Detecting Exaggerated Numerals in Financial Text

A Project/Thesis submitted in partial fulfillment for the Degree of **Master of Computer Application** at

Jadavpur University

By

RIMA ROY

Registration No.: 154239 of 2020-2021

Examination Roll No.: MCA2360038

Class Roll No.: 002010503031

Under the Guidance of

Dr. Sudip Kumar Naskar

Department of Computer Science and Engineering

Jadavpur University, Kolkata-700032

2023

Faculty of Engineering and Technology

Jadavpur University

To whom it may concern

I hereby recommend that the thesis entitled "**Detecting Exaggerated Numerals in Financial Text**" has been carried out by **RIMA ROY** (University Registration No.: 154239 of 2020-2021, Class Roll No.: 002010503031, Examination Roll No.: MCA2360038) under my guidance and supervision that has been accepted in partial fulfillment of the requirement for the Degree of Master of Computer Application in Department of Computer Science and Engineering, Jadavpur University.

_____

Dr. Sudip Kumar Naskar

Project Supervisor

Department of Computer Science and Engineering,

Jadavpur University, Kolkata-700032

_____

Prof. Nandini Mukherjee

Head of the Department

Department of Computer Science and Engineering,

Jadavpur University, Kolkata-700032

_____

Prof. Ardhendu Ghoshal

Dean

Department of Computer Science and Engineering,

Jadavpur University, Kolkata-700032

Faculty of Engineering and Technology
Jadavpur University

Certificate of Approval

This is to certify that the thesis entitled " **Detecting Exaggerated Numerals in Financial Text**" is a bonafide record of work carried out by **Rima Roy** in partial fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood by this approval that the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approves the thesis only by the purpose which it has been submitted for.

_____

Signature of Examiner 1

Date:

_____

Signature of Examiner 2

Date:

Faculty of Engineering and Technology
Jadavpur University

Declaration of Originality and Compliance with Academic Ethics

I hereby declare that this thesis entitled " **Detecting Exaggerated Numerals in Financial Text** " contains a literature survey and original research work by the undersigned candidate, as part of his Master of Computer Application studies. All information in this document has been obtained and presented by following the academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials along with the results that are not original to this work.

_____

Name – RIMA ROY
Registration No. - 154239 of 2020-2021
Examination Roll No.: MCA2360038
Class Roll No. - 002010503031
Thesis Title — Detecting Exaggerated Numerals in Financial Text.

Acknowledgment

The writing of the thesis as well as the related work has been a long journey with input from many individuals right from the first day till the development of the final project.

I would like to express my deepest gratitude to my supervisor, **Dr. Sudip Kumar Naskar**, Associate Professor, Department of Computer Science and Engineering, Jadavpur University for his admirable guidance, care, and patience and for providing me with an excellent atmosphere for doing research. Our numerous scientific discussions and his many constructive comments have greatly improved this work. I feel deeply honoured that I got the opportunity to work under him.

I would like to thank all the faculty members of the Department of Computer Science and Engineering of Jadavpur University for their continuous support and I also, would like to thank Sohom Ghosh, a researcher at the computer science and engineering department of Jadavpur University who helped me a lot by sharing knowledge throughout this work.

This thesis would not have been completed without the inspiration and support of several wonderful individuals including my batchmates of Master of Computer Application in Jadavpur University — my thanks and appreciation to all of them for being part of this journey as well as making this thesis possible.

RIMA ROY

Registration No.: 154239 of 2020-2021

Examination Roll No.: MCA2360038

Class Roll No.: 002010503031

Department of Computer Science & Engineering

Jadavpur University

Abstract

Numerics play a very important role in the financial text. If we change a particular numeric within a financial text then the meaning of the text is different. For example- "Google earnings increase 10%" If we change the numeric '10' and mistyped one extra zero then the sentence becomes "Google earnings increase 100%". So, the meaning of these two sentences is different in the financial aspect. So, changing the numeric in a financial text will lead to different outcomes in financial forecasting systems.

In this thesis, I try to give an idea of making an automated system that helps to detect this type of exaggerated numeric in the financial text by predicting the range of the numeric in a position of the text using natural language processing. For this task, I use the standard data dataset called 'numeracy-600k' which has two subsets that contain a large set of market comments and article titles also. The work involves the contextual embedding of a numeric token using the BERT-SEC-NUM model, exploring different machine learning and deep learning model like logistic regression, random forest, XGBoost, light GBM, CNN, LSTM, MLP, etc. Evaluating the performance of the model and discussing the challenges through comprehensive experiments.

Keywords *- Natural Language Processing (NLP), Contextual Embedding for a given token (number), Supervised Learning, Deep Learning.*

CONTENTS

List of figures and tables

Chapter 1-Introduction

We know that the market comments and the financial text contains numerals as a part of the text. This numerical plays a very important role in the text. Without the numerical in-market comments, we will miss a lot of useful information. For example, we see some instances of market comments–

Comment 1 – market comments

Apple's share increased by 10%.

Comment 2 – financial statement

Apple's 2018 revenue is $265.5 Balian.

Comment 3 – product analysis

Apple Q2 Mac sales of 4.6 million units vs 4.1 million units last year.

Comment 4 – analyst report

Apple Canaccord Genuity raises price target to $600

From the example, we can see that numerals provide more detailed information than the words in the text. For example, in comment 1 we can see that Apple's share increases but we cannot obtain the percentage change or the price quote without the numerical. Furthermore, comment 3 provides crucial information such as the date (Q2) and the number of sales with numerals (4.6 and 4.1). These examples show the crucial role of numerals in the financial aspect.

In the thesis, we try to give an idea of a system that predicts the range of the numeric for a particular position of the text using the context of the text. Let's take an example from our dataset-

Text- 2013 Ironman Coeur d'Alene results

Here I used the English text only.

So, we give the input to our model as-

---------- Ironman Coeur d'Alene results

And our model can predict the range or magnitude of the numeric of the blank position of the text. So, for this above example, our model predicts the range of the numeric '2013'. By using this magnitude, we detect whether the numeric is exaggerated or not.

Here we use classification, a part of supervised learning to make this type of system. For the training of the model, we use a sample of a large dataset called 'Numerecy_600k' which contains 600k market comments and then evaluate the different types of machine learning and neural network model.

Chapter 2-Related Work

Murakami et al. (2017) attempted to generate market comments from stock prices. Their work used only two kinds of numerals: the latest price, and the difference of closing price between two days. As we describe earlier, however, market comments describe various kinds of topics along with numerals. In this thesis, we will provide experimental results for general market comments and show the numeracy of various supervised machine learning and deep learning models.

Spithourakis and Riedel (2018) used language models to predict numerals in clinical and scientific datasets. They do not touch on numeral prediction in financial market comments. In this project, we examine whether the different supervised machine learning and deep learning models can learn numeracy to insert proper information into market comments, by predicting the correct range or magnitude of the numeric to the particular position of the market comment and with the help of this information detect the numeric is exaggerated or not.

Several different approaches have been used to detect false information and fake news. **Wang et al. (2018a)** used both text information and images in tweets to detect misleading information. **Tschiatschek et al. (2018)** identified fake news via crowd signals, namely, Facebook users' flags of fake news. As mentioned by **Shu et al. (2017)**, "the underlying characteristics of fake news have not been fully understood." In this project, we concentrate on market comments and focus on exaggerated numeral identification in the comments.

Chung-Chi Chen and Hen-Hsen Huang (2019) proposed the task of detecting exaggerated numerals on the numeracy_600k dataset using different neural network models. In this paper, they used both dataset numeracy_600k comments and numeracy_600k articles. They used different machine learning models and deep learning models mainly neural network-based models for predicting the range of the numeric in a particular position of the text. They calculated accuracy, f1_micro, and f1_macro to evaluate their model. Here in this project, I also used the same dataset numeracy_600k to predict the magnitude which means a range of the numerical in a particular position of the text but the process is different. In their project, they used word-by-word input, Countvectorizer for word embedding, etc. But in this project, I used a pre-trained deep learning model SEC-BERT-NUM for word embedding and embedding of the targeted numeric token provided as input to the model.

