Predicting Daily Cases Per Country

In this notebook data of a number of countries is used to train a model per each country to be used to predict the daily cases on a specifc day.

In [0]:

```
import numpy as np
import pickle
import pandas as pd
from pandas import DataFrame
```

In [0]:

```
countries = {"Germany", "France", "Italy"}
```

Loading the data

In [0]:

```
def load_pickle(file_name):
  with open(file_name, 'rb') as f:
    return pickle.load(f)
```

Mounting Google Drive

We mount google drive to access the data stored there

In [181]:

```
from google.colab import drive
drive.mount('./gdrive')
```

```
Drive already mounted at ./gdrive; to attempt to forcibly remount, call drive.mount("./gdrive",
force_remount=True).
```

In [0]:

```
drive_base_path = "./gdrive/My Drive"
data_path = "{}/COVID-19".format(drive_base_path)
```

Loading the Weather Data of the Countries and Joining Them

In [0]:

```
weather_data_base_path = "{}/weather-features".format(data_path)
```

In [0]:

```
wind_speed_dict = dict(filter(lambda x: x[0] in countries, load_pickle("{}/windspeedKmph_dict.pickle".format
(weather_data_base_path)).items()))
tempreture_dict = dict(filter(lambda x: x[0] in countries, load_pickle("{}/tempC_dict.pickle".format(weather
_data_base_path)).items()))
humidity_dict = dict(filter(lambda x: x[0] in countries, load_pickle("{}/humidity_dict.pickle".format(weathe
r_data_base_path)).items()))
sun_hour_dict = dict(filter(lambda x: x[0] in countries, load_pickle("{}/sunHour_dict.pickle".format(weather
_data_base_path)).items()))
```

In [0]:

```
germany_df = DataFrame()
germany_df['wind_speed'] = wind_speed_dict['Germany']
germany_df['tempreture'] = tempreture_dict['Germany']
germany_df['humidity'] = humidity_dict['Germany']
germany_df['sun_hour'] = sun_hour_dict['Germany']
```

In [0]:

```
italy_df = DataFrame()
italy_df['wind_speed'] = wind_speed_dict['Italy']
italy_df['tempreture'] = tempreture_dict['Italy']
italy_df['humidity'] = humidity_dict['Italy']
italy_df['sun_hour'] = sun_hour_dict['Italy']
```

In [0]:

```
france_df = DataFrame()
france_df['wind_speed'] = wind_speed_dict['France']
france_df['tempreture'] = tempreture_dict['France']
france_df['humidity'] = humidity_dict['France']
france_df['sun_hour'] = sun_hour_dict['France']
```

Loading "Our World in Data" Dataset and Merging it With the Weather Data

In [0]:

```
our_world_data_base_path = "{}/our-world-in-data".format(data_path)
```

In [0]:

```
def merge_dataframes_filter_with_date(main_df, to_be_merged_df, column_names, start_date, end_date):
  to_be_merged_df['date'] = pd.to_datetime(to_be_merged_df['date'])
  to_be_merged_df = to_be_merged_df[(to_be_merged_df['date'] >= start_date) & (to_be_merged_df['date'] <= en
d_date)]
  to_be_merged_df.index = [x for x in range(main_df.shape[0])]
  main_df['date'] = to_be_merged_df['date']
  for name in column_names:
    main_df[name] = to_be_merged_df[name]
```

In [0]:

```
new_cases_dict = dict(filter(lambda x: x[0] in countries, load_pickle("{}/new_cases_dict.pickle".format(our_
world_data_base_path)).items()))
```

In [0]:

```
new_deaths_dict = dict(filter(lambda x: x[0] in countries, load_pickle("{}/new_deaths_dict.pickle".format(ou
r_world_data_base_path)).items()))
```

We have to restrict the days of the data to be the same as the days of weather data to be able to join them

In [0]:

```
start_date = '2020-01-22'
end_date = '2020-03-21'
```

In [0]:

```
germany_new_cases_df = DataFrame(new_cases_dict['Germany'], columns=['date', 'new_cases'])
merge_dataframes_filter_with_date(germany_df, germany_new_cases_df, ['new_cases'], start_date, end_date)
germany_new_deaths_df = DataFrame(new_deaths_dict['Germany'], columns=['date', 'new_deaths'])
merge_dataframes_filter_with_date(germany_df, germany_new_deaths_df, ['new_deaths'], start_date, end_date)
```

In [0]:

```
italy_new_cases_df = DataFrame(new_cases_dict['Italy'], columns=['date', 'new_cases'])
merge_dataframes_filter_with_date(italy_df, italy_new_cases_df, ['new_cases'], start_date, end_date)
italy_new_deaths_df = DataFrame(new_deaths_dict['Italy'], columns=['date', 'new_deaths'])
merge_dataframes_filter_with_date(italy_df, italy_new_deaths_df, ['new_deaths'], start_date, end_date)
```

