

**Department of Artificial Intelligence**

**Amrita School of Engineering**

**Coimbatore- 641 112, Tamil Nadu, India**

***A project submitted***

***in partial fulfilment of the requirements for the degree of***

***Master of Technology in Artificial Intelligence***

**By**

**Pratima Chinta**

**(CB.SC.P2AIE23021)**

**&**

**Sneha Nair**

**(CB.SC.P2AIE23022)**

**Supervised by:**

**Dr. SenthilKumar T**

**Dec, 2023**

**MACHINE LEARNING**

**Table 1: Team Members**

|  |  |  |  |
| --- | --- | --- | --- |
| **Roll No** | **Name** | **Official Email id** | **Contribution** |
| CB.SC.P2AIE23021 | Pratima Chinta | cb.sc.p2aie23021@cb.students.amrita.edu | Text Preprocesing, Noise removal, Lasso, Ridge regression, Kmeans with SVM, Fuzzy C-means with KNN, Multinomial Naive Bayes, Kmeans with CNN, RNN-LSTM |
| CB.SC.P2AIE23022 | Sneha Nair | cb.sc.p2aie23022@cb.students.amrita.edu | Descriptive Statistics, Smote algorithm, Linear, Logistic regression, Kmeans with KNN, Gaussian Naive Bayes, Decision Tree, AdaBoost Classifier, RNN-GRU |

**GitHub URL of the project page:**

<https://github.com/PratimaChinta/ML_mini_project->

**Kaggle URL of the dataset page:**

<https://www.kaggle.com/code/sachinsharma1123/deceptive-reviews-classification-acc-91/input>

**Colab links:**

[ml proj\_part\_1.ipynb - Colaboratory (google.com)](https://colab.research.google.com/drive/1TqDBRR2jgz30NY6TYfQAh2RV7al69tC2) - Preprocessing, Regression models without hyperparameters, Clustering models without hyperparameters- K Means with KNN and SVM, Smote.

[ml proj\_part 2.ipynb - Colaboratory (google.com)](https://colab.research.google.com/drive/1AKLqIhVWmCjEGJFzYNysTgVrfIehV7EV) -descriptive statistics, Regression models with hyperparameters, All Clustering models with hyperparameters and Fuzzy C-means with KNN with and without hyperparameters, Bayesian classifiers- Naive bayes, decision tree, Neural network- Kmeans with CNN (with and without hyperparameters)

[ml\_proj\_part\_3.ipynb - Colaboratory (google.com)](https://colab.research.google.com/drive/1CdJk6bX6-P0gy7bp-l2LPeuf6yqeMYqC#scrollTo=fo8t-CsH8R7J) -  LSTM (RNN), GRU, AdaBoost

[ml-proj-statistical tests.ipynb - Colaboratory (google.com)-](https://colab.research.google.com/drive/1KhGAbnWshnelRizvt6FcW3R004BLCWkP#scrollTo=NheVpG3sZUdH) Statistical Feature Analysis

**SECTION-1**

**1.1 Application Name:** Fake reviews detection using Machine Learning Algorithms

**1.2 Description:**

In the modern business and commerce world, online reviews play a major role. When making an online purchase decision, reviews are a major factor by the users themselves. As a result, shady people or organizations attempt to influence product reviews to further their own agendas. To identify fraudulent online reviews, it presents a few semi-supervised and supervised text mining models and evaluates the effectiveness of the two methods using a dataset of hotel reviews. Positive reviews of the target product can draw in more clients and boost sales; Negative reviews can drive away customers and reduce sales. These reviews are usually created with the intention of misleading potential customers to deceive, defile, or induce them to do business. It is false or fraudulent. The goal of our work is to determine whether the review is false or factual. In our work, we employ various machine learning algorithms i.e. regression models, clustering algorithms, bayesian algorithms, decision tree and neural networks.

**1.3 A set of analytical questions:**

**Why:**

* Why is fake review detection needed?
* Why is maintaining user trust essential for the success of online platforms?
* Why is it important to minimize false positives in fake review detection? Why are specific features chosen for fake review detection models?
* Why use machine learning algorithms and neural networks for fake review detection?

**What**:

* What are the potential consequences of relying on fake reviews for businesses, consumers or online platforms?
* What advantages do these technologies offer over traditional methods?
* What features or characteristics of reviews are indicative of fakeness?
* What kind of data is needed for training fake review detection models?
* What machine learning algorithms are suitable for fake review detection?
* What metrics should be used to evaluate the performance of fake review detection models?

**How:**

* How can effective fake review detection mitigate the impact on businesses and consumers?
* How can important features be extracted or engineered for use in machine learning models?
* How do the algorithms compare in terms of accuracy, efficiency, and scalability?
* How should textual data from reviews be pre-processed to make it suitable for machine learning models?
* How do you choose the most appropriate machine learning algorithm or neural network architecture for fake review detection?
* How do you deploy a fake review detection model in a real-world scenario?
* How can the model be monitored and updated over time to adapt to evolving patterns of fake reviews?

**1.4 A set of questions for prediction**

* Can we predict whether a given review is genuine or fake based on its textual content?
* How accurately can a machine learning model differentiate between real and fake reviews?
* Can the model provide a confidence score or probability for each prediction, indicating the likelihood of a review being fake?
* How can prediction confidence be used to prioritize reviews for manual inspection?
* How does the diversity of the training set contribute to the model's ability to generalize to unseen data?
* How will ensemble methods, such as combining multiple models, be considered for improving fake review prediction accuracy?
* How are the models trained on labelled data?
* How can a model be fine-tuned to improve its performance on specific types of fake reviews or adapt to changing patterns over time?
* Why are specific hyperparameters adjusted, and how do they impact the model's accuracy and generalization?

**1.5 Technologies Used:**

|  |  |
| --- | --- |
| **Language** | Python |
| **Environment** | Google collab |
| **Libraries** | Numpy , pandas,matplotlib, seaborn, sklearn, nltk |

**1.6 Why is this Application required?**

Fake reviews detection using machine learning algorithms is a crucial task in today's digital world, where online reviews significantly influence consumer decisions. The prevalence of fake reviews, intentionally written to mislead consumers about a product or service, has necessitated the development of advanced methods to identify and remove them. Machine learning algorithms offer a powerful solution to this problem by analysing various features of reviews to distinguish genuine ones from fabricated ones.

Significance of Fake Reviews Detection:

1. Protecting Consumer Interests: Fake reviews can lead consumers to make uninformed decisions, potentially purchasing inferior products or services. Accurate detection of fake reviews helps safeguard consumer interests by ensuring that online reviews reflect genuine experiences.
2. Enhancing Brand Reputation: Businesses rely on online reviews to establish credibility and attract customers. Fake reviews can damage a brand's reputation and erode consumer trust. By removing fake reviews, businesses can maintain a positive online presence and foster trust among their customers.
3. Promoting Fair Market Practices: Fake reviews can distort market competition, giving an unfair advantage to businesses that engage in this practice. Identifying and eliminating fake reviews helps maintain a level playing field and promotes fair competition.
4. Improving E-commerce Reliability: E-commerce platforms depend on authentic reviews to guide consumer choices. Accurate fake reviews detection enhances the reliability of e-commerce transactions, boosting consumer confidence in online shopping.

**1.6 List of similar applications**:

|  |  |
| --- | --- |
| **Application Name** | **URL** |
| Yelp's Fake Review Detection System | <https://blog.yelp.com/businesses/yelp-review-filter-explained/> |
| Google's Fake Review Detection System | <https://research.aimultiple.com/fake-review-detection/> |
| TripAdvisor's Fake Review Detection System | <https://www.revinate.com/blog/how-to-report-fraud-on-tripadvisor/> |
| TrustPilot's Fake Review Detection System | <https://www.trustpilot.com/blog/trends-in-trust/how-to-spot-a-fake-review> |

**1.7 What is unique in your project**

This fake review detection project using machine learning (ML) is unique in its approach to identifying deceptive or fraudulent reviews among the vast amount of user-generated content. Fake review detection projects heavily rely on natural language processing (NLP) techniques to analyze the textual content of reviews. ML models are trained to recognize patterns and linguistic cues associated with fake or deceptive language. Fake review detection often deals with imbalanced datasets where genuine reviews significantly outnumber fake ones. I have tried to implement ML models that can handle this class imbalance to avoid biases and ensure accurate predictions. Detecting fake reviews often involves addressing adversarial challenges, where individuals intentionally try to deceive the system. ML models must be robust enough to handle adversarial attacks and continuously improve to stay ahead. Analyzing the patterns of legitimate users versus potential fake reviewers contributes to more accurate predictions. This project experiments with all kinds of machine learning and neural networks along with hyperparameter tuning for each model to determine which algorithm gives better performance in correctly classifying reviews.

