General Language Models Case Study

The AutOntology tool can work in conjunction with already-existing language models. This can be done with “out of the box” packages. One specific example is the spaCy model, which has several pre-trained models. One example is “en_core_web_sm”. This can be used in conjunction with additional entity annotation for heliophysics or another relevant field, which is known as transfer learning. Examples of how to use pre-trained spaCy models for Named Entity Recognition are shown below:

```
nlp = spacy.load("en_core_web_sm")
```

Fig. 1 – With the en_core_web_sm model, which is the default English model

```
nlp = spacy.load("en_core_web_md")
```

Fig. 2 – With the en_core_web_md model, which is an English optimised for CPU usage.

The model can also be used in conjunction with language models to bring out certain features in the text. An overview of some language models is provided in this document, as well as how they can potentially fit into the AutOntology model.

Unigram Probability Model

This language model estimates the probability of a given word appearing within any sequence of words.

```
# code based on https://medium.com/mti-technology/n-gram-language-model-
b7c2fc322799

def tokenize_raw_text(raw_text_path: str, token_text_path: str) -> None:
    """
    Read a input text file and write its content to an output text file in
the form of tokenized sentences
    :param raw_text_path: path of raw input text file
    :param token_text_path: path of tokenized output text file
    """
    with open(raw_text_path) as read_handle, open(token_text_path, 'w') as
write_handle:
        for paragraph in read_handle:
            paragraph = paragraph.lower()
            paragraph = replace_characters(paragraph)

            for tokenized_sentence in
generate_tokenized_sentences(paragraph):
                write_handle.write(','.join(tokenized_sentence))
                write_handle.write('\n')


def generate_tokenized_sentences(paragraph: str) -> Iterator[str]:
    """
    Tokenize each sentence in paragraph.
    For each sentence, tokenize each words and return the tokenized
sentence one at a time.
```

```
    :param paragraph: text of paragraph
    """
    word_tokenizer = RegexpTokenizer(r'[-\'\w]+')

    for sentence in sent_tokenize(paragraph):
        tokenized_sentence = word_tokenizer.tokenize(sentence)
        if tokenized_sentence:
            tokenized_sentence.append('[END]')
            yield tokenized_sentence

def replace_characters(text: str) -> str:
    """
    Replace tricky punctuations that can mess up sentence tokenizers
    :param text: text with non-standard punctuations
    :return: text with standardized punctuations
    """
    replacement_rules = {'“': '"', '”': '"', '’': "'", '--': ','}
    for symbol, replacement in replacement_rules.items():
        text = text.replace(symbol, replacement)
    return text

# Each text is tokenized and saved to a new file

tokenize_raw_text('path_to_file/train_raw.txt',
''path_to_file/train_tokenized.txt')
tokenize_raw_text(''path_to_file/dev1_raw.txt',
''path_to_file/dev1_tokenized.txt')
tokenize_raw_text(''path_to_file/dev2_raw.txt',
''path_to_file/dev2_tokenized.txt')

# Each unigram will be used to increment the count in the counts attribute,
a dict that maps each unigram to its count in the training text. Then, this
class stores the total number of words.

train_counter = UnigramCounter('../data/train_tokenized.txt')
print(train_counter.token_count)
print(train_counter.counts)

train_model = UnigramModel(train_counter)
train_model.train(k=1)
print(train_model.probs)

# Finally, the model is evaluated.

dev1_counter = UnigramCounter('../data/dev1_tokenized.txt')
dev2_counter = UnigramCounter('../data/dev2_tokenized.txt')

dev1_avg_log_likelihood = train_model.evaluate(dev1_counter)
dev2_avg_log_likelihood = train_model.evaluate(dev2_counter)
```

N-gram Probability Model

This language model estimates the probability of a given sequence of length n of words appearing within any sequence of words.

```
# code based on https://nlpforhackers.io/language-models/ and
https://medium.com/analytics-vidhya/a-comprehensive-guide-to-build-your-
own-language-model-in-python-5141b3917d6d
```

```
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

with open(Path to corpus file) as f:
  contents = f.read()

corpus = nltk.sent_tokenize(contents)

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for sentence in corpus.sents():
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        model[(w1, w2)][w3] += 1

# Transform the counts into probabilities
for w1_w2 in model:
    total_count = float(sum(model[w1_w2].values()))
    for w3 in model[w1_w2]:
        model[w1_w2][w3] /= total_count
```