In [0]:

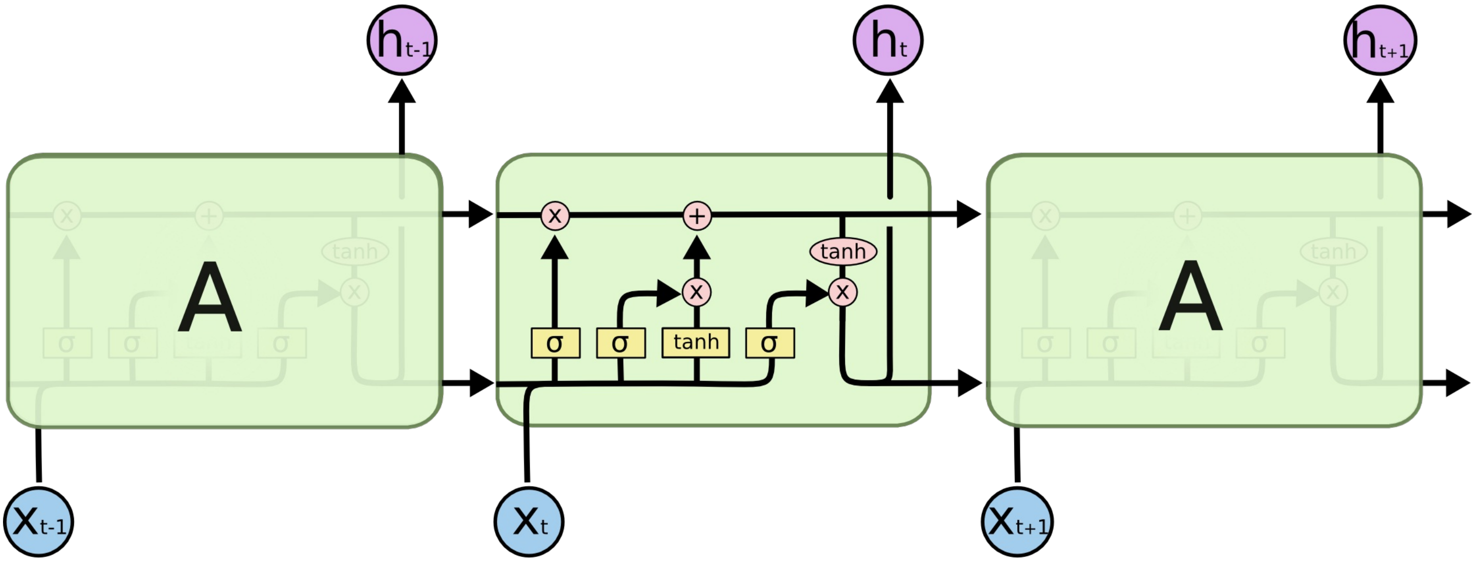
```
france_new_cases_df = DataFrame(new_cases_dict['France'], columns=['date', 'new_cases'])
merge_dataframes_filter_with_date(france_df, france_new_cases_df, ['new_cases'], start_date, end_date)
france_new_deaths_df = DataFrame(new_deaths_dict['France'], columns=['date', 'new_deaths'])
merge_dataframes_filter_with_date(france_df, france_new_deaths_df, ['new_deaths'], start_date, end_date)
```

Defining the Models

In this section we are going to try 2 approaches.
The first one is to consider the datat as a time series and to use LSTM in the model and train it to predict the cases of a day based on a previous day. The time series forecasting problem can be trained to have differnet types of inputs and outputs. One approach is to input a day and output a day. A second approach is to have an input sequence of days and output a day. Another approach is to have a input sequence of days and output another sequence of days. We are going to use the first approach with the input day as one day before the output day. The number of days the input is before the output day is a hyper parameter that can be changed to get the best results. This [paper (https://www.researchgate.net/publication/341089678_Neural_Network_Model_for_Prediction_of_Covid-19_Confirmed_Cases_and_Fatalities)](https://www.researchgate.net/publication/341089678_Neural_Network_Model_for_Prediction_of_Covid-19_Confirmed_Cases_and_Fatalities) is used as a reference for the parameters used in the LSTM model.
The second approach we are going to use is a fully connected neural network with one hidden layer. We use root mean sqaure error as the error function to evaluate the models.
The models are tested using Germany, France and Italy's data.

LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. Intuitively, the cell is responsible for keeping track of the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function of the LSTM gates is often the logistic sigmoid function.



