Project Proposal – Code Completion

* Give us a review of the problem. If you changed the scope of the problem, please mention that too.
* Describe the model **precisely**. Don't just say "we use MDP" -- say how the model applies to your problem (what are the features, states, variables, etc.).
* Describe the algorithm you use. You can use prose, bullets, diagrams, or pseudocode if you need to. You do not have to include the actual code (although you should have made some progress in implementing the algorithm).
* Walk us through a **concrete example** that demonstrates how the model and algorithm work. Something like "Consider the input \_\_\_. I converted it to a factor graph in Figure 1 with parameters \_\_\_. After running the algorithm we get \_\_\_, which matches the gold label / does not match the label because \_\_\_."
* Compare the current result (evaluation metric score) with the baseline and oracle. Discuss how you improve upon the baseline and how you plan to move toward the oracle. You should have **concrete numbers** (which means you should have written some code and performed some experiments, especially with the baseline and the oracle.)

Questions to answer:

What is our task?

What are our proposed models & algorithms for tackling this task?

What are concrete examples to demonstrate how the model and algorithm work?

* Precise description how the methods we have chose apply to this problem
  + What are the variables, factors, states?
* What is our current progress? Where did we start and so on…

What to claim as baseline – in paper – version of HMM – all that is important

* normally would be P(‘str’/’string’) – but we are not doing that because …. But since need to solve to one doen’t make too much sense
* instead they are using logistic regression – why they aren’t doing counting thing – we don’t have ‘str’ paired with every single word
* 10 features on pg 335 for logistic regression
* they use lambda for smoothing – pg 336
* for Viterbi – which means they return top n results – so user can tab through to the other options
* we can discuss the run time and explain why it happens to be this time – write TODO for susan if confused
* take full expresssions for what we are trying to maximize pg 335

Comment about what the seperators are – anything that is not a name number or string. If there are multiple in a row, susan just treats that as 1 seperator

What we (susan) has done is set up the HMM, but have haven’t done the logistic regression in order to come up with factors between the partial keywords and the actual intended words

Our contributions to this beyond the model: for the logistic regression, their featuresd are mostly about similarity of emitted thing with the state – look at files and tokenizer tells you where in the file it is from, instead – from wehre you currently are, how far you are from the last occurance of the word

* feature – if the word that you are trying to match with also contains in the same file # lines up
* aside from logistic regression – of what you autocompleted so far, does it make sense – checking to see if you what you auto completed can be parsed
* susan chatted how to do the parsing to us
  + unexpected end of line using build in parser python thing
  + not sure how useful this would be – only thing that would matter are the key words – for i in in….

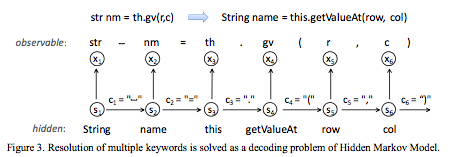
The paper they use a hidden markov model to implement Abreviation Completion as a means to support code completion of multiple keywords based on a non-predefined abbreviated input in order to improve the efficiency of code writing.

Input/output examples for python:

Input (what you type) - output (what you would get)

Motivation: Many programmers use code completion to accelerate code-writing by avoiding typing the whole character sequence of keywords. Our initial strategy for applying the material in from class and see what we could do in the autocompletion world, was to try and predict the next keyword that the programmer would type before he started entering the character sequence of that keyword. For our baseline we used an ngrams approach. After our initial exploratory attempts, which included our ngrams model, we realized that solving this problem would both be really tough and not add too much practical value in practice for the programmer. If you could very accurately predict the next keyword in the sequence based on the scope and previous keywords, why not predict the next entire sequence of key words? That would theoretically save a ton of time if it could be done accurately. Unfortunately, it is a really tough problem, especially without any initial or partial characters to narrow down the pool of keywords from which to choose. Consequently, our refocused efforts have been, in line with a paper titled Code Completion From Abbreviated Input (cite 1), to be able to autocomplete a whole sequence of keywords at once based on a partial input. As argued in the Miller paper, this approach overcomes keystroke limitations imposed by conventional code completion systems, which can only complete one keyword at a time, so the amount of extra keystrokes grows in proportion to the number of keywords. Our project goal is to recreate and improve upon the Abbreviation Completion HMM model implementation described in the HMM model.

If you were to formulate the base problem we are trying to tackle it would look like Figure 3 below (redo this ???).



Describe how the HMM works in more depth (susan ☺). List out what potential inputs and outputs would be.

This standard HMM does have limitations when it comes to modeling the problem that we are trying to tackle. Given a keyword like “string,” there are infinitely many permuations of characters that could be considered attempts to abbreviate or incorrectly stand for “string,” therefore the P(“str”/”string”) is something that is not practical to try and calculate or approximate.

