# Contents Page No.

1. Different tools used for Implementation 2 - 4
2. Libraries Used 5 - 6
3. Models Created 7 - 11
4. References 12 - 13

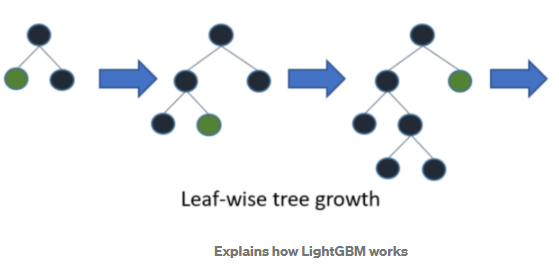
# Report (functionality)

1. **Different tools used for the implementation:**
   1. **Choice of environment:**

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs. Colab is free to use (research.google.com, n.d.).

* 1. **LGBM Regressor**

It is a tree based framework that uses gradient boosting algorithm. LightGBM grows trees leaf-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.



Light GBM is prefixed as ‘Light’ because of its high speed. Light GBM can handle the large size of data and takes lower memory to run. Another reason of why Light GBM is popular is because it focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development. it is not advisable to use LGBM on small datasets. Light GBM is sensitive to overfitting and can easily overfit small data (Mandot, 2018).

It is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm. LightGBM extends the gradient boosting algorithm by adding a type of automatic feature selection as well as focusing on boosting examples with larger gradients. This can result in a dramatic speedup of training and improved predictive performance.

* 1. **Wavenet**

WaveNet is a deep neural network for generating raw audio. It was created by researchers at London-based artificial intelligence firm DeepMind. The technique, outlined in a paper in September 2016, is able to generate relatively realistic-sounding human-like voices by directly modelling waveforms using a neural network method trained with recordings of real speech. Tests with US English and Mandarin reportedly showed that the system outperforms Google's best existing text-to-speech (TTS) systems, although as of 2016 its text-to-speech synthesis still was less convincing than actual human speech. WaveNet's ability to generate raw waveforms means that it can model any kind of audio, including music (Wikipedia, 2021).

* 1. **LSTM**

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. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications (Wikipedia Contributors, 2018).

* 1. **GRU**

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate. GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling and natural language processing was found to be similar to that of LSTM. GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets (Wikipedia Contributors, 2019).

* 1. **BPPS matrix**

The bpps are pre-calculated NumPy arrays for each RNA sequence.

Biophysically speaking, this matrix gives the probability that each pair of nucleotides in the RNA forms a base pair (given a particular model of RNA folding). At the simplest level -- it's a symmetric square matrix with the same length as the sequence, so you can get N more features out of it, if you want them. Each column and each row should sum to one (up to rounding error), but more than one entry in each column/row will be nonzero -- usually somewhere between 1-5 entries.

1. **Libraries used**
   1. **Pandas:** Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series (Wikipedia, 2021).
   2. **NumPy:** NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python (www.w3schools.com, n.d.).
   3. **Matplotlib:** Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib’s APIs (Application Programming Interfaces) to embed plots in GUI applications (ActiveState, n.d.).
   4. **Json:** JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand (W3schools.com, 2019).
   5. **ast:** The ast module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change with each Python release; this module helps to find out programmatically what the current grammar looks like. An abstract syntax tree can be compiled into a Python code object using the built-in compile() function (docs.python.org, n.d.).
   6. **seaborn:** Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics (seaborn.pydata.org, n.d.).
   7. **os:** The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The \*os\* and \*os. path\* modules include many functions to interact with the file system.
   8. **sklearn:** Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction (Analytics Vidhya, 2015).
   9. **itertools:** This module implements a number of iterator building blocks inspired by constructs from APL, Haskell, and SML. Each has been recast in a form suitable for Python. The module standardizes a core set of fast, memory efficient tools that are useful by themselves or in combination. Together, they form an “iterator algebra” making it possible to construct specialized tools succinctly and efficiently in pure Python (docs.python.org, n.d.).
   10. **Tqdm:** tqdm is a Python library that allows you to output a smart progress bar by wrapping around any iterable. A tqdm progress bar not only shows you how much time has elapsed, but also shows the estimated time remaining for the iterable (Steen, 2020).
2. **The different models used**
   1. **LGBM Regressor**

First the train, test and the sample submission files are loaded into the system. All these files are in the .json format. The shapes of the train and the test dataset are checked. The **train dataset has 2400 rows with 19 different columns while the test dataset has 3634 rows with 7 different columns.** The different column head names is printed to make the programmer accustomed with the dataset.

