Toxic Comment Classification

NaN-Prediction Pending

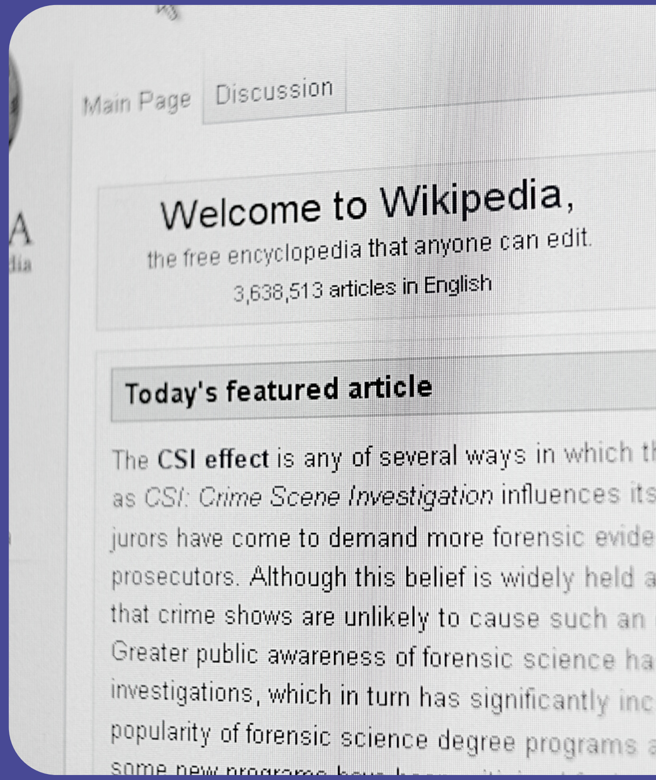
Aneri Dalwadi AU1940153
Mananshi Vyas AU1940289
Nandini Bhatt AU1940283
Kairavi Shah AU1940177

Introduction

The data contains 1.5 million comments from Wikipedia's talk page and has to be classifies into multiclass-classification - toxic, obscene, identity hate, severe, etc. And here we try to classify toxicity and calculate its severity using and comparing various Machine Learning algorithms.

Problem Statement

Social media sites are one of the most popular websites on the internet today flooding with comments. It is vital to manage the user-generated offensive content on many of these sites that can make a user's online experience unpleasant.



GANTT CHART


Tasks
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Choosing a Problem Statement
Visualizing and Pre-processing data
Looking at various approaches
Model Selection
Model Training and Result Analysis
MId-Sem Presentation and Feedback
Hyper-parameter fine tuning, new models train, test and result analysis
Documentation and Presentation

Existing Body Of Work

WORK 1

Toxic Comment Classification on SocialMedia Using SVM and ChiSquare Feature Selection

In this paper, authors have used SVM with TF-IDF as the feature extraction and Chi Square as the feature selection. The best performance obtained using the SVM model with a linear kernel, without implementing Chi Square, and using stemming and stopwords removal with the F 1 - Score equal to 76.57%.

WORK 2

Detecting Offensive Tweets via Topical Feature Discovery over a Large Scale Twitter Corpus

