's a good example, if you look at the end of the sentence. It's non minimal because there are a lot of direct translations you could map out of that set, and because 'in' is mapped to two separate phrases next to each other. This example is on the stranger end, and parsing this alignment wasn't feasible.

An example alignment string for phrase translation might look like '1-1 1-2 1-3 2-1 2-2 2-3'. Here a series of two words in the source maps to a series of three words in the translation.

The solution to this issue is to collapse the phrases on both sides to a single terminal. So we get '1-1' only in this case, since it's one phrase to one phrase. Again, this does change the number of terminals in the final sentence, but for the purposes of the grammar tree, any phrase is really one unit, and it doesn't make sense to count all the phrases' individual words as being separate terminals, since that doesn't add any more information to the final tree. There's no grammatical reason that those three words are being "grouped", it's really more of an issue of dictionary translation, and reordering in general deals more with what to do with the dictionary elements rather than how to create them in the first place.

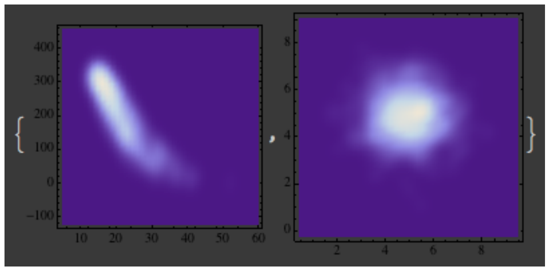
CLASSIFICATION

(TA Note: The *classification* portion of this project (everything after the feature table to having a trained SVM model) was done for the CS221 class. I've included this section to mostly summarize the results.)

After implementing a parser that could generate these trees, the next step was to train a classifier. To that end, we extracted many features (which are discussed more in the next section) and generated a training table. From this point on, the issue of detecting which language (if any) was a machine learning problem. The job of the classifier was to give a probability that a given alignment string was in the desired language. For example if we gave the string "1-1 2-2 3-3", and language "french", we would expect some number close to 1.

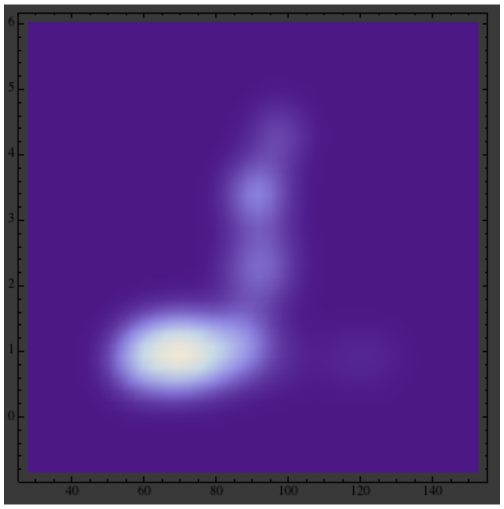
Our first attempts ended in failure, as we tried to do a traditional classification where we expected every point in our feature extracted space to be given one label. As expected, the classifier tended to do very poorly, because it would try to force a single point that had many possible language classifications to only have one label, and as such fail on a large swath of other language examples for which we expected the result to be positive as well.

The next attempt was with GMMs and other clustering models. These also tended to do very poorly since the data was very lattice based - there were many points on the EXACT same location in the space, and therefore we were trying to model what was effectively discrete data with a continuous distribution. The other issue was that the data wasn't Gaussian in the least - it had some other bizarre shape. The easiest way to tell this is to look at a centrality-density plot.



centrality-density plot of a gaussian

On the right is a gaussian distribution in 2D. You can see the telltale high density center and fringe, as expected. Notice the shape of the centrality-density plot on the left. The x-axis shows centrality, where the left side is "central" and the right is "fringe". The y-axis goes from "sparse" to "dense" (high y-values). Again, as expected, there is a high density center (left side points have high density), and the fringe is low density. Now let's look at our data set:



centrality-density of hindi features

The axes work the same way, but we see that the description of this structure is the "opposite" of the gaussian distribution - the center has very low density (and there are a lot of points there), and there seem to be some clusters of higher density on the *fringe*! It's no wonder that gaussian distributions don't work.

We eventually were able to get the desired results using a multilabel SVM classifier that trained an individual model for each class that we were interested in. It then would ouput multiple individual labels for each example, and we would take precision/recall per standard jacard distance metrics. The final f1-score on the model when trained using French, Arabic and German was 93%. Hindi was a good exploratory case, but we were unable to find a large enough corpus to use in our research.

