Hi and welcome to the Covera NLP exercise!!

Data Science Exercise

Hello! On behalf of Covera Health, we are excited that you are considering joining our team!

Part of your interview process consists in solving a data science exercise.

Please make sure you have gone through the setup process as defined in the README.md.

We really hope that you will enjoy working on this exercise with us; it's a small glimpse into what we do here on a day-to-day basis! Have fun! :)

Background/Context

Radiology has been transformative in patient care. Being able to identify pathologies for diagnoses through advanced technologies such as MRI and CT has helped countless lives. However, Radiology is a very complicated discipline, and even the most qualified radiologists make mistakes from time to time while interpreting images and describing their findings in diagnostic reports. For example, a recent study published in the Spine Journal demonstrated error rates of lumbar MRIs to be as high as 43%. These errors in diagnosis have significant impact to patient care, as all of the downstream treatment decisions are predicated on the initial diagnosis.

Covera Health is focused on improving patient outcomes by estimating radiology provider error rates. We rely on expert reviews of selected medical diagnostic reports to inform our error rate estimates. Our task is to analyse diagnostic reports automatically to augment the expert reviews.

A first step in what we do is to extract and identify the diseases mentioned in a biomedical test. For the purposes of this exercise In this notebook, we'll be using a dataset from the NCBI disease database (https://www.ncbi.nlm.nih.gov/CBBresearch/Dogan/DISEASE/) which is publicly available and contains 6,293 English sentences from 793 PubMed abstracts. Each sentence has been annotated by the named diseases they contain. Our goal to is develop a model which can automatically extract named diseases from the text.

1. Let's load the data and give it a browse

In [1]:
```
import pandas as pd

data = pd.read_csv('data/ncbi_disease_dataset.csv',encoding='latin1')

data.fillna(method="ffill",inplace=True)
print('Total Number of Sentences: ',data['Sentence #'].unique().shape[0
])
```

```
Total Number of Sentences:  6293
```

In [2]:
```
data.head(10)
```

Out[2]:

	Word	Tag	Sentence #
0	Identification	O	Sentence :0
1	of	O	Sentence :0
2	APC2	O	Sentence :0
3	,	O	Sentence :0
4	a	O	Sentence :0
5	homologue	O	Sentence :0
6	of	O	Sentence :0
7	the	O	Sentence :0
8	adenomatous	B-Disease	Sentence :0
9	polyposis	I-Disease	Sentence :0

In [3]:
```
data['Tag'].value_counts()/data['Tag'].value_counts().sum()
```

Out[3]:
```
O            0.917684
I-Disease    0.044972
B-Disease    0.037344
Name: Tag, dtype: float64
```

Looks like we've got a very sparse dataset, with 'O' (outside of entity) being the vast majority of tags, which makes sense. A lot of langauge is grammar, pronouns, broader ideas, etc. and not named entities.

What are the different entities?

In [4]:

```
all_tags = data['Tag'].unique().tolist()
all_tags.remove('O')
all_tags.sort(key = lambda x: x.split('-')[1])
for tag in all_tags:
    print('________')
    print(tag)
    print('________')
    print(data[data['Tag']==tag]['Word'].head(20))
```

```
________
B-Disease
________
8          adenomatous
15         adenomatous
61               colon
173              colon
197             cancer
208              HNPCC
221        colorectal
244        hereditary
251              HNPCC
308        colorectal
332              HNPCC
354        colorectal
367        colorectal
476              HNPCC
487        colorectal
534            cancers
537        colorectal
553        colorectal
603       endometrial
615     premenopausal
Name: Word, dtype: object
________
I-Disease
________
9      polyposis
10          coli
11        tumour
16     polyposis
17          coli
18             (
19           APC
20             )
21        tumour
62     carcinoma
174    carcinoma
222       cancer
245          non
246            -
247    polyposis
248       cancer
249     syndrome
309       cancer
368       cancer
488            ,
Name: Word, dtype: object
```

The NER tags follow standard tagging convention defined [here (https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging))](https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)) and B-disease is the beginning of a disease tag and an I-Disease in insidea a disease tag. For eg: **colorectal cancer** is comprised of two separate tags as follows: colorectal: **B-Disease and cancer: I-Disease .**

In [5]:
```
data['Tag'].value_counts()/data['Tag'].value_counts().sum()
```

Out[5]:
```
O            0.917684
I-Disease    0.044972
B-Disease    0.037344
Name: Tag, dtype: float64
```

In [6]:
```
data.describe(
)
```

Out[6]:

	Word	Tag	Sentence #
count	184447	184447	184447
unique	10798	3	6293
top	.	O	Sentence :981
freq	8361	169264	204

2. Preparing text for model inputs

Question 1: How should you prepare the data for modeling?