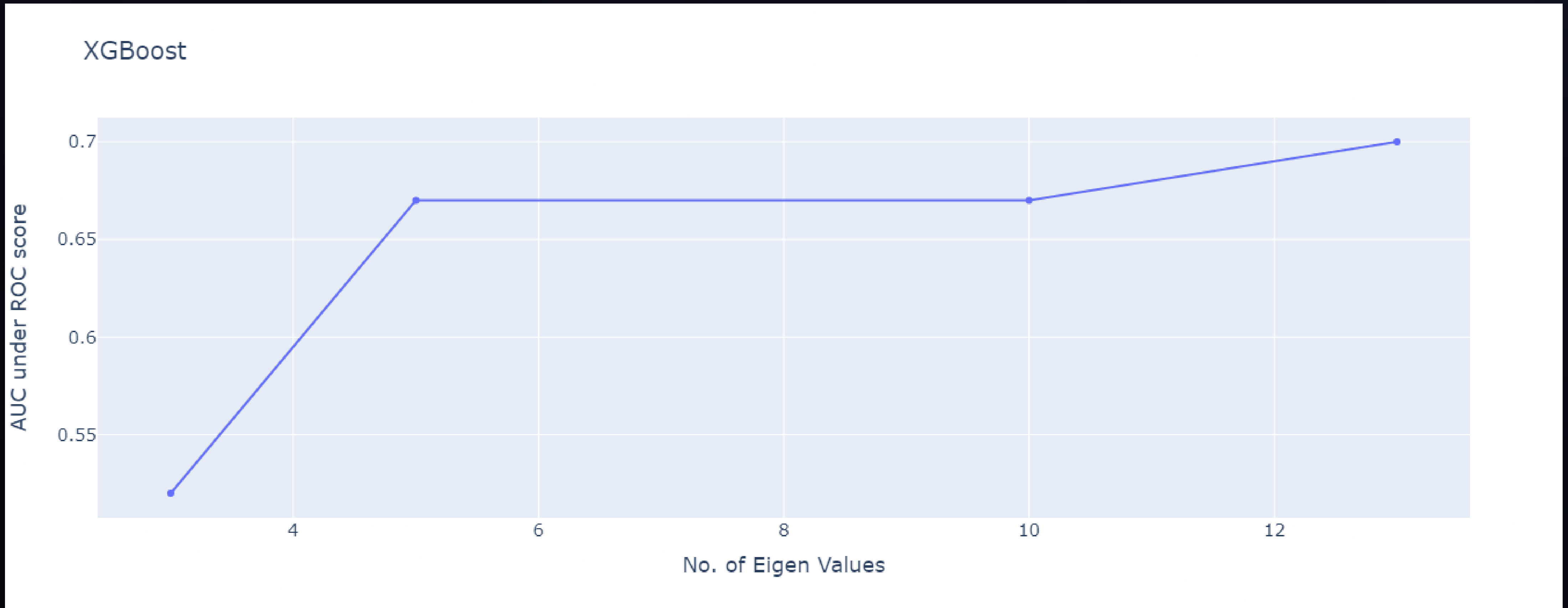
In this paper, authors have used a semi-supervised approach to detect profanity-related offensive content on Twitter. They achieved a 75.1% TP rate with Logistic Regression and a 69.7 % TP rate with popular keyword matching baseline. The false-positive rate was identical for both at about 3.77%.