In [0]:

```
import math
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.callbacks import EarlyStopping
```

Germany's Models

In [0]:

```
germany_df.drop(columns='date', inplace=True)
```

The LSTM Model

Preparing the train and Test Sets

First thing to do is to add a new column wich is a shift by one day of the new cases model to used as the output for the LSTM model for the training phase

In [0]:

```
germany_df['output_new_cases'] =germany_df['new_cases'].shift(-1)
germany_df.drop(axis=0,index=[germany_df.shape[0]-1], inplace=True)
```

We split the data into a training, test and valdiation sets 70%, 10%, 20% splits respectively, after normalizing the data using MinMaxScaler in order to remove the dominance of large valued features. The sets have o be reashaped to [examples, timesteps, features] to be used as inputs to the LSTM model. In our case the timesteps used is 1 as only 1 day is considered as input not a sequence of days

In [0]:

```
germany_dataset = germany_df.values
germany_dataset = germany_dataset.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
germany_dataset = scaler.fit_transform(germany_dataset)
train_size = int(len(germany_dataset) * 0.70)
val_size = int(len(germany_dataset) * 0.10)
test_size = int(len(germany_dataset) * 0.20)
train, val ,test = germany_dataset[0:train_size,:], germany_dataset[train_size:val_size + train_size,:], ger
many_dataset[val_size + train_size:len(germany_dataset),:]
train_X, train_Y = train[:,:-1], train[:,-1]
test_X, test_Y = test[:, :-1], test[:, -1]
val_X, val_Y = val[:, :-1], val[:, -1]
train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))
val_X = np.reshape(val_X, (val_X.shape[0], 1, val_X.shape[1]))
```

Training the LSTM Model

In [244]:

```
lstm_model = Sequential()
lstm_model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2])))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(1))
lstm_model.compile(loss='mean_squared_error', optimizer='adam')

history = lstm_model.fit(train_X, train_Y, epochs=20, batch_size=70, verbose=1, shuffle=False,
                         validation_data=(val_X, val_Y), callbacks=[EarlyStopping(monitor='val_loss', patien
ce=10)])
lstm_model.summary()
```

```
Train on 41 samples, validate on 5 samples
Epoch 1/20
41/41 [==============================] - 0s 8ms/step - loss: 0.0018 - val_loss: 7.6273e-04
Epoch 2/20
41/41 [==============================] - 0s 147us/step - loss: 0.0015 - val_loss: 0.0013
Epoch 3/20
41/41 [==============================] - 0s 125us/step - loss: 0.0018 - val_loss: 0.0018
Epoch 4/20
41/41 [==============================] - 0s 91us/step - loss: 0.0019 - val_loss: 0.0022
Epoch 5/20
41/41 [==============================] - 0s 97us/step - loss: 0.0025 - val_loss: 0.0022
Epoch 6/20
41/41 [==============================] - 0s 94us/step - loss: 0.0014 - val_loss: 0.0022
Epoch 7/20
41/41 [==============================] - 0s 104us/step - loss: 0.0015 - val_loss: 0.0021
Epoch 8/20
41/41 [==============================] - 0s 90us/step - loss: 0.0012 - val_loss: 0.0019
Epoch 9/20
41/41 [==============================] - 0s 97us/step - loss: 9.8731e-04 - val_loss: 0.0016
Epoch 10/20
41/41 [==============================] - 0s 89us/step - loss: 0.0010 - val_loss: 0.0014
Epoch 11/20
41/41 [==============================] - 0s 97us/step - loss: 0.0013 - val_loss: 0.0012
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_9 (LSTM)                (None, 64)                18432     
_________________________________________________________________
dropout_19 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 65        
=================================================================
Total params: 18,497
Trainable params: 18,497
Non-trainable params: 0
_________________________________________________________________
```

Predicting Using the LSTM Model

In [245]:

```
# Make a prediction
yhat = lstm_model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
test_Y = test_Y.reshape((len(test_Y), 1))

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.309
```

The Dense Model

Preparing the Train and Test Sets

The data is split and normalized as done before the LSTM model but not reshaped as the input doesn't contain the timesteps dimension.