**SECTION 2**

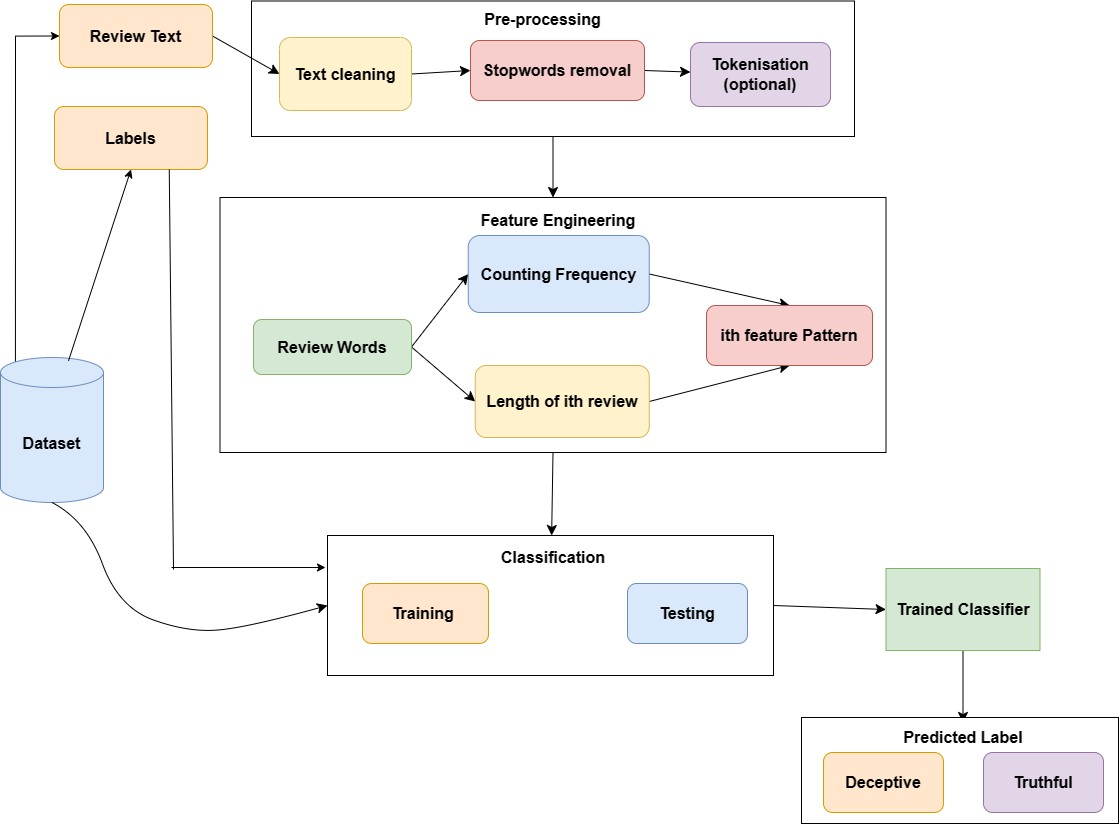
**2.1 Conference Papers related to our project**

|  |  |  |
| --- | --- | --- |
| **Paper Name** | **Conference Name/Citation** | **DOI/Authors** |
| Semi-Supervised Learning Based Fake Review Detection | H. Deng et al. 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Guangzhou, China, 2017, pp. 1278-1280, | Doi:10.1109/ISPA/IUCC.2017.00195.  Authors:  [Huaxun Deng](https://ieeexplore.ieee.org/author/37086234085); [Linfeng Zhao](https://ieeexplore.ieee.org/author/37086236727); [Ning Luo](https://ieeexplore.ieee.org/author/37086236171); [Yuan Liu](https://ieeexplore.ieee.org/author/37085569505); [Guibing Guo](https://ieeexplore.ieee.org/author/37894495700); [Xingwei Wang](https://ieeexplore.ieee.org/author/37293609500); and more |
| Review of Fake Product Review Detection Techniques | 2022, Second International Conference on Artificial Intelligence and Smart Energy (ICAIS), Coimbatore, India, 2022, pp. 771-776, | Doi:10.1109/ICAIS53314.2022.9742735.  Authors:  B. V. Santhosh Krishna, S. Sharma, K. Devika, Y. Sahana, K. N. Sharanya and C. Indraja |
| [Multi-task Ensemble Learning for Fake Reviews Detection and Helpfulness Prediction: A Novel Approach](https://aclanthology.org/2023.ranlp-1.78). | Melleng et al., RANLP 2023, In Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing, pages 721–729, Varna, Bulgaria. INCOMA Ltd., Shoumen, Bulgaria. | Authors:  Alimuddin Melleng, Anna Jurek-Loughrey, and Deepak P. |
| Research on K-Value Selection Method of K-Means Clustering Algorithm | *J* 2019, Graduate institute, Space Engineering University, Beijing 101400, China, | Doi: <https://doi.org/10.3390/j2020016>  Authors: Chunhui Yuan and Haitao Yang |
| KNN, k-Means and Fuzzy c-Means for 16-QAM Demodulation in Coherent Optical Systems | June 2019  Conference: 2019 IEEE Colombian Conference on Communications and Computing (COLCOM) | DOI:[10.1109/ColComCon.2019.8809116](http://dx.doi.org/10.1109/ColComCon.2019.8809116)  Authors: Alejandro Escobar P, [University of Antioquia | UdeA · Departamento de Ingeniería de Telecomunicaciones,](https://www.researchgate.net/institution/University_of_Antioquia?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InByb2ZpbGUiLCJwcmV2aW91c1BhZ2UiOiJwdWJsaWNhdGlvbiJ9fQ)  Master of Telecommunications Engineers |
| Reliable probability estimates based on support vector machines for large multiclass datasets**.** | In: Iliadis, L., Maglogiannis, I., Papadopoulos, H., Karatzas, K., Sioutas, S. (eds.) AIAI 2012, Part II. IFIP AICT, vol. 382, pp. 182–191. Springer, Heidelberg (2012) | Authors: Lambrou,  A., Papadopoulos,  H.,Nouretdinov,  I. Gammerman |
| Naive Bayesian Prediction of Japanese Annotated Corpus for Textual Semantic Word Formation Classification | Luoyang Normal University, Luoyang Henan 471934, China Correspondence should be addressed to Zhoushao Hao; haozs@lynu.edu.cn Received 13 January 2022; Revised 11 February 2022; Accepted 19 February 2022; Published 16 March 2022 | Authors: Zhoushao Hao |
| Decision-tree Induction from Time-series Data Based on a Standard-example Split Test. | Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA | Authors**:** Hideto Yokoi,  [KagawaUniversity, Department of Medical Informatics](https://www.researchgate.net/institution/Kagawa-University?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InByb2ZpbGUiLCJwcmV2aW91c1BhZ2UiOiJwdWJsaWNhdGlvbiJ9fQ) |
| Japanese Historical Character Recognition using Deep Convolutional Neural Network (DCNN) with DropBlock Regularization | July 2019  [International Journal of Recent Technology and Engineering (IJRTE)](https://www.researchgate.net/journal/International-Journal-of-Recent-Technology-and-Engineering-IJRTE-2277-3878?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19) 8(2):3510-3515 | DOI:[10.35940/ijrte.B2923.078219](http://dx.doi.org/10.35940/ijrte.B2923.078219)  Authors:Sujata Saini  [Tokyo Metropolitan University, TMU · Faculty of System Design and Graduate School of System Design](https://www.researchgate.net/institution/Tokyo-Metropolitan-University?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InByb2ZpbGUiLCJwcmV2aW91c1BhZ2UiOiJwdWJsaWNhdGlvbiJ9fQ)  Doctor of Philosophy |
| LSTM, GRU, Highway and a Bit of Attention:  An Empirical Overview for Language Modeling in Speech Recognition | Human Language Technology and Pattern Recognition, Computer Science Department, RWTH Aachen University, Aachen, Germany @cs.rwth-aachen.de | Authors:  Kazuki Irie, Zolt´an T¨  uske, Tamer Alkhouli, Ralf Schl¨uter, Hermann Ne |