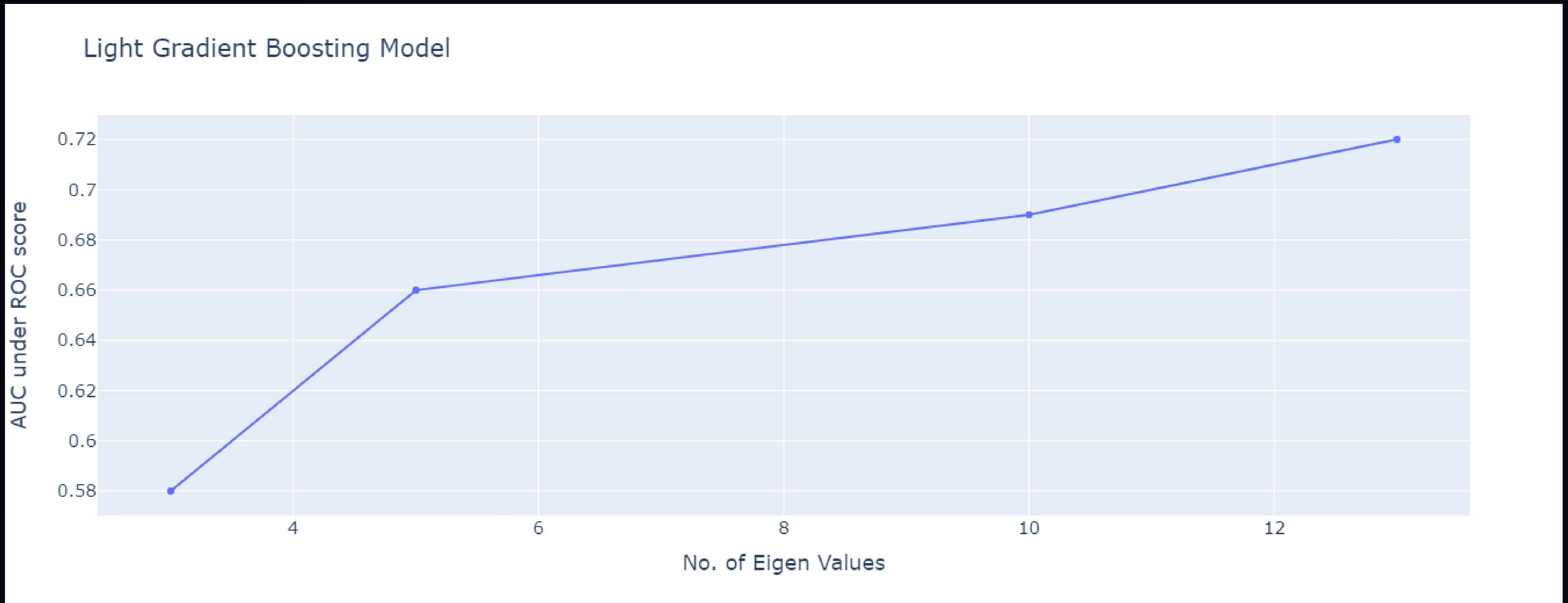
Our Approach

- Data cleaning
- Splitting dataset into tokens
- Selecting 2 character shingles
- Calculate tf_idf
- Create 3D matrix with tf_idf values of these shingles
- Calculate Eigen values to get numeric input features
- Train, tune and analyze models like LGBM, XGBoost and CatBoost
- Cross validate using AUC under ROC matrix since data is imbalanced
