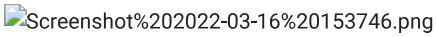
Problem Statement

You are a **Data Scientist** working at a pharmaceutical company (J&J, Pfizer, Target etc).

Studying adverse drug reactions in patients is central to drug development in healthcare.

The company wants to find relationship between drug and adverse events/ symptoms leveraging the review data. Your task is to build a mechanism

- to identify the problems (adverse events/side-effects/symptoms etc)
- tag them to the drug name which is causing them



Why Information Extraction?

1. To find and understand limited relevant parts of text
2. Gather information
3. Produce a structure representation of relevant information

How to solve this problem?

- A model that will be trained on various annotated data and help us tag or identify different entities of our interest.
- **Information Extraction** is the art of extracting useful information from large piece of text and using it directly or indirectly to gain insights.
- It is basically extracting important information based on the semantic(logical) meaning of the sentence.

Goals:

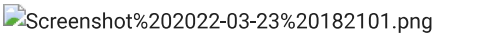
1. organize information
2. put info in a semantically precise form, to be inferred by other algo

IE system extracts clear and factual information

Some Examples:

But with this limited information a lot of problems can be solved. Let's take a simple example of

1. how **email service provider, suggest a meeting** by detecting a time frame in the mail body. 

2. handling the **customer support department** of an electronic store with multiple branches worldwide,


   -- Now, if you pass it through the Named Entity Recognition API, it pulls out the entities Bandra (location) and Fitbit (Product).

   -- This can be then used to categorize the complaint and assign it to the relevant department within the organization that should be handling this.

3. A map service provider can **detect a address and search it in the app** directly.

What is NER?

Named-entity recognition (NER), in general, (also known as entity identification or entity extraction) is a **subtask of information extraction** (text analytics) that aims at finding and categorizing specific entities.

NER involves the identification of proper names in texts, and the classification of these names into a set of predefined categories of interest like **person, location, organization, drug, time, etc.**

It refers to extracting **'named-entities'** from text. Named-entities denotes to words in a sentence representing real-world objects with proper names like:

- Person's name (Ramu, Raja, Seeta, etc.),
- Countries (India, Sri Lanka, etc),
- Organization (Google, Facebook, etc.)
- or anything that has been given a specific name.

Types of NER

- **Classical Approaches (rule-based or Dictionary)**
- **ML Approaches**
  - **Multi-class classification**
  - **Conditional Random Field (CRF)**
- **DL Approaches**
  - **Bidirectional LSTM-CRF**
  - **Bidirectional LSTM-CNNs**
  - **Bidirectional LSTM-CNNS-CRF**
  - **Pre-trained language models (Elmo and BERT)**
- **Hybrid Approaches (DL + ML)**

Classical Approaches:

Dictionary based

This is the simplest NER approach. Here we will be having a **dictionary that contains a collection of vocabulary** .

Basic string-matching algorithms are used to check whether the entity is occurring in the given text to the items in vocabulary.

Cons: The method has limitations as it is required to update and maintain the dictionary used for the system.

Rule Based

Here, the model uses a pre-defined set of rules for information extraction.

Mainly two types of rules are used,

Pattern-based rules, which depend upon the morphological pattern of the words used, and

context-based rules, which depend upon the context of the word used in the given text document. A simple example for a context-based rule is

- Let's suppose you have been asked to mask **PII (Personally identifiable information) from free text**
- We can have a rule based NER that if a person's title is followed by a proper noun, then that proper noun is the name of a person.

Sentence ="**The medicine was prescribed by Dr. Sharma and the patient responded in a positive manner.**"

- In this case **Dr. Sharma** will be labelled as name.

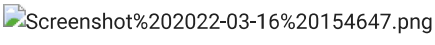
Cons: But we will miss on a lot of names, if it doesn't start with a title like Mr, Mrs, Ms, Dr, etc.

Machine Learning:

Multi-class Classification

Named entities are the labels so we can apply different classification algorithms.

There are mainly two phases while we use an ML-based solution for NER.

1. Training the ML model on the annotated documents.
2. The trained model can be used to annotate the raw documents. 

Cons: The problem here is that identifying and labelling named entities require thorough understanding of the context of a sentence and sequence of the word labels in it, which this method ignores that.

Problems in NER:

There exist some issues which need to be addressed.

- **Segmentation(Detection) 'ambiguity'.**

Consider 'New York'. This can be detected as 'New' & 'York' as separate entities.

Hence, deciding over the boundary is crucial whether to consider 'New York' as a single entity or 'New' and 'York' 2 different entities.

- **Tag assignment(Recognition) 'ambiguity'.**

We have 'Nirma' as a girl's name (PERSON)& as a detergent brand(Organization in India).

- In order to overcome these ambiguities, we introduce a new concept called **Sequence Labelling**

What is Sequence labelling?

It refers to assigning labels/tags to each element of a sequence being passed as an input using an algorithm or machine learning model.

