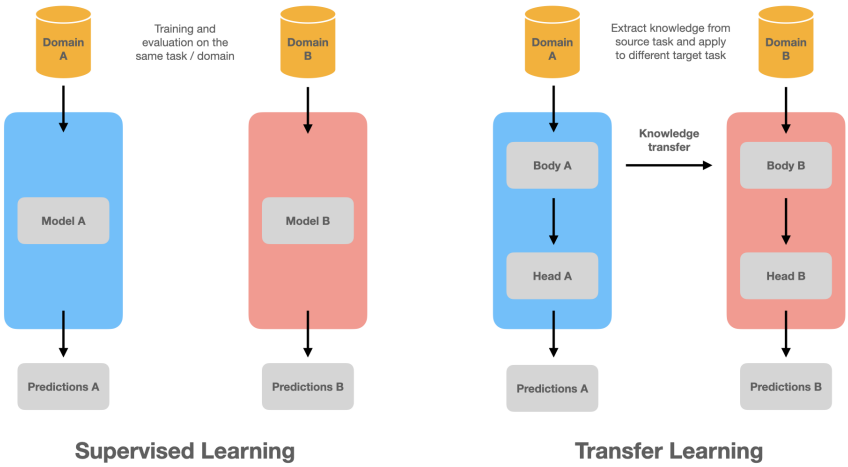
**Natural Language Processing & Transformers (21AI53)**
**CIE 2**
**Scheme and Solution**

| SL. No | | Questions | M |
|---|---|---|---|
| 1 | a) | • Install NLTK<br>• Import NLTK<br>• Download Brown corpus --- nltk.download('brown')<br>• Load the Categorized Corpus---from nltk.corpus import brown<br>• Accessing Categories---categories=brown.categoriesdocuments_in_news = brown.fileids(categories='news')<br>• Accessing Texts---text = brown.words(fileids=documents_in_news[0])<br>• Tokenization and Preprocessing---words = word_tokenize(text) | 5 |
| | b) | Sequential backoff tagger is mainly used to increase the accuracy of the pos tagging process<br>Backoff tagging is one of the core features of SequentialBackoffTagger. It allows you to chain taggers together so that if one tagger doesn't know how to tag a word, it can pass<br>the word on to the next backoff tagger. If that one can't do it, it can pass the word on to the<br>next backoff tagger, and so on until there are no backoff taggers left to check.<br><br>Defbackoff_tagger(train_sents,tagger_classes,backoff=None):<br>for cls in tagger_classes:<br>backoff = cls(train_sents, backoff=backoff)<br>return backoff<br>>>> from tag_util import backoff_tagger<br>>>> backoff = DefaultTagger('NN')<br>>>> tagger = backoff_tagger(train_sents, [UnigramTagger, BigramTagger, TrigramTagger], backoff=backoff)<br>>>> tagger.evaluate(test_sents)<br>0.8806820634578028 | 5 |
| 2 | a) | • Pos ---process of converting words into list of tuples (i.e., word,tag)<br>• Tag signifies whether the word is noun, adjective, verb et cetera.<br>• It is needed to extract phrases from sentence using tag patterns, grammar analysis and word sense disambiguation.<br>• Most taggers are trainable and use list of tagged sentence as their training data<br>Any 2 approaches---default tagger and unigram tagger with example code | 5 |
| | b) | from nltk.tag import SequentialBackoffTagger<br>from nltk.corpus import names<br>class NamesTagger(SequentialBackoffTagger):<br>def __init__(self, *args, **kwargs):<br>SequentialBackoffTagger.__init__(self, *args, **kwargs)<br>self.name_set = set([n.lower() for n in names.words()])<br>def choose_tag(self, tokens, index, history):<br>word = tokens[index] | 5 |