**2.2 Journal Details**

|  |  |  |
| --- | --- | --- |
| **Paper name** | **Journal name /Citation** | **Authors** |
| Detecting fake online reviews using supervised and semisupervised learning | An UGC-CARE Approved Group-II Journal [www.ijearst.co.in](http://www.ijearst.co.in).  International journal of engineering in advanced research science and technology Volume.01, IssueNo.03, April-2022, Pages: 696-703 | Authors: Mr .B .B. K. Prasad , B. Naga Siva Likhitha , K.Ramesh |
| Fake online reviews detection using semi-supervised and supervised learning | International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538 Volume 10 Issue IV Apr 2022- Available at [www.ijraset.com](http://www.ijraset.com) | Authors: Ajanta Chettri, Amal George, Dr. A. Rengarajan, Feon Jaison, School of CS and IT, Jain University, Bangalore, Professor, School of CS and IT, Jain University, Bangalore |
| Detection of fake online reviews using semi-supervised and supervised learning | International Research Journal of Engineering and Technology (IRJET) Volume: 08 Issue: 04 | Apr 2021 [www.irjet.net](http://www.irjet.net) Impact Factor value: 7.529 | ISO 9001:2008 Certified Journal | Page 650 | Authors: N. Kumaran1, chapalamadugu haritha chowdary2, devarapalli sreekavya3 1n. Kumaran, Assistant professor, Dept. of Computer Science and Engineering, SCSVMV (Deemed to be University), Tamil nadu |
| Detection Of Fake Reviews Using Machine Learning Algorithm | International Journal of Future Generation Communication and Networking ,Vol. 13, No. 1, (2020), pp. 415-419, AISSMS IOIT,Pune | Girish .J.Navale    Dr. Sanjay Kurkute , Amey.R.Kamble Kshitija Shinde |
| Fake Reviews Detection using Supervised Machine Learning | (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 12, No. 1, 2021 | Ammar Mohammed3 Department of Computer Science Misr International University, Egypt Faculty of Graduate Studies of Statistical Research Cairo University, Egypt |
| Fake review identification and utility evaluation model using machine learning | Original research article  Front. Artif. Intell., 19 January 2023 Sec. Machine Learning and Artificial Intelligence Volume 5 - 2022 | <https://doi.org/10.3389/frai.2022.1064371> | Wonil Choi  Student / Intern  Sungkyunkwan University  Jongno-gu, Republic of Korea |
| Fake Reviews Detection through Machine learning Algorithms: A Systematic Literature Review | :September 9th, 2022 DOI: https://doi.org/10.21203/rs.3.rs-2039197/v1 License:   This work is licensed under a Creative Commons Attribution 4.0 International License. | Mohammed Ennaouri  ( ◊ mohammed\_ennaouri@um5.ac.ma ) École Nationale Supérieure d'Informatique et d'Analyse des Systèmes Ahmed Zellou  École Nationale Supérieure d'Informatique et d'Analyse des Systèmes |
| Fake Reviews Detection Using NLP Model and Neural Network Model | Paper ID **:** IJERTV12IS050063  Volume & Issue **:**[Volume 12, Issue 05 (May 2023)](https://www.ijert.org/volume-12-issue-05-may-2023)  Published (First Online**):**10-05-2023  ISSN (Online) **:** 2278-0181  Publisher Name **:** IJERT  License:This work is licensed under a [Creative Commons Attribution 4.0 International License](http://creativecommons.org/licenses/by/4.0/) | Authors : Abhijeet A Rathore , Gayatri L Bhadane , Ankita D Jadhav , Kishor H Dhale, Jayshree D Muley |
| Review of Machine Learning Algorithms for Fake Review Detection using Amazon Dataset | Research Scholar, Suresh Gyan Vihar University, Jaipur, Rajasthan, India 2Professor, Suresh Gyan Vihar University, Jaipur, Rajasthan, India 3Professor, Poornima Institute of Engineering and Technology, Jaipur, Rajasthan, India | Navin Kr.Goyal1 , Bright Keswani2 , DineshGoyal |
| Fake Reviews Detection Using Machine Learning | Information Technology Department, Fr. Conceicao Rodrigues College of Engineering, University of Mumbai, India 1 swijeld[at]gmail.com 2 ridhi01bauskar[at]gmail.com 3 apoorvashet11[at]gmail.com | Swijel Dmello1 , Ridhi Bauskar2 , Apoorva Shet |

**2.3 Model Diagram:**



*Figure 1. Model Diagram*

Below is a simplified methodology for fake review detection using a deceptive opinion dataset:

* **Data Collection:**

Collect or choose a dataset of reviews containing labels indicating whether each review is deceptive or truthful.

* **Data Preprocessing:**

Clean the text data: Remove stopwords, punctuation, and perform text normalization.

Split the dataset into training and testing sets.

* **Feature Extraction:**

Use techniques such as TF-IDF or Count Vectorization to convert text data into numerical features. Apply other techniques like one-hot encoding for additional feature representation.

* **Model Training:**

Train various machine learning models on the training set. Common models include:

Logistic Regression

Support Vector Machines (SVM)

Ensemble methods (Random Forest, Gradient Boosting)

Neural Networks

* **Clustering (Optional):**

Apply clustering algorithms, such as K-Means, to group reviews based on similarity. This step may help identify patterns or anomalies.

* **Model Evaluation and Prediction:**

Model prediction refers to the process of using a trained model to make predictions or estimates on new, unseen data. Once a machine learning model has been trained on a dataset, its primary purpose is to generalize its learning to make predictions on new instances or observations. Evaluate model performance on the testing set using metrics like accuracy, precision, recall, F1-score, and confusion matrix.

* **Fine-Tuning:**

Fine-tune hyperparameters based on model performance and, if necessary, revisit feature engineering.

**2.4 Dataset Description**

**Tabulation:** 1600 rows × 5 columns

A dataset which comprises 1600 reviews has been selected, of which:

* 800 show favorable sentiment polarity and,
* remaining 800 show negative sentiment polarity.
* 400 of the 800 reviews with favorable sentiment polarity are genuine and remaining 400 are fraudulent.
* 400 of the other 800 evaluations with negative emotion polarity are honest and remaining 400 are fake reviews.

Reviews with a tag of "1" indicate authenticity, whereas reviews with a tag of "0" indicate falsity for evaluation purposes. The Reviews from Kaggle are gathered. Reviews of it can be found online n TripAdvisor, MTurk, and other sites.

Each of the above datasets consist of 20 reviews for each of the 20 most popular Chicago hotels

Columns:

* deceptive(truthful or deceptive)
* hotel (name of hotel)
* polarity (positive or negative)
* source (website it is taken from)
* text (the review text)

**SECTION-3**

**3.1. Preprocessing**

**3.1.1. Statistical Feature Analysis**

A statistical test is a way to determine whether the random variable is following the null hypothesis or alternate hypothesis. It basically tells whether the sample and population or two/ more samples have significant differences. You can use various descriptive stats such as mean, median, mode, range, or standard deviation for this purpose. However, we generally use the mean. The statistical test gives you a number which is then compared with the p-value. If its value is more than the p-value you accept the null hypothesis, else you reject it.

The procedure for implementing each statistical test will be as follows:

* We calculate the statistic value using the mathematical formula
* We then calculate the critical value using statistic tables
* With the help of critical value, we calculate the p-value
* If p-value> 0.05 we accept the null hypothesis else we reject it

## Z-Test

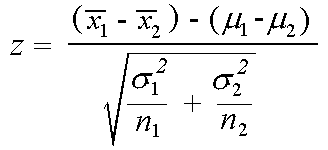
A Z-test is used to compare the mean of two given samples and infer whether they are from the same distribution or not. We do not implement Z-test when the sample size is less than 30.

A Z-Test may be a one-sample Z test or a two-sample Z test.

The One-Sample t-Test determines whether the sample mean is statistically different from a known or hypothesized population mean. The two-sample Z-test compares 2 independent variables.

We will implement a two-sample Z test.

Z statistic is denoted by



Google Image

A two-sample Z-test is typically used to compare the means of two independent samples, assuming the samples are normally distributed and the variances are known. However, the 'source', ‘hotel' and ‘deceptive' columns seem to be categorical, not continuous. As a result, a Z-test for comparing means might not be suitable for the dataset chosen.

In understanding the relationship or dependence between the 'source' and 'deceptive' columns, we can consider using a chi-squared test for independence. The chi-squared test can determine whether there is a statistically significant association between two categorical variables.

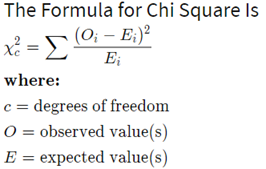
## Chi-Square Test

This test is applied when you have two categorical variables from a population. It is used to determine whether there is a significant association or relationship between the two variables.

There are 2 types of chi-square tests: chi-square goodness of fit and chi-square test for independence, we will implement the latter one.

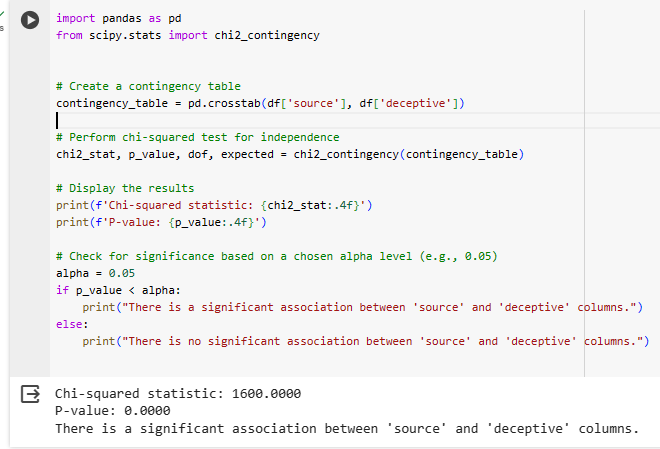
The degree of freedom in the chi-square test is calculated by (n-1)\*(m-1) where n and m are numbers of rows and columns respectively.