This sequence can be **words of a sentence passed in the same order as in the sentence** .

A LSTM can be taken as a Sequence labeller.

NER can be done using a number of Sequence Labelling methods listed below alongside Rule-Based methods:

1. Linear Chain Conditional Random Fields (Linear Chain CRF)
2. Maximum Entropy Markov Models
3. Bi-LSTM

Why Conditional Random Fields?

- well suited for handling NER problems, because it takes **context into account**
- when a CRF model makes a prediction, it factors in the impact of neighbouring samples by modelling the prediction as a graphical model.
- A **linear chain CRF** confers to a labeller in which tag assignment(for present word, denoted as y_i) **depends only on the tag of just one previous word**(denoted by y_{i-1}).

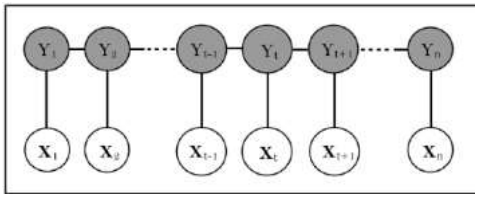
Deciding Feature Functions:

We need to extract multiple features per word of the sentence that will be assisting in recognizing a Named Entity. For this, we need functions called as feature functions that will assist in generating unique features.

- These features function can consider any logic(depends on programmer) but the output has to be either True:1 or False:0.

Optional: Math of Linear CRF

Diagram:



Fundamental Equation:

Consider the below formula:

CRF:

$$p_\theta(y|x) = \frac{exp(\sum_j w_j F_j(x,y))}{\sum_{y'} exp(\sum_j w_j F_j(x,y'))}$$

, where $F_j(x,y) = \sum_{i=1}^{L} f_j(y_{i-1}, y_i, x, i)$

Here,

pθ (y|x) refers to the probability of calculating a Label sequence(y) given a word sequence(x).

- The inner summation goes from i=1 to i=length of sentence 'L'. Hence we are summating the value of any feature function for all words of the sentence

if we have a sentence 'Ram is cool', the inner summation will add values of the output of the j^{th} feature function for all 3 words of the sentence

- The outer summation goes from j=1 to the total number of feature functions. It is doing something like this

W_1 * Σfeature_function$_1$ + W_2 * Σfeature_function$_2$……

- W_j refers to weights assigned to a feature_function$_j$. The denominator is referred to as Normalizing constant. It looks quite similar to the numerator with a couple of changes in the formula y' in place of y A third summation over y'.

Let us understand this. y' refers to all the possible Label Sequence that can be assigned to a word sequence (sentence).

Bringing back 'Ram is cool'. If you remember the named Entity Tag table mentioned above, we have a number of possible tags like PER, LOC, VEH, ORG, etc. Hence the assigned label sequence can be [PER, PER, LOC], [O, O, O], [VEH, ORG, O] and many many more!!!

Now, if I wish to calculate the P([PER, PER, LOC] | 'Ram is cool')=

- **Numerator = exp ($\Sigma_j w_j \Sigma_i F_j$('Ram is cool','PER PER LOC'))**
- **Denominator =exp ($\Sigma_j w_j \Sigma_i F_j$('Ram is cool','O O O'))+exp ($\Sigma_j w_j \Sigma_i F_j$('Ram is cool',' VEH ORG O'))+exp ($\Sigma_j w_j \Sigma_i F_j$('Ram is cool','PER ORG ORG'))….**

Hence, the 3rd summation in denominator refers to summation for all label sequence possible for the sentence, and y' refers to these possible combinations.

A couple of points note:

- As we know the **correct sequence label for 'Ram is cool' is [PER O O].**

Hence, P([PER O O]|'Ram is cool') should be highest amongst all other possible sequences for 'Ram is cool' if our CRF is trained well.

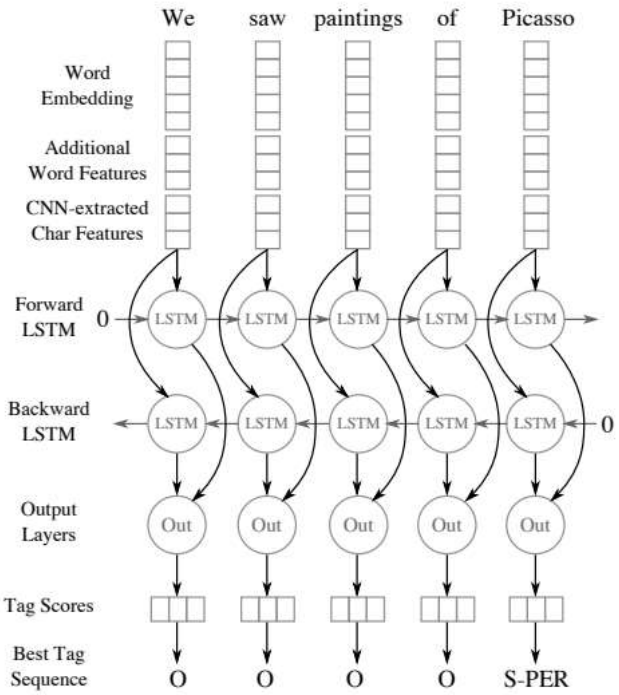
- Linear CRF appears quite similar to logistic regression.