| | | | |
|---|---|---|---|
| | | if word.lower() in self.name_set:<br>return 'NNP'<br>else:<br>return None<br><br>>>> from taggers import NamesTagger<br>>>> nt = NamesTagger()<br>>>> nt.tag(['Jacob'])<br>[('Jacob', 'NNP')] | |
| 3 | a) |  <br>It is common practice in NLP use *transfer learning* to train a convolutional neural network on one task and then adapt or *fine-tune* it on a new task, thus making use of the knowledge learned in the original task. Architecturally, this usually works by splitting the model in terms of a *body* and *head*, where the head is a task-specific network. During pretraining, the weights of the body learn broad features of the source domain, and it is these weights which are used to initialize the new model for the new task. Compared to traditional supervised learning, this approach typically produces high-quality models that can be trained much more efficiently on a variety of downstream tasks, and with much less labelled data. | 5 |
| | b) | Subword Tokenization is a Natural Language Processing technique(NLP) in which a word is split into subwords and these subwords are known as tokens. This technique is used in any NLP task where a model needs to maintain a large vocabulary and complex word structures. The concept behind this, frequently occurring words should be in the vocabulary whereas rare words are split into frequent subwords. For example, the word “unwanted” might be split into “un”, “want”, and “ed”. The word “football” might be split into “foot”, and “ball”.<br>from transformers import AutoTokenizer<br>model_ckpt = "distilbert-base-uncased"<br>tokenizer = AutoTokenizer.from_pretrained(model_ckpt)<br>text="I love machine learning!. Tokenization is awesome"<br>encoded_text=tokenizer(text)<br>print(encoded_text)<br>tokens=tokenizer.convert_ids_to_tokens(encoded_text.input_ids)<br>print(tokens)<br>['[CLS]', 'i', 'love', 'machine', 'learning', '!', '.', 'token', '##ization', 'is', 'awesome', '[SEP]'] | 5 |

| | | | |
|---|---|---|---|
| **4** | **a)** | *Self-attention is a sequence-to-sequence operation: a sequence of vectors goes in, and a sequence of vectors comes out. Let's call the input vectors x1, x2,..., xt and the corresponding output vectors y1, y2,..., yt. The vectors all have dimension k.* To produce output vector yi, the self attention operation simply takes a weighted average over all the input vectors, the simplest option is the dot product.<br><br>$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$ | **5** |
| **4** | **b)** | Language barrier---zero shot training<br>Data<br>Biases<br>Blackbox models<br>Long documents | **5** |
| **5** | **a)** | **Let the input text is : text** = """Dear Amazon, last week I ordered an Optimus Prime action figure \from your online store in Germany. Unfortunately, when I opened the package, \I discovered to my horror that I had been sent an action figure of Megatron \ instead! As a lifelong enemy of the Decepticons, I hope you can understand my \ dilemma. To resolve the issue, I demand an exchange of Megatron for the \ Optimus Prime figure I ordered. Enclosed are copies of my records concerning \ this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""<br>**from transformers import** set_seed<br>set_seed(42)<br>generator = pipeline("text-generation")<br>response = "Dear Bumblebee, I am sorry to hear that your order was mixed up."<br>prompt = text + "**\n\n**Customer service response:**\n**" + response<br>outputs = generator(prompt, max_length=200)<br>**print**(outputs[0]['generated_text'])<br><br>Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.<br>Dear Amazon, last week I ordered an Optimus Prime action figure from your online<br>> store in Germany. Unfortunately, when I opened the package, I discovered to<br>> my horror that I had been sent an action figure of Megatron instead! As a<br>> lifelong enemy of the Decepticons, I hope you can understand my dilemma. To<br>> resolve the issue, I demand an exchange of Megatron for the Optimus Prime<br>> figure I ordered. Enclosed are copies of my records concerning this purchase.<br>> I expect to hear from you soon. Sincerely, Bumblebee.<br><br>Customer service response:<br>Dear Bumblebee, I am sorry to hear that your order was mixed up. The order was<br>> completely mislabeled, which is very common in our online store, but I can<br>> appreciate it because it was my understanding from this site and our customer<br>> service of the previous day that your order was not made correct in our mind<br>> and that we are in a process of resolving this matter. We can assure you that<br>> your order | **7** |
| **5** | **b)** | Transformer models have a maximum input sequence length that is referred to as the maximum context size.<br>Let dataset be df and let the feature name be "text". We can find the maximum | **3** |

possible length of words in sequence and plot a graph using the following code:

```
df["Words Per Tweet"] = df["text"].str.split().apply(len)
df.boxplot("Words Per Tweet", by='label_name', grid=False, showfliers=False,
color='black', )
plt.suptitle("")
plt.xlabel("");
```