It is denoted by:



Google Image

**Output for chi-squared test on the 'source' and 'deceptive' column:**



*Figure- Statistical test- Chi-Squared*

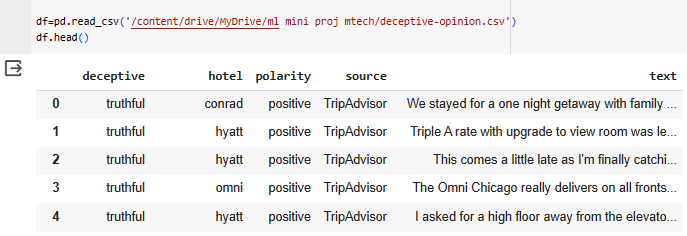
Here, Null hypothesis: "There is a significant association between 'source' and 'deceptive' columns."

Alternate hypothesis: “There is no significant association between 'source' and 'deceptive' columns."

Thus, Null hypothesis is rejected.

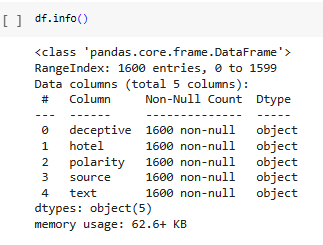
**3.1.2. Descriptive Statistics**

**df.head() –** It prints the first 5 rows of all the features.



*Figure 2. Df.head()*

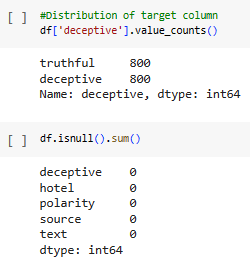
**df.info() –** It gives the number of values of each feature, displays if it has null values and also gives the type of Dtype.



*Figure 3. Df.info()*

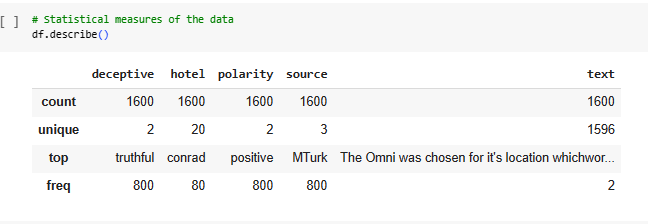
**df[‘target’].value\_counts() –** It prints out the number of values corresponding to one category and same for the other categories.

**df.isnull().sum() –** It check if the sum of all the null values for each feature in the dataset.



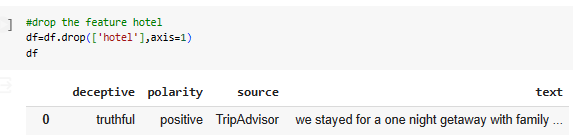
*Figure 4. Df.isnull().sum() and value\_counts*

**df.describe() –** It gives the different statistical measures of the data.



*Figure 5. Df.describe()*

**Df.drop([‘hotel’]) –** This deletes the column hotel as it is not selected as a feature in my project.



*Figure 6. Df.drop(‘hotel’)*

**3.1.3. Text cleaning using nltk**

Cleaning the Text Data: Text data often contains a lot of noise and irrelevant information that can hinder the performance of natural language processing (NLP) tasks.

Removing Stop words: Stop words are common words that do not add much meaning to a text. They are often removed from text data because they can increase the dimensionality of the data and make it more difficult to train NLP models.

Removing Punctuation: Punctuation can also be considered noise in text data. It can be removed from the text using regular expressions or other string manipulation techniques.

Text Normalization: Text normalization involves converting all text to lowercase or uppercase, removing extra whitespace, and converting numbers to their word forms. This can help to improve the consistency of the data and make it easier to match words together.

Explanation: text.lower(): Converts the text to lowercase.

re.sub('\[.\*?\]', '', text): Removes text in square brackets using a regular expression.

re.sub("\\W"," ",text): Removes special characters by substituting them with a space.

re.sub('https?://\S+|www\.\S+', '', text): Removes links using a regular expression.

re.sub('<.\*?>+', '', text): Removes HTML tags using a regular expression.

re.sub('[%s]' % re.escape(string.punctuation), '', text): Removes punctuation using a regular expression.

re.sub('\n', '', text): Removes newline characters.

re.sub('\w\*\d\w\*', '', text): Removes words containing numbers using a regular expression.

The apply method is then used to apply the text\_cleaning function to each element in the 'text' column of the DataFrame (df). The cleaned text is then stored back in the 'text' column. The purpose of these cleaning operations is often to prepare the text data for further analysis or natural language processing tasks.

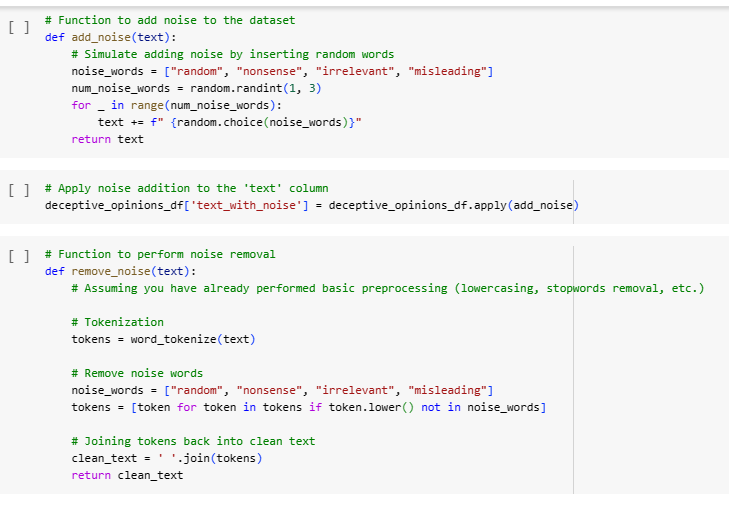


*Figure 7. data pre-preprocessing using nltk*

**3.1.4. Noise Removal**

I am adding new noise words in the data and removing it to show the functionality of noise removal. Noise can produce inconsistent results if uncleaned data is fed to machine learning models. Therefore, it is crucial to remove noise from the text data before feeding it to the model to reduce unnecessary computation and provide clean data.

* Functionality of noise removal: Tokenizes the input text into a list of words. Removes any words from the noise\_words list and joins the remaining tokens back into a clean text.
* Explanation: word\_tokenize(text) tokenizes the input text into a list of words. List comprehension is used to filter out words that match the noise words. The filtered tokens are then joined back into a clean text.



*Figure 8. noise removal*

**3.1.5. Feature Engineering**

Feature engineering is a critical aspect of building effective models for fake review detection. It involves transforming raw data into a format that is more suitable for machine learning algorithms, allowing the model to capture relevant patterns and information. Here is one feature engineering techniques commonly used in fake review detection:

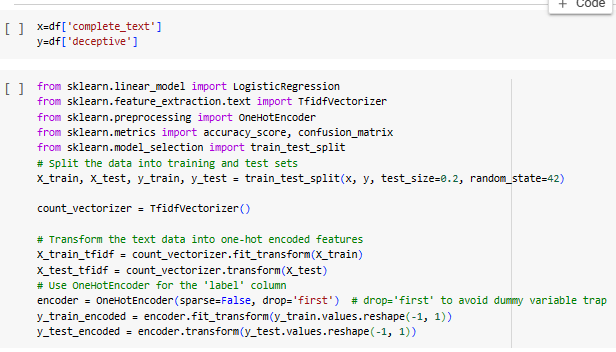
Text Features:

**TF-IDF** (Term Frequency-Inverse Document Frequency): Convert text data into numerical vectors that reflect the importance of words in a document relative to the entire dataset.

**One hot encoding:**

One-hot encoding is a method for representing categorical variables as binary vectors. Each category is represented by a binary vector, with a 1 in the position corresponding to the category and 0 in all other positions. OneHotEncoder is used to convert categorical labels (y\_train and y\_test) into one-hot encoded vectors.

* fit\_transform is used on the training labels to fit the encoder and transform the training labels.
* transform is used on the test labels to transform them using the categories learned from the training set.



*Figure 9. Vectorising input text and Encoding labels*

**3.1.6. SMOTE Algorithm**

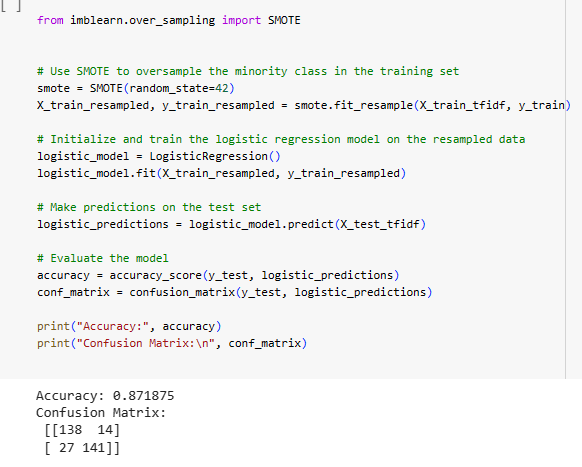
SMOTE (Synthetic Minority Over-sampling Technique) is a technique used to address class imbalance in classification problems by oversampling the minority class. By generating synthetic samples, SMOTE effectively increases the number of instances in the minority class, making it closer in size to the majority class. This helps prevent the machine learning model from being biased toward the majority class and improves its ability to generalize well to unseen data.