- As per results: LGBM model performs the best

Final Results

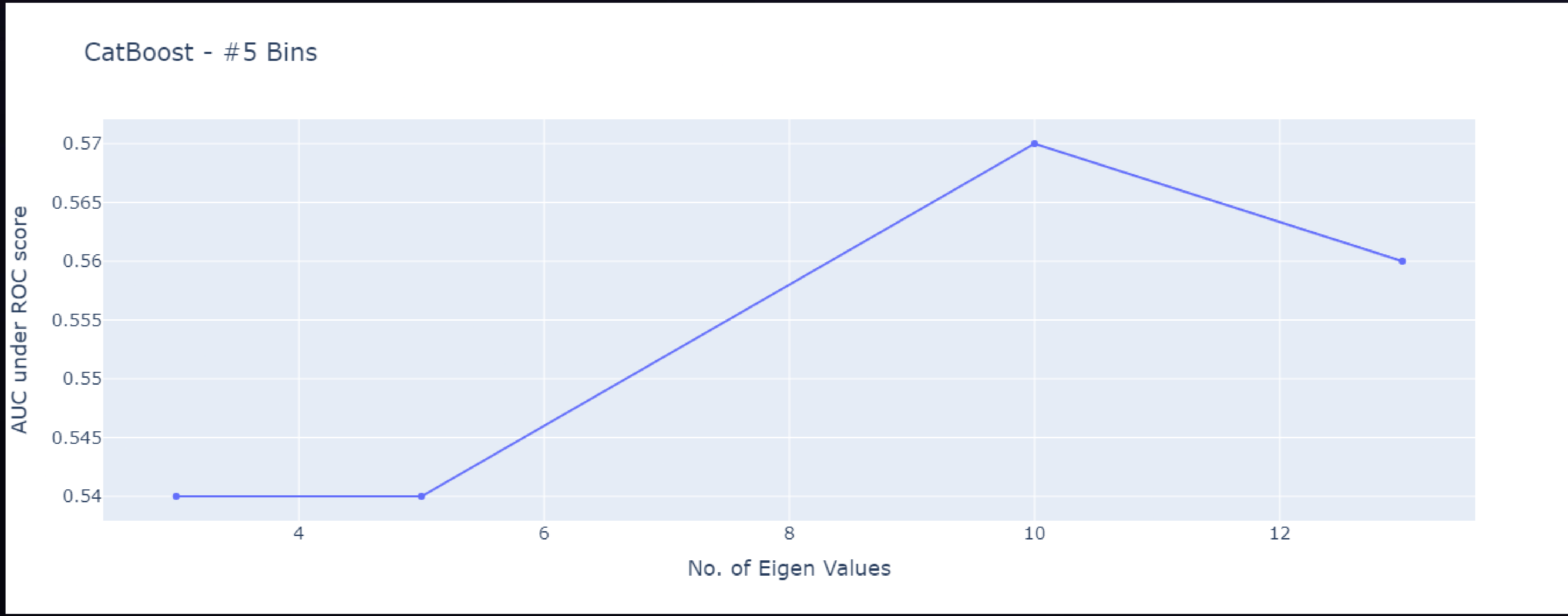
- Applied 3 gradient bosting models amogst which LGBM outperforms with 13 eigen values with accuracy of 0.72 compared to XGBoost but underperforms in terms of training time .
- AUC Score drops on varying character shingles & n-gram for LGBM.
- Hyperparameter tuned are depth of tree, no. of leaves.