Deep Learning:

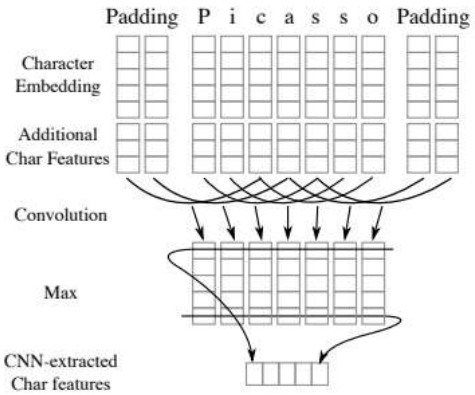
Bi-directional LSTM with CNN

- Recurrent neural networks (RNNs) are a powerful family of connectionist models that capture time dynamics via cycles in the graph.
- But they **fail due to the gradient vanishing/exploding problems.** LSTMs are variants of RNNs designed to cope with these gradient vanishing problems.
- For many sequence labelling tasks it is beneficial to have access to both past (left) and future (right) contexts.

- However, the **LSTM's hidden state (ht) takes information only from past, knowing nothing about the future**. An elegant solution whose effectiveness has been proven by previous work is bi-directional LSTM
- The basic idea is to present each sequence **forwards and backwards to two separate hidden states to capture past and future information**, respectively. Then the two hidden states are concatenated to form the final output.



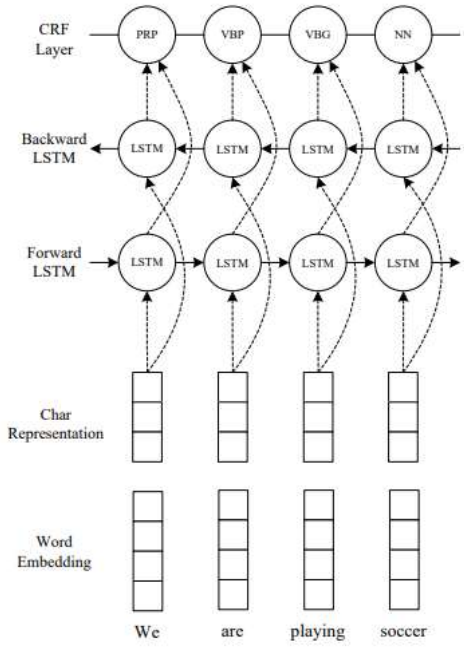
- Convolutional neural networks **(CNN)** have also been investigated for **modelling character-level information**, among other NLP tasks. So in NER CNNs are used to extract character-level features for and POS-tagging respectively



Bi-directional LSTM CRF

We have how a Bi-directional LSTM model works with another layer of CNN, but for sequence labelling we can consider **CRF**, as

- it is beneficial to consider the **correlations between labels in neighbourhoods** and jointly decode the best chain of labels for a given input sentence.
  - For example, in POS tagging an adjective is more likely to be followed by a noun than a verb,
  - In NER with standard BIO2 annotation I-ORG cannot follow I-PER.
  - Therefore, we model label sequence jointly using a conditional random field (CRF) instead of decoding each label independently.



Hybrid

- Hybrid methods are composed of **rule-based and deep learning** approaches, which facilitates the extraction of entities out of texts of domains that lack generic named entities labelled domain data sets.
- The proposed approach takes the advantages of both deep learning and clustering approaches but separately, in combination with a knowledge-based approach by using a postprocessing module

Training a NER from scratch

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import keras


# Hyperparams if GPU is available
if tf.test.is_gpu_available():
    BATCH_SIZE = 512  # Number of examples used in each iteration
    EPOCHS = 5  # Number of passes through entire dataset
    MAX_LEN = 75  # Max length of review (in words)
    EMBEDDING = 40  # Dimension of word embedding vector
# Hyperparams for CPU training
else:
    BATCH_SIZE = 32
    EPOCHS = 5
    MAX_LEN = 75
    EMBEDDING = 20
```

```
WARNING:tensorflow:From <ipython-input-2-98cbf56d884d>:2: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated an
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
```

Reading Data

```
!gdown 1ao5aINQqedAWHSXGLpAVzW1iEZj7sZ5_
!gdown 15RuVEDqO592th-44iVEpcGbtvX9RjDWR
```

```
Downloading...
From: https://drive.google.com/uc?id=1ao5aINQqedAWHSXGLpAVzW1iEZj7sZ5_
To: /content/ner_dataset.csv
100% 15.2M/15.2M [00:00<00:00, 48.0MB/s]
```