How SMOTE Works:

* Identify Minority Class: SMOTE focuses on the minority class, which is the class with fewer instances.
* Select a Minority Instance: Randomly pick an instance from the minority class.
* Find k Nearest Neighbors: Identify the k-nearest neighbors of the selected instance from the same class.
* Generate Synthetic Samples: Create synthetic instances along the line segments connecting the selected instance and its neighbors.
* The number of synthetic instances generated is determined by a user-defined ratio.
* Repeat: Repeat the process until the desired balance between classes is achieved.

Example using imbalanced-learn:

Use SMOTE class from the imbalanced-learn library. The fit\_resample method is applied to the training data (X\_train and y\_train), creating synthetic samples for the minority class. The resulting X\_train\_resampled and y\_train\_resampled can be used for training a model that better handles class imbalance. Here I have used logistic regression.



*Figure 10. Smote algorithm*

**SECTION- 4**

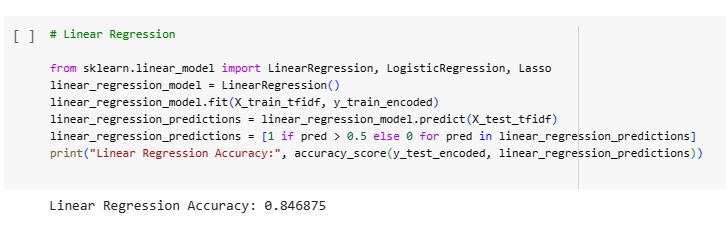
**4.1 Regression**

Regression is a statistical technique in machine learning that models the relationship between a dependent variable (or outcome) and one or more independent variables (or predictors). The goal of regression is to understand and quantify the relationship, enabling predictions of the dependent variable based on new or unseen data. It is commonly used for tasks where the target variable is continuous and involves finding the best-fitting line or curve to represent the data.

**4.1.1. Linear regression**

Linear regression is a core supervised machine learning method for predicting continuous outcomes based on input features. It assumes a linear link between independent and dependent variables. In simple regression, there's one input, while in multiple regression, there are many. The model seeks the optimal line minimizing the sum of squared differences between predictions and actual values. The equation, y = mx + b, defines the relationship, where y is the outcome, x is the input, m is the slope, and b is the y-intercept. Coefficients m and b are determined in training, often using the least squares approach. Applied broadly in fields like economics and finance, linear regression aids in modelling and understanding variable relationships. Evaluation metrics like Mean Squared Error or R-squared gauge its predictive performance.

**Linear Regression without Hyperparameters**

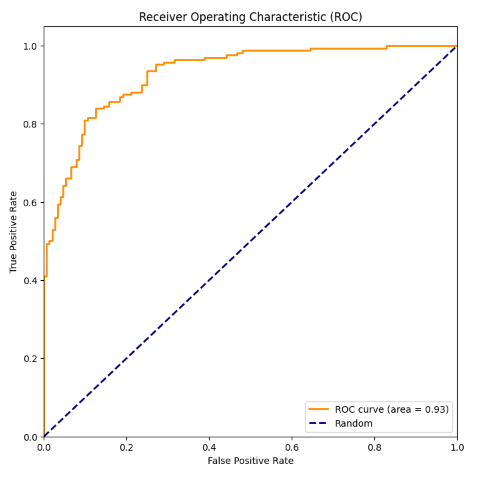


*Figure 11. Linear regression –default hyperparameters*

Parameters for Linear regression without Hyperparameter

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| Test size | To split the dataset into training and testing in a ratio | 0.2 |
| Random State | This parameter sets the seed for the random number generator used by the data splitter. By using a fixed seed, the random splitting process becomes deterministic, allowing for result reproducibility. | 42 |

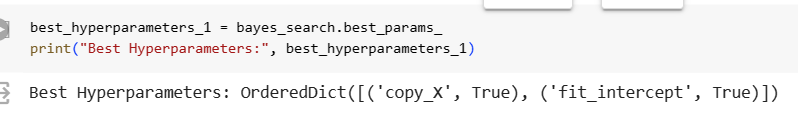
**Output:**

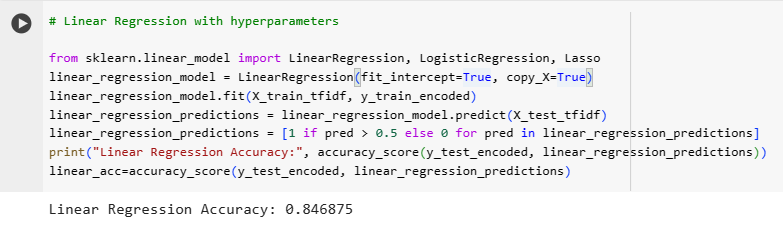


*Figure 12. Linear regression –default hyperparameters- output*

**Linear Regression with Hyperparameter**

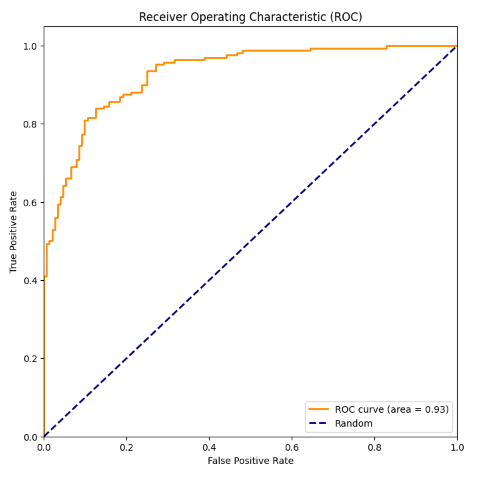
**Hyperparameter Tuning approach used: GridSearch**





*Figure 13. 2 screenshots above- Linear regression –different hyperparameters*

**Output:**



*Figure 14. Linear regression –different hyperparameters-output*

Parameters for Linear regression with Hyperparameters

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| Copy\_x | If True, the input features (X) will be copied. If False, the model may overwrite the input data | True |
| fit\_intercept | Determines whether to calculate the intercept for the model. If set to False, the model will be forced to pass through the origin (0,0). | True |
| n\_jobs | Specifies the number of jobs to use for the computation. If set to -1, all available CPU cores will be used. | None |
| positive | If True, forces the coefficients to be positive. This constraint is applied during the fit. | False |

**Inference:**

* **Initializing Linear Regression Model:** A linear regression model is initialized with specific hyperparameters, such as including an intercept term in the regression equation. A linear regression model is initialized with specified hyperparameters. **fit\_intercept=True** means that the model will include an intercept term in the regression equation, and **copy\_X=True** indicates that a copy of the input data will be made.
* **Training the Model:** The linear regression model is trained on the training data.
* **Making Predictions:** Predictions are made on the test data.
* **Converting Predictions to Binary:** Continuous predictions are converted into binary form using a threshold of 0.5. If the predicted value is greater than 0.5, it's assigned a class label of 1; otherwise, it's assigned a class label of 0.
* **Evaluating the Model:** The accuracy of the linear regression model is calculated. The accuracy score indicates how well the model's predictions align with the true labels in the test set.

The two models of Linear Regression I have used don’t have any change as the best parameters defined by Grid Search is same as the default values.

**4.1.2 Logistic Regression**

This model uses the training data to fit the Logistic Regression model. During training, the model learns the optimal weights (coefficients) for each feature to make predictions.

Logistic Regression is a classification algorithm commonly used for binary classification tasks. It models the probability that a given instance belongs to a particular class.

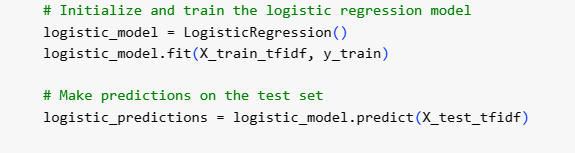
It's worth noting that Linear Regression is not the typical choice for classification problems, and Logistic Regression is more commonly used. Logistic Regression is designed for binary classification and provides predicted probabilities that can be directly thresholded for classification decisions.