Distributional word representations (**Turian et al., 2010; Mikolov et al., 2013; Pennington et al., 2014)** trained in an unsupervised manner on large-scale corpora are widely used in modern natural language processing systems. However, these approaches only obtain a single global representation for each word, ignoring their context. Different from traditional word representations, contextual embeddings move beyond word-level semantics in that each token is associated with a representation that is a function of the entire input sequence. These context-dependent representations can capture many syntactic and semantic properties of words under diverse linguistic contexts.

Peters et al., 2018; Devlin et al., 2018; Yang et al., 2019; Raffel et al., 2019 have shown that contextual embeddings pre-trained on large-scale unlabelled corpora achieve state-of-the-art performance on a wide range of natural language processing tasks, such as text classification, question answering and text summarization. This project is also a classification project and here I used the contextual embedding of numerical tokens.

Further analysis **(Liu et al., 2019a; Hewitt and Liang, 2019; Hewitt and Manning, 2019; Tenney et al., 2019a)** demonstrate that contextual embeddings are capable of learning useful and transferable representations across languages. I classify the range of the numeric in a particular position of the text and in this project, I also used the contextual embedding of a targeted token as an input to the mode. For this purpose, I used the SEC-BERT-NUM model which is deep learning pre-trained model that gives the contextual embedding of the targeted numerical in the text.

A line of work has proposed methods to improve the numeracy ability of the neural model. For example, **Geva et al. (2020)** improve numeracy by generating more training data including numbers. **Zhang et al. (2020)** transform the number in the text into a scientific notation form. However, it does not guarantee that the models can learn numerical knowledge from the perspective of magnitude and mathematical notation. Other studies attempt to design a numeral embedding to improve numeracy ability. They demonstrate that their designed embeddings can perform well in probing tasks on numeracy **(Jiang et al. 2019; Sundararaman et al. 2020)**. However, none of them pre-trains their model with numeral embeddings on a large corpus in a self-supervised manner. In this project I also t used the contextual embedding of a numerical token and tried to check the numerical ability of the model by predicting the magnitude of the numerical on a particular position of the text.

Chapter 3-Methodology

3.1) Text Pre-processing:

Data pre-processing is a technique of converting the raw data into a structured data format that machine learning models can understand.

In Natural Language Processing, text pre-processing is a method to clean the data and make it in a structured format for model training. Text pre-processing helps the machine learning model to predict more accurately.

There are several steps to clean the data as follows-

Punctuations delete, lower all the text, **steaming** means reducing the word into a word stem but it does not guarantee that its root form will not lose its meaning, l**emmatization** means the process also stems the word to its root form and does not lose its meaning. It has a predefined dictionary for the English language that stores the context of words and uses it during the diminishing. **Removing stop words** mean**s** removing the words that occur commonly across all the documents in the corpus, **Tokenizations** means breaking the sentences into a word called a token.

Here in this project, we use the BERT model which is a deep learning-based pre-trained model used for contextual embedding. So, for tokenizing the text, we use the BERT tokenizer and we do not any text preprocessing techniques other than tokenization because preprocessing is not needed when using pre-trained language representation models like BERT. In particular, it uses all of the information in a sentence, even punctuation, and stop-words, from a wide range of perspectives by leveraging a multi-head self-attention mechanism.

3.2) Word Embedding:

We know that the machine learning model can't understand the text. It understands the numeric value which is called a vector. In natural language processing (NLP), a word embedding is a representation of a word into a vector. The embedding is used in text analysis. Typically, the representation is a real-valued vector that encodes the meaning of the word in such a way that, the words that are closer in the vector space are expected to be similar in meaning. It extracts all the features from the text fed into the machine learning model as input to process the text data. It helps in reducing the dimensionality and capturing the word having a similar meaning. It helps in capturing the word having a similar meaning.

In this project, we predict the magnitude of the numeric to a particular position of the text based on the context of the other words of the text. So, we provide **contextualized word embedding** of the targeted token as input to the model.

There are several types of word embedding techniques – Bag of Word, TFIDF, Word2Vec, etc. But this word embedding technique gives the fixed embedding means it gives always a fixed vector for a particular word but the problem arises when a particular word is used with a different meaning in different sentences. Let's discuss it with an example-

Let’s our corpus has their sentences –

1. The **Bank** of Baroda
2. The River **Bank**
3. English Question **Bank** book

Here the word **Bank** is used for different meanings in these three sentences. For the first sentence, it means a financial organization, for the second sentence it means the bank of the river, for the third sentence it means a book. So, we clearly distinguish the meaning of the particular word based on the presence of the other word in the text which means the context of the other word.

But if we use this fixed embedding technique then it provides the same vector for the word bank for these three sentences although the purpose of using the word bank is different for these sentences. So, it is clear that fixed embedding techniques can't distinguish the meaning of the word based on the context of the other word of the text.

But we easily solve this problem using a **contextualized word embedding**. Contextualized word embedding is a technique that captures word semantics in a context such that it can represent differently under different sentences even though it is the same word. It Build a vector for each word conditioned on its context! That is its representation for each token is a function of the entire input sentence.

In this project, use the BERT model for getting the contextualized word embedding for a token. **Bidirectional Encoder Representations from Transformers** (**BERT**) is a family of masked-language models introduced in 2018 by researchers at Google. A 2020 literature survey concluded that "in a little over a year, BERT has become a ubiquitous baseline in Natural Language Processing (NLP) experiments counting over 150 research publications analyzing and improving the model.”

BERT was originally implemented in the English language at two model sizes: (1) $\text{BERT}_{\text{BASE}}$: 12 encoders with 12 bidirectional self-attention heads totaling 110 million parameters, and (2) $\text{BERT}_{\text{LARGE}}$: 24 encoders with 16 bidirectional self-attention heads totaling 340 million parameters. It is a deep learning pre-trained model used for contextualized embedding. Both models were pre-trained on the Toronto Book (800M words) and English Wikipedia (2,500M words). It transforms text into a 768-dimension feature vector.

BERT is based on the Transformer architecture. Specifically, BERT is composed of a Transformer encoder layer. It is a deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus. Context-free models such as word2vec generate a single-word embedding representation for each word in the vocabulary, where BERT takes into account the context for each occurrence of a given word. For instance, whereas the vector for "Bank" will have the same word2vec vector representation for both of its occurrences in the sentences "The Bank of Baroda" and "Question Bank English Book", BERT will provide a contextualized embedding that will be different according to the sentence.

Here in this project first try to use the BERT model as follows-

Let's take an example text from our corpus:

Text- National Garden Festival begins Saturday, June 23rd.

Here are targeted numeric is 23. So, we aim to get a contextualized embedding for the token 23 using BERT.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased',output_hidden_states=True)

# Add the special tokens.
marked_text = "[CLS] " + text + " [SEP]"

# Split the sentence into tokens.
tokenized_text = tokenizer.tokenize(marked_text)
```

BERT tokenizer uses what is called a **WordPiece** tokenizer. It works by splitting words either into full forms (e.g., one word becomes one token) or into *word pieces* — where one word can be broken into multiple tokens.
Bert Tokenizer tokenized the example text as follows –

0 [CLS]	101
1 national	2,120
2 garden	3,871
3 festival	2,782
4 begins	4,269
5 saturday	5,095
6 ,	1,010
7 June	2,238
8 23rd	13,928
9 [SEP]	102

Here faces the **first challenge**, because the targeted token '23' is not present as the BERT tokenizer tokenized the text in such a way that the token is 23rd not 23. So can't get the embedding of the targeted token as the targeted token is not present. This problem happens also if the targeted numeric is float. Suppose the targeted numeric is 10.38 then the BERT tokenizer tokenized it three different tokens '10', '.', '38'. So can't get the embedding of the targeted token 10.38.
We solve the above problem using the **SEC-BERT** model instead of using the BERT model for making the contextual embedding of the targeted token.
SEC-BERT is a family of BERT models for the financial domain, intended to assist financial NLP research and FinTech applications.

SEC-BERT consists of the following models:

SEC-BERT-BASE:

Same architecture as BERT-BASE trained on financial documents.

SEC-BERT-NUM (This model used in this project):

Same as SEC-BERT-BASE but we replace every number token with a [NUM] pseudo-token handling all numeric expressions in a uniform manner, disallowing their fragmentation).

SEC-BERT-SHAPE:

Same as SEC-BERT-BASE but we replace numbers with pseudo-tokens that represent the number's shape, so numeric expressions (of known shapes) are no longer fragmented, e.g., '53.2' becomes '[XX.X]' and '40,200.5' becomes '[XX,XXX.X]'.

SEC-BERT-NUM model was pre-trained on 260,773 10-K filings from 1993-2019, publicly available at U.S. Securities and Exchange Commission (SEC).

In this project, we used the pre-trained SEC-BERT-NUM model from the hugging face library. Before using this model, we replace the targeted numeric using a special token '[NUM]' so that when the model tokenized the text then we get a token '[NUM]'. So, we get the embedding corresponding to the '[NUM]' token based on the context of the other word in the text. In this way, we get the contextual embedding of the targeted numeric token using this model. Let's discuss it with an example from our corpus -

Original text -'100 Most Anticipated books releasing in 2010'

here the targeted numeric is 2010. So first replace the targeted numeric using the '[NUM]' token. So processed text is

processed_text -'100 Most Anticipated books releasing in [NUM]'
Then used the pre-trained SEC-BERT-NUM model from the hugging face library

