POLITECNICO
DI MILANO

AN2DL: Challenge 2

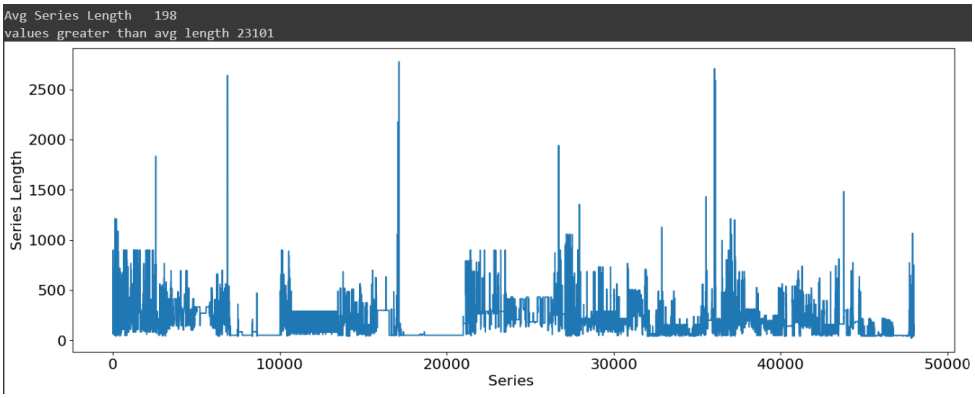
Authors:

Sanjay Shivakumar Manohar, 10864950
Satvik Bisht, 10886954
Duggaraju Venkata Sraavya, 10879043

Time Series Forecasting

First Approach

Analysis of the given data series was done. They had huge differences in their length. Some of them were in counts of tens and few were in counts of thousands. On average each series was around hundreds in length(~200). Also, counts of each category of the series were observed. Category F seemed to have a very lesser count compared to others.




```
(array(['A', 'B', 'C', 'D', 'E', 'F'], dtype='<U1'),
 array([ 5728, 10987, 10017, 10016, 10975,   277]))
```

As the time series was univariate, belonging to six different domains a single general model with one input feature was observed to be enough.

For the preprocessing, initially, the time series data was rolled into sequences of length (window size) 200, and a number of future values to forecast(Telescope) was set to 10. The stride length was set to 4. Not all the values were taken from the initial training data but only those above index 1500 as most values before were zero in most of the cases. This was a tradeoff of some accuracy lost due to omitting data to cut high computation requirements. The data was already normalised, so this process was skipped.

```
Model: "k_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 200, 1)]          0

 bidirectional_lstm (Bidire  (None, 200, 128)          33792
 ctional)

 bidirectional_lstm1 (Bidir  (None, 128)               98816
 ectional)

 dense_9 (Dense)             (None, 512)               66048

 dropout_6 (Dropout)         (None, 512)               0

 dense_10 (Dense)            (None, 1024)              525312

 dropout_7 (Dropout)         (None, 1024)              0

 dense_11 (Dense)            (None, 10)                10250

=================================================================
Total params: 734218 (2.80 MB)
Trainable params: 734218 (2.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The custom model given in the notebook by the professor was the first model that was tried, with few changes. Data was split into training and validation only; the testing dataset wasn't required, as the one on CodaLab would have been used for this purpose. The model used an input layer taking a sequence of length

200 and a single feature (200,1). The number of epochs was set to 200 and the loss function was Mean Square Error with ReduceLR plateau and Early stopping callbacks. The train_loss and val_loss were around 0.0025 and 0.005. The model produced **MSE = 0.0057 and MAE = 0.054** when tested on the Codalab development phase. Also tried other models to compare the results. Simpler models like conv1d and rnn were not able to show better results. After taking the hints provided in the logbook, we tried implementing resnet with conv1d and used robust scaling to normalise data. All these models initially showed underfitting, so we tried the models with different complexities by adding/removing layers. The best results were obtained from lstm and its variants.

We also loaded the time series data and category information from a specified path filtering out invalid data based on valid_periods. Encoded the categorical data into numerical format. We also performed some preprocessing which converted the time series data into a suitable format for the model which involved creating sequences of specific lengths for seq_length and their corresponding future values to be predicted forecast_length. Shuffled the data to ensure randomness. The model consists of a GRU (Gated Recurrent Unit) layer, a dropout layer for regularisation, and a dense layer for output. and split the data into training and validation sets. Configured the model with an 'adam' optimizer and 'mean squared error' as the loss function. This model produces **MSE= 0.00587** and **MAE = 0.00545** in codalab in the development phase.

Second Approach

After the first approach, the preprocessing step was changed to involve the maximum amount of data possible. The training dataset was now produced taking the start of the final index values from periods numpy file. This ensured all the data for each series was taken into the training process. Other preprocessing steps were the same as before.

Smaller models were tried out as the feature seemed to be simple to learn and to avoid overfitting. The models seemed to perform surprisingly well although its complexity was

less compared to the previous model. The callbacks and loss functions were the same as before. This model produced a **MSE = 0.0054** and **MAE =0.053** which was our highest in the development phase.