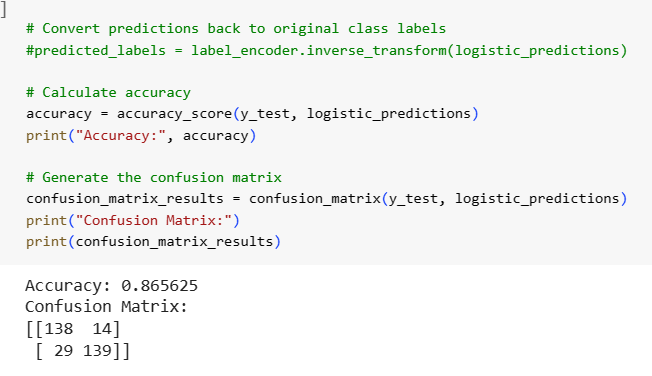
**Logistic regression without Hyperparameter**

Logistic regression default hyperparameters

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| c | Inverse of regularization strength. Smaller values of C specify stronger regularization. | 1.0 |
| penalty | The regularization term used in the logistic regression model. 'l2' refers to Ridge regularization. | 12 |
| solver | Algorithm to use in the optimization problem. 'lbfgs' is one of the solvers available. | lbfgs |
| **max\_iter** | Maximum number of iterations taken for the solver to converge | 100 |
| **multi\_class** | Determines the strategy for handling multiple classes. 'auto' automatically selects 'ovr' (one-vs-rest) or 'multinomial' based on the type of problem. | auto |
| **Fit\_intercept** | Specifies whether to calculate the intercept for the model. | True |

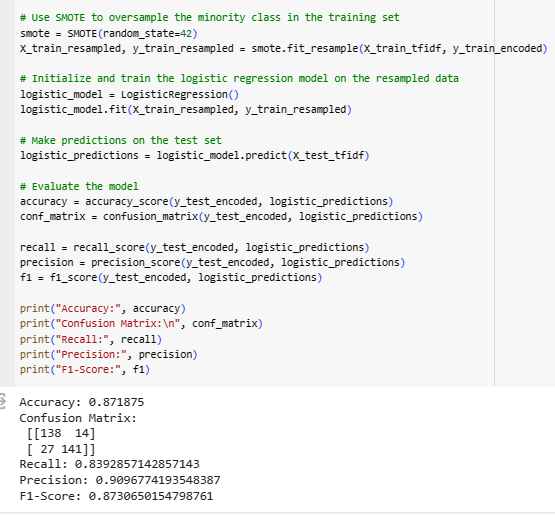
**Output: (Without SMOTE)**





*Figure 15. Logistic regression –default parameters*

With SMOTE and default parameters:



*Figure 16. Logistic regression –default parameters and Smote*

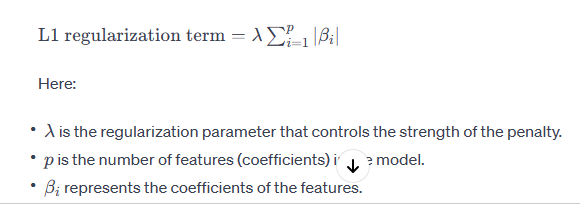
**Logistic Regression with Hyperparameters**

(With SMOTE)

**Hyperparameter Tuning approach used:** GridSearch (L2 penalty gives better results)

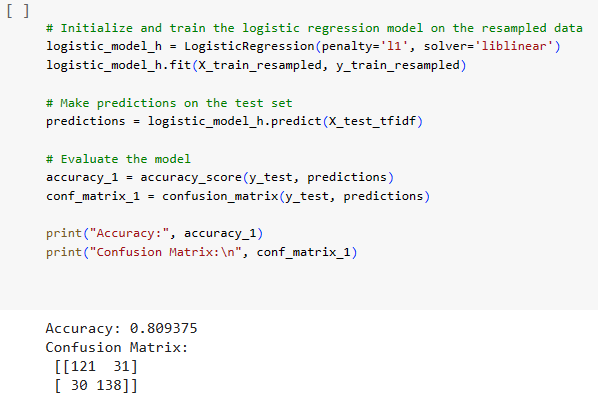
Parameters for Logistic Regression with Hyperparameter

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| penalty | The regularization term used in the logistic regression model. 'l1' refers to Lasso regularization. | l1 |
| solver | Algorithm to use in the optimization problem | Liblinear |



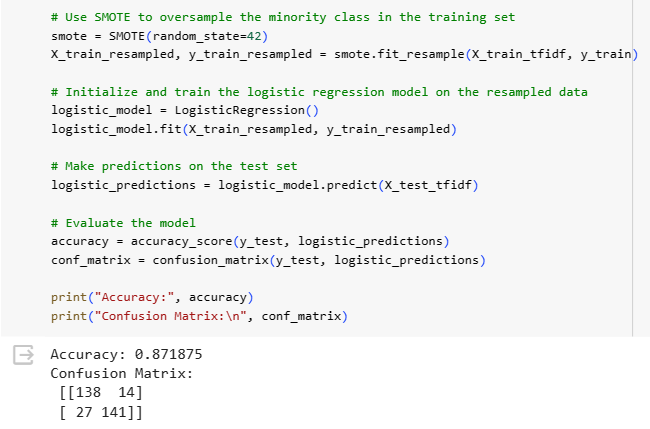
*Figure 17. L1 regularisation for penalty parameter*

Remaining parameters are default.



*Figure 18. Logistic regression with hyperparameters*

**Output:**



*Figure 19. Logistic regression with hyperparameters and smote-output*

**Inference:** Here I have experimented with both scenarios (with and without SMOTE) and evaluated the Logistic Regression model using appropriate metrics to determine which approach yields better results and the L2 default penalty with SMOTE yields higher accuracy as show above.

**4.1.3 Lasso Regression**

This creates an instance of the Lasso Regression model with a regularization parameter (alpha) set to 0.1. The alpha parameter controls the strength of regularization.

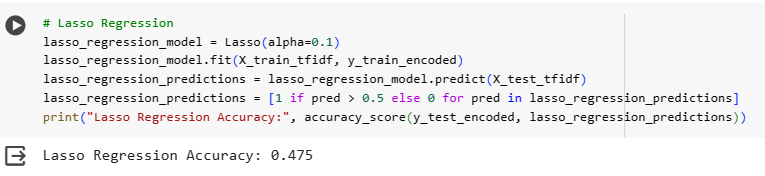
Lasso Regression is a linear regression technique with L1 regularization, where the regularization term penalizes the absolute values of the coefficients. It can be used for feature selection by driving some coefficients to exactly zero. In this example, it's applied to a binary classification task, and the accuracy is evaluated as a performance metric. Adjusting the alpha parameter can control the strength of regularization, and you may experiment with different values based on your specific dataset and requirements.

**Lasso Regression without Hyperparameter**

Default parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **alpha** | The regularization strength parameter. It controls the degree of regularization applied to the model. Higher values of alpha result in stronger regularization, potentially leading to more coefficients being exactly zero | 1.0 |
| **fit\_intercept** | Indicates whether to fit an intercept term in the model. If set to True, the model will have an intercept term; if set to False, the model will not have an intercept term. | True |
| **normalize** | This is a boolean parameter that indicates whether to normalize the input features before fitting the model. If set to True, the input features will be normalized, which can be useful when the features have different scales. | False |
| **precompute** | Whether to use a precomputed Gram matrix to speed up calculations. If set to 'auto', the Gram matrix is precomputed only if the number of samples is smaller than a certain threshold. | False |
| **positive** | Constrains the coefficients to be non-negative if set to True. This can be useful in scenarios where you have prior knowledge that the relationships between features and the target variable should be positive | False |
| **selection** | This parameter determines the algorithm used to update the coefficients in each iteration. The default value 'cyclic' updates the coefficients cyclically, while 'random' selects a random feature to update in each iteration. | Cyclic |

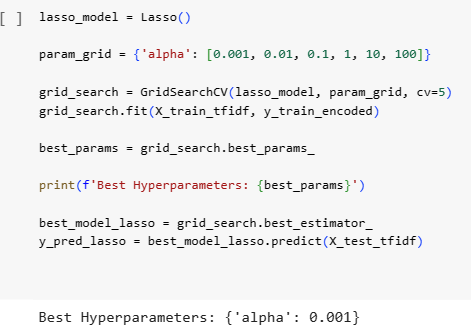
**Output:**



*Figure 20. Lasso regression-with alpha=0.1*

**Lasso Regression with Hyperparameter**

**Hyperparameter Tuning approach used: GridSearch**



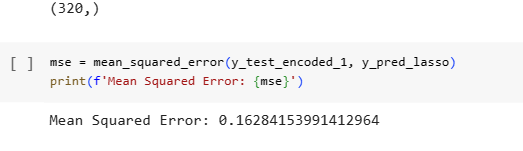
*Figure 21. Lasso regression-with alpha=0.001*

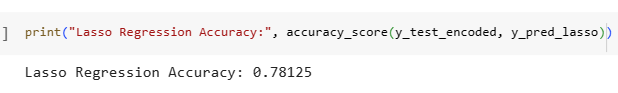
**Parameters for Lasso Regression with hyperparameter**

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| alpha | The regularization strength parameter. | 0.001 |

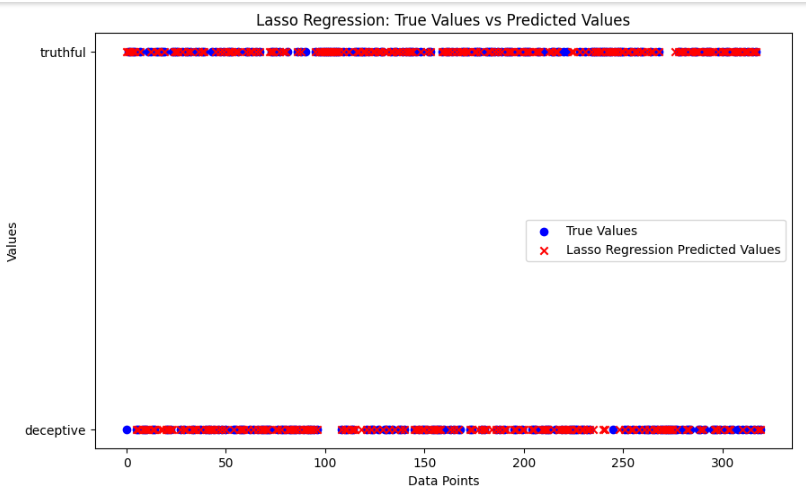
Remaining parameters are default

**Output:**





*Figure 22. Lasso regression-with alpha-0.001 output*



*Figure 23. Lasso regression-with alpha-0.001 predicted values*

**Inference:**

As shown above the Lasso regression, after using Hyperparameter Tuning, the alpha value chosen affects the accuracy of the model. The accuracy score measures how well the model's predictions align with the true labels in the test set.