The values of the first 3 rows of the train dataset is visualized followed by the column names in the test dataset.

A list named *train\_data[]* is defined. The unique ‘*id*’ values are checked in the training dataset and the records having unique id’s are retrieved. The length of the different columns in the training set is 68 (each column is a 1x68 vector). Hence, a loop with range 68 is run to retrieve all the values from these records and these values are appended in the list *train\_data[]*.

The *train\_data[]* list is converted to a DataFrame and the column names are set to be equivalent to that of the train dataset. The first 5 values of this newly created DataFrame is printed.

Histograms have been used to show the frequency distribution of the different columns in the dataframe.

Similar to the training dataset, a list named *test\_data[]* has been defined. The records with the unique values have been retrieved. The values under the 5 different columns (*id*, *sequence*, *structure*, *seq\_scored*, *predicted\_loop\_type*) present in the records are extracted and then appended in the list *test\_data[]*. This list is converted to a DataFrame and the column names are set equivalent to the column heads of the test dataset. The first 5 rows in the test DataFrame gets printed.

The first bpps matrices for the first 25 RNA sequences is loaded and displayed.

The values of the *Reactivity*, *deg\_Mg\_pH10*, *deg\_Mg\_50C*, *deg\_pH10* and *deg\_50C* in the training DataFrame is rounded to 2 places off decimal.

Signal to Noise can be stated as mean(measurement value over 68 nts )/mean( statistical error in measurement value over 68 nts).

A distribution plot has been used to show the distribution of the S/N filter in the training data.

The distribution of the 2 types of sequence lengths of the RNA sequences (107 in the train set & public test set and 130 in the private test set) is visualized using bar plots.

The mean values of each record of the 5 target columns (*reactivity*, *deg\_Mg\_pH10*, *deg\_Mg\_50C*, *deg\_pH10*, *deg\_50C*) is calculated and histograms has been plotted to show their frequency distribution throughout the training set.

These mean values have been assigned to the records in the *submission.csv* file.

The distribution of the values of the 5 target columns without their errors is visualized using graphs.

A pairplot has been used to visualize the relationship among the mean values of the 5 target columns of the training set. The hue has been set to SN\_filter to map plot aspects to different colours.

Since the sequence length is 107, a loop of range 107 is initiated, the sequence column in the train set and the test set is converted to a categorical type. The dataset has been split into the train and test data and Light GBM Regressor has been used for predicting the target values from the test set.

The predicted average values have been assigned to the respective columns in the submission.csv file and the column names has been renamed accordingly (for example, ‘*mean\_reactivity\_pred*’ to ‘*reactivity*’).

The predicted target values in the *submission.csv* file has been visualized using histograms. The first 10 rows of the file are displayed.

The sequence features have been expanded and this time LGBM regressor has been used to train the model on the sum of the sequence, structure and predicted loop type columns. The predicted values of the target columns have been stored in a *sample\_submission.csv* file. The values of each of the predicted target columns is visualized using histograms.

Regplots has been used to show the distribution of the first 25 samples of the 5 target variables in the training set.

* 1. **Better LGBM Regressor**

3 lists named SEQUENCE\_COLS[], STRUCTURE\_COLS[] and PRED\_LOOP\_TYPE\_COLS[] are defined. A range of 130 is set because the sequence length is 130 in the private test set. In the loop, the values of the sequence, structure and the predicted\_loop\_type columns have been stored in a dataframe. The records in the sequence and structure columns have been appended in the SEQUENCE\_COLS and the STRUCTURE\_COLS.