```
tokenizer = AutoTokenizer.from_pretrained("nlpaueb/sec-bert-num")
model = BertModel.from_pretrained('nlpaueb/sec-bert-num')

# Split the sentence into tokens.

tokenized_text = tokenizer.tokenize(processed_text)

# Map the token strings to their vocabulary indices.

indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
index = tokenized_text.index('[NUM]')
tokens_tensor = torch.tensor([indexed_tokens])
```

```
# Put the model in "evaluation" mode, meaning feed-forward operation.

model.eval()
with torch.no_grad():
  last_hidden_states = model(tokens_tensor)[0]

# Getting the contextual embedding of the targeted token (numeric) and converting this from torch tensor to NumPy array

embedding_of_num = last_hidden_states[:,index,:]
embedding_of_num_use = embedding_of_num[0].cpu().detach().numpy()
```

In this project, the dataset is too large and for each sample of the dataset there is a targeted numeric and for each targeted numeric we make an embedding using this model. Since this is a deep-learning complex model so it takes a lot of time to produce the embedding. To avoid this issue, we periodically dumped the embedding into a file when the model produced the embedding so that in the future, we load the embedding from the file and use it as an input to the model and the model predicts the magnitude of the numeric as an output.

3.3) Train-Test-Split:

The train-test split is used to estimate the performance of machine learning, and deep learning algorithms that apply to prediction-based Algorithms or Applications. This method is a fast and easy procedure to perform such that we can compare our machine learning model results to machine results.

We need to split a dataset into train and test sets to evaluate how well our machine learning, deep learning model performs. The train set is used to fit the model, and the statistics of the train set are known. The second set is called the test dataset, this set is solely used for predictions and the statistics of the test set are not known. The model can predict the test dataset based on the statistics that which machine learns during the training.

In this project, we split our whole dataset into a training set and test set where the training set contains 90% data and the test set contains 10% data. For the experiment, we use two types of train test split in our project. One is **a random split** used by the Sklearn library and the other is **out of time** train test split means the training set contains the data of the before-a-period and the test set contains the data of the after-period.

3.4) Model used for Classification:

3.4.1) Supervised learning model:

Supervised learning is one of the types of machine learning in which a machine is trained with well-labeled data and predicts the output based on the data provided. This learning aims to find a mapping function to map the input variable(x) with the labeled or output variable (y).

The working of this learning is shown by the following figure:

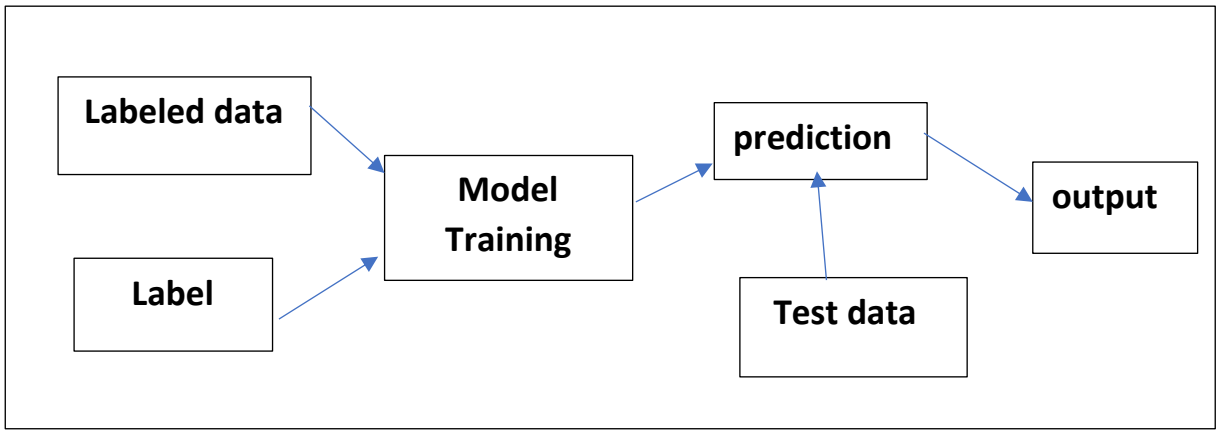


Figure 3.4-1- Working Principle of Supervised Learning

We know that there are mainly two types of supervised learning algorithms-

i) Classification
ii) Regression

For our project purpose, we use a classification-type supervised algorithm. In this project, we use four different classification machine learning algorithms –

i) Random Forest
ii) Logistic Regression
iii) XGBoost
iv) Light Gradient Boosting Mechanism

Let's discuss this algorithm one by one.

3.4.1.1) Random Forest Classifier:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of combining multiple classifiers to solve a complex problem and improve the performance of the model. Since our project is based on classification so we used a random forest classifier for our project.

As the name suggests, ***"Random Forest is a classifier that*** contains ***several decision trees on various subsets of the given dataset and takes the majority voting to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, predicts the final output.

The below diagram explains the working of the Random Forest algorithm:

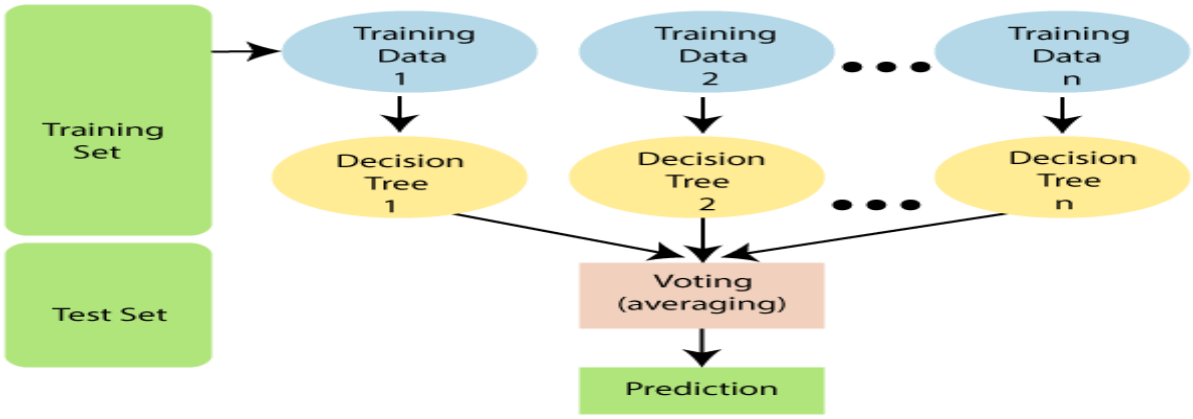


Figure 3.4-2- Working Principle of random forest classifier

The Working process can be explained in the below steps:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for the decision trees that you want to build.

Step-4: Repeat Steps 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Some important hyperparameter for the random forest:

N_estimators: As we have already discussed a random forest has multiple trees and we can set the number of trees we need in the random forest. This is done using a hyperparameter "**n_estimators**".

Max_feature: It is one of the parameters that we can tune to randomly select the number of features at each node.

Max depth: We can decide the depth to which a tree can grow using "**max_depth**", and this can be considered as one of the stopping criteria that restrict the growth of the tree.

In this project, we use a random forest classifier with different hyperparameters for the experiment and try to predict the range of the targeted numeric.

For 1 Lakh sample dataset we use the random forest classifier with hyperparameter n_estimators is 100 and max depth is 7.

For the 5k sample dataset we use the random forest classifier with hyperparameter n_estimators is 180, max depth is 13, max_leaf_nodes is 231, and min_samples_split is 60.

For the 10k dataset sample dataset we use the random forest classifier with hyperparameter n_estimators is 100, max depth is 12, max_leaf_nodes is 295, min_samples_split is 26, n_jobs is -1.

For finding the best hyperparameter for the model we use a hyperparameter tuning framework called Optuna.

3.4.1.2) Logistic regression:

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. **it gives the probabilistic values which lie between 0 and 1**.

Logistic Regression is much similar to Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.

Logistic Regression is linear Regression plus sigmoid function which is an activation function similar to a single neuron **that's why we consider Logistic Regression as a single-neuron neural network.**

In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

The below image is showing the logistic function:

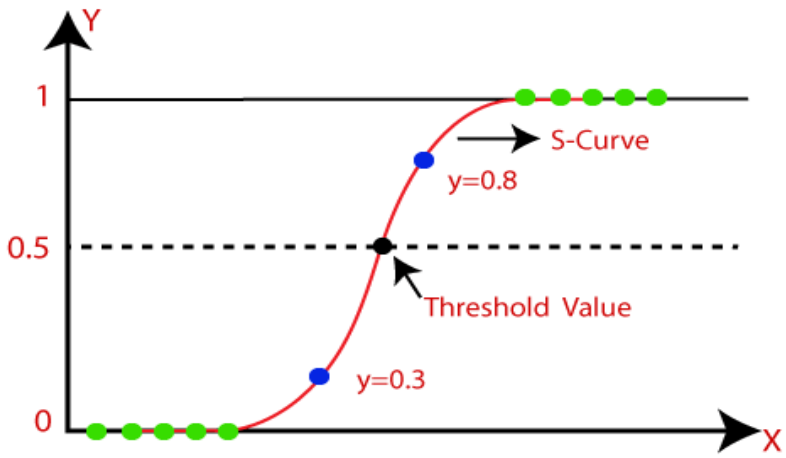


Figure 3.4-3- Working Principle of logistic regression

Some important hyperparameters of logistic regression:

Solver: It is the algorithm to use in the optimization problem. The choices are *{'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}*, default='lbfgs'.

Penalty (or regularization): It intends to reduce model generalization error, and is meant to disincentivize and regulate overfitting. The technique discourages learning a more complex model, to avoid the risk of overfitting. The choices are {'l1', 'l2', 'elasticnet', 'none'}, default='l2'. However, some penalties may not work with some solvers.