Lasso Regression introduces a penalty term based on the absolute values of the coefficients, encouraging sparsity in the model. This can result in some coefficients being exactly zero, effectively performing feature selection and potentially improving the model's interpretability.

* Low Alpha (α): When alpha is set to a very small value (close to zero), the regularization term becomes negligible. In this case, the Lasso Regression model behaves similarly to standard linear regression. The model is less constrained, and it may overfit the training data, potentially capturing noise in the dataset.
* Intermediate Alpha: As alpha increases, the regularization term becomes more pronounced. The model is forced to simplify by shrinking some coefficients towards zero. Features that contribute less to the predictive power of the model may have their coefficients reduced to exactly zero. Intermediate values of alpha balance the trade-off between fitting the training data well and avoiding overfitting.
* High Alpha: For very high values of alpha, the regularization term dominates the loss function. This results in more aggressive shrinkage of coefficients. Many coefficients may be driven to exactly zero, effectively performing feature selection. The model becomes simpler and more interpretable.

**4.1.4 Ridge Regression**

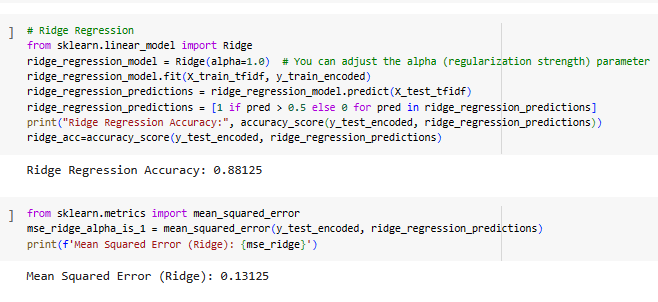
Ridge Regression is a technique used in linear regression to handle multicollinearity, where predictor variables are highly correlated. It adds a penalty term to the traditional linear regression cost function, discouraging large coefficients. This regularization helps stabilize the model and improve its performance by balancing the trade-off between bias and variance. Ridge Regression does not eliminate variables but rather shrinks their coefficients. It is particularly useful when dealing with datasets with correlated features.

**Ridge regression without Hyperparameter**

Default parameters:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **alpha** | The regularization strength parameter. It controls the amount of regularization applied to the model. Higher values of alpha result in stronger regularization. Similar to Lasso, the purpose is to prevent overfitting by penalizing large coefficients. | 1.0 |
| **fit\_intercept** | Indicates whether to fit an intercept term in the model. If set to True, the model will have an intercept term; if set to False, the model will not have an intercept term. | True |
| **normalize** | This is a boolean parameter that indicates whether to normalize the input features before fitting the model. If set to True, the input features will be normalized, which can be useful when the features have different scales. | False |
| **max\_iter** | Maximum number of iterations for the solver to converge. If set to None, the solver will iterate until convergence is reached. | None |
| **copy\_X** | If True, the input features X will be copied; if False, the input features may be overwritten. It is generally safe to leave this parameter at its default value. | True |
| **solver** | This parameter determines which solver to use for the optimization problem. The default value 'auto' automatically chooses the most appropriate solver based on the input data. | Auto |
| **tol** | Tolerance to declare convergence. The solver will stop iterating when the change in the coefficients is smaller than this tolerance. | 1e-3 |

**Output:**



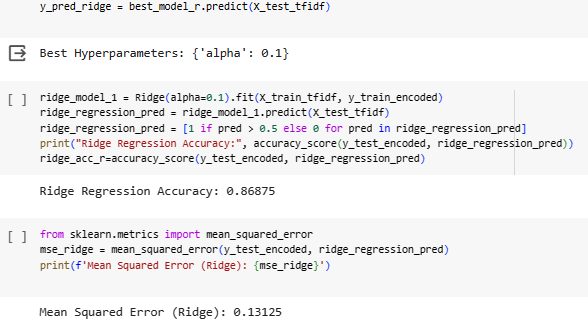
*Figure 24. Ridge regression-with alpha-1.0*

**Ridge Regression with Hyperparameter**

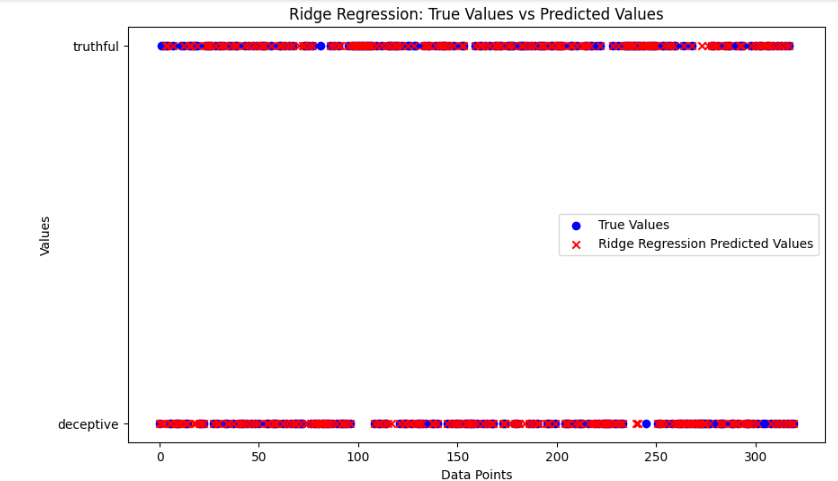
**Hyperparameter Tuning approach used: Grid Search**

Parameters changed for Ridge Regression

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| alpha | The regularization strength parameter. | 0.1 |

  
  
*Figure 25. Ridge regression-with alpha-0.1-with hyperparameters*

**Output:**

  
  
*Figure 26. Ridge regression-with alpha-0.1-output scatter plot*

**Inference:**

The regularization strength (α) is set to 1.0 in this example. Adjusting it, allows you to control the amount of regularization applied. Higher alpha values lead to stronger regularization. Predictions have a threshold of 0.5 to convert them into binary form. This is a common practice for binary classification tasks, where values above the threshold are assigned to one class, and values below the threshold are assigned to the other class. From above experiment of using hyperparameter tuning and without it, better accuracy was achieved when using alpha value as 1.0 (default).

**Comparison of All Regression Models:**

**Evaluation Parameters for best models of each type of regression**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **MSE** | **Accuracy** | **Precision** | **F1** | **Recall** |
| **Linear** | 0.153125 | 0.846875 | 0.87898089 | 0.8492307 | 0.821428 |
| **Logistic** | - | 0.871875 | 0.909677419 | 0.873065 | 0.8392857 |
| **Ridge** | 0.11875 | 0.88125 | 0.91666 | 0.882716 | 0.851190 |
| **Lasso** | 0.21875 | 0.78125 | 0.7916666 | 0.7916666 | 0.7916666 |

As shown above, ridge regression performs better than other regression models with accuracy 88%

**SECTION-5**

**5.1 Clustering models and Classifiers**

**Clustering** models are unsupervised learning algorithms that group data points into clusters based on their similarities. They achieve this by identifying patterns and relationships within the data without any prior knowledge of what those patterns might be. There are many different types of clustering models, each with its own strengths and weaknesses, most famous one is:

**K-means clustering:** This is a simple and popular algorithm that partitions data points into a fixed number of clusters, defined by the user. It works by iteratively assigning each data point to the nearest cluster centroid (the average of all points in the cluster) and then recomputing the centroids based on the newly assigned points. It is a particularly popular choice for its simplicity and efficiency. It is easy to implement and works well for data that is well-separated into spherical clusters. However, it can be sensitive to outliers and may not work well for data that is not well-separated or has clusters of different shapes.