In [0]:

```
dense_model_germany_df = germany_df
#Add a counter as a feature to indicate the day from the start of the pandemic
dense_model_germany_df['days_from_pandemic_start'] = [x for x in range(dense_model_germany_df.shape[0])]
g_new_cases_df = dense_model_germany_df['new_cases']
dense_model_germany_df.drop(columns='new_cases', inplace=True)
dense_model_germany_df['new_cases'] = g_new_cases_df
germany_dense_model_dataset = dense_model_germany_df.values
germany_dense_model_dataset = germany_dense_model_dataset.astype('float32')
dense_model_scaler = MinMaxScaler(feature_range=(0, 1))
germany_dense_model_dataset = dense_model_scaler.fit_transform(germany_dense_model_dataset)
train_size = int(len(germany_dataset) * 0.70)
val_size = int(len(germany_dataset) * 0.10)
test_size = int(len(germany_dataset) * 0.20)
train, val ,test = germany_dense_model_dataset[0:train_size,:], germany_dense_model_dataset[train_size:val_s
ize + train_size,:], germany_dense_model_dataset[val_size + train_size:len(germany_dataset),:]
train_X, train_Y = train[:,:-1], train[:,-1]
test_X, test_Y = test[:, :-1], test[:, -1]
val_X, val_Y = val[:, :-1], val[:, -1]
```

Defining the Dense Model

In [250]:

```
model = Sequential()
model.add(Dense(64, input_dim=train_X.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(train_X, train_Y, epochs=20, batch_size=70, verbose=1, shuffle=False,
                    validation_data=(val_X, val_Y), callbacks=[EarlyStopping(monitor='val_loss', patience=10
)])

model.summary()
```

```
Train on 41 samples, validate on 5 samples
Epoch 1/20
41/41 [==============================] - 0s 2ms/step - loss: 0.0706 - val_loss: 0.0497
Epoch 2/20
41/41 [==============================] - 0s 124us/step - loss: 0.0712 - val_loss: 0.0421
Epoch 3/20
41/41 [==============================] - 0s 97us/step - loss: 0.0471 - val_loss: 0.0349
Epoch 4/20
41/41 [==============================] - 0s 104us/step - loss: 0.0694 - val_loss: 0.0278
Epoch 5/20
41/41 [==============================] - 0s 97us/step - loss: 0.0540 - val_loss: 0.0220
Epoch 6/20
41/41 [==============================] - 0s 101us/step - loss: 0.0663 - val_loss: 0.0168
Epoch 7/20
41/41 [==============================] - 0s 102us/step - loss: 0.0579 - val_loss: 0.0126
Epoch 8/20
41/41 [==============================] - 0s 56us/step - loss: 0.0401 - val_loss: 0.0095
Epoch 9/20
41/41 [==============================] - 0s 59us/step - loss: 0.0221 - val_loss: 0.0073
Epoch 10/20
41/41 [==============================] - 0s 108us/step - loss: 0.0332 - val_loss: 0.0059
Epoch 11/20
41/41 [==============================] - 0s 58us/step - loss: 0.0424 - val_loss: 0.0052
Epoch 12/20
41/41 [==============================] - 0s 61us/step - loss: 0.0367 - val_loss: 0.0051
Epoch 13/20
41/41 [==============================] - 0s 54us/step - loss: 0.0199 - val_loss: 0.0056
Epoch 14/20
41/41 [==============================] - 0s 69us/step - loss: 0.0332 - val_loss: 0.0063
Epoch 15/20
41/41 [==============================] - 0s 56us/step - loss: 0.0250 - val_loss: 0.0073
Epoch 16/20
41/41 [==============================] - 0s 58us/step - loss: 0.0224 - val_loss: 0.0084
Epoch 17/20
41/41 [==============================] - 0s 57us/step - loss: 0.0219 - val_loss: 0.0096
Epoch 18/20
41/41 [==============================] - 0s 60us/step - loss: 0.0182 - val_loss: 0.0110
Epoch 19/20
41/41 [==============================] - 0s 75us/step - loss: 0.0209 - val_loss: 0.0122
Epoch 20/20
41/41 [==============================] - 0s 60us/step - loss: 0.0197 - val_loss: 0.0133
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_32 (Dense)             (None, 64)                512
_________________________________________________________________
dropout_20 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_33 (Dense)             (None, 1)                 65
=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
_________________________________________________________________
```

Predicting Using the Dense Model

In [251]:

```python
# make a prediction
yhat = model.predict(test_X)
test_Y = test_Y.reshape((len(test_Y), 1))

# calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.231
```

The same steps are repeated for the France and the Italy's datasets.

