Report:

In this question, we are asked to find the most probable meaning of several ambiguous words (also called 'entities') given their respective context. To do so, three models were developped.
The first is a simple baseline which looks at the most frequent sense according to each lemma counts in every wordnet sense candidates. In partiular, we return the synset *s* that is $\underset{s}{argmax}\{frequency(s)\}$ where $frequency(s)=\sum_{lemma\,l \in s} l.count()$. The second model uses the nltk Lesk's algorithm, which looks at the context words in order to choose the appropriate sense of a given ambiguous word. For instance, it returns $\underset{s}{argmax}\{overlap(s,c)\}$ where $overlap(s,c)=\|s.definition() \cap c\|$ = number of tokens that are both in the definition of synset *s* and in the context *c* in which the ambiguous word occurs. Eventually our third model is a combination of both the frequency signal and the overlap signal: we choose the sense *s* that has the highest *f(s)* where $f(s)=\alpha\, frequency(s)+\beta\, overlap(s,c)$ with *c* being the context in which the ambiguous word occurs. One tiny change in the *overlap* function from the second model is that in this case, we took the *TreeTagger* lemmas of each definition word instead of the actual words. This made more sense since the context words were already lemmatized. We can think of parameters *alpha* and *beta* as how much weight we should put on each metric. These weights were first hard-coded, then learned with Linear Regression, but eventually reverted back to hard-coded values as explained later. We evaluated our three models by measuring *recall*, *precision*, and *F1* scores. Note that if we make predictions on all our entities, since we make only one synset prediction per entity, *recall* and *precision* will be the same thus making *F1* also identical. Indeed, *precision* = number of predicted synsets (1 per entity) that are actually true (1 or 0 for each entity); *recall* = number of true synsets (can be more than 1 per entity) that are predicted. However, since we predict only 1 synset the recall value is also 1 or 0 for each entity. Therefore we have that *precision* = *recall* = *F1*. Of course, one naive way of having perfect *precision* (but zero *recall*) is to predict nothing, we will come back to this idea at the end of the report.

In each model we query wordnet to get the list of all candidate synsets for a given ambiguous word. At this step, we considered providing (or not) the part-of-speech (POS) tags of the entity in order to restrict (or not) the selection of valid synset candidates. We found that providing the POS tags improved every models accuracy as shown below in *Table 1*. Note that this was tested both with and without stop words in the context, but the conclusion was the same: using POS tags is better. Moreover, we experimented with and without the removal of english stop words form the context and found that the performance is slightly better when including the stop words as we can see in *Table 1*. Note that this was tested both when we used POS tags and when we didn't, but the conclusion was the same: keeping stop words is slightly better.

Precision / Recall / F1 (in %)	Use POS = False	Use POS = True	Remove stop words = False	Remove stop words = True
Model 1	54.34 / 54.34 / 54.34	60.14 / 60.14 / **60.14**	60.14 / 60.14 / **60.14**	60.14 / 60.14 / 60.14
Model 2	34.07 / 34.07 / 34.07	38.41 / 38.41 / **38.41**	38.41 / 38.41 / **38.41**	35.66 / 35.66 / 35.66
Model 3	54.34 / 54.34 / 54.34	60.41 / 60.41 / **60.41**	60.41 / 60.41 / **60.41**	60.28 / 60.28 / 60.28

Table 1: precision, recall, F1 scores (in %) for each model on the test set depending on POS usage and stop words removal.

The first thing that striked us is that the nltk's implementation of Lesk's algorithm (ie: Model 2) performs much less than the baseline model 1 where we only look at sense frequencies. This means that looking at the context words is actually missleading the prediction of the correct synset. We have two hypotheses that could explain such low score. The first is that the implementation of nltk Lesk's algorithm looks at the intersection between the sense candidate definition words and the context.

However in our case, the context is made of word lemmas, thus making the intersection sometimes smaller than what it should actually be.

%	Precision/ Recall/ F1
Model 2: nltk.wsd.lesk	38.41 / 38.41 / 38.41
Custom Lesk: lemmatize definition	**40.55 / 40.55 / 40.55**

Table 2: precision, recall, F1 scores (in %) on the test set for different Lesk's algorithms having used POS tags, and keeping english stop words.

We were able to validate this hypothesis by implementing our own version of Lesk's algorithm that would do the same as nltk's implementation but after lemmatizing the synset candidate definition words.

As we can see in *Table2*, our own implementation increased the *precision*, *recall*, and *F1* score. The second reason that could explain such a low score for the Lesk's algorithm is that in our case, the context size is not large enough to infer some useful information about the overall scene in which the ambiguous word is used. In order to test this we tried to plot a diagram that shows the accuracy of that model according to each instance's context length.

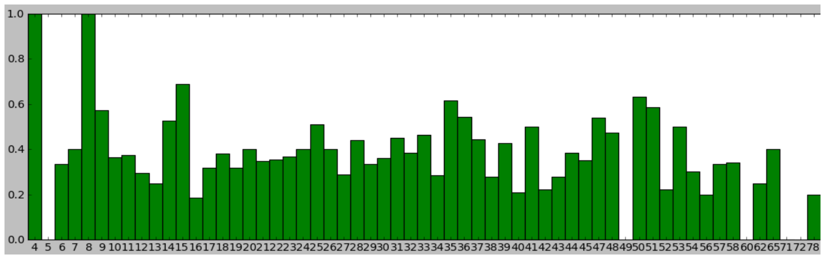

Figure 1: Accuracy score for Model 2 (nltk.wsd.lesk) according to context length.

Unfortunately, as we can see in *Figure1* the context length doesn't seem to really influence the accuracy score of the model in any convincing way. We may even say that longer contexts have poorer accuracy, which goes against our initial assumption.

Further more, we note that combining the metrics of the first two models didn't improve a lot the performance (cf: Model 3 in *Table1*). This is probably due to the fact that Model 2 does a poor job at predicting the correct wordnet synset, and doesn't add any useful information to Model 1. In any case, in this last model we combined the sense frequency metric (Model 1) with the sense definition & context overlap (Model 2). Each metric are respectively weighted with parameters *alpha* and *beta*. We first explored the different values that they could take and found *alpha* = 1.0 and *beta* = 0.5 to work well with our development dataset. We then tried to learn those weights using the nltk logistic regression library by building an X and a Y matrix of the following form: X has two columns, the first with sense frequency metric, and the second with sense definition & context overlap. Each row represents a candidate synset for a particular ambiguous entity. The Y matrix is a column vector with value 1 when the corresponding candidate synset is the true one, and 0 otherwise. After applying logistic regression on these matrices, we used the weights computed by this aproximation algorithm and used them as *alpha* and *beta* parameter values in our third model. Unfortunately, it turned out that these approximated weights produced a slightly worse *F1* score for Model 3. We thus sticked with our initial hard-coded values of 1.0 and 0.5.

Eventually, we previously said that predicting nothing would yield a perfect *precision* score but a null *recall*. In order to improve the *F1* score, one can think of not giving a prediction when the model "is not sure enough" of its prediction, thus slightly increasing the *precision* at the expense of *recall*. The tricky part here is how do we define the model's confidence about a prediction and how to set its threshold. Moreover, some improvement could be made by using better metrics than the default Lesk's algorithm. Once better metrics are found, another improvement could also be made when trying to learn the 'combination' weights of each metric in the third model by introducing a more complex machine learning algorithm than logistic regression. To conclude, we believe that for such a complex task, a much larger dataset (with more various context lengths) would be required to be able to improve our scores.