Homework 2: The relationship between lexical information and syntax

Language usually provides lots of clues to the structure of sentences. For example, English allows sentences like the following:

1. "The doctor gave the embassy the prize."
2. "The doctor gave the prize to the embassy."

This structure, which is known as the *dative alternation*, exists in many of the world's languages. The first syntactic structure is often called a "double object (**DO**) dative", while the second is called a "prepositional object (**PO**) dative." This is because the case of (1) looks like it contains two direct objects while (2) contains the preposition "to" to mark the indirect object in English. Both the direct and indirect objects take the forms of **noun phrases (NPs)**. Noun phrases are syntactic units that, in English, can look like these:

- Proper noun: Doctor, Dr. Jacobs, Bank of America, etc.
- Determiner Noun: the/a/some/our/their/one/any doctor/apple/apples
- Compounds: the pepperoni pizza, the housecat, the side door
- Plural Nouns: apples/houses

NPs can be larger and contain more than just nouns, containing lots of modifiers (adjuncts), including relative clauses or other noun phrases:

- The doctor wearing the pink scrubs
- The doctor who likes doing surgery
- The doctor (who) the patients like
- The doctor (who) the patients of the famous hospital like
- The famous and skilled doctor from the mountain town of Telluride

Here are some more examples of DO structures and their PO alternatives:

Direct object	Indirect object	DO sentence	PO sentence
The pepperoni pizza	The first grade teachers	The delivery person brought [indirect_object the first grade teachers] [direct_object the pepperoni pizza]	The delivery person brought [direct_object the pepperoni pizza] [indirect_object to the first grade teachers]
The briefcase of money	The police	The detective sent [indirect_object the police] [direct_object the briefcase of money].	The detective sent [direct_object the briefcase of money] [indirect_object to the police].

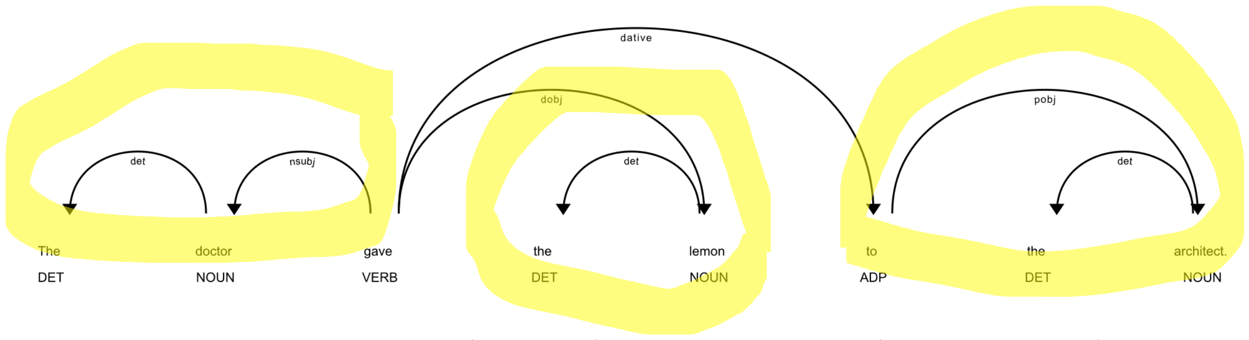
The production of these structures is strongly associated with many different linguistic properties. In particular, there are many trade-offs in how people order the arguments (direct and indirect objects) in their sentences. In English, people typically produce long noun phrases

(“The doctor gave the embassy the prize for having the cleanest building”) second, or include a prepositional form when the noun phrases are confusable (“The vet returned the kitten to the cat”).

Your job in this homework will be to **(Q1)** *extract the direct and indirect objects* from double object dative sentences and *classify the noun phrases* into either direct objects and indirect objects:

A. **Subject**. The *agent* doing the action (typically a highly animate being, e.g., "the doctor") but may also be inanimate (“the tree”). It will typically have the `nsubj` dependency.
B. **Verb**. This is the action that is being performed, e.g., “give”. It will typically have the `VERB` or `ROOT` dependency.
C. **Direct object.** “the prize”, which is the *patient* of the giving action, is usually a non-animate object, and can be more concrete (“the cash”) or more abstract (“the idea”). It will typically have the `dobj` dependency in spaCy and `obj` in stanza.
D. **Indirect object.** The recipient of the giving action, i.e., “the embassy”, who is the *recipient* of the giving action, which is also usually highly animate (“the doctor”). It will typically have the `dative` dependency in spacy but `obl` or `iobj` in stanza (if in a prepositional object dative or a double object dative, respectively).

A representative dependency parse can be seen here, in which the VERB (“gave”) attaches to the noun phrase “The doctor” which has the `nsubj` relation, as well as the `dobj` relation (“the lemon”) and the `dative` relation (“to the architect”). Note that we are interested in the entire **subtree** (the token.subtree object in spacy) which contains *all* of the **children** of a particular node (e.g., all the children of “doctor” and all of their children also). The subtree features will correspond to an entire noun phrase. It is easier to see these “subtrees” when looking at the visualized parse:



And then **(2)** create a set of syntactic and lexical features that predict whether the direct or indirect object come first in the sentence. In **(3)** you will use a neural language model to get a sense of how well a sentence encoder “knows” the syntactic structure of the sentence without using specific information.