France's Models

In [0]:

```
france_df.drop(columns='date', inplace=True)
```

The LSTM Model

Preparing the train and Test Sets

In [0]:

```
france_df['output_new_cases'] =france_df['new_cases'].shift(-1)
france_df.drop(axis=0,index=[france_df.shape[0]-1], inplace=True)
```

In [0]:

```
france_dataset = france_df.values
france_dataset = france_dataset.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
france_dataset = scaler.fit_transform(france_dataset)
train_size = int(len(france_dataset) * 0.70)
val_size = int(len(france_dataset) * 0.10)
test_size = int(len(france_dataset) * 0.20)
train, val ,test = france_dataset[0:train_size,:], france_dataset[train_size:val_size + train_size,:], franc
e_dataset[val_size + train_size:len(france_dataset),:]
train_X, train_Y = train[:,:-1], train[:,-1]
test_X, test_Y = test[:, :-1], test[:, -1]
val_X, val_Y = val[:, :-1], val[:, -1]
train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))
val_X = np.reshape(val_X, (val_X.shape[0], 1, val_X.shape[1]))
```

Defining the LSTM Model

In [271]:

```
lstm_model = Sequential()
lstm_model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2])))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(1))
lstm_model.compile(loss='mean_squared_error', optimizer='adam')

history = lstm_model.fit(train_X, train_Y, epochs=20, batch_size=70, verbose=1, shuffle=False,
                         validation_data=(val_X, val_Y), callbacks=[EarlyStopping(monitor='val_loss', patien
ce=10)])
lstm_model.summary()
```

```
Train on 41 samples, validate on 5 samples
Epoch 1/20
41/41 [==============================] - 0s 8ms/step - loss: 0.0015 - val_loss: 0.0048
Epoch 2/20
41/41 [==============================] - 0s 147us/step - loss: 0.0011 - val_loss: 0.0043
Epoch 3/20
41/41 [==============================] - 0s 123us/step - loss: 8.8675e-04 - val_loss: 0.0038
Epoch 4/20
41/41 [==============================] - 0s 122us/step - loss: 6.7794e-04 - val_loss: 0.0034
Epoch 5/20
41/41 [==============================] - 0s 117us/step - loss: 9.4882e-04 - val_loss: 0.0033
Epoch 6/20
41/41 [==============================] - 0s 120us/step - loss: 6.8105e-04 - val_loss: 0.0032
Epoch 7/20
41/41 [==============================] - 0s 119us/step - loss: 8.0140e-04 - val_loss: 0.0032
Epoch 8/20
41/41 [==============================] - 0s 125us/step - loss: 6.9709e-04 - val_loss: 0.0033
Epoch 9/20
41/41 [==============================] - 0s 119us/step - loss: 7.9482e-04 - val_loss: 0.0034
Epoch 10/20
41/41 [==============================] - 0s 112us/step - loss: 6.1630e-04 - val_loss: 0.0036
Epoch 11/20
41/41 [==============================] - 0s 121us/step - loss: 7.0715e-04 - val_loss: 0.0038
Epoch 12/20
41/41 [==============================] - 0s 115us/step - loss: 5.5393e-04 - val_loss: 0.0041
Epoch 13/20
41/41 [==============================] - 0s 114us/step - loss: 4.4567e-04 - val_loss: 0.0043
Epoch 14/20
41/41 [==============================] - 0s 115us/step - loss: 7.5241e-04 - val_loss: 0.0044
Epoch 15/20
41/41 [==============================] - 0s 114us/step - loss: 5.2984e-04 - val_loss: 0.0045
Epoch 16/20
41/41 [==============================] - 0s 106us/step - loss: 5.0908e-04 - val_loss: 0.0045
Epoch 17/20
41/41 [==============================] - 0s 93us/step - loss: 5.5986e-04 - val_loss: 0.0043
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_12 (LSTM)               (None, 64)                18176     
_________________________________________________________________
dropout_23 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_36 (Dense)             (None, 1)                 65        
=================================================================
Total params: 18,241
Trainable params: 18,241
Non-trainable params: 0
_________________________________________________________________
```

Predicting Using the LSTM Model

In [272]:

```python
# make a prediction
yhat = lstm_model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
test_Y = test_Y.reshape((len(test_Y), 1))

# calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.572
```

The Dense Model

Preparing the Train and Test Sets

In [0]:

```
dense_model_france_df = france_df
#Add a counter as a feature to indicate the day from the start of the pandemic
dense_model_france_df['days_from_pandemic_start'] = [x for x in range(dense_model_france_df.shape[0])]
g_new_cases_df = dense_model_france_df['new_cases']
dense_model_france_df.drop(columns='new_cases', inplace=True)
dense_model_france_df['new_cases'] = g_new_cases_df
france_dense_model_dataset = dense_model_france_df.values
france_dense_model_dataset = france_dense_model_dataset.astype('float32')
dense_model_scaler = MinMaxScaler(feature_range=(0, 1))
france_dense_model_dataset = dense_model_scaler.fit_transform(france_dense_model_dataset)
train_size = int(len(france_dataset) * 0.70)
val_size = int(len(france_dataset) * 0.10)
test_size = int(len(france_dataset) * 0.20)
train, val ,test = france_dense_model_dataset[0:train_size,:], france_dense_model_dataset[train_size:val_siz
e + train_size,:], france_dense_model_dataset[val_size + train_size:len(france_dataset),:]
train_X, train_Y = train[:,:-1], train[:,-1]
test_X, test_Y = test[:, :-1], test[:, -1]
val_X, val_Y = val[:, :-1], val[:, -1]
```