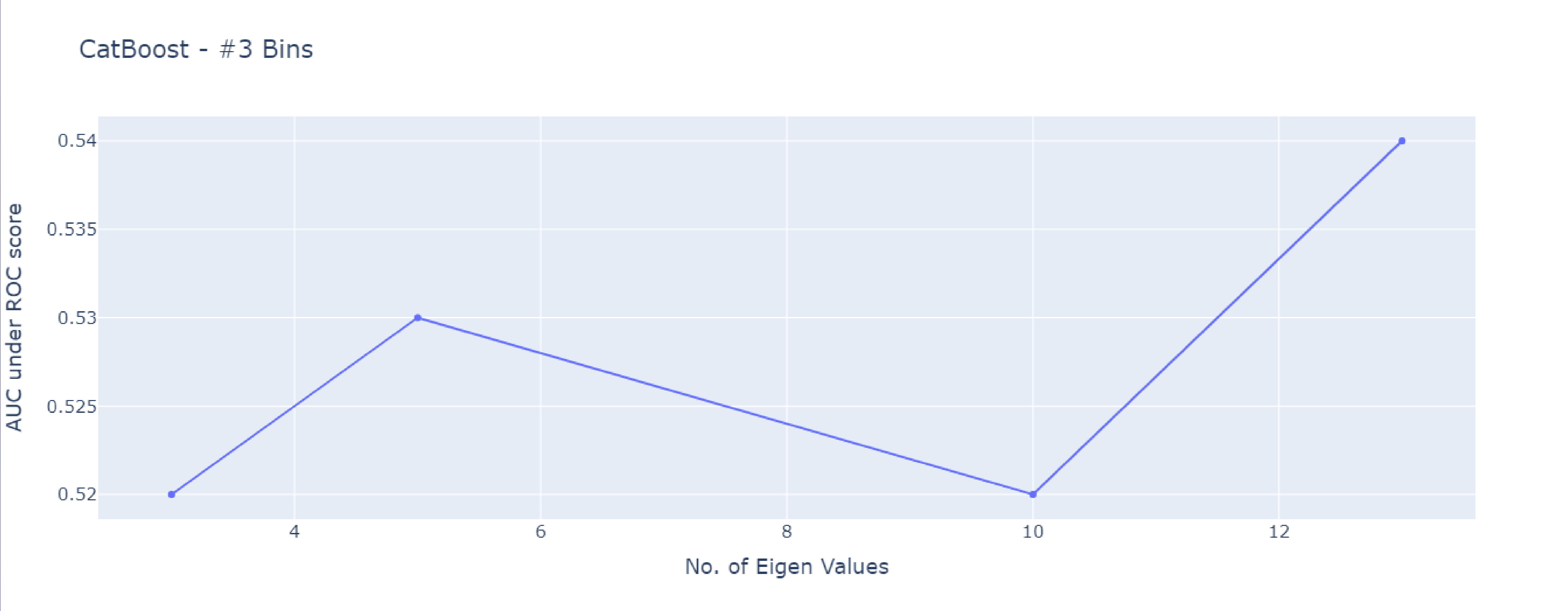




Final Results

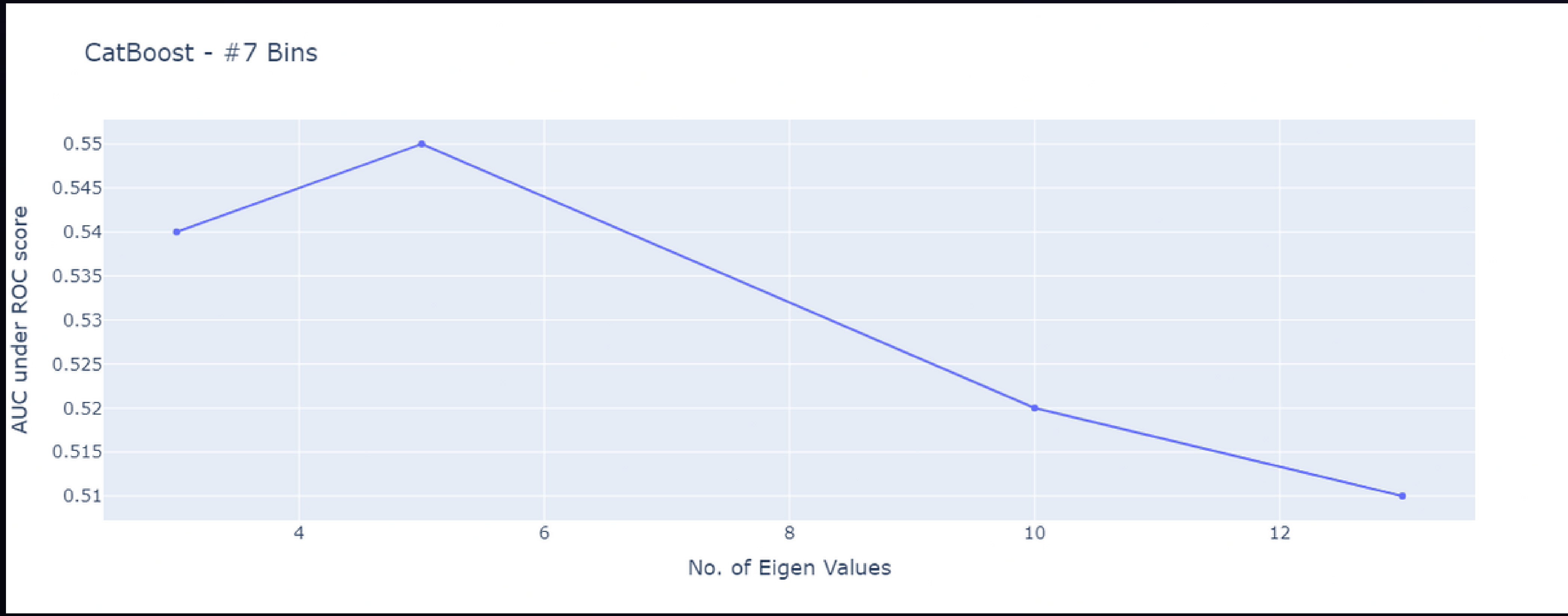
- Increasing valuee of eigrn values showed non-linear trend with AUC score around 0.5.
- Learning rate was fixed.