```
Downloading...
From: https://drive.google.com/uc?id=15RuVEDqO592th-44iVEpcGbtvX9RjDWR
To: /content/drugsComTrain_raw.csv
100% 1.38M/1.38M [00:00<00:00, 119MB/s]
```

```
data = pd.read_csv("/content/ner_dataset.csv", encoding="latin1")
```

```
# Show the first 10 rows
data.head(n=10)
```

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O
5	NaN	through	IN	O
6	NaN	London	NNP	B-geo
7	NaN	to	TO	O
8	NaN	protest	VB	O
9	NaN	the	DT	O

We see in column "Sentence #" sentence 1 is followed by NaN, so we need to fill the same we whatever the value is in top

```
data = data.fillna(method="ffill")
data.head(n=24)
```

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	Sentence: 1	of	IN	O
2	Sentence: 1	demonstrators	NNS	O
3	Sentence: 1	have	VBP	O
4	Sentence: 1	marched	VBN	O
5	Sentence: 1	through	IN	O
6	Sentence: 1	London	NNP	B-geo
7	Sentence: 1	to	TO	O
8	Sentence: 1	protest	VB	O
9	Sentence: 1	the	DT	O
10	Sentence: 1	war	NN	O
11	Sentence: 1	in	IN	O
12	Sentence: 1	Iraq	NNP	B-geo
13	Sentence: 1	and	CC	O
14	Sentence: 1	demand	VB	O
15	Sentence: 1	the	DT	O
16	Sentence: 1	withdrawal	NN	O
17	Sentence: 1	of	IN	O
18	Sentence: 1	British	JJ	B-gpe
19	Sentence: 1	troops	NNS	O
20	Sentence: 1	from	IN	O
21	Sentence: 1	that	DT	O
22	Sentence: 1	country	NN	O
23	Sentence: 1	.	.	O

Now, this looks better.

As we see the dataset has 4 columns

- Sentence # (the sentence number)
- Word
- POS
- Tag (the annotation which is manually labelled against each word)

This method of labelling is known as sequence labelling. Let's take a look at some of the numbers./

```
data.shape
```

```
(1048575, 4)
```

```
print("Number of sentences: ", len(data.groupby(['Sentence #'])))
```

```
Number of sentences:  47959
```

```
words = list(set(data["Word"].values))
n_words = len(words)
print("Number of unique words in the dataset: ", n_words)
```

```
Number of unique words in the dataset:  35178
```

The number of words is less than number of sentence. Its becuase we can have same word in different sentences.

```
n_tags = len(data.groupby(['Tag']))
print("Number of Labels: ", n_tags)
```

```
Number of Labels:  17
```

```
tags = list(set(data["Tag"].values))
print("Tags:", tags)
```

```
Tags: ['B-gpe', 'I-nat', 'B-eve', 'B-art', 'B-per', 'I-art', 'B-nat', 'B-geo', 'O', 'I-geo', 'B-org', 'I-org', 'B-tim', 'I-per', 'I-gpe'
```

- **geo = Geographical Entity**
- **org = Organization**
- **per = Person**
- **gpe = Geopolitical Entity**
- **tim = Time indicator**
- **art = Artifact**
- **eve = Event**
- **nat = Natural Phenomenon**

```
#let's check the frequency of each tag
data['Tag'].value_counts()
```

```
O        887908
B-geo     37644
B-tim     20333
B-org     20143
I-per     17251
B-per     16990
I-org     16784
B-gpe     15870
I-geo      7414
I-tim      6528
B-art       402
B-eve       308
I-art       297
I-eve       253
B-nat       201
I-gpe       198
I-nat        51
Name: Tag, dtype: int64
```

We can see the **GEO (gepgraphic location) tag has the max frequency** , followed by time entity.

Preprocessing

For Preprocessing, we will first put each sentence through a function which will **put each of the words along with its POS and Tag together in a tuple**. Each sentence will form a list of tuples.

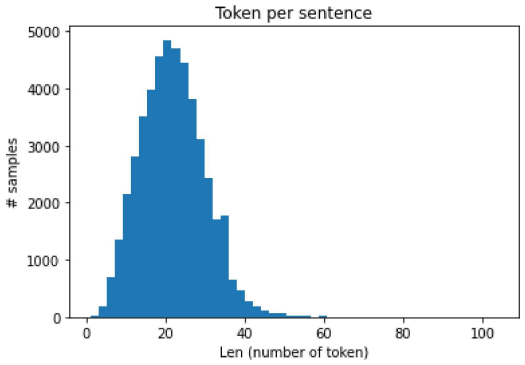
```
class SentenceGetter(object):
    """Class to Get the sentence in this format:
    [(Token_1, Part_of_Speech_1, Tag_1), ..., (Token_n, Part_of_Speech_n, Tag_n)]"""
    def __init__(self, data):
        """Args:
            data is the pandas.DataFrame which contains the above dataset"""
        self.n_sent = 1
        self.data = data
        self.empty = False
        agg_func = lambda s: [(w, p, t) for w, p, t in zip(s["Word"].values.tolist(),
                                                           s["POS"].values.tolist(),
                                                           s["Tag"].values.tolist())]
        self.grouped = self.data.groupby("Sentence #").apply(agg_func)
        self.sentences = [s for s in self.grouped]

    def get_next(self):
        """Return one sentence"""
        try:
            s = self.grouped["Sentence: {}".format(self.n_sent)]
            self.n_sent += 1
            return s
        except:
            return None

getter = SentenceGetter(data)
sent = getter.get_next()
print('This is what a sentence looks like:')
print(sent)
```