Defining the Dense Model

In [275]:

```
model = Sequential()
model.add(Dense(64, input_dim=train_X.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(train_X, train_Y, epochs=20, batch_size=70, verbose=1, shuffle=False,
                    validation_data=(val_X, val_Y), callbacks=[EarlyStopping(monitor='val_loss', patience=10
)])

model.summary()
```

```
Train on 41 samples, validate on 5 samples
Epoch 1/20
41/41 [==============================] - 0s 2ms/step - loss: 0.0481 - val_loss: 0.0164
Epoch 2/20
41/41 [==============================] - 0s 86us/step - loss: 0.0414 - val_loss: 0.0128
Epoch 3/20
41/41 [==============================] - 0s 65us/step - loss: 0.0371 - val_loss: 0.0102
Epoch 4/20
41/41 [==============================] - 0s 70us/step - loss: 0.0397 - val_loss: 0.0083
Epoch 5/20
41/41 [==============================] - 0s 53us/step - loss: 0.0212 - val_loss: 0.0070
Epoch 6/20
41/41 [==============================] - 0s 59us/step - loss: 0.0224 - val_loss: 0.0060
Epoch 7/20
41/41 [==============================] - 0s 60us/step - loss: 0.0227 - val_loss: 0.0055
Epoch 8/20
41/41 [==============================] - 0s 57us/step - loss: 0.0214 - val_loss: 0.0053
Epoch 9/20
41/41 [==============================] - 0s 60us/step - loss: 0.0262 - val_loss: 0.0052
Epoch 10/20
41/41 [==============================] - 0s 56us/step - loss: 0.0264 - val_loss: 0.0053
Epoch 11/20
41/41 [==============================] - 0s 60us/step - loss: 0.0152 - val_loss: 0.0055
Epoch 12/20
41/41 [==============================] - 0s 63us/step - loss: 0.0178 - val_loss: 0.0057
Epoch 13/20
41/41 [==============================] - 0s 73us/step - loss: 0.0191 - val_loss: 0.0059
Epoch 14/20
41/41 [==============================] - 0s 65us/step - loss: 0.0222 - val_loss: 0.0059
Epoch 15/20
41/41 [==============================] - 0s 72us/step - loss: 0.0264 - val_loss: 0.0059
Epoch 16/20
41/41 [==============================] - 0s 69us/step - loss: 0.0199 - val_loss: 0.0057
Epoch 17/20
41/41 [==============================] - 0s 68us/step - loss: 0.0160 - val_loss: 0.0054
Epoch 18/20
41/41 [==============================] - 0s 72us/step - loss: 0.0245 - val_loss: 0.0051
Epoch 19/20
41/41 [==============================] - 0s 76us/step - loss: 0.0208 - val_loss: 0.0047
Epoch 20/20
41/41 [==============================] - 0s 81us/step - loss: 0.0115 - val_loss: 0.0043
Model: "sequential_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_39 (Dense)             (None, 64)                512       
_________________________________________________________________
dropout_25 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_40 (Dense)             (None, 1)                 65        
=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
_________________________________________________________________
```

Predicting Using the Dense Model

In [276]:

```python
# make a prediction
yhat = model.predict(test_X)
test_Y = test_Y.reshape((len(test_Y), 1))

# calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.316
```

In [0]:

```python

```

Italy's Models

In [0]:

```
italy_df.drop(columns='date', inplace=True)
```

The LSTM Model

Preparing the train and Test Sets

In [0]:

```
italy_df['output_new_cases'] =italy_df['new_cases'].shift(-1)
italy_df.drop(axis=0,index=[italy_df.shape[0]-1], inplace=True)
```

In [0]:

```
italy_dataset = france_df.values
italy_dataset = italy_dataset.astype('float32')
scaler = MinMaxScaler(feature_range=(0, 1))
italy_dataset = scaler.fit_transform(italy_dataset)
train_size = int(len(italy_dataset) * 0.70)
val_size = int(len(italy_dataset) * 0.10)
test_size = int(len(italy_dataset) * 0.20)
train, val ,test = italy_dataset[0:train_size,:], italy_dataset[train_size:val_size + train_size,:], italy_d
ataset[val_size + train_size:len(italy_dataset),:]
train_X, train_Y = train[:,:-1], train[:,-1]
test_X, test_Y = test[:, :-1], test[:, -1]
val_X, val_Y = val[:, :-1], val[:, -1]
train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))
val_X = np.reshape(val_X, (val_X.shape[0], 1, val_X.shape[1]))
```