Question 1: Extract noun phrases and sentence structures from data (20 points)

First, we want you to create a function called `get_sentence_structure(sentence)`. This function will take as input the sentence and return either “DO” (as a string), “PO” (as a string), or None (NoneType). You may want to refer back to the notebooks on using the `DependencyMatcher` to see whether a sentence contains a particular structure. Note that the direct and indirect objects *must be* siblings (otherwise they belong to different clauses) so it is not sufficient to just search for a single dependency.

Then, we want you to create a function called `extract_direct_object(sentence)` that takes a sentence as input and returns **a string** containing only the direct object. We want you to do the *same thing* for a function called `extract_indirect_object(sentence)`. Note that the direct and indirect objects *must be* siblings (otherwise they belong to different clauses), so it is not sufficient to just search for a single dependency.

We will be evaluating you on the syntactic wellformedness of your noun phrases (they must be noun phrases) and on the amount of words that overlap with a gold standard. When training your model, refer to the `DO_sample.jsonl` and `PO_sample.jsonl` for representative examples of how your results should look. You may assume that if your model gets these same noun phrases, that your model will pass tests.

Note a few things about the corpus:

- Your training data (training_data.tsv) will be a tab-delimited file with real-world conversational sentences that contain the double object dative structure and the identity of the direct and indirect objects so you can assess accuracy.
- There will be some typos, errors, and tricky cases in your training data so *use the extracted NPs as a guideline for building your system*. For example, some of the output of your functions will be incorrect – the notebooks from class will give you an idea of some types of errors to expect.

We will have leaderboards of the accuracy along several dimensions, but you will receive full points for your code as long as you meet some minimum thresholds:

- <u>Accuracy of the classification</u> (DO/PO) – Should beat majority class (83%)
- <u>Accuracy of NP extraction</u> compared with our implementation of the DependencyMatcher – Should exceed 90% accuracy
- <u>Accuracy of extraction on difficult test sentences</u> designed to test a variety of cases (typos, errors, and intentional tests) – Should exceed 60% accuracy

Question 2: Extract NP features (30 points)

One of the ways that work in computational linguistics is unique is that many simple solutions exist for complex problems. Often, linguists or linguistic knowledge can save you a lot of time when building models. So, for this question you will be asked to extract different linguistic features associated with these sentences for the direct and indirect objects in each sentence. We want you to process noun phrases within a sentence and extract simple linguistic features that stand for different properties of the direct and indirect objects. You should create three wrapper functions (which may depend on any number of other functions you create) which should have the following specific names and arguments and return values of a specific type.

- `extract_feature_1(noun_phrase, sentence): integer`
- `extract_feature_2(noun_phrase, sentence): string`
- `extract_feature_3(noun_phrase, sentence): float`

Make sure that your functions above only return a single element and that element is of the right type. The first function should return an integer; the second a string; the third a float. If you have some missing values, make sure it is also the right type (e.g., not a number, the empty string, etc.). Your functions should work for both types of noun phrase and return the same types of features for both. Your model should work for words that are not necessarily in your vocabulary – return something for *every single data point*, such as `None` or `NaN`. Some features you might want to consider:

- **Integer valued** features of potential interest
    - Length of the noun phrase in number of words
    - Length of the noun phrase in number of characters
    - Syntactic depth of noun phrase
    - Whether the word "to" is present (1) or not (0)
    - Whether the head of a noun phrase is animate (1) or not (0)
    - Wordnet-based semantic features (e.g., vertebrate or not)
      NB: You may want to consider log transforming many of these, depending on how they are distributed.
- **String valued** features (anything that appears less than 5 times will be replaced with UNK)
    - Part-of-speech tag sequence (e.g., pronoun, proper nouns, singular/plural)
    - Verb string or lemma (e.g., "give")
    - Head noun string or lemma (e.g., "doctor")
    - The head noun part-of-speech (e.g., PRON, NN, NNS, NNP, NNPS)
      NB: You may choose to normalize the strings in any way that suits you but anything that appears less than 5 times will be replaced with UNK
- **Float valued** features
    - Unigram, bigram, or trigram probability of words in noun phrase
    - Log median or mean word frequency of all words in noun phrase
    - Max cosine similarity of noun phrase to other noun phrases in the sentence
    - The likelihood of each word from a trigram model or each subword from a neural language model
      NB: You may choose to use any model or any dataset that you would like to compute these numbers

We will be using your feature sets to train a classifier to predict whether a string from a given sentence is the *direct object* or the *indirect object*. Thus, we expect your functions to return features that we can use to train this DO/PO classifier.