This can be used to predict the occurrence of a particular word. Here is a script that generates a random piece of text using the model:

```
# code based on https://nlpforhackers.io/language-models/

import random

# starting words
text = ["today", "the"]
sentence_finished = False

while not sentence_finished:
  # select a random probability threshold
  r = random.random()
  accumulator = .0

  for word in model[tuple(text[-2:])].keys():
      accumulator += model[tuple(text[-2:])][word]
      # select words that are above the probability threshold
      if accumulator >= r:
          text.append(word)
          break

  if text[-2:] == [None, None]:
      sentence_finished = True

print (' '.join([t for t in text if t]))
```

This can be used to create predictive text based on the corpus.

Text Generation Using Keras

The aim of this model is to create legible, sensible text that resembles that from the original document. This can potentially be used to bolster the amount of text training examples.

code based on https://medium.com/analytics-vidhya/a-comprehensive-guide-to-build-your-own-language-model-in-python-5141b3917d6d

```
import numpy as np
import pandas as pd
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import LSTM, Dense, GRU, Embedding
from keras.callbacks import EarlyStopping, ModelCheckpoint

with open(Path to corpus file) as f:
  contents = f.read()

import re

def text_cleaner(text):
    # lower case text
    newString = text.lower()
    newString = re.sub(r"'s\b","",newString)
    # remove punctuations
    newString = re.sub("[^a-zA-Z]", " ", newString)
    long_words=[]
    # remove short word
    for i in newString.split():
        if len(i)>=3:
            long_words.append(i)
    return (" ".join(long_words)).strip()

# preprocess the text
data_new = text_cleaner(contents)

def create_seq(text):
    length = 30
    sequences = list()
    for i in range(length, len(text)):
        # select sequence of tokens
        seq = text[i-length:i+1]
        # store
        sequences.append(seq)
    print('Total Sequences: %d' % len(sequences))
    return sequences

# create sequences
sequences = create_seq(data_new)

from sklearn.model_selection import train_test_split

# vocabulary size
vocab = len(mapping)
sequences = np.array(sequences)
# create X and y
X, y = sequences[:,:-1], sequences[:,-1]
# one hot encode y
y = to_categorical(y, num_classes=vocab)
# create train and validation sets
X_tr, X_val, y_tr, y_val = train_test_split(X, y, test_size=0.1,
random_state=42)

print('Train shape:', X_tr.shape, 'Val shape:', X_val.shape)
```

```
# define model
model = Sequential()
model.add(Embedding(vocab, 50, input_length=30, trainable=True))
model.add(GRU(150, recurrent_dropout=0.1, dropout=0.1))
model.add(Dense(vocab, activation='softmax'))
print(model.summary())

# compile the model
model.compile(loss='categorical_crossentropy', metrics=['acc'],
optimizer='adam')
# fit the model
model.fit(X_tr, y_tr, epochs=100, verbose=2, validation_data=(X_val,
y_val))

# generate a sequence of characters with a language model
def generate_seq(model, mapping, seq_length, seed_text, n_chars):
	in_text = seed_text
	# generate a fixed number of characters
	for _ in range(n_chars):
		# encode the characters as integers
		encoded = [mapping[char] for char in in_text]
		# truncate sequences to a fixed length
		encoded = pad_sequences([encoded], maxlen=seq_length,
truncating='pre')
		# predict character
		yhat = model.predict_classes(encoded, verbose=0)
		# reverse map integer to character
		out_char = ''
		for char, index in mapping.items():
			if index == yhat:
				out_char = char
				break
		# append to input
		in_text += char
	return in_text

inp = ‘solar flares’
print(len(inp)
print(generate_seq(model, mapping, 30, inp.lower(), 15))
```

These general language models can be used to augment the already-existing corpus. Potential uses include the generation of additional text in areas where there are few examples, as well as providing statistical analysis about frequent topics in the text.