Defining the LSTM Model

In [280]:

```
lstm_model = Sequential()
lstm_model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2])))
lstm_model.add(Dropout(0.5))
lstm_model.add(Dense(1))
lstm_model.compile(loss='mean_squared_error', optimizer='adam')

history = lstm_model.fit(train_X, train_Y, epochs=20, batch_size=70, verbose=1, shuffle=False,
                         validation_data=(val_X, val_Y), callbacks=[EarlyStopping(monitor='val_loss', patien
ce=10)])
lstm_model.summary()
```

```
Train on 41 samples, validate on 5 samples
Epoch 1/20
41/41 [==============================] - 0s 8ms/step - loss: 0.0015 - val_loss: 0.0032
Epoch 2/20
41/41 [==============================] - 0s 149us/step - loss: 0.0024 - val_loss: 0.0032
Epoch 3/20
41/41 [==============================] - 0s 266us/step - loss: 0.0024 - val_loss: 0.0033
Epoch 4/20
41/41 [==============================] - 0s 145us/step - loss: 0.0011 - val_loss: 0.0033
Epoch 5/20
41/41 [==============================] - 0s 139us/step - loss: 0.0012 - val_loss: 0.0032
Epoch 6/20
41/41 [==============================] - 0s 109us/step - loss: 0.0011 - val_loss: 0.0031
Epoch 7/20
41/41 [==============================] - 0s 105us/step - loss: 0.0012 - val_loss: 0.0032
Epoch 8/20
41/41 [==============================] - 0s 110us/step - loss: 0.0012 - val_loss: 0.0031
Epoch 9/20
41/41 [==============================] - 0s 121us/step - loss: 9.4214e-04 - val_loss: 0.0030
Epoch 10/20
41/41 [==============================] - 0s 117us/step - loss: 0.0010 - val_loss: 0.0030
Epoch 11/20
41/41 [==============================] - 0s 116us/step - loss: 0.0011 - val_loss: 0.0028
Epoch 12/20
41/41 [==============================] - 0s 108us/step - loss: 8.6113e-04 - val_loss: 0.0027
Epoch 13/20
41/41 [==============================] - 0s 110us/step - loss: 8.3132e-04 - val_loss: 0.0026
Epoch 14/20
41/41 [==============================] - 0s 109us/step - loss: 0.0011 - val_loss: 0.0027
Epoch 15/20
41/41 [==============================] - 0s 88us/step - loss: 7.8429e-04 - val_loss: 0.0028
Epoch 16/20
41/41 [==============================] - 0s 82us/step - loss: 6.3349e-04 - val_loss: 0.0030
Epoch 17/20
41/41 [==============================] - 0s 112us/step - loss: 8.1473e-04 - val_loss: 0.0031
Epoch 18/20
41/41 [==============================] - 0s 98us/step - loss: 0.0010 - val_loss: 0.0032
Epoch 19/20
41/41 [==============================] - 0s 97us/step - loss: 7.0902e-04 - val_loss: 0.0032
Epoch 20/20
41/41 [==============================] - 0s 124us/step - loss: 6.8554e-04 - val_loss: 0.0032
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_13 (LSTM)               (None, 64)                18432
_________________________________________________________________
dropout_26 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 65
=================================================================
Total params: 18,497
Trainable params: 18,497
Non-trainable params: 0
_________________________________________________________________
```

Predicting Using the LSTM Model

In [281]:

```python
# make a prediction
yhat = lstm_model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
test_Y = test_Y.reshape((len(test_Y), 1))

# calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.493
```

The Dense Model

Preparing the Train and Test Sets

In [0]:

```
dense_model_italy_df = italy_df
#Add a counter as a feature to indicate the day from the start of the pandemic
dense_model_italy_df['days_from_pandemic_start'] = [x for x in range(dense_model_italy_df.shape[0])]
g_new_cases_df = dense_model_italy_df['new_cases']
dense_model_italy_df.drop(columns='new_cases', inplace=True)
dense_model_italy_df['new_cases'] = g_new_cases_df
italy_dense_model_dataset = dense_model_italy_df.values
italy_dense_model_dataset = italy_dense_model_dataset.astype('float32')
dense_model_scaler = MinMaxScaler(feature_range=(0, 1))
italy_dense_model_dataset = dense_model_scaler.fit_transform(italy_dense_model_dataset)
train_size = int(len(italy_dataset) * 0.70)
val_size = int(len(italy_dataset) * 0.10)
test_size = int(len(italy_dataset) * 0.20)
train, val ,test = italy_dense_model_dataset[0:train_size,:], italy_dense_model_dataset[train_size:val_size
+ train_size,:], italy_dense_model_dataset[val_size + train_size:len(italy_dataset),:]
train_X, train_Y = train[:,:-1], train[:,-1]
test_X, test_Y = test[:, :-1], test[:, -1]
val_X, val_Y = val[:, :-1], val[:, -1]
```