C (or regularization strength)**:** It must be a positive float. Regularization strength works with the penalty to regulate overfitting. Smaller values specify stronger regularization and a high value tells the model to give high weight to the training data.

In this project, we use logistic regression with different hyperparameters for the experiment and try to predict the range of the targeted numeric.

For the 1 Lakh sample dataset, we use the logistic regression with hyperparameter solver 'ibfgs' and max_iter is l Lakh.

For the 5k sample dataset we use the logistic regression with hyperparameter solver 'ibfgs' and max_iter is lakh.

For the 10k sample dataset we use the logistic regression with hyperparameter solver 'liblinear' and max_iter is lakh and c are 0.01.

For finding the best hyperparameter for the model we use a hyperparameter tuning framework called Optuna.

3.4.1.3) XGBoost:

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models. It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction. XGBoost stands for "Extreme Gradient Boosting" and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.
One of the key features of XGBoost is its efficient handling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing. Boosting is an ensemble modelling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

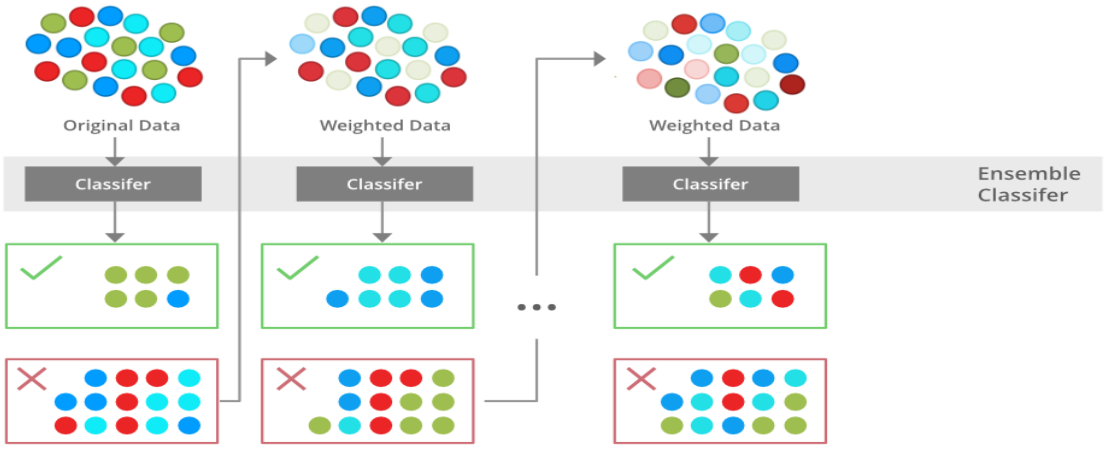


Figure 3.4-4- Working Principle of Boosting Mechanism

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of the predecessor as labels.

XGBoost is an implementation of Gradient Boosted decision trees. In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers or predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

In this project, we use the XGBoost classifier model for prediction. First, install the XGBoost model in the notebook then from this import XGBoost Classifier with the default hyperparameter. Since at first, our dataset sample is too large (it contains 1lakh data sample) so XGBoost took too much time to train it takes nearly about 4 hours and my system started hanging. So, solving this issue reduced our data sample size.

3.4.1.4) Light Gradient Boosting Machine:

LightGBM is a gradient-boosting framework based on decision trees to increase the efficiency of the model and reduces memory usage. It uses two novel techniques-

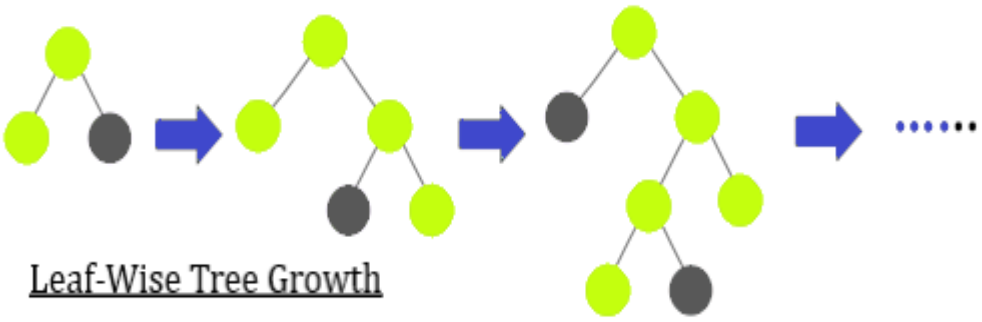
1. **Gradient-based One Side Sampling**
2. **Exclusive Feature Bundling (EFB)**

which fulfills the limitations of the histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. They comprise together to make the model work efficiently and provide it a cutting edge over other Gradient Boosting Decision Tree frameworks and this model takes less time for training than other Gradient Boosting Decision Tree models.

The architecture of light GBM:

LightGBM splits the tree leaf-wise as opposed to other boosting algorithms that grow tree level-wise. It chooses the leaf with the maximum delta loss to grow. Since the leaf is fixed, the leaf-wise algorithm has a lower loss compared to the level-wise algorithm. Leaf-wise tree growth might increase the complexity of the model and may lead to overfitting in small datasets.

Below is a diagrammatic representation of Leaf-Wise Tree Growth –



.

Figure 3.4-5- Leaf wise tree growth

Some important hyperparameters of light GBM -

Num_leaves: This is the main parameter to control the complexity of the tree model. Theoretically, we can set num_leaves=2^(max_depth) to obtain the same number of leaves as a depth-wise tree. However, this simple conversion is not good in practice. The reason is that a leaf-wise tree is typically much deeper than a depth-wise tree for a fixed number of leaves. Unconstrained depth can induce over-fitting. Thus, when trying to tune the num_leaves, we should let it be smaller than 2^(max_depth). For example, when the max_depth=7 the depth-wise tree can get good accuracy, but setting num_leaves to 127 may cause over-fitting, and setting it to 70 or 80 may get better accuracy than depth-wise.

Max_depth: we also can use max_depth to limit the tree depth explicitly and this can be considered as one of the stopping criteria that restrict the growth of tree.

Learning rate: Controls the rate or speed at which the model learns. Specifically, it controls the amount of apportioned error that the weights of the model are updated with each time they are updated, such as at the end of each batch of training examples. When the overfitting problem arises, we can set the learning rate to a small value to avoid this problem.

In this project, we use light GBM to predict the range of the numeric with the hyperparameter tuning. For finding the best hyperparameter we use the hyperparameter tuning framework called Optuna and the best parameter are –

For 1lakh data we use this model with the hyperparameter learning_rate=0.05, max_depth=5, num_leaves=30, num_iteration=1000

3.4.2) Deep learning model:

Deep learning is a subset of machine learning and artificial intelligence that behaves identically as a human brain gathers information or knowledge. Neural Networks are a type of Deep Learning which uses interconnected nodes or neurons in a layered structure identical to the human brain. Neural Network is a method in artificial intelligence that makes the computer learn to process the data similar to the human brain.

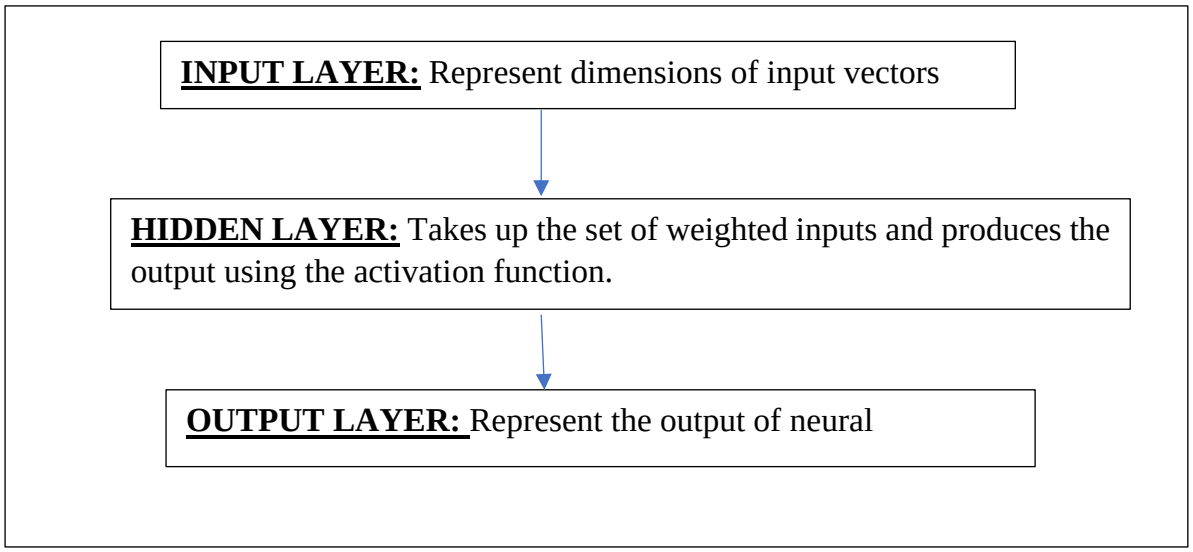


Figure 3.4-6- General architecture of a deep learning network

3.4.2.1) MLP:

A multilayer artificial neuron network is an integral part of deep learning. A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP). An MLP is a typical example of a feedforward artificial neural network. The number of layers and the number of neurons is referred to as hyperparameters of a neural network, and these need tuning. The weight adjustment training is done via backpropagation. Deeper neural networks are better at processing data. However, deeper layers can lead to vanishing gradient problems. Special algorithms are required to solve this issue.

The MLP learning procedure is as follows:

- Starting with the input layer, propagate data forward to the output layer. This step is forward propagation.
- Based on the output, calculate the error (the difference between the predicted and known outcome). The error needs to be minimized.
- Backpropagate the error. Find its derivative concerning each weight in the network, and update the model.

Repeat these three steps given above over multiple epochs to learn ideal weights. Finally, the output is taken via a threshold function which is called the activation function to obtain the predicted class labels.

The MLP Network is shown in Figure –

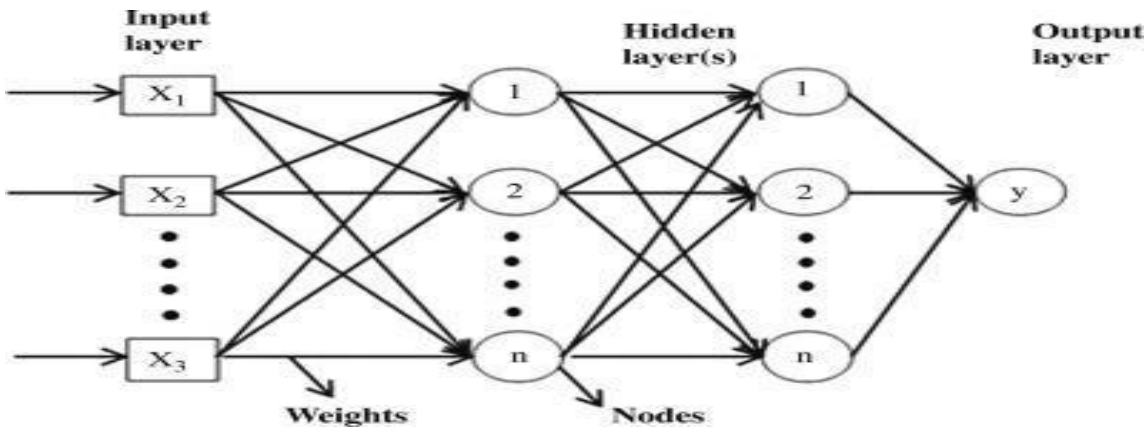


Figure 3.4-7- General architecture of MLP

In this project, we used the MLP model also for prediction. We use this model from the Sklearn library and then tune its hyperparameter to find the best-hidden layer size, learning rate, etc. Here we use softmax as an activation function to the last hidden layer.

The best parameter for this model is for different-size samples –