To train your model, that assume *we* will be processing your code in the following way:

- Upload a zip file containing all of your relevant python files, one of which is HW2.py
- Load in corpus using `load_corpus` function
- `import HW2` must work
- Question 1
    - Extract direct, indirect objects from each sentence (Question 1)
    - Return either DO or PO for a sentence – these will be your y's
        - "DO": "The man gave [indirect_object, the child] [direct_object, a hotdog]."
        - "PO": "The man gave [direct_object, a hotdog] [indirect_object to the child]."

    - None: Sentence does not contain the dative structure of interest
- Question 2
  - Return linguistic features for an arbitrary noun phrase in a sentence
- Turning each of the three feature lists from above into a list of triples using the function `concatenate_features` from
  - Direct object features
  - Indirect object features
- Train a classifier (a Multilayer Perceptron) whose job is to predict whether the direct and indirect objects appeared in the DO/PO order. You may test your performance on your training data using `evaluate.py`.

Question 3: Extract sentence embeddings (15 points)

A natural comparison between the methods used above and modern approaches is to use a neural model to extract embeddings from `sentencetransformers`. You may use any model that `sentencetransformers` supports but keep in mind that Gradescope images aren't terribly fast so check out `all-distilroberta-v1` and `all-MiniLM-L6-v2` ;)

If you want to build your own sentence embedder, hgere are some suggestions: RoBERTa, GPT-2, DistilBERT, DistilGPT. If you're the type to play around with the embeddings, try different pooling functions for creating sentence embeddings.

Bonus Question (2 points): We know that neural language models are extremely sensitive to all kinds of subtle changes in their input. Thus, you might wonder what factors the language model is most sensitive to and whether changing the input to the model would make the model more or less effective. For this bonus question, you may also use the function `alter_sentence` to change the form of a sentence in any way you please – e.g., you can change all the nouns, replace the nouns with semantic neighbors, or simplify the language anywhere in the sentence in a way that will make the classification task easier. Think of this as a way of using what we learned in (2) to make performance in (3) even better. Get creative! If you attempt this bonus, you must also answer Bonus Question 1!

- `extract_sentence_embedding(sentence)`: returns a sentence embedding in the `.numpy()` format (i.e., of type `numpy.array`). The output of your functions below should should the embedding associated with an input sentence from any of the models in the form of a numpy array (not a pytorch or tensorflow Tensor).
- **BC1 (3 points):** `alter_sentence(sentence)`: returns a string that *may* be an altered version of the input sentence. If you do not wish to attempt the bonus, you must not change this function.

Short Answer Questions

1. **10 points.** Dependency relations are sometimes tricky for parsers to get right. Using the spacy and spacy-stanza parsers, generate plots of the dependency graphs for "The doctor gave the spotted lemon to the doctor" and include them in your submission. Compare the areas where the two graphs are the same and cases where they are different. What factors do you think influenced the behavior of the parsers?

2. **15 points.** Describe the features you used in Coding question 2. Did you use features from the list of possible features? Why or why not? If you used features from the lists, did you try multiple combinations of features to get the best performance? Provide a justification for using the features that you did.
3. **10 points.** Which model did you choose to use for `sentencetransformers`? What reasons did you select this model? If you considered other models, how did you evaluate which one to use? Does your embedding model perform better than your feature-based model? Why do you think it does/doesn't?
   Bonus (2 points): Report the performance of a classifier (the MultilayerPerceptron from the supplementary code) that is trained on features extracted from the training data. Build a model that contains both symbolic features and embeddings features. Does this "combined" model perform better or worse than the symbolic-only and embeddings-only models? You do not need to upload your model -- just use the model training code to obtain your performance statistics.
4. **Bonus Question (3 points):** Describe the function you wrote to change the input to the sentence embedding generation model. Why did you make the changes you did? Did you think your changes would help or hurt performance? Why do you think that is?

Collaboration

You may partner with your classmates for this assignment but you MUST tell us who you worked with in a file called collaborators.txt.

Resources

- SUBTLEX-US frequency norms: https://www.ugent.be/pp/experimentele-psychologie/en/research/documents/subtlexus
- Animacy norms: https://osf.io/4t3cu/
- Packages of potential interest and use:
  - spacy
  - stanza
  - spacy-stanza
  - benepar
  - nltk with punkt and wordnet submodule (don't forget to download it)
  - scikit-learn

Note about Gradescope submission

Files to submit. You do not need to create any additional files – everything will live in HW2.py. You may prototype your submission using a Jupyter notebook but your final submission must be converted to a .py file.

Folder structure. HW2.py should be in the top level directory. If you have any additional files to add or folders (e.g., norms you use), you must include them in your submission but they may be anywhere in the zip file as long as your code references these resources correctly. The folder structure should remain the same from what you see when you unzip the homework assignment. When you are done editing the code, you must create a zip structure where you select only the *files* needed for your submission to work.

Software requirements: You may use packages specified in the homework (i.e., transformers, sentence-transformers, spacy, nltk, scikit-learn) and any python standard modules (e.g., collections, os, etc.); no packages other than these and their dependencies will be installed on the autograder. If you want to use packages other than the four listed, please consult with the instructor (cxjacobs@buffalo.edu) the TAs (Sean [shayanaf@buffalo.edu] or Magalí [solmagal@buffalo.edu]).