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/**](https://nlpforhackers.io/language-models/) **and** [**https://medium.com/analytics-vidhya/a-comprehensive-guide-to-build-your-own-language-model-in-python-5141b3917d6d**](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/**](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.