1 lakh sample dataset hidden_layer_sizes=(200, 100) and learning_rate_init=0.01, random_state=5, verbose=True, with softmax activation to the last hidden layer.

For 10k sample dataset hidden_layer_sizes=(300,200,100,50) and random_state=5, verbose=True, max_iter=500, learning_rate_init=0.000001.

3.4.2.2) CNN:

In Convolutional Neural Network, variations of multilayer perceptron are used, and here the connection between the nodes does not form a cycle. It is a class of deep neural networks. Convolutional Neural Networks' applications mainly revolve around visual classification like face recognition, object detection, etc. But it proved to be good enough for performing Natural Language Processing tasks.

Convolutional Neural Network has convolutional and downsampling layers. The neurons in the convolutional layer can scan inputs for patterns, and downsampling layers reduce the feature map dimensionality, which eventually improves actual performance. Downsampling layers are placed after convolutional layers in a ConvNet or Convolutional Neural Network, and after that, one or more multilayer perceptrons are placed.

The words in a text can be represented as vectors in a vector space based on the entire vocabulary. In-text classification with Convolutional Neural Network, the sequential data is processed by CNN, so one-dimensional convolutions are used. It can identify patterns in a text by changing the size of kernels and merging their outputs, and the output of each convolution is fired when a particular pattern is found. That is how patterns can be identified in a sentence, even if they are present anywhere.

In this project, we used CNN to predict the range of the target numeric. For 1 lakh sample data CNN takes lots of time to train and that's why my runtime is disconnected, since the training was done on the Google Colab platform and my system started hanging. So that I have not used CNN for the 1 lakh sample dataset and we reduced our data set sample to 10k. In this data set CNN runs smoothly.

Here we used a sequential model from TensorFlow by the Keras API to design the CNN model. TensorFlow is a deep learning framework powered by Google and Keras is API which is run over the tensorflow. First, we give embedding produced by the SEC-BERT-NUM model as an

input to this sequential model. The embedding size is 9000x768 (since our training dataset has a 9000-sample size and each sample has a targeted numeric and for each numeric, the BERT model produced a 768-dimension feature vector). So, our input layer has 768 neurons. Then used three conv1D layers as a hidden layer. For first Convolution layer used 500 neurons and a filter size 7, for second convolution layer used 400 neurons and a filter size of 5, and for third convolution layer used 300 neurons and a filter size of 4. For these three layers, I used the nonlinear activation function RELU and padding same. Then used a GlobalMaxPooling1D layer to reduce the size followed by this layer. Then used two fully connected dense layers. The first dense layer has 17 neurons and an activation function RELU, and the second dense layer which is an output layer has 7 neurons because our dataset has 7 different classes of numeric. So, each of the 7 neuron representations of each class. This last layer used softmax as an activation function. So, our CNN model is ready and I compile the model using optimizer='adam' and loss function 'sparse_categorial_crossentropy', then fit the model using train set and test set and our model is ready to predict the range of the numeric on test data.

3.4.2.3) LSTM:

In typical neural networks, the input is not dependent on the output and vice versa. Still, when we have to predict the next word in a text, we need to remember the previous terms, and these phenomena are solved by a Recurrent Neural Network in the form of a hidden layer. The previous output step is used as input for the current step in the Recurrent Neural Network. So, at a point in time, two inputs are fed into the current step in the Recurrent Neural Network. One is the current step input and the other is the previous step output. The most important feature of a Recurrent Neural Network is the memory cell, it stores the information of the previous calculations. Unlike the other neural networks in Recurrent Neural Networks, the same parameters are used for all the inputs, and the same task is performed. Thus, the complexity is less here. Here first, the input is given, and then the current state is calculated with the help of current input and previous state output, and now the current state becomes the previous step for the next step. After all the time steps are done, the final state calculates the result.

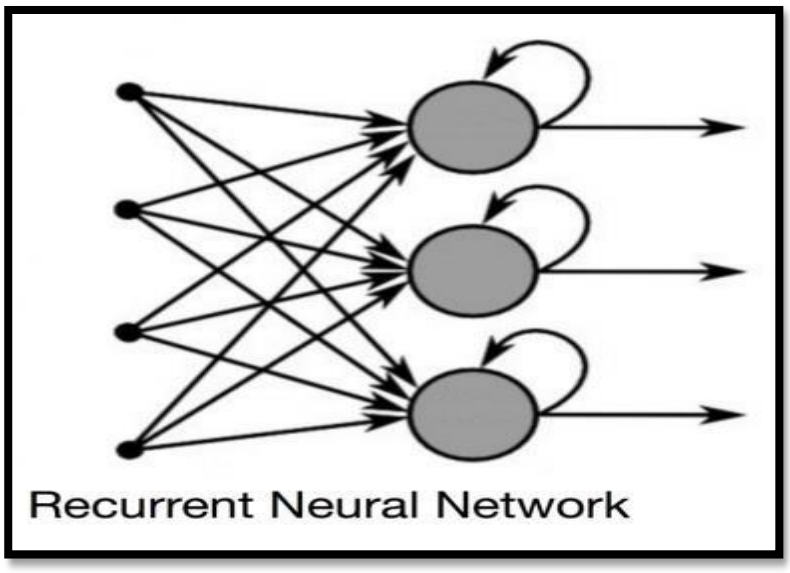


Figure 3.4-8- General architecture of RNN

But the problem in RNN is its short-term memory that is In a Recurrent Neural Network, some previous information is preserved with the help of a memory cell. Still, suppose there is a large chunk of text, a recurrent neural network may omit some essential details because it has difficulty passing information from previous time steps to later time steps. This is called Short-Term Memory, and Recurrent Neural Network suffers from it.

Here comes the LSTM or Long Short-Term Memory, is a special kind of Recurrent Neural Network. It can remember information for a longer period because it only remembers relevant information and forgets all the information that is not relevant.

The key feature of Long Short-Term Memory is Cell State, it acts as the memory of the network. The Cell State carries only the important information while processing the sequence, and new relevant information gets added to it. The old non-relevant information gets chopped and the effects of short-term memory are reduced, and all this happens by using some gates. These gates are neural networks that decide which information is to be kept and which to forget in the cell state while training.

In Long Short-Term Memory, there is a forget gate that decides if the information is important enough or not. Here, the information from the previous step and the current input is fed into the sigmoid function, giving the output between zero and one. If the value is closer to 0, the information is meant to forget, and if the value is more relative to 1, then the information is kept.

Now we discuss the working principle of the LSTM network:

The only difference in the workflow of LSTM and RNN is that the Internal Cell State (c_i) is passed with the hidden state (h_i). The Input Gate is used to update the cell state. The hidden state information of a current state is passed in the tanh activation function, which gives the output between -1 and 1 and then the output of the sigmoid and the output of tanh are multiplied. Thus, the sigmoid output becomes the deciding factor in keeping the tanh output. A hidden state does the prediction and preserves previous states' information. The previous hidden state and current state input are fed into the sigmoid activation function. Modified cell states are passed into the tanh activation function. Then, the tanh and sigmoid output multiply and give the hidden state output. The modified cell state and hidden form are carried over in the next time step.