Functions used:

* *expand\_columns*() 🡪 used to add columns to the dataframe passed as parameter for each of the sequences.
* *parse\_sample\_submission()* 🡪 id and sequence positions are split for ach record in sample submission.
* *get\_train\_long()* 🡪 All the columns have been padded with a constant value 107, a dataframe created, appropriate column heads added and the newly created dataframe is finally returned.
* *get\_test\_long()* 🡪 Columns *id*, *seqpos*, *sequence*, *structure* and *predicted\_loop\_type* are padded appropriately and the corresponding dataframe is finally returned.
* *add\_long\_features()* 🡪Records with sequence less than or equal to 106 are filtered out, grouped by *id* column, merged with the dataframe passed as parameter. Check is being done if merged key is unique in the filtered dataframe being merged. The process is done for *sequence*, *structure* and *predicted\_loop\_type* columns. These are further shifted using the sliding window technique.

LGBMRegressor is run with a learning rate of 1% and no. of boosted trees to fit as 100. The *importance\_type = ‘gain’* hyperparameter tuning allows the result to contain total gains of splits which use the feature. A horizontal bar graph is plotted based on the relative importance of the dataframe columns. Finally, histograms are plotted for all the 5 target columns based on training and test data.

* 1. **WaveNet + GRU**

Train and Public Test lengths:

* sequence, structure and predicted\_loop\_type have a length of 107
* reactivity, deg\_Mg\_pH10, deg\_pH10, deg\_Mg\_50C and deg\_50C have a length of 68

Private Test lengths:

* sequence, structure and predicted\_loop\_type have a length of 130
* reactivity, deg\_Mg\_pH10, deg\_pH10, deg\_Mg\_50C and deg\_50C have a length of 91

The steps followed are:

* Pre-process and tokenize the sequence, secondary structure and loop type.

Functions used:

* + - *preprocess\_inputs()* 🡪 transforms features to 3d format
    - *cmcrmse()* 🡪 custom loss function
    - *build\_model()* 🡪 function to build the wave net model. Default sequence length used is 107 with a dropout ratio of 10%. We used Adam optimizer. Loss and metric have also been calculated.
    - *Train\_and\_evaluate()* 🡪 Kfold cross validation has been used. *ReduceLROnPlateau* from Keras has been used for scheduling learning rate with *patience* hyperparameter set to 5.
    - We train a bi-directional GRU model having layers and dropout. Out-of-folds predictions have also been done.
    - *inference\_format()* 🡪 formats our predictions and writes the submission to file *submission.csv*.
* Use all the information to train a model on degradations recorded by the researchers from OpenVaccine.
* Ran our model on the public test set (shorter sequences) and the private test set (longer sequences), and submit the predictions.
  1. **GRU + LSTM**

The function seed\_everything() has been used to generate a random number everytime.

The train set, test set and the sample submission files have been loaded. The names of the columns in the train set has been print.

The train set has 2400 rows and 19 columns and no missing values have been found in the training set.

The test set has 3634 rows with 7 columns and no missing values have been found.

The format of the sample submission file has been checked.

A KDE plot and a count plot has been used to visualize the signal to noise distribution.

Here we can see that the data we have is the sequence and the predicted structure and loop type of each base in the RNA. The feature reactivity measures the degradation at each base. The higher the reactivity the more likely the RNA is to degrade at that base. The train set’s head has been transposed and the values with their category names has been displayed.

The *SN\_filter* is the signal-to-noise filter capturing which RNA molecules passed the evaluation criteria defined by the Stanford researchers. This means we will drop the rows with *SN\_filter = 0*. There are some RNAs that have quite a large amount of noise which is filtered by the *SN\_filter*. It has been found that among the 2400 records in the training set, 2096 has S/N ratio > 1, 1589 has *SN\_filter* equal to 1 and 509 samples have S/N ration greater than 1 and *SN\_filter* equal to 0.

