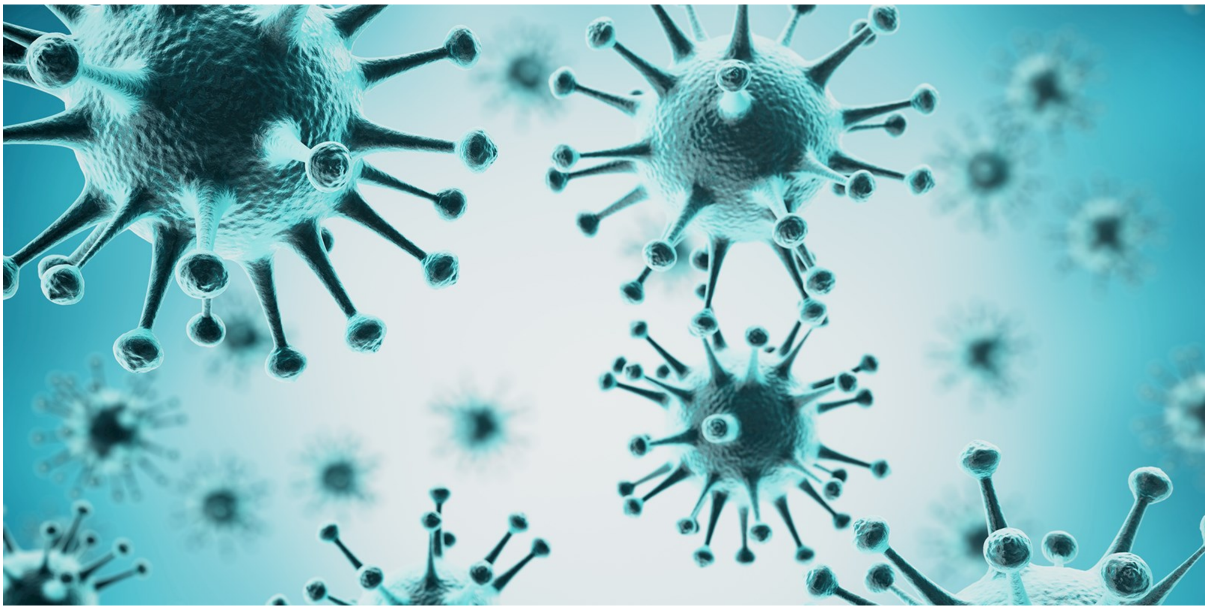


Artificial Neural Network

PART 1: Data Preprocessing

In [1]:

```
# Importing the libraries
import numpy as np
import pandas as pd
import tensorflow as tf
tf.__version__
```

Out[1]:

```
'2.1.0'
```

In [2]:

```
# Importing the dataset
dataset = pd.read_csv('symptom_frequency.csv')
X = dataset.iloc[:,:20].values
y = dataset.iloc[:, -1].values

display(dataset.head())
print('');
dataset.info()
```

	Breathing Problem	Fever	Dry Cough	Sore throat	Running Nose	Asthma	Chronic Lung Disease	Headache	Heart Disease	Diabetes
0	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes
1	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No
2	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
3	Yes	Yes	Yes	No	No	Yes	No	No	Yes	Yes
4	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes

5 rows × 21 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5434 entries, 0 to 5433
Data columns (total 21 columns):
Breathing Problem                          5434 non-null object
Fever                                      5434 non-null object
Dry Cough                                  5434 non-null object
Sore throat                                5434 non-null object
Running Nose                               5434 non-null object
Asthma                                     5434 non-null object
Chronic Lung Disease                       5434 non-null object
Headache                                   5434 non-null object
Heart Disease                              5434 non-null object
Diabetes                                   5434 non-null object
Hyper Tension                              5434 non-null object
Fatigue                                    5434 non-null object
Gastrointestinal                           5434 non-null object
Abroad travel                              5434 non-null object
Contact with COVID Patient                 5434 non-null object
Attended Large Gathering                   5434 non-null object
Visited Public Exposed Places              5434 non-null object
Family working in Public Exposed Places    5434 non-null object
Wearing Masks                              5434 non-null object
Sanitization from Market                   5434 non-null object
COVID-19                                   5434 non-null object
dtypes: object(21)
memory usage: 891.6+ KB
```

In [3]:

```
dataset['COVID-19'].value_counts(normalize=True)
```

Out[3]:

```
Yes    0.806588
No     0.193412
Name: COVID-19, dtype: float64
```

ENCODING CATEGORICAL DATA

In [4]:

```
# Label Encoding to binary column, from Yes/no -> 1/0
# X's
i = 0
for i in dataset.columns:
    dataset[i] = dataset[i].replace('No', 0).replace('Yes', 1)

X = dataset[[x for x in dataset.columns if x not in 'COVID-19']]

# y
y = dataset['COVID-19'].replace('No', 0).replace('Yes',1)
```

In [5]:

```
from sklearn.model_selection import StratifiedShuffleSplit

# Initialize object
sss = StratifiedShuffleSplit(n_splits=1, test_size = 0.3)

# Create indexes
train_index, test_index = next(sss.split(X, y))

# Assign set
X_train, X_test = X.loc[train_index], X.loc[test_index]
y_train, y_test = y.loc[train_index], y.loc[test_index]
```

PART 2: Building the ANN

In [6]:

```
# Initializing the ANN
ann = tf.keras.models.Sequential()

# Adding the input layer and the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the second hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

# Adding the output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

PART 3: Training the ANN

In [7]:

```
# Compiling the ANN
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [9]:

```
# Training the ANN on the Training set
ann.fit(X_train.values, y_train.values, batch_size = 32, epochs = 10)
```

```
Train on 3803 samples
Epoch 1/10
3803/3803 [==============================] - 0s 54us/sample - loss: 0.0613 -
accuracy: 0.9740
Epoch 2/10
3803/3803 [==============================] - 0s 52us/sample - loss: 0.0584 -
accuracy: 0.9753
Epoch 3/10
3803/3803 [==============================] - 0s 51us/sample - loss: 0.0588 -
accuracy: 0.9748
Epoch 4/10
3803/3803 [==============================] - 0s 54us/sample - loss: 0.0576 -
accuracy: 0.9740
Epoch 5/10
3803/3803 [==============================] - 0s 55us/sample - loss: 0.0561 -
accuracy: 0.9776
Epoch 6/10
3803/3803 [==============================] - 0s 54us/sample - loss: 0.0558 -
accuracy: 0.9769
Epoch 7/10
3803/3803 [==============================] - 0s 54us/sample - loss: 0.0543 -
accuracy: 0.9771
Epoch 8/10
3803/3803 [==============================] - 0s 53us/sample - loss: 0.0540 -
accuracy: 0.9784
Epoch 9/10
3803/3803 [==============================] - 0s 53us/sample - loss: 0.0541 -
accuracy: 0.9771
Epoch 10/10
3803/3803 [==============================] - 0s 53us/sample - loss: 0.0530 -
accuracy: 0.9774
```

Out[9]:

```
<tensorflow.python.keras.callbacks.History at 0x2e8bd62b388>
```

PART 4: Making the predictions

In [11]:

```
# Predicting the Test set results
y_pred = ann.predict(X_test)
y_pred
```

Out[11]:

```
array([[0.9962257 ],
       [1.        ],
       [0.99999964],
       ...,
       [0.10707814],
       [0.9977544 ],
       [1.        ]], dtype=float32)
```

PART 5: Interpreting

In [101]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
y_pred_cm = (y_pred > 0.5)
cm = confusion_matrix(y_test, y_pred_cm)
print(cm)
```

```
[[ 302   13]
 [  15 1301]]
```

Precision = tp / tp + fp, measuring how many true positives it predicted out of all predicted positives Recall = tp / tp + + fn, measuring how many true positives it predicted out of all positives F1-score 2 * (Precision + Recall) / Precision + Recall, pretty much just balances the metrics

In [102]:

```
# Classification report
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_scc

print(classification_report(y_test, y_pred_cm))
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       315
           1       0.99      0.99      0.99      1316

    accuracy                           0.98      1631
   macro avg       0.97      0.97      0.97      1631
weighted avg       0.98      0.98      0.98      1631
```