The above-stated work is shown below:

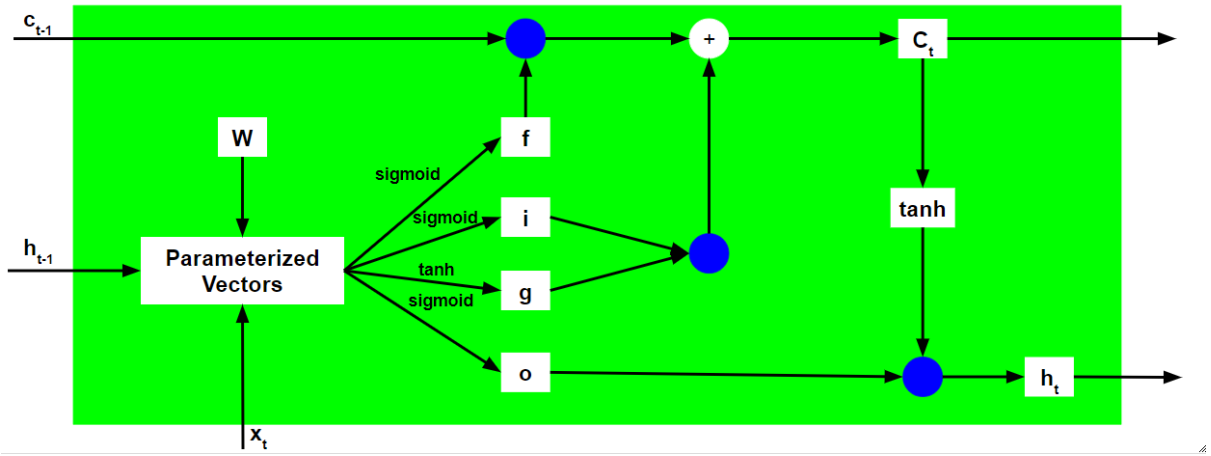


Figure 3.4-9- General architecture of LSTM

In this project, we used LSTM for predicting the range of the target numeric. For 1 lakh sample data, LSTM takes lots of time to train and that's why my runtime is disconnected, since the training was done on the Google Colab platform and my system started hanging. So, that I have not used CNN for a 1 lakh sample dataset and we reduced our data set sample to 10k. In this data set CNN runs smoothly.

Here we used a sequential model from TensorFlow by the Keras API to design the CNN model. TensorFlow is a deep learning framework powered by Google and Keras is an API that is run over TensorFlow. First, we give embedding produced by the SEC-BERT-NUM model as input

to this sequential model. The embedding size is 9000x768 (since our training dataset has a 9000-sample size and each sample has a targeted numeric and for each numeric, the BERT model produced a 768-dimension feature vector). So, our input layer has 768 neurons. Then used three LSTM layers as hidden layers having neurons 300, 200, 100 respectively. Then used two fully connected dense layers. The first dense layer has 17 neurons and the activation function RELU, the second dense layer which is an output layer has 7 neurons because our dataset has 7 different classes of numeric. So, each of the 7 neuron representations of each class. This last layer used softmax as an activation function. So, our LSTM model is ready and I compile the model using optimizer='adam' and loss function 'sparse_categorial_crossentropy', then fit the model using train set and test set and our model is ready to predict the range of the numeric on test data.

Chapter 4-Experiment, Results, and Analysis

4.1) Corpus:

For this project, I used the dataset called Numerecy_600k. There are two parts of this dataset. One is numeracy_600k_article which contains a large amount of news of news articles related to the financial domain and the other is numeracy_600k_comment which contains a large number of market comments. Since the dataset is too large so for convenience l used the different samples of these two datasets. First, experiment on a 1lakh sample of numeracy_600k_article dataset. Here I face some challenges which will discuss later. So that l reduces the dataset size and experiment on a 5k sample of numeracy_600k_article dataset and third for an experiment I increase the dataset size 5k to 10k where I am combined sample of numeracy_600k_article and numeracy_600k_comments. In this dataset the data is divided into 8 categories that represent the magnitude of the numeric. So, the dataset has 8 classes.

This dataset has 5 fields Title, Number, Offset, Length, Magnitude where I use title as input and magnitude as label or target. The title contains the text. The number contains the targeted numeric whose magnitude is predicted. The offset represents the starting position of the targeted numeric. Length is the length of that numeric and magnitude represent the in which range that numeric belongs that should be predicted by the model.

In this dataset, the range of the magnitude is determined by the following-

magnitude 0 if numeric in the range 0 to 1

magnitude 1 if numeric in the range 1 to 10

magnitude 2 if numeric in the range 10 to 10^2

magnitude 3 if numeric in the range 10^2 to 10^3

magnitude 4 if numeric in the range 10^3 to 10^4

magnitude 5 if numeric in the range 10^4 to 10^5

magnitude 6 if numeric in the range 10^5 to 10^6

magnitude 7 if the numeric in the range is greater than 10^6

The dataset is as follows-

Title	Number	Offset	Length	Magnitude
Learn to Play Ice Skating Classes for Kids 3+	3.0	43	1	1
Tech 101: capturing EVPs with a digital recorder	101.0	5	3	3
Rat City Roller Girl bout 3 this Saturday	3.0	26	1	1
Age Discrimination of People in Their 40s	40.0	38	2	2
10 best films of 2009 - What's on your list?	2009.0	17	4	4

Table 4.1-1- Dataset used in the experiment

Let's discuss about the data statistics of the original dataset numeracy_600k_article as well as different sample size of the original dataset. Here we used different sample size 1 lakh, 5k, 10k of this dataset. Sample size 1 lakh and 5k of numeracy_600k_article have 8 classes and sample size 10k of combined numeracy_600k_article, numeracy_600k_comment has 7 classes because after sampling class 7 is not present in the dataset sample. The distribution of the classes of original dataset and different sample size of original dataset are given bellow-

Sample size	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
Whole 600k article dataset	1879	218260	188838	41292	145829	2030	1840	32
1 lakh sample size	311	36294	31597	6958	24222	315	296	7
5k sample size	10	1848	1562	327	1230	14	9	32
10k sample size	1219	3740	2790	940	1250	29	32	_

Table 4.1-2- Dataset Statistics

4.2) Working Algorithm:

Step 1: Import all dependencies and load the data into the Pandas DataFrame.

Step 2: Create a list called corpus and add all titles to this corpus.

Step 3: Replace the targeted number of each title in the corpus with the special token called '[NUM]'.

Step 4: Create the contextual embedding of the '[NUM]' token using the SEC-BERT-NUM model and dumped this embedding into a file for convenience.

Step 5: Load the embedding from the file to the Numpy array or Pandas DataFrame this is our input data and the corresponding magnitude is the label. Now split the processed data that is embedded and its corresponding label into a training set and test set. (For 1 lakh and 5k sample datasets used out of time split, for 10k dataset used random split).

Step 6: Train the model by the training set and find the best parameter for the model by hyperparameter tuning.

Step 7: Test the model on the test data and node down the accuracy, f1 score macro, and f1 score micro for that model.

4.3) Results:

Here in this project, I used different machine learning, deep learning model, different data size and different train test split methods for the experiment. I used Out of time train test split for a 1 lakh dataset and a random train test split for a 10k sample dataset. Now I discuss the results which means the accuracy, f1_macro, and f1_micro of this different model.

Model name	Sample size	Accuracy on the train set	Accuracy on the test set	F1 micro on the train set	F1 micro on the test set	F1 macro on the train set	F1 macro on the test set
Random forest classifier	1 lakh	0.5773	0.5413	0.5773	0.5413	0.2263	0.2107
Random forest classifier	5k	0.8090	0.5274	0.8090	0.5274	0.3169	0.2052
Random forest classifier	10k	0.8135	0.513	0.8135	0.513	0.6292	0.4045

Model name	Sample size	Accuracy on the train set	Accuracy on the test set	F1 micro on the train set	F1 micro on the test set	F1 macro on the train set	F1 macro on the test set