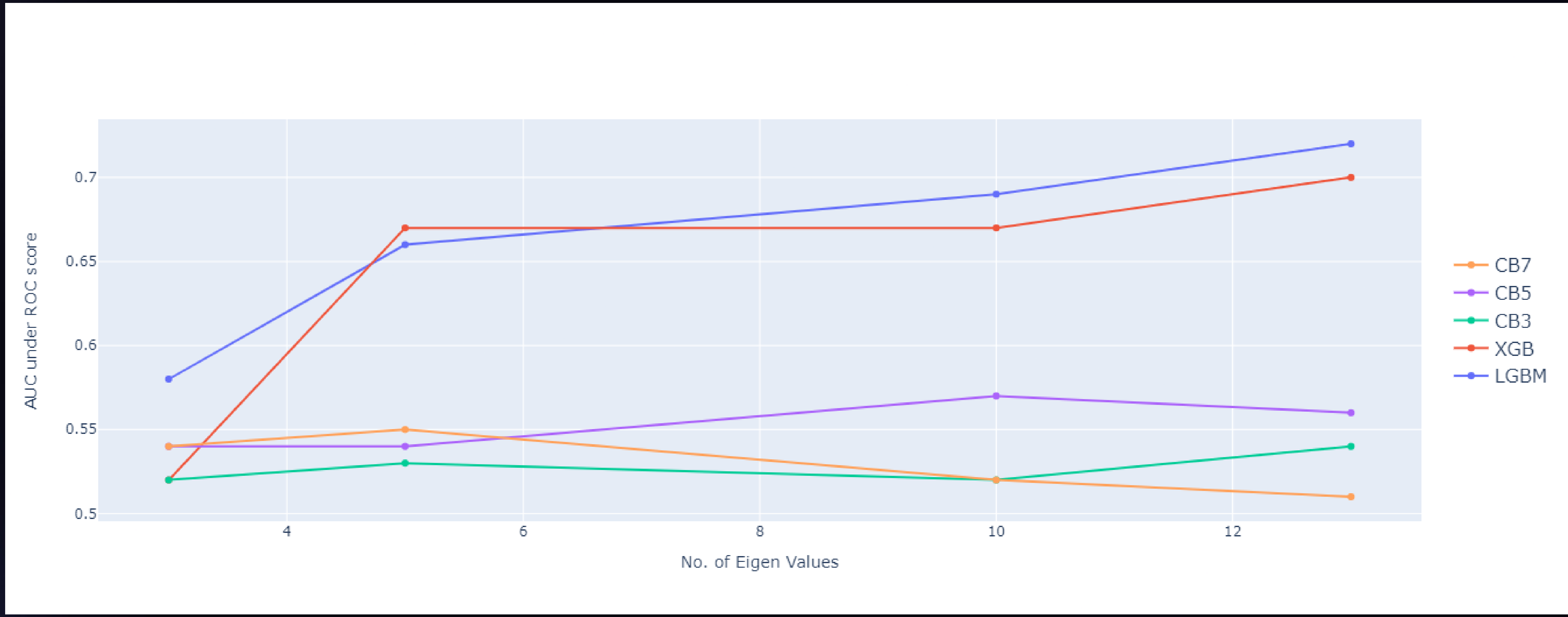




Final Results

- LGBM perform best with 13 eigen vectors with less computation time of 6.5 hours & give AUC under ROC score of 0.72.
- Two character shingles works best for classification for our data.
- Threshold value for AUC curve is 0.7.





Conclusions

Model Used (Time Taken in sec & hrs)	3	5	10	13
LGBM (AUC)	0.58 (12600s [3.5 hrs])	0.66 (16200s [4.5 hrs])	0.69 (21600s [6 hrs])	0.72 (23400s [6.5 hrs])
XGBoost	0.52 (18000s [5 hrs])	0.67 (21600s [6 hrs])	0.67 (28800s [8 hrs])	0.70 (34200s [9.5 hrs])
CatBoost (No. of bins): 3	0.52 (10800s [3 hrs])	0.53 (16200s [4.5 hrs])	0.52 (25200s [7 hrs])	0.54 (28800s [8 hrs])
CatBoost (No. of bins): 5	0.54 (12600s [3.5 hrs])	0.54 (16200s [4.5 hrs])	0.57 (26280s [7.3 hrs])	0.56 (28080s [7.8 hrs])
CatBoost (No. of bins): 7	0.54 (10800s [3 hrs])	0.55 (14400s [4 hrs])	0.52 (25200s [7 hrs])	0.57 (27000s [7.5 hrs])

Number of eigen values

ROLE

ANERI

Implemented codes in R and ggplot for various inferences. Implementation and interpretation of base and gradient boosting models. Feature selection and hyper-parameter tuning.

MANANSHI

Implemented codes in Apache Spark for data cleaning and pre-processing. Understanding and interpreting models. Finding appropriate approach to solve the problem and formulating eigen value matrix.

NANDINI

Helped in implementing pre-processing with use of regex and apache spark. Interpretation of use of eigen values and fine tuning hyperparameters for model implementation.

KAIRAVI

Helped with implementation of models and conversion to csv files and reducing the data size. Interpretation of different eigen values and fine tuning hyperparameters for model implementation.