**Bpps:** We have this readily available N x N matrix. This is used to determine the likely structures of mRNA. *bpps* is base probability matrix, a matrix of the probability of each base dating with every other base in the mRNA. So, assuming the mRNA molecule consists of 10 characters: ACAUUGGGAU. Then the bpps is a 10 x 10 matrix. Take the first base 'A'. So we have the probability of the first base 'A' pairing up with the remaining 9 bases. I guess A pairing with itself is 0 and the sum of the probabilities of the remaining 9 bpp will be the total probability of A being paired. So, *P(the probability of 'i' th base in the sequence seq being paired) = sum of probabilities of P(i,j) being paired) from j=0 to len(seq)-1*.

The bpps files have been loaded.

A function named *generate\_bpps\_sum()* has been created to generate sum of the values under each columns in the dataframe. These sums are then appended to an array which has been returned.

A function named *generate\_bpps\_max()* has been created to generate row-wise max value in Numpy matrix from *'bpps'* folder against each molecule id.

The function *generate\_bpps\_nb()* has been used to check the number of non-0 bpps files in the dataframe and calculate their mean value. These values have been appended to a list which have been returned.

The 3 list returned from the above 3 functions have been added to the training and the testing set.

200 random samples have been selected from the train set and the test set and their bpps sum, max row values and mean values have been generated using the above 3 functions.

The 3 values from the newly added columns in the dataframe has been utilised for visualizing the distribution of the maximum value in the bpps files, the sum of the bpps files and the mean values in the bpps files.

A list named *target\_cols* have been defined which stores the column names of the target columns.

The different characters used in the sequences have been enumerated.

The *preprocess\_inputs()* function has been used to preprocess the values of the input columns ‘*sequence’*, ‘*structure’* and ‘*predicted\_loop\_types*’ based on the key-value pairs in the enumeration. The preprocessed values have been converted to a list which in turn has been converted to an array and stored in an array named *base\_features*. This array has then been transposed. This array along with the *bpps\_sum* and *bpps\_max* columns in the dataframe (which were converted to a list) were appended in a list named *mylist[]*. This list has been returned.

The loss functions that has been used here are the root mean squared error and the mean column-wise root mean squared.

GRU and LSTM layers were used for building this model.

**GRU layer:** Based on available runtime hardware and constraints, this layer will choose different implementations (cuDNN-based or pure-TensorFlow) to maximize the performance. NVIDIA CUDA Deep Neural Network (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. It provides highly tuned implementations of routines arising frequently in DNN applications.

**LSTM layer:** Long short-term memory (LSTM) is an artificial Recurrent Neural Network (RNN) architecture used in the field of deep learning.

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.

The model has been built using the different rnn layers.

2 sets of learning curves has been plotted to show the variation between the training loss and the validation loss from the GRU and LSTM layers.

It can be observed from the predicted values that when the SN filter is not applied, then the values of the target columns come almost accurate to the values in the training set. However, when the *SN\_filter* is applied, then the values have been lowered. These predicted values gets stored in the *submission.csv* file.