Logistic Regression	1 lakh	0.6335	0.6006	0.6335	0.6006	0.6234	0.3378
Logistic Regression	5k	0.7994	0.5071	0.7994	0.5071	0.9085	0.2274
Logistic Regression	**10k**	**0.6395**	**0.571**	**0.6395**	**0.571**	**0.5440**	**0.4907**
XGBoost	1 lakh	0.8854	0.6084	0.8854	0.6084	0.9428	0.3067
Light GBM	1 lakh	0.6702	0.5909	0.6702	0.5909	0.6534	0.3043
MLP	**1 lakh**	**0.7500**	**0.6222**	**0.7500**	**0.6222**	**0.4902**	**0.3459**
MLP	10k	0.6003	0.5420	0.6003	0.5420	0.4180	0.3844
CNN	10k	0.4608	0.4040	0.4608	0.4040	0.2399	0.2138
LSTM	10k	0.4421	0.3970	0.4421	0.3970	0.2416	0.2042

Table 4.3-1- Results of the experiment

The results after using k fold cross validation on 10k dataset –

Model name	Sample size	Accuracy on the train set	Accuracy on the test set	F1 micro on the train set	F1 micro on the test set	F1 macro on the train set	F1 macro on the test set
Random forest classifier	10k	0.8243	0.5239	0.8243	0.5239	0.6493	0.3901
Logistic Regression	10k	0.6744	0.5571	0.6744	0.5571	0.6438	0.4808
XGBoost	10k	0.9998	0.5638	0.9998	0.5638	0.9998	0.4803
Light GBM	10k	0.6730	0.5090	0.6730	0.5090	0.7428	0.3832

Table 4.3-2- Results of the experiment after using k fold cross validation

4.4) Challenges and Discussions:

Since this project is new for me so I face many challenges during the working of this project. Here I discuss these challenges and How I try to solve these challenges –

The first challenge that I faced that is how to find the contextual embedding of a particular numeric token. First, I used BERT to create the contextual embedding of a numeric token but this gave an error because the BERT model not properly tokenized the text containing the numeric token that I discussed earlier in the word embedding section. I overcome this challenge by using a particular type of BERT model named SEC-BERT-NUM. This model is used to create the contextual embedding of a given numeric token in the text.

Here for this project, I used the numerecy_600k dataset which has two subsets those are numeracy_600k_article which contains 600k news articles related to financial text and numeracy_600k_comments which contains 600k market comments. First, I used the numeracy_600k dataset for my project since this dataset set is too large it is difficult to handle in the Colab notebook so I take a sample of size 1 lakh of this dataset. But here I faced some problems during training time that is 1 lakh sample takes too much time when I trained machine learning and deep learning model and the Colab runtime is disconnected during the training, my laptop was also to much heated and hanging during the training process so overcome this problem I reduced first my sample size from 1 lakh to 5 thousand but here the data sample size is very small concerning the original size of dataset so this effects on predictions and overfitting is occur. So, then I again increase the sample data size from 5 thousand to 10 thousand and I make this sample of size 10k with the combination of numeracy_600k_article and numeracy_600k_comments. So, in this project, I experiment with different samples of the original dataset of sizes 1 lakh, 5k, and 10k respectively.

After training I try to calculate the accuracy of the model in the training set as well as the test set then I see that overfitting is occur that is model is performing well in the training set but not perform well in the test set that is model not generalized the feature of data of test set. So, to overcome this I used the hyperparameter tuning technique.

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned. The same kind of machine learning model can require different constraints, weights, or learning rates to generalize different data patterns. These measures are called hyperparameters and have to be tuned so that the model can optimally solve the machine-learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss. There are many techniques and frameworks for hyperparameter tuning like Gridsearchcv, Randomizedsearchcv, Optuna, etc. In my project first I manually try to find the optimal hyperparameter for the model then I used Optuna for finding the optimal hyperparameter.

Optuna is a software framework for automating the optimization process of these hyperparameters. It automatically searches for and finds optimal hyperparameter values by trial and error for excellent performance. It uses Bayesian optimization, which is faster and allows for a flexible search space.

The step for using Optuna is described below –

Step 1- Define an objective function

Step 2 - Define a set of hyperparameters to try

Step 3 – Define the variable or metrics you want to optimize

Step 4 – Finally run the function

Here you need to mention the number of trials. In the Optuna world, the term Trial is a single call of the objective function, and multiple such Trials together are called Study**.** In this project, I used number of trials is 100.

Using this hyperparameter technique I overcome the overfitting problem that is I minimize the difference between the accuracy of the training set and test set so using this technique I get the balanced fit model by determining the optimal hyperparameter of the model.

Cross-validation is often used to estimate this generalization performance. Here first I used a fixed training set and test set then I try to check whether my dataset have high variance or not means if I change the training set then the accuracy in the test set is changed or not so for this used k-fold cross-validation. In this process, the dataset is split into k fold and each fold is used as a test set and calculate the accuracy. Finally, the average of all this accuracy is considered the final accuracy of the model. But in my case k fold cross validation results is almost similar to when I used fixed training size. So, from this, we can say that the dataset does not have high variance.

4.5) Analysis:

In this project, I used different machine learning and deep learning model for experiments. From the performance, we can see that the accuracy of the random forest classifier is 54%, the accuracy of logistic regression is 60%, the accuracy of XGBoost is 60%, the accuracy of the light GBM model is 59%, the accuracy of MLP is 62%, the accuracy of CNN is 40% and accuracy of LSTM is 39%. So, from the performance of the model, we see that MLP is perform better than the other model. MLP is a neural network-based model which is suitable for classification prediction problems where inputs are assigned a class or label. This project is also a classification project. For MLP we used multiple layers of neurons for prediction. They are very flexible and can be used generally to learn a mapping from inputs to outputs and we know that the deep learning model performs well in selecting the underlying feature than any machine learning model and based on this feature they mapped to input into output. I think for that reason MLP performs well than the other model. Here I used CNN and LSTM which also are deep learning models but the performance of these two models is very poor because I think that here, we have not used the sequence of words of the text. Here in my approach, I used only the contextual embedding of the targeted numerical token but we know that LSTM and CNN perform better when we provide the sequence of data as input. So, for this reason I think the performance of LSTM and CNN is poor.