It seems it predicts positive too often. However, this is preferable in the COVID situation

In [111]:

```
# Correct vs wrong, index of wrongs to compare what went wrong
correct = 0
wrong = 0
index_of_wrongs = []
i = 0

#looping
for j in y_pred_cm:
    ii = y_test.index[i]
    if (j == True) and (y_test[ii] == 1):
        correct += 1
    elif (j == False) and (y_test[ii] == 0):
        correct += 1
    elif (j == True) and (y_test[ii] == 0):
        wrong += 1
        index_of_wrongs.append(ii)
    elif (j== False) and (y_test[ii] ==1):
        wrong += 1
        index_of_wrongs.append(ii)
    i += 1

# Inspect it
index_of_wrongs;
```

PART 6: Saving and loading model

In [104]:

```
# Save model
ann.save('Covid_model_2021')
```

```
WARNING:tensorflow:From C:\Users\brosb\AppData\Roaming\Python\Python37\site-
packages\tensorflow_core\python\ops\resource_variable_ops.py:1786: calling B
aseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_o
ps) with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: Covid_model_2021\assets
```

In [112]:

```
# Load model
from tensorflow import keras

loaded_model = keras.models.load_model('Covid_model_2021')
```

In [113]:

```
# Trying it out
y_pred = loaded_model.predict(X_test.values)
y_pred = (y_pred > 0.5) # MAYBE LEAVE THIS PART OUT, SO WE GET PROBABILITIES
```

In [114]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[ 302   13]
 [  15 1301]]
```

Appendix A: Hyperparameter search

In [35]:

```
# Importing the libraries
import numpy as np
import pandas as pd
import tensorflow as tf
tf.__version__

# Importing the dataset
dataset = pd.read_csv('symptom_frequency.csv')
X = dataset.iloc[:,:20].values
y = dataset.iloc[:, -1].values

# Label Encoding to binary column, from Yes/no -> 1/0
i = 0
for i in dataset.columns:
    dataset[i] = dataset[i].replace('No', 0).replace('Yes', 1)
X = dataset[[x for x in dataset.columns if x not in 'COVID-19']]
y = dataset['COVID-19'].replace('No', 0).replace('Yes',1)

# Train test split
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size = 0.3)
train_index, test_index = next(sss.split(X, y))

# Assign set
X_train, X_test = X.loc[train_index], X.loc[test_index]
y_train, y_test = y.loc[train_index], y.loc[test_index]
```

In [ ]:

```
# from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_s
from sklearn import metrics

ann_versions_list = list()
ann_versions_dict = dict()

# ANN CREATION AND PREDICTION
def make_ann_predict(optimizer = 'adam', loss = 'binary_crossentropy',
                     epochs = 10, batch_size = 32, positive_limit = 0.7,
                     description = ''):

    # ANN
    ann = tf.keras.models.Sequential()
    ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
    ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
    ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
    ann.compile(optimizer = optimizer, loss = loss, metrics = ['Accuracy'])
    ann.fit(X_train.values, y_train.values, batch_size = batch_size, epochs = epochs)
    y_pred = ann.predict(X_test)

    # METRICS
    y_pred_binary = (y_pred > positive_limit)
    if description not in ann_versions_dict.keys():
        ann_versions_list.append(description)
        ann_versions_dict[description]  = y_pred_binary

# Metrics to try out
optimizer_list = ['Adam', 'SGD', 'Adadelta', 'Adagrad', 'Adamax', 'Nadam', 'Ftrl']
epochs_list = list(range(50, 200, 50))
positive_limit_list = list(np.arange(0.0, 1.1, 0.25))
```

In [51]:

```
# Run the models
for limit in positive_limit_list:
    for epochs in epochs_list:
        for optimizer in optimizer_list:
            make_ann_predict(optimizer = optimizer, epochs = epochs, positive_limit = limit
```