```
Model: "k_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 200, 1)]          0

 lstm (LSTM)                 (None, 200)               161600

 dense (Dense)               (None, 512)               102912

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 269642 (1.03 MB)
Trainable params: 269642 (1.03 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Final Phase

For the final phases, there were some issues with regard to development phase models. To start with, the output shape was just (10,1) which wasn't fit to predict 18 values (18,1). We used our GRU model by changing the forecast_length to 18 and re-trained the model which scored MSE= 0.0115 and MAE = 0.074.

Retraining the simple model to predict 18 values also didn't work out well with accuracy. So a different approach was made. Instead of predicting 18 values, the model was built just to predict the next two values. Also instead of taking a window of 200 values, a smaller scale of 20 was used.As we trained the model using a sliding window like approach, the prediction logic too was changed to make prediction in a sliding window manner. The window included the previous predicted value to predict the next value.

```
Model: "k_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 20, 1)]           0

 lstm (LSTM)                 (None, 64)                16896

 dense (Dense)               (None, 128)               8320

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 25474 (99.51 KB)
Trainable params: 25474 (99.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
def predict(self, X, categories):
  row,colum=X.shape
  total_pred=18
  window=20
  out=[]
  for i in range(total_pred):
    r = self.model.predict(X[0:row,colum-window+i:colum+i])
    r=r[:,0]
    r=np.reshape(r, (len(X),1))
    X = np.hstack((X, r))
    out.append(r)
  out = np.array(out)
  out = out.reshape(total_pred,len(X))
  out = out.T
  out = np.array(out)
  return out
```

This produced a model capable of not just handling a prediction of 18 values, but can be modified in code to predict much larger or smaller numbers of values. This model produced **MSE = 0.0086** and **MAE = 0.0614** in the codalab Final phase.

Contributions

Sanjay(10864950):

1. Data analysis- Analysing count of each category and variable lengths of each series.
2. Data preprocessing- Trying out different data sequences of smaller lengths to optimise memory usage and data transformation to different window sizes and strides.
3. Sliding window kind of approach for predictions.
4. Trying multiple complex and simpler models using early stopping and LR plateau callbacks with different epoch ,batch settings and multi gpu training.

Satvik Bisht (10886954):

1. Preprocesses time series data by converting it into sequences suitable for model input, considering specified sequence and forecast lengths.
2. Handled categorical data encoding and ensures data validity by filtering out periods with insufficient information.
3. Implemented GRU model and used different techniques like early stopping, dropout,different optimizers and loss functions.

Sraavya (10879043):

1. Data analysis and preprocessing: Analysed different periods of time series and their count. Different preprocessing techniques like robust scaling.
2. Implemented different models like LSTM, BiLSTM, RNN, ResNet-Conv1D to analyse the results.
3. Tried different complexities of models to reduce underfitting and different combinations of epochs, optimizers, loss function and early stopping to improve the results.