Also, here we calculated the f1_macro for the evaluation and we get 34% highest f1_macro score on a 1 lakh size sample of numeracy_600k_article by the model MLP which gives the best accuracy also and we get 49% highest f1_macro score on 10k size sample of

numeracy_600k_article and numeracy_600k_comments dataset. From the performance table we can see that f1_macro score increases for 10k dataset than 1 lakh dataset. I think this is happen because for the 10k sample size dataset, I create a dataset using the combination of numeracy_600k_article and numeracy_600k_comments so this dataset contains the news article and market comments both related text data whereas for the 1 lakh sample size dataset, I used only numeracy_600k_comments only so it contains only the news article related text data that's why the data distribution for 10k sample size dataset is different from 1 lakh sample size dataset. So, for that, we see that the f1_macro score increases in the 10k sample size dataset.

Here I try to find for which text the model predicts wrong. In a 10k sample size, the logistic regression performs well. So Now I try to find out for which text this model predicts wrong. I see that for the text where the comparison between the numerals happens that type of text predicts wrong. Also, see that in the text where the target numerals are a reference to other numerals, the model predicts the wrong range for that type of text. For some market data, the range of the numeric predicts wrongly because this type of data is changeable every day based on the market. The text where the targeted numerical is date or year that type of data predict wrongly by this model and also, we see that in the text that contains the donation, sales-related data where the numerals vary from man to man this type of data predict wrongly by this model. Here I give some examples where the model predicts the wrong magnitude for targeted numerals–

Text	Original Magnitude	Predict Magnitude	Remarks
FIRE VICTOR PCL <FIREm.BK> QTRLY PROFIT FOR THE PERIOD <NUM> MLN BAHT VS 16 MLN BAHT	1	3	Comparison between the numerals
PROPHASE LABS INC <PRPH.O> Q3 SALES <NUM> MLN VS $5.4 MLN	1	2	Comparison between the numerals
QUALSTAR CORP <QBAK.O> QUARTERLY REVENUE FELL 25.7 PCT TO <NUM> MLN	1	2	Reference to other numerals
MDC HOLDINGS INC <MDC.N> SHARES UP 3.7 PCT AT <NUM>	2	1	Reference to other numerals
NYSE ORDER IMBALANCE <XON.N> <NUM> SHARES ON BUY SIDE	6	5	Market data
Laptop spying suits settled by the school district for <NUM>	6	2	Varying amount
Year in review: <NUM> top Crime Examiner stories (photos)	4	1	Date

Table 4.5-1- Error analysis

Chapter 5-Conclusion

Based on several tests on different models of machine learning, random forest like Logistic Regression, XGBoost, Light GBM, and deep learning models like MLP, CNN, and LSTM it concluded that the deep learning model MLP performed well in comparison to other models and also logistic regression model performance almost closed to MLP model. Logistic Regression means linear regression with a sigmoid activation function so we can view a logistic regression as a single-neuron neural network and MLP is a multilayer neural network. CNN and LSTM are also neural network models but this model performs better when the sequence of data is maintained. So here we conclude that the neural network model where the sequential data is not important that's a type of model that performs better than another model. Here we also try the random forest model but the performance of the random forest is not good compare to MLP, Logistic Regression. We know that random forest is a bagging-type ensemble model and this model performs better when the dataset has high variance but using k-fold cross-validation we see that our dataset has low variance means the test set accuracy does not so much depend on the choosing of the training set. So, I think this is a reason that random forests do not perform so well. Here also used two boosting type ensemble models one is XGBoost and the other is light GBM and from the performance, we see that these two models perform better than the random forest model which is a bagging type model. From this, we can conclude that for this project the boosting type model performs better than the bagging type model. In this project, we get the highest accuracy 62% by the model MLP and the highest f1_macro 48% by the model Logistic Regression. So, from this, we conclude that based on accuracy the MLP is the best model (also the accuracy of logistic regression is 60% which is close to MLP), and based on f1_macro the model logistic regression is best.

References

1. Soichiro Murakami, Akihiko Watanabe, Akira Miyazawa, Keiichi Goshima, Toshihiko Yanase, Hiroya Takamura, and Yusuke Miyao. 2017. Learning to generate market comments from stock prices. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), volume 1, pages 1374–1384.

2. Georgios Spithourakis and Sebastian Riedel. 2018. Numeracy for language models: Evaluating and improving their ability to predict numbers. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2104–2115.

3. Sebastian Tschiatschek, Adish Singla, Manuel Gomez Rodriguez, Arpit Merchant, and Andreas Krause. 2018. Fake news detection in social networks via crowd signals. In Proceedings of the 2018 Web Conference, pages 517–524.

4. Yaqing Wang, Fenglong Ma, Zhiwei Jin, Ye Yuan, Guangxu Xun, Kishlay Jha, Lu Su, and Jing Gao. 2018a. Eann: Event adversarial neural networks for multi-modal fake news detection. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 849– 857.

5. Chen, C.-C.; Huang, H.-H.; Takamura, H.; and Chen, H.-H. 2019. Numeracy-600k: Learning numeracy for detecting exaggerated information in market comments. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 6307–6313.

6. Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543.

7. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

8. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805

9. John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4129–4138.

10. John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 2733–2743, Hong Kong, China. Association for Computational Linguistics.

11. Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Multi-task deep neural networks for natural language understanding. arXiv preprint arXiv:1901.11504.

12. Dongling Xiao, Han Zhang, Yukun Li, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. 2020. Ernie-gen: An enhanced multi-flow pre-training and fine-tuning framework for natural language generation. arXiv preprint arXiv:2001.11314.

13. Geva, M.; Gupta, A.; and Berant, J. 2020. Injecting numerical reasoning skills into language models. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 946–958.