```
This is what a sentence looks like:
[('Thousands', 'NNS', 'O'), ('of', 'IN', 'O'), ('demonstrators', 'NNS', 'O'), ('have', 'VBP', 'O'), ('marched', 'VBN', 'O'), ('through',
```

```
# Get all the sentences
sentences = getter.sentences
# Plot sentence by lenght
plt.hist([len(s) for s in sentences], bins=50)
plt.title('Token per sentence')
plt.xlabel('Len (number of token)')
plt.ylabel('# samples')
plt.show()
```



From the histogram, we can see most of the sentences have around **15 to 30 tokens** . Tokens can we referred as words ignoring punctuation marks.

- **Further PreProcessing:**
- We will convert each text word to a corresponding integer ID using the **word2idx dictionary**. Doing so saves a lot of memory.
- In order to feed the data to out Bi-LSTM-CRF model, we need to ensure that all text should be of the same length.
    - The method sequence.pad_sequences() and variable MAX_LEN are used for this purpose.
    - The text which are shorter than MAX_LEN are padded to get them to the same length, whereas text which are longer than MAX_LEN are truncated.

```
import sklearn

# Vocabulary Key:word -> Value:token_index
# The first 2 entries are reserved for PAD and UNK
word2idx = {w: i + 2 for i, w in enumerate(words)}
word2idx["UNK"] = 1 # Unknown words
word2idx["PAD"] = 0 # Padding

# Vocabulary Key:token_index -> Value:word
idx2word = {i: w for w, i in word2idx.items()}

# Vocabulary Key:Label/Tag -> Value:tag_index
# The first entry is reserved for PAD
tag2idx = {t: i+1 for i, t in enumerate(tags)}
tag2idx["PAD"] = 0

# Vocabulary Key:tag_index -> Value:Label/Tag
idx2tag = {i: w for w, i in tag2idx.items()}
print("The word Obama is identified by the index: {}".format(word2idx["Obama"]))
print("The labels B-geo(which defines Geopraphical Enitities) is identified by the index: {}".format(tag2idx["B-geo"]))
```

```
The word Obama is identified by the index: 33417
The labels B-geo(which defines Geopraphical Enitities) is identified by the index: 8
```

```
#from keras.preprocessing.sequence import pad_sequences
from keras_preprocessing.sequence import pad_sequences

# Convert each sentence from list of Token to list of word_index
X = [[word2idx[w[0]] for w in s] for s in sentences]

# Padding each sentence to have the same lenght
X = pad_sequences(maxlen=MAX_LEN, sequences=X, padding="post", value=word2idx["PAD"])

# Convert Tag/Label to tag_index
y = [[tag2idx[w[2]] for w in s] for s in sentences]

# Padding each sentence to have the same lenght
y = pad_sequences(maxlen=MAX_LEN, sequences=y, padding="post", value=tag2idx["PAD"])
from keras.utils import to_categorical

# One-Hot encode
y = [to_categorical(i, num_classes=n_tags+1) for i in y]  # n_tags+1(PAD)
from sklearn.model_selection import train_test_split
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.1)
X_tr.shape, X_te.shape, np.array(y_tr).shape, np.array(y_te).shape
```

```
print('Raw Sample:\n ', ' '.join([w[0] for w in sentences[0]]))
print('\n ' )
print('Raw Label:\n ', ' '.join([w[2] for w in sentences[0]]))
print('\n ' )
print('After processing, sample:\n', X[0])
print('\n ' )
print('After processing, labels:\n', y[0])
```

```
Raw Sample:
  Thousands of demonstrators have marched through London to protest the war in Iraq and demand the withdrawal of British troops from tha


Raw Label:
  O O O O O O B-geo O O O O O B-geo O O O O O B-gpe O O O O O


After processing, sample:
 [10207  1277  6629 20616 15605 14764 14515 12779  1278 15048 17680 20918
 23613 32038 13909 15048  6336  1277 28724 32859  9404  3914  7714  5905
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0]


After processing, labels:
 [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
```