FEATURES

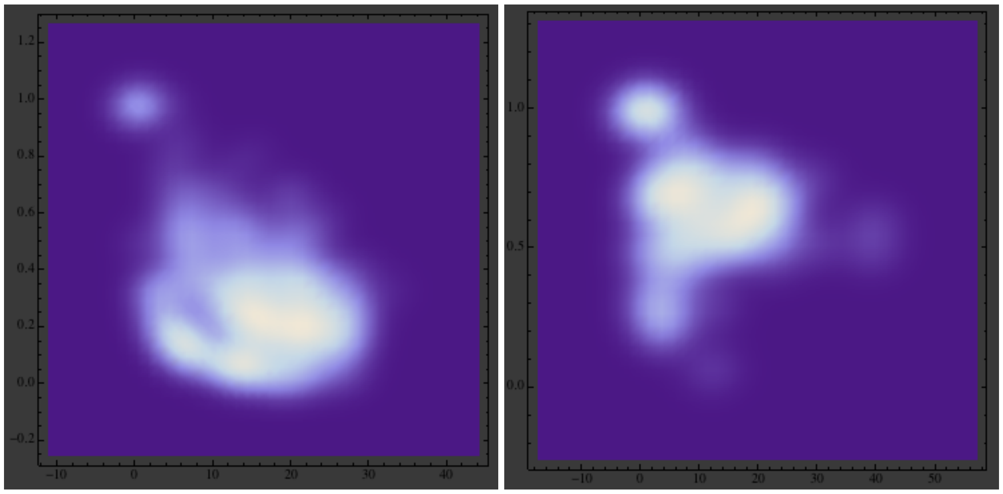
Our features for this project looked like:

- Basics
    - length of the sentence
        - (highest value terminal in the tree)
    - number of nodes in the tree `done`
    - number of each operation
        - N, R, 4one etc
    - depth `done`
- Positionality
    - max/min height of each operation (not N)
    - mean height of each operation (not N)
        - height of each 'R' subtree and take average
    - mean height of tree
        - height from each leaf averaged
- Ranges
    - max range of operation
        - how many nodes does the operation span
    - mean range of operation
        - mean # of nodes the operation spans (so R(1,3) spans 2, etc)
    - max children of a node for each operation type in flattened tree

        - this is slightly different from above since it doesn't include sections where another operation is working.
- Markov Chains
    - for each path from a leaf to a parent, take the probability of order 0 and order 1 events and then compute a probability table. Take the average of these probability tables as the overall probability table for the tree.

Along with these features, there were another duplicate set run against the compressed tree. This was briefly mentioned before, but compressing the tree amounts to taking nested similar operations and flattening them. So a tree like `(N (N 0 1) 2)` would become `(N 0 1 2)`. Of course, only same operation name children could be flattened, and if a node was a hybrid, then it was not flattened either.

Here's an example of a feature - length of the sentence versus depth of the compressed tree:



On the left is French and on the right is Hindi. We can see that this feature has a line along the middle of the graph where the features are completely separate! This is a very interesting result that confirms that Hindi, a language that is qute different from English, has a much more complex tree than French!

Another interesting result was that the usage of operations outside N and R were not signigifcant enough to include as part of our data set (they tended to be quite rare), which is also a very important consideration, as it suggests that DeKai Wu's original asssessment of the problem was correct - related information does stay grouped together across languages, and therefore it's not necessary to consider the whole alignment from scratch; building it in pieces might be a better idea.

Martin Kay suggested the markov chain features, and they ended up being the best indicator style feature - for example in Hindi, the presence of 'R' followed by 'R' in a tree was in 63% of all trees, and in French it was only 7%. In each language pair, there was a similar result, where either one or a combination of the 4 order-1 markov operations were seen to exist only on one side.

Another important result was that each feature present was only useful when given "context" by normalizing with respect to whichever of the main "dimensions" the feature corresponded to (lengh, depth, or # of nodes in the tree). For example, knowing the number of reverse nodes as a function of length was not an interesting feature, but knowing the percentage of reverse nodes as a function of length was in fact a very interesting feature. The mutual information of the features almost doubled when normalized with respect to the appropriate other feature.

PHRASAL RUN

Finally, the prediction of the classifier was made into an indicator feature in Phrasal. The baseline score was 15, and it jumped to 17.011 when this classifier was added to the mix.

CONCLUSIONS AND NEXT STEPS

The features we found, and the 93% classifier rate would probably be more useful if used as a float feature rather than an indicator. There is probably a better correlation to be found by considering how sure the classifier is in its judgement rather than just blindly taking it's prediction.

That being said, the 2 point increase is pretty significant, and is certainly higher than other trivial features in the Phrasal classifier that were implemented earlier in the quarter.

There are further features we could look for. One idea is to integrate parts of speech as the leaves of the tree, and include those in the markov table. It would make the number of features expand dramatically, which would necessitate more aggresive feature selection and component analysis, but that's hardly a problem considering that the number of training examples is very high.

There's also a lot of tuning left to be done! Due to time constraints, and the large number of moving pieces in this project (figuring out how to correctly generate the trees was a large task, as was getting the classfier working), there wasn't as much time to do final tuning on the BLEU score.