**Classifiers:** In the world of machine learning, a classifier model is a powerful tool used to categorize data points into predefined groups or classes. Think of it as a sophisticated sorting machine that, instead of separating laundry by color, can analyze data based on various features and assign it to specific categories.

**Classifier model: KNN (K-Nearest Neighbors)**:

Concept: This algorithm classifies data points based on the majority vote of their k nearest neighbors. It assumes that similar data points are likely to belong to the same class.

Steps:

* Input: Data points with features and their corresponding class labels.
* Choose a value for k: This determines the number of nearest neighbors considered for classification.
* For each new data point:
  + Calculate the distance to all other data points.
  + Find the k nearest neighbors.
  + Predict the class of the new data point based on the majority class of its k neighbors.

**Classifier Model: SVM (Support Vector Machine)**: This algorithm aims to find the hyperplane that maximizes the margin between different classes. The data points closest to the hyperplane are called support vectors, and they play a crucial role in defining the decision boundary.

Steps:

* Input: Data points with features and their corresponding class labels.
* Choose a kernel function: This determines how the data points are transformed into a higher-dimensional space where the hyperplane can be more easily found.
* Find the optimal hyperplane: This involves solving an optimization problem that maximizes the margin between the classes.
* Predict the class of new data points: This is done by comparing their distance to the hyperplane and assigning them to the class on the other side.

**5.1.1 K-Means with KNN (Check code for implementation**)

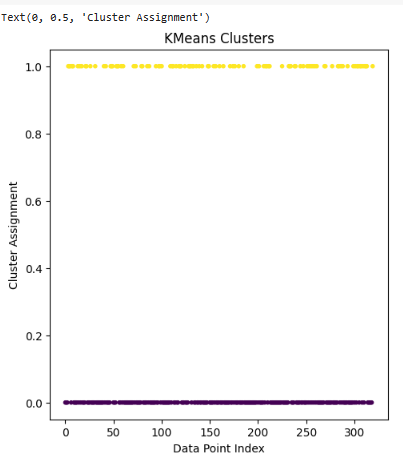
* K-Means Clustering:
  + Apply K-Means clustering to your dataset to partition it into K clusters.
  + Assign each data point to the cluster it belongs to.
* Feature Engineering:
  + Represent each data point by its cluster assignment. This can be done by creating binary features for each cluster, indicating whether the data point belongs to that cluster or not.
* KNN Classification on Clustered Data:
  + Use the cluster assignments as features and the original class labels as the target variable.
  + Split the data into training and testing sets.
  + Train a KNN classifier on the training data, using the cluster assignments as features.
  + Evaluate the classifier on the test data.

**Kmeans with KNN without hyperparameter tuning:**

Default parameters of K – Means

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **n\_clusters** | Specifies the number of clusters (K) the algorithm should create. Set k based on the understanding of the underlying data and the number of clusters you want to identify. | 8 |
| **init** | This parameter determines the method for initializing the initial cluster centroids. 'k-means++' is the default, which selects initial centroids that are spaced out and often leads to faster convergence. Other options include 'random', which randomly selects initial centroids, and ndarray, allowing you to pass your own array of centroids. | k-means++ |
| **n\_init** | The number of times the algorithm will run with different centroid seeds. The final result will be the best output in terms of inertia (sum of squared distances to the nearest centroid) across multiple runs. | 10 |
| **max\_iter** | Maximum number of iterations for each single run of the algorithm. If convergence is not reached within this number of iterations, the algorithm stops. | 300 |
| **tol** | Tolerance to declare convergence. If the change in the centroid positions is smaller than this value, the algorithm is considered to have converged. | 1e-4 |
| **random\_state** | Seed for random number generation. If you want reproducibility of results, you can set this parameter to a specific integer value. | None |
| **algorithm** | Determines the algorithm used. 'auto' chooses 'full' for dense data and 'elkan' for sparse data. 'full' is the classical K-Means algorithm, while 'elkan' can be more efficient for sparse data. | Auto |

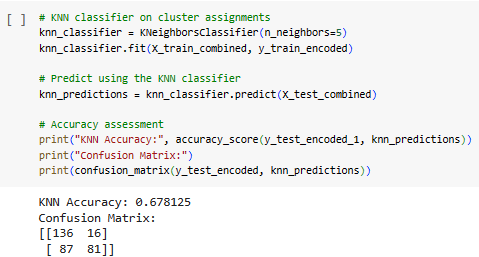
**Output: 2 clusters used**



*Figure 27. Kmeans Clustering plot*

Default parameters of KNN

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **n\_neighbors** | Specifies the number of neighbors to consider when making predictions. You need to set this parameter based on your understanding of the underlying data and the characteristics of the problem. | 5 |
| **weights** | Determines the weight assigned to each neighbor. 'uniform' gives equal weight to all neighbors, while 'distance' gives more weight to closer neighbors. | uniform |
| **algorithm** | Specifies the algorithm used to compute the nearest neighbors. 'auto' chooses the most appropriate algorithm based on the input data, but you can also explicitly set it to 'ball\_tree', 'kd\_tree', or 'brute'. | Auto |
| **leaf\_size** | Leaf size passed to the BallTree or KDTree algorithms. It can affect the speed and memory usage of the construction and query, with larger values generally leading to faster but less memory-efficient trees. | 30 |
| **p** | The power parameter for the Minkowski distance metric. When **p=1**, it corresponds to the Manhattan distance, and when **p=2**, it corresponds to the Euclidean distance. | 2 for Euclidean distance |
| **metric** | The distance metric used for the tree. The default 'minkowski' is a generalization of both the Euclidean and Manhattan distances. | minkowski |



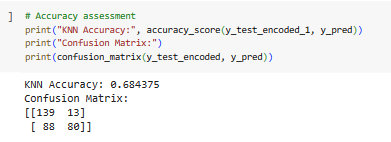
*Figure 29. Kmeans with KNN Output*

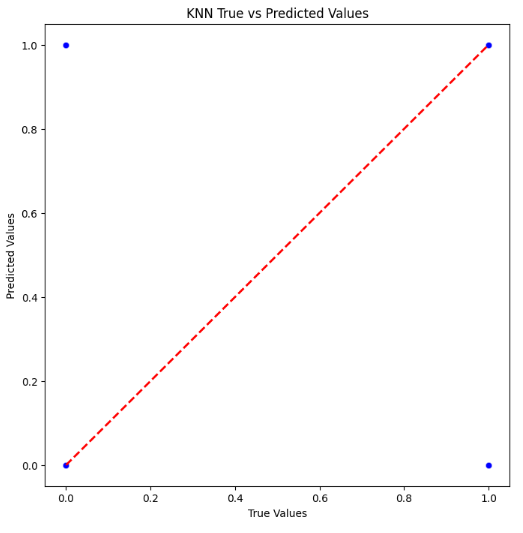
Here n\_neighbours=5 (default)

**Hyperparameter Tuning approach used: GridSearch**

**Result:** best\_neighbours= 9, remaining parameters are default

**Output:**





*Figure 29. Kmeans with KNN Output- with hyperparameters*

**Inference:** The number of clusters (k) in both k-means and KNN (n\_neighbors) needs to be chosen based on the characteristics of your data i.e. vectorised text data and the nature of your classification task i.e. classifying into deceptive and genuine review. The above process integrates both clustering and classification steps, allowing us to leverage the clustering information for improved classification.

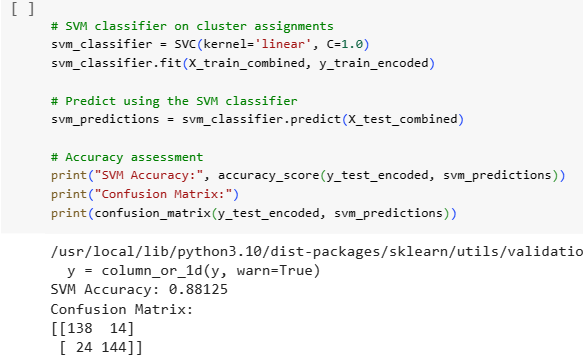
**5.1.2. K-Means with SVM (Check code for implementation)**

Default parameters of SVM:

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **C** | Regularization parameter. It influences the trade-off between achieving a low training error and a low testing error. Higher values of C allow the model to fit the training data more closely but might lead to overfitting | 1.0 |
| **kernel** | Specifies the kernel type to be used in the algorithm. Common choices include 'linear', 'poly' (polynomial), 'rbf' (radial basis function), and 'sigmoid'. The choice of the kernel affects the decision boundary shape. | **'**rbf' - Radial basis function |
| **degree** | Degree of the polynomial kernel function ('poly'). Ignored by all other kernels. It is the degree of the polynomial used by the 'poly' kernel. | 3 |
| **gamma** | Kernel coefficient. It influences the shape of the decision boundary. A low value of gamma leads to a more flexible decision boundary, while a high value makes the decision boundary less flexible. | Scale |
| **coef0** | Independent term in the kernel function. It is significant in 'poly' and 'sigmoid' kernels and can be tuned. | 0.