```
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```

Model Building - Bi-LSTM + CRF

```
!pip install tensorflow_addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow_addons
  Downloading tensorflow_addons-0.18.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 6.7 MB/s
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from tensorflow_addons) (2.7.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tensorflow_addons) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->tensorflow_addons) (3
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.18.0
```

```
# from keras.models import Model, Input
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional
#from keras_contrib.layers import CRF
from tensorflow_addons.layers import CRF

# Model definition
input = Input(shape=(MAX_LEN,))
model = Embedding(input_dim=n_words+2, output_dim=EMBEDDING, # n_words + 2 (PAD & UNK)
                  input_length=MAX_LEN, mask_zero=True)(input)  # default: 20-dim embedding
model = Bidirectional(LSTM(units=50, return_sequences=True,
                           recurrent_dropout=0.1))(model)  # variational biLSTM
model = TimeDistributed(Dense(50, activation="relu"))(model)  # a dense layer as suggested by neuralNer
crf = CRF(n_tags+1)  # CRF layer, n_tags+1(PAD)
out = crf(model)  # output
model = Model(input, out)
model.compile(optimizer="rmsprop", loss='categorical_crossentropy', metrics ='accuracy')
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 75)]              0

 embedding (Embedding)       (None, 75, 20)            703600

 bidirectional (Bidirectiona  (None, 75, 100)          28400
 l)

 time_distributed (TimeDistr  (None, 75, 50)           5050
 ibuted)

 crf (CRF)                   [(None, 75),              1278
                              (None, 75, 18),
                              (None,),
                              (18, 18)]

=================================================================
Total params: 738,328
Trainable params: 738,328
Non-trainable params: 0
_________________________________________________________________
```

Now, let's again go back to our original ask.

- We need to find relation between drug name and problems it is causing.

But there are **2 obstacles** :

- But we **don't have any entity** tags which talk either about **drug name** or **side effects/problems/diseases** .
- We **need a lot of annotated data from a specific domain** (in this case BioMedical data)

How do we tackle it?

The answer is we use a pre-trained domain specific NER. There are domain specific NERs such as banking, e-commerce, manufacturing, bio-medical NER.

Before that, let's check the dataset which has review data from a pharma company.

```
raw_df=pd.read_csv('/content/drugsComTrain_raw.csv', encoding='iso-8859-1')
raw_df.head()
```

	uniqueID	drugName	condition	review	rating	date	usefulCount
0	206461.0	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9.0	20-May-12	27.0
1	95260.0	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8.0	27-Apr-10	192.0
2	92703.0	Lybrel	Birth Control	"I used to take another oral	5.0	14-Dec-	17.0

We can see, the data has these importand columns:

1. **drugName**
2. **condition**
3. **review** Let's take a look at one of review, and understand how a **SME would have labelled it manually.**

```
print('Review Number:',raw_df['uniqueID'][0])
print('Drug Name:',raw_df['drugName'][0])
print('Review:',raw_df['review'][0])
```

```
Review Number: 206461.0
Drug Name: Valsartan
Review: "It has no side effect, I take it in combination of Bystolic 5 Mg and Fish Oil"
```

And we see, for review number 206461, there were **no side-effects or problem** being caused by the drug Valsartan. Let's take a look at another example.

```
print('Review Number:',raw_df['uniqueID'][15])
print('Drug Name:',raw_df['drugName'][15])
print('Review:',raw_df['review'][15])
```

```
Review Number: 81890.0
Drug Name: Liraglutide
Review: "I have been taking Saxenda since July 2016.  I had severe nausea for about a month once I got up to the 2.6 dosage.  It has sin
```

Liraglutide is a molecule used to treat type 2 diabetes and chronic weight management. Now, we can see there are some side-effects attached to it

- **dry mouth**
- **nausea**

So, as we get closer to the ask. We need to map Liraglutide with dry mouth and nausea.

But, how?

Let's check what are the tools we have in hand. But before that, let's take a look at how a word cloud on this review column would look like:

```
# let's see the words cloud for the reviews

# most popular drugs

from wordcloud import WordCloud
from wordcloud import STOPWORDS

stopwords = set(STOPWORDS)

wordcloud = WordCloud(background_color = 'yellow', stopwords = stopwords, width = 1200, height = 800).generate(str(raw_df['review']))

plt.rcParams['figure.figsize'] = (10, 15)
plt.title('Word Cloud - Review', fontsize = 25)
print(wordcloud)
plt.axis('off')
plt.imshow(wordcloud)
plt.show()
```

```
<wordcloud.wordcloud.WordCloud object at 0x7f8e47db0f90>
```

Not much information, is inferred from the word cloud.