Defining the Dense Model

In [290]:

```
model = Sequential()
model.add(Dense(64, input_dim=train_X.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')

history = model.fit(train_X, train_Y, epochs=20, batch_size=70, verbose=1, shuffle=False,
                    validation_data=(val_X, val_Y), callbacks=[EarlyStopping(monitor='val_loss', patience=10
)])

model.summary()
```

```
Train on 41 samples, validate on 5 samples
Epoch 1/20
41/41 [==============================] - 0s 2ms/step - loss: 0.0468 - val_loss: 0.1060
Epoch 2/20
41/41 [==============================] - 0s 69us/step - loss: 0.0552 - val_loss: 0.0952
Epoch 3/20
41/41 [==============================] - 0s 99us/step - loss: 0.0517 - val_loss: 0.0851
Epoch 4/20
41/41 [==============================] - 0s 80us/step - loss: 0.0436 - val_loss: 0.0764
Epoch 5/20
41/41 [==============================] - 0s 65us/step - loss: 0.0197 - val_loss: 0.0692
Epoch 6/20
41/41 [==============================] - 0s 66us/step - loss: 0.0359 - val_loss: 0.0623
Epoch 7/20
41/41 [==============================] - 0s 72us/step - loss: 0.0334 - val_loss: 0.0556
Epoch 8/20
41/41 [==============================] - 0s 64us/step - loss: 0.0275 - val_loss: 0.0496
Epoch 9/20
41/41 [==============================] - 0s 60us/step - loss: 0.0300 - val_loss: 0.0442
Epoch 10/20
41/41 [==============================] - 0s 64us/step - loss: 0.0321 - val_loss: 0.0394
Epoch 11/20
41/41 [==============================] - 0s 71us/step - loss: 0.0253 - val_loss: 0.0355
Epoch 12/20
41/41 [==============================] - 0s 66us/step - loss: 0.0258 - val_loss: 0.0323
Epoch 13/20
41/41 [==============================] - 0s 68us/step - loss: 0.0230 - val_loss: 0.0295
Epoch 14/20
41/41 [==============================] - 0s 69us/step - loss: 0.0269 - val_loss: 0.0272
Epoch 15/20
41/41 [==============================] - 0s 70us/step - loss: 0.0354 - val_loss: 0.0253
Epoch 16/20
41/41 [==============================] - 0s 65us/step - loss: 0.0158 - val_loss: 0.0237
Epoch 17/20
41/41 [==============================] - 0s 67us/step - loss: 0.0188 - val_loss: 0.0224
Epoch 18/20
41/41 [==============================] - 0s 68us/step - loss: 0.0283 - val_loss: 0.0216
Epoch 19/20
41/41 [==============================] - 0s 68us/step - loss: 0.0199 - val_loss: 0.0210
Epoch 20/20
41/41 [==============================] - 0s 68us/step - loss: 0.0146 - val_loss: 0.0206
Model: "sequential_34"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_46 (Dense)             (None, 64)                512       
_________________________________________________________________
dropout_29 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 65        
=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
_________________________________________________________________
```

Predicting Using the Dense Model

In [291]:

```python
# make a prediction
yhat = model.predict(test_X)
test_Y = test_Y.reshape((len(test_Y), 1))

# calculate RMSE
rmse = np.sqrt(mean_squared_error(test_Y, yhat))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.471
```

Conclusion

The results are close when using the LSTM model or the dense model with the dense model having slightly better results. The amount of the data was a limitation with only 60 days used as the weather data is the resitricting factor. Results would have been better and a better comparison would have been made if more data samples are used in training.

References

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)
- https://towardsdatascience.com/time-series-analysis-visualization-forecasting-with-lstm-77a905180eba (https://towardsdatascience.com/time-series-analysis-visualization-forecasting-with-lstm-77a905180eba)
- https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/ (https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/)
- https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/)
- https://towardsdatascience.com/lstm-for-time-series-prediction-de8aeb26f2ca (https://towardsdatascience.com/lstm-for-time-series-prediction-de8aeb26f2ca)
- https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/ (https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/)
- https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/ (https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/)