```
Train on 3803 samples
Epoch 1/50
3803/3803 [==============================] - 1s 217us/sample - loss: 0.582
8 - Accuracy: 0.0000e+00
Epoch 2/50
3803/3803 [==============================] - 0s 63us/sample - loss: 0.3958
- Accuracy: 0.0000e+00
Epoch 3/50
3803/3803 [==============================] - 0s 63us/sample - loss: 0.2877
- Accuracy: 0.0000e+00
Epoch 4/50
3803/3803 [==============================] - 0s 67us/sample - loss: 0.1816
- Accuracy: 0.0000e+00
Epoch 5/50
3803/3803 [==============================] - 0s 71us/sample - loss: 0.1314
- Accuracy: 0.0000e+00
Epoch 6/50
3803/3803 [==============================] - 0s 57us/sample - loss: 0.1094
- Accuracy: 0.0000e+00
Epoch 7/50
```

In [87]:

```
ann_versions_list
```

Out[87]:

```
['Adam, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'SGD, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'Adadelta, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'Adagrad, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'Adamax, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'Nadam, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'Ftrl, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit',
 'Adam, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'SGD, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'Adadelta, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'Adagrad, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'Adamax, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'Nadam, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'Ftrl, binary crossentropy, 100 epochs, 32 batchsize, 0.0 limit',
 'Adam, binary crossentropy, 150 epochs, 32 batchsize, 0.0 limit',
 'SGD, binary crossentropy, 150 epochs, 32 batchsize, 0.0 limit',
 'Adadelta, binary crossentropy, 150 epochs, 32 batchsize, 0.0 limit',
 'Adagrad, binary crossentropy, 150 epochs, 32 batchsize, 0.0 limit',
```

In [76]:

```
f1_score_list = list()

for version in ann_versions_list:
    print(version, '\n')
    print(classification_report(y_test, ann_versions_dict[version]))
    total_f1_score = classification_report(y_test, ann_versions_dict[version], output_dict
    print('Total f1- score: ', total_f1_score)
    f1_score_list.append(total_f1_score)
    print('---------------------------------------------------------------------------')
```

```
-
Adagrad, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       315
           1       0.81      1.00      0.89      1316

    accuracy                           0.81      1631
   macro avg       0.40      0.50      0.45      1631
weighted avg       0.65      0.81      0.72      1631

Total f1- score:  0.8931116389548694
--------------------------------------------------------------------------
-
Adamax, binary crossentropy, 50 epochs, 32 batchsize, 0.0 limit

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       315
```

Find max f1-score

In [84]:

```
max_f1 = max(f1_score_list)
[i for i, j in enumerate(f1_score_list) if j == max_f1]
```

Out[84]:

```
[49]
```

The best ANN version

In [94]:

```
print(ann_versions_list[49])
print('\n')
print(classification_report(y_test, ann_versions_dict[ann_versions_list[49]]))
```

```
Adam, binary crossentropy, 100 epochs, 32 batchsize, 0.5 limit


              precision    recall  f1-score   support

           0       0.94      0.96      0.95       315
           1       0.99      0.99      0.99      1316

    accuracy                           0.98      1631
   macro avg       0.97      0.97      0.97      1631
weighted avg       0.98      0.98      0.98      1631
```

Appendix B: Correlations

In [64]:

```
df.columns
```

Out[64]:

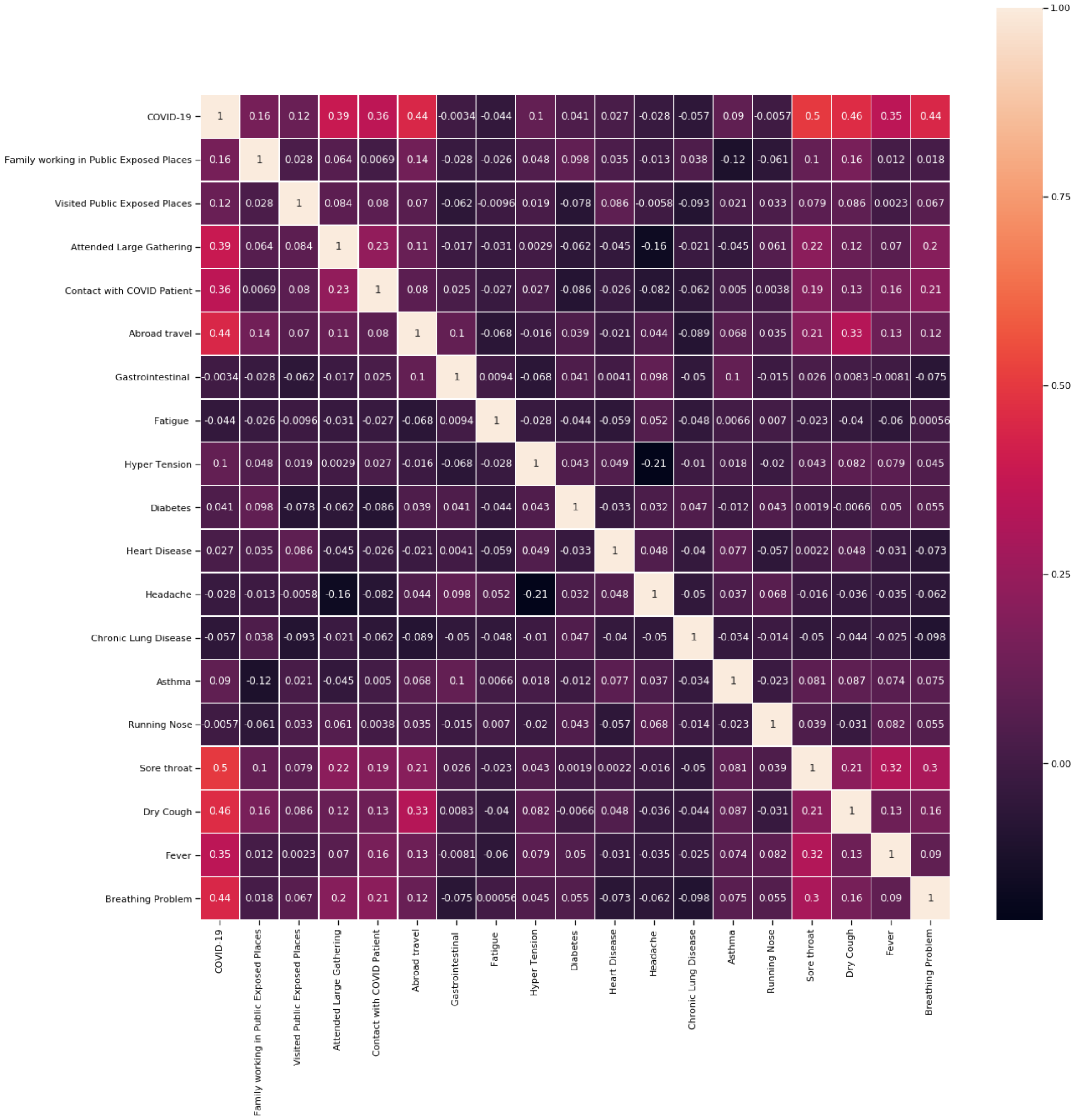
```
Index(['Breathing Problem', 'Fever', 'Dry Cough', 'Sore throat',
       'Running Nose', 'Asthma', 'Chronic Lung Disease', 'Headache',
       'Heart Disease', 'Diabetes', 'Hyper Tension', 'Fatigue ',
       'Gastrointestinal ', 'Abroad travel', 'Contact with COVID Patient',
       'Attended Large Gathering', 'Visited Public Exposed Places',
       'Family working in Public Exposed Places', 'Wearing Masks',
       'Sanitization from Market', 'COVID-19'],
      dtype='object')
```

In [71]:

```
# Importing the dataset
df = pd.read_csv('symptom_frequency.csv')

# Label Encoding to binary column, from Yes/no -> 1/0
# X's
i = 0
for i in df.columns:
    df[i] = df[i].replace('No', 0).replace('Yes', 1)

df.drop('Wearing Masks', inplace = True, axis = 1)
df.drop('Sanitization from Market', inplace = True, axis=1)
```

In [81]:

```
import seaborn as sn
import matplotlib.pyplot as plt

sn.set_context('notebook')

# Plot
plt.figure(figsize=(20,20))
ax = sn.heatmap(df.corr(), annot=True, linewidth=0.5)
ax.set_xlim([19,0])
ax.set_ylim([0,21])
plt.show()
```



TechLabs 2020-2021, AI FOR GOOD