In [7]:
```
# since we are doing sentence level, we will make the each value in X a
 list of the tokens for each sentence
# and for y we will make each value a list of the tags for the tokens in
the corresponding sentence

# we will also lower each token

X = data.groupby('Sentence #').Word.apply(lambda x: [s.lower() for s in
x])
y = data.groupby('Sentence #').Tag.apply(list)
```

In [8]:
```
X.head()
```

Out[8]:
```
Sentence #
Sentence :0        [identification, of, apc2, ,, a, homologue, of...
Sentence :1        [., the, adenomatous, polyposis, coli, (, apc,...
Sentence :10       [., a, common, msh2, mutation, in, english, an...
Sentence :100      [., we, report, that, heterozygosity, for, a, ...
Sentence :1000     [., eleven, of, the, 16, homozygous, subjects,...
Name: Word, dtype: object
```

```
In [9]: y.head()
```

```
Out[9]: Sentence #
        Sentence :0       [O, O, O, O, O, O, O, O, B-Disease, I-Disease,...
        Sentence :1       [O, O, B-Disease, I-Disease, I-Disease, I-Dise...
        Sentence :10      [O, O, O, O, O, O, O, O, O, O, B-Disease, O, O...
        Sentence :100     [O, O, O, O, O, O, O, O, O, O, O, O, O, O, O, ...
        Sentence :1000    [O, O, O, O, O, O, O, O, O, O, O, O, O, B-Dise...
        Name: Tag, dtype: object
```

```
In [10]: #Remove cell
         #We're going to be creating a sequence based model with batching, which
          means we need to define a maximum sequence length
         max_len = 75
```

```
In [11]: data.groupby('Sentence #').Word.count().hist()
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7febf06150d0>
```

```
In [12]: # To make the word2idx dictionary we first get all of the words and map
          the to lewer strings
         # then we sort by their frequency (in case we want to drop infrequent to
         kens and reduce our vocab size)
         # This will give us a dataframe with the word as the index, and the coun
         ts as the column
         # In order to give it a unique idx, we create a new column where the val
         ue is just the order of the word
         # now we can convert this to a dictionary, where the key is a word and t
         he value is its unique index

         word_counts = data.Word.dropna().map(str.lower).value_counts().rename('c
         ounts').to_frame()
         word_counts['index_num'] = range(1, word_counts.shape[0]+1)
         word2idx = word_counts['index_num'].to_dict()

         # we'll set a special idx for tokens that were not part of the original
          vocab
         word2idx['UNK'] = 0
         # we'll also set a special idx to pad our sentence vectors
         word2idx['PAD'] = -1
```

```
In [16]: tag_counts = data.Tag.dropna().value_counts().rename('counts').to_frame
         ()
         tag_counts['index_num'] = range(0, tag_counts.shape[0])
         tag2idx = tag_counts['index_num'].to_dict()

         tag2idx['PAD'] = -1

         tag2idx
```

```
Out[16]: {'O': 0, 'I-Disease': 1, 'B-Disease': 2, 'PAD': -1}
```

Question 2: Why do we need a max length? We set the maximum sequence length to 75 words, do you agree with this choice, why or why not?

Assuming we're translation words and labels to indices write some code to create a hash table that can map our strings to numerical values.

In [17]:
```
# word2idx =
# tag2idx =
```

In [18]:
```
from keras.preprocessing.sequence import pad_sequences

import numpy as np
X_word = [[word2idx[w] for w in s] for s in X.tolist()]
X_word = pad_sequences(maxlen=max_len, sequences=X_word, value=word2idx[
"PAD"], padding='post', truncating='post')
```

In [ ]:
```
```

We'll do the same thing for our `TAG` values which are the labels for the named entities

In [19]:
```
y = [[tag2idx[w] for w in s] for s in y]
y = pad_sequences(maxlen=max_len, sequences=y, value=tag2idx["PAD"], pad
ding='post', truncating='post')
```

Bonus question: let's assume there are lots of misspellings in our training data, and we have no way to correct them before they get to the model. How would you deal with this if at all?

In [ ]:
```
# As opposed to something like a dictionary-based method where you might
be able
# to return a result for an exact phrase match,
# Instead statistical named entity recognition is more robust to mispell
ings b/c the model
# learns to predict the tag not just on the token istelf but also on the
neighboring tokens as well
```

We've got the data into a format that can be fed into our model. Now let's decide how to split our data to train, validate and test our model. Write some code to split the data.

```
In [ ]: from sklearn
        def create_splits(X, Y, ):
            """creates function that takes in X, Y"""
            pass





        X_train =
        X_test =
        X_val =
        y_train =
        y_test =
        y_val =
```

Question 3: Now that we've got our data prepared, we're going to start building a model. Are there any additional steps we should take before feeding our data to a model?

```
In [ ]: # if we are utilizing the tokens as is, I would utilize
```

Question 4: How do we ensure that our model generalizes and that our metrics are representative of real world data?

Question 5: How would you select the best model for the problem?

```
- What are some considerations when selecting the best model?
- What are some things you can do to ensure that the model you've selected i
s indeed the best one?
```

Build a Model

```
In [ ]: def create_model():
            model = None
            return model
```

Question 6: Explain whether we should use the following code, and if so, why?

In [ ]:
```python
from keras.callbacks import EarlyStopping,ModelCheckpoint
from keras.models import load_model

model = create_model()

callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=3
    ),
    ModelCheckpoint(
        filepath='models/ner_checkpoint_test.h5',
        monitor='val_loss',
        save_best_only=True
    )
]



history = model.fit(X_train,
                    y_train,
                    validation_split=0.1, #change to validation data
                    callbacks = callbacks,
                    verbose=1
)
```

3. Validation

Now let's assume we've got predictions out of our model. We'll load some dummy predictions to evaluate the performance of our hypothetical model.

In [15]:
```python
#load in prepared predictions and evaluate how well we predicted
predictions = pd.read_csv('predictions/ncbi_disease_predictions.csv')
tag2idx = dict()
for idx,i in enumerate(sorted(predictions['Tag'].unique().tolist())):
    tag2idx[i] = idx
predictions['true_index'] = predictions['Tag'].map(tag2idx)
predictions['predicted_index'] = predictions['predicted'].map(tag2idx)
```

```
In [16]: predictions.head()
```

Out[16]:

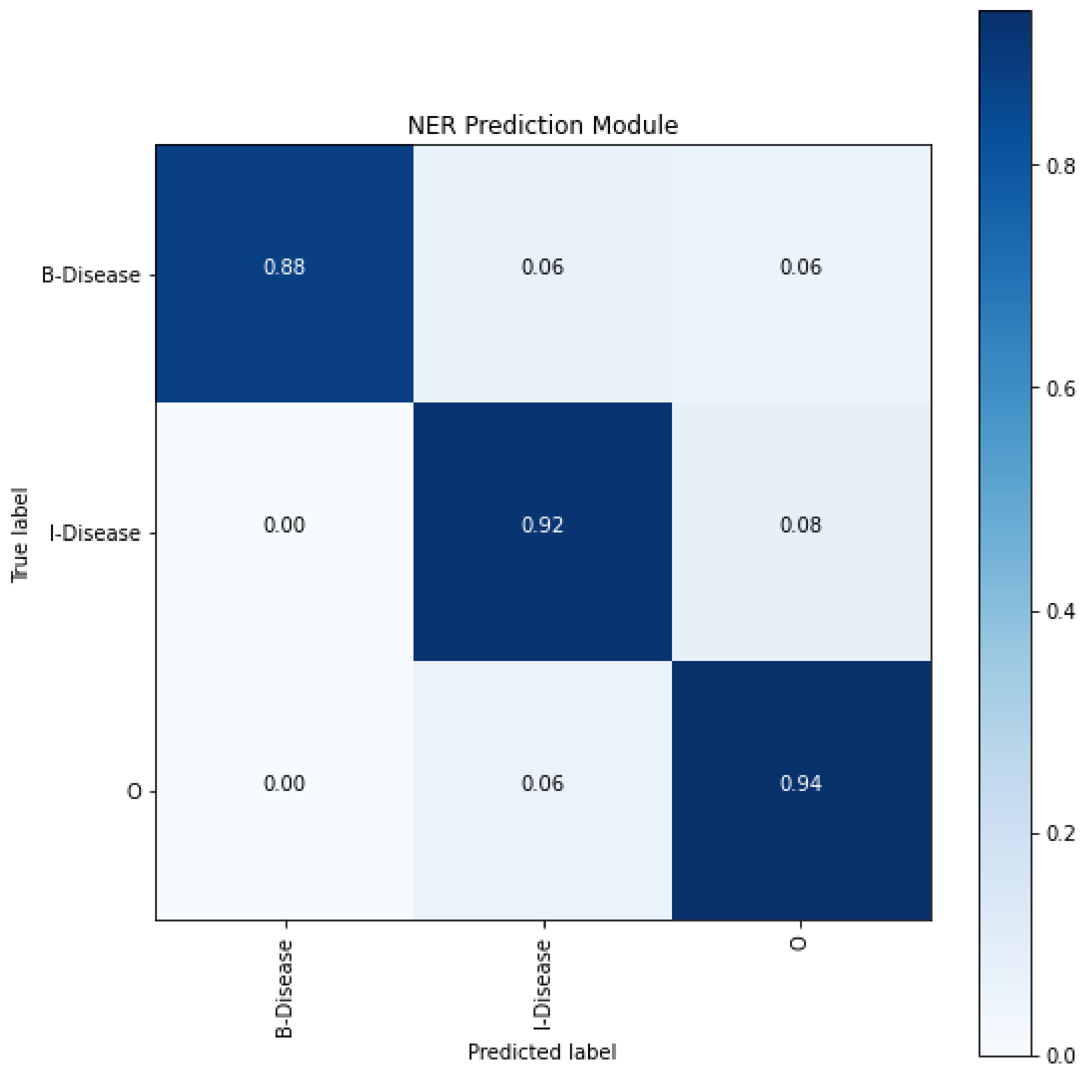
	Unnamed: 0	Word	Tag	Sentence #	true_index	predicted_index	predicted
0	0	Clustering	O	Sentence :0	2	2	O
1	1	of	O	Sentence :0	2	2	O
2	2	missense	O	Sentence :0	2	2	O
3	3	mutations	O	Sentence :0	2	2	O
4	4	in	O	Sentence :0	2	2	O

And we'll use a confusion matrix to see how we did on each prediction.

In [17]:
```
import matplotlib.pyplot as plt
import numpy as np
import itertools

def plot_confusion_matrix(cm,
                          classes,
                          normalize=False,
                          title='Confusion matrix',
                          save_file = 'confusion_matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    --inputs--
    cm - the confusion matrix produced by sklearns confusion_matrix func
tion
    classes - the labels for the classes in the confusion matrix

    --outputs--
    matplotlib figure displaying the confusion matrix, and the saved png
of the figure

    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    #print(cm)
    plt.figure(figsize=(8,8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)



    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1
])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.savefig('figures/{}.png'.format(save_file))
    plt.show()
```

In [18]:
```python
from sklearn.metrics import confusion_matrix
def switch_keys_values_dict(d):
    switch_dict = dict()
    for t in tag2idx.keys():
        switch_dict[d[t]] = t
    return switch_dict

idx2tag = switch_keys_values_dict(tag2idx)

labels = list()
for i in range(0,len(idx2tag)):
    labels.append(idx2tag[i])

test_labels = predictions['true_index'].values
pred_labels = predictions['predicted_index'].values

cnf_matrix = confusion_matrix(test_labels,pred_labels)
cnf_title = 'NER Prediction Module'
save_all = 'confusion_matrix_all'
plot_confusion_matrix(
    cnf_matrix,
    classes=labels,
    normalize=True,
    title=cnf_title,
    save_file=save_all
)
```

```
Normalized confusion matrix
```



Question 7: Can you think of some single metrics that summarize the performance of our model? What are the tradeoffs we make when we optimize for certain metrics? Can you think of pros and cons of using different metrics?

Question 8: We are seeing some strange results where words that are not diseases are being classified as diseases. The clinical members on the team who are experts in medical language are happy to help you troubleshoot. What would you do to best use their time?

```
In [ ]:
```

4. Named Entity Prediction Module

Now that you've built a model, our engineering team needs to use it on raw text to generate predictions. With pseudo code, explain how you would make this model available to teams outside of AI/Data Science

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Question 9: What would you do if you had 10x the data? What would you do if you had 1/10 the data?

Question 10: what happens if your model starts doing poorly? How would you understand that your model is doing poorly? How do you measure data drift without labels for new data?

Bonus questions

Bonus question 1: What if instead of predicting the disease Tag we had to predict the disease directly?

Bonus question 2: How would you adapt your model from PubMED abstracts to extracting diseases from clinical notes?

Bonus question 3: Imagine you had access to all the data in the world and an unlimited annotation budget can you design your ideal dataset?

Now that you've built a model, our engineering team needs to use it on raw text to generate predictions. With pseudo code, explain how you would make this model available to teams outside of AI/Data Science

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Question 9: What would you do if you had 10x the data? What would you do if you had 1/10 the data?

Question 10: what happens if your model starts doing poorly? How would you understand that your model is doing poorly? How do you measure data drift without labels for new data?

Bonus questions

Bonus question 1: What if instead of predicting the disease Tag we had to predict the disease directly?

Bonus question 2: How would you adapt your model from PubMED abstracts to extracting diseases from clinical notes?

Bonus question 3: Imagine you had access to all the data in the world and an unlimited annotation budget can you design your ideal dataset?

```
In [ ]:
```