1. **References:**

* research.google.com. (n.d.). Colaboratory – Google. [online] Available at: https://research.google.com/colaboratory/faq.html#:~:text=Colaboratory%2C%20or%20%E2%80%9CColab%E2%80%9D%20for.
* Mandot, P. (2018). What is LightGBM, How to implement it? How to fine tune the parameters? [online] Medium. Available at: https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc.
* Wikipedia. (2021). WaveNet. [online] Available at: https://en.wikipedia.org/wiki/WaveNet#:~:text=WaveNet%20is%20a%20deep%20neural%20network%20for%20generating%20raw%20audio.&text=The%20technique%2C%20outlined%20in%20a [Accessed 15 Apr. 2021].
* Wikipedia Contributors (2019). Gated recurrent unit. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Gated\_recurrent\_unit [Accessed 11 May 2019].
* Wikipedia Contributors (2018). Long short-term memory. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Long\_short-term\_memory.
* Wikipedia. (2021). pandas (software). [online] Available at: https://en.wikipedia.org/wiki/Pandas\_(software)#:~:text=pandas%20is%20a%20software%20library [Accessed 15 Apr. 2021].
* www.w3schools.com. (n.d.). Introduction to NumPy. [online] Available at: https://www.w3schools.com/python/numpy/numpy\_intro.asp#:~:text=NumPy%20is%20a%20Python%20library.
* ActiveState. (n.d.). What Is Matplotlib In Python? How to use it for plotting? [online] Available at: https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/ [Accessed 15 Apr. 2021].
* W3schools.com. (2019). What is JSON. [online] Available at: https://www.w3schools.com/whatis/whatis\_json.asp.
* docs.python.org. (n.d.). ast — Abstract Syntax Trees — Python 3.9.2 documentation. [online] Available at: https://docs.python.org/3/library/ast.html#:~:text=The%20ast%20module%20helps%20Python.
* seaborn.pydata.org. (n.d.). seaborn: statistical data visualization — seaborn 0.11.1 documentation. [online] Available at: https://seaborn.pydata.org/#:~:text=Seaborn%20is%20a%20Python%20data.
* Analytics Vidhya. (2015). SKLearn | Scikit-Learn In Python | SciKit Learn Tutorial. [online] Available at: https://www.analyticsvidhya.com/blog/2015/01/scikit-learn-python-machine-learning-tool/#:~:text=Scikit%2Dlearn%20is%20probably%20the [Accessed 15 Apr. 2021].
* docs.python.org. (n.d.). itertools — Functions creating iterators for efficient looping — Python 3.9.4 documentation. [online] Available at: https://docs.python.org/3/library/itertools.html#:~:text=This%20module%20implements%20a%20number [Accessed 15 Apr. 2021].
* Steen, D. (2020). Progress bars for Python with tqdm. [online] Medium. Available at: https://towardsdatascience.com/progress-bars-for-python-with-tqdm-4dba0d4cb4c [Accessed 15 Apr. 2021].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Model Type** | **Owner** | **Validation/Loss Function Used** | **Accuracy** |
| 1. | LightGBM Regressor | Self | KFold Cross validation | 0.0176721 |
| 2. | LightGBM Regressor | (fernandoramacciotti, 2020) | RMSE | Reactivity: 0.672821 +0.058203  deg\_Mg\_pH10: 0.591186 + 0.023849  deg\_pH10: 1.20183 + 0.102108  deg\_Mg\_50C: 0.775368 + 0.053296  deg\_50C: 1.06625 + 0.117316 |
| 3. | LightGBM Regressor | (omarvivas, 2020) | CatBoost, RMSE, LGBM RMSE, XGB RMSE | catBoost RMSE: 1.015502  LGBM RMSE: 1.020465  XGB RMSE: 1.022722 |
| 4. | GRU + LSTM | Self | MCRMSE | GRU :-  Loss: 0.2512  Val\_loss: 0.3069  LSTM:-  Loss: 0.2851  Val\_loss: 0.3451 |
| 5. | GRU + LSTM | (shadowburning, 2020) | MCRMSE | Mean Validation Loss: 0.392550 |
| 6. | GRU + LSTM | (gagankarora, 2020) | MCRMSE | Loss: 0.2549  Val\_loss: 0.2491 |
| 7. | GRU + LSTM | (vbmokin, 2020) | MCRMSE | Mean Validation Loss: 0.387129956 |
| 8. | GRU + LSTM | (ruko, 2020) | MCRMSE | GRU Mean Fold MCRMSE: 0.388990  LSTM Mean Fold MCRMSE: 0.3921980 |
| 9. | GRU + LSTM | (anzhemeng, 2020) | MCRMSE | GRU Mean Fold Validation Loss: 0.2133493  LSTM Mean Fold Validation Loss: 0.2146221 |
| 10. | WaveNet + GRU | Self | MCRMSE | MCRMSE: 0.84072194 |