- We need a model which is pre-trained on Bio-Medical data and would be able to identify drug names and side-effects/symptoms and disease name. So here comes, stanza

Bio-Medical NER - Stanza

At a high level, Stanza currently provides packages that support **Universal Dependencies (UD)-compatible syntactic analysis** and named entity recognition (NER) from both English biomedical literature and clinical note text. Officially offered packages include:

- 2 UD-compatible biomedical syntactic analysis pipelines, trained with human-annotated treebanks;
- 1 UD-compatible clinical syntactic analysis pipeline, trained with silver data;

Category	Treebank	Package Name	New Tokenization	Source Corpora	Treebank Doc
Bio	CRAFT	craft	Yes	Full-text biomedical articles related to the Mouse Genome Informatics database; general English Web Treebank.	CRAFT homepage
	GENIA	genia	No	PubMed abstracts related to “human”, “blood cells”, and “transcription factors”.	GENIA homepage
Clinical	MIMIC	mimic	Yes	All types of MIMIC-III clinical notes; general English Web Treebank.	MIMIC-III homepage

- 8 accurate biomedical NER models augmented with contextualized representations;
- 2 clinical NER models, including one specialized in radiology reports.

Category	Corpus	Package Name	Supported Entity Types
Bio	AnatEM	anatem	`ANATOMY`
	BC5CDR	bc5cdr	`CHEMICAL`, `DISEASE`
	BC4CHEMD	bc4chemd	`CHEMICAL`
	BioNLP13CG	bionlp13cg	16 types in Cancer Genetics (* see below for a full list)
	JNLPBA	jnlpba	`PROTEIN`, `DNA`, `RNA`, `CELL_LINE`, `CELL_TYPE`
	Linnaeus	linnaeus	`SPECIES`
	NCBI-Disease	ncbi_disease	`DISEASE`
	S800	s800	`SPECIES`
Clinical	i2b2-2010	i2b2	`PROBLEM`, `TEST`, `TREATMENT`
	Radiology	radiology	`ANATOMY`, `OBSERVATION`, `ANATOMY_MODIFIER`, `OBSERVATION_MODIFIER`, `UNCERTAINTY`

The 16 entity types in the BioNLP13CG model include, some of which are:

- ORGAN
- ORGANISM
- PROBLEM
- TREATMENT
- DISEASE
- CHEMICAL

Let's import stanza

```
!pip install stanza
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting stanza
  Downloading stanza-1.4.2-py3-none-any.whl (691 kB)
     |████████████████████████████████| 691 kB 6.9 MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from stanza) (4.64.1)
Requirement already satisfied: protobuf in /usr/local/lib/python3.7/dist-packages (from stanza) (3.17.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from stanza) (1.21.6)
Requirement already satisfied: torch>=1.3.0 in /usr/local/lib/python3.7/dist-packages (from stanza) (1.12.1+cu113)
Collecting emoji
  Downloading emoji-2.2.0.tar.gz (240 kB)
     |████████████████████████████████| 240 kB 37.9 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from stanza) (2.23.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from stanza) (1.15.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torch>=1.3.0->stanza) (4.1.1)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->stanza) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->stanza) (2022.9.24)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->stanza) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->stanza)
Building wheels for collected packages: emoji
  Building wheel for emoji (setup.py) ... done
  Created wheel for emoji: filename=emoji-2.2.0-py3-none-any.whl size=234927 sha256=2336e1483cbb5158ef67cb4d0f3e8b46372131260e688d1a3da0
  Stored in directory: /root/.cache/pip/wheels/f3/e3/f2/1de1c2e3ed742e1df73e0f15d58864e50c7e64f607b548d6cf
Successfully built emoji
Installing collected packages: emoji, stanza
Successfully installed emoji-2.2.0 stanza-1.4.2
```

```
import stanza
stanza.download('en', package='mimic',processors={'ner':'i2b2'})
stanza.download('en', package='mimic',processors={'ner':'bc5cdr'})
```

```
Downloading https://raw.githubusercontent.com/stanfordnlp/stanza-resources/main/resources_1.4.1.json: 193k/? [00:00<00:00, 1.59MB/s]
INFO:stanza:Downloading these customized packages for language: en (English)...
============================
| Processor       | Package |
----------------------------
| tokenize        | mimic   |
| pos             | mimic   |
| lemma           | mimic   |
| depparse        | mimic   |
| ner             | i2b2    |
| backward_charlm | mimic   |
| forward_charlm  | mimic   |
| pretrain        | mimic   |
============================

Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/tokenize/mimic.pt: 100% 645k/645k [00:00<00:00, 897kB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/pos/mimic.pt: 100% 39.2M/39.2M [00:00<00:00, 61.9MB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/lemma/mimic.pt: 100% 4.19M/4.19M [00:00<00:00, 16.5MB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/depparse/mimic.pt: 100% 109M/109M [00:01<00:00, 81.7MB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/ner/i2b2.pt: 100% 49.8M/49.8M [00:00<00:00, 68.6MB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/backward_charlm/mimic.pt: 100% 18.9M/18.9M [00:00<00:00, 33.5MB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/forward_charlm/mimic.pt: 100% 18.9M/18.9M [00:00<00:00, 38.0MB/s]
Downloading https://huggingface.co/stanfordnlp/stanza-en/resolve/v1.4.1/models/pretrain/mimic.pt: 100% 82.7M/82.7M [00:01<00:00,
```

Because, within the data we already have drug name, we need PROBLEM and DISEASE entity identification only, we are downloading only two NER models

1. **i2b2**
2. **bc5cdr**

So, let us create two pipeline with these

```
| Processor       | Package |
nlp1 = stanza.Pipeline('en', package='mimic', processors={'ner':'i2b2'})
nlp2= stanza.Pipeline('en', package='mimic', processors={'ner':'bc5cdr'})
```

```
INFO:stanza:Checking for updates to resources.json in case models have been updated.  Nc
```

```
Downloading                                                    193k/?
https://raw.githubusercontent.com/stanfordnlp/stanza-          [00:00<00:00,
resources/main/resources_1.4.1.json:                           2.63MB/s]
```

```
INFO:stanza:Loading these models for language: en (English):
=======================
| Processor | Package |
-----------------------
| tokenize  | mimic   |
| pos       | mimic   |
| lemma     | mimic   |
| depparse  | mimic   |
| ner       | i2b2    |
=======================

INFO:stanza:Use device: cpu
INFO:stanza:Loading: tokenize
INFO:stanza:Loading: pos
INFO:stanza:Loading: lemma
INFO:stanza:Loading: depparse
INFO:stanza:Loading: ner
INFO:stanza:Done loading processors!
INFO:stanza:Checking for updates to resources.json in case models have been updated.  Nc
```

```
Downloading                                                    193k/?
https://raw.githubusercontent.com/stanfordnlp/stanza-          [00:00<00:00,
resources/main/resources_1.4.1.json:                           4.29MB/s]
```

```
INFO:stanza:Loading these models for language: en (English):
=======================
| Processor | Package |
-----------------------
| tokenize  | mimic   |
| pos       | mimic   |
| lemma     | mimic   |
| depparse  | mimic   |
| ner       | bc5cdr  |
=======================

INFO:stanza:Use device: cpu
INFO:stanza:Loading: tokenize
INFO:stanza:Loading: pos
```

```
#Let's check a sample sentence with i2b2
doc = nlp1("This is my first time using any form of birth control. I&#039;m glad I went with the patch, I have been on it for 8 months. At f
doc.entities
```

```
[{
   "text": "birth control",
   "type": "TREATMENT",
   "start_char": 40,
   "end_char": 53
 }, {
   "text": "the patch",
   "type": "TREATMENT",
   "start_char": 81,
   "end_char": 90
 }, {
   "text": "my cramps",
   "type": "PROBLEM",
   "start_char": 305,
   "end_char": 314
 }, {
   "text": "cramps",
   "type": "PROBLEM",
   "start_char": 372,
   "end_char": 378
 }, {
   "text": "birth control",
   "type": "TREATMENT",
   "start_char": 392,
   "end_char": 405
 }, {
   "text": "the patch",
   "type": "TREATMENT",
   "start_char": 437,
   "end_char": 446
 }]
```

So if we see what is being detected through NER,

- **cramps (problem)**
- **birth control (treatment)**
- **patch (treatment)**

Named Entity Recognition:

" This is my first time using any form of birth control .
I& # 039 ; m glad I went with the patch , I have been on it for 8 months .
At first
It decreased my libido but that subsided .
The only downside is that it made my periods longer (5 - 6 days to be exact) I used to only have periods for 3 - 4 days max also made my cramps intense for the first two days of my period , I never had cramps before using birth control .
Other than that in happy with the patch "

Let's do the same with **bc5cdr**

```
#Let's check a sample sentence with i2b2
doc = nlp2("This is my first time using any form of birth control. I&#039;m glad I went with the patch, I have been on it for 8 months. At f
doc.entities
```

```
[{
   "text": "cramps",
   "type": "DISEASE",
   "start_char": 308,
   "end_char": 314
 }, {
   "text": "cramps",
   "type": "DISEASE",
   "start_char": 372,
   "end_char": 378
 }]
```

Named Entity Recognition:

This is my first time using any form of birth control .
I& #039 ; m glad I went with the patch , I have been on it for 8 months .
At first
It decreased my libido but that subsided .
The only downside is that it made my periods longer (5 - 6 days to be exact) I used to only have periods for 3 - 4 days max also made my cramps intense for the first two days of my period , I never had cramps before using birth control .
Other than that in happy with the patch

We see almost the same results, just the name of the entities differ. We will choose bc5cdr, as of now.

```
#slicing the first 50 rows, keeping the time factor in mind

df = raw_df.head(50)
df['Side-Effects']=''

for i, row in df.iterrows():
    doc=nlp2(row["review"])
    doc_entity=doc.entities
    #print(doc_entity)
    ae=[]

    for token in doc_entity:
        #print(token.type)
        if token.type=="DISEASE":
            ae.append(token.text)
            #print(ae)

    df.at[i,'Side-Effects']=ae
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  after removing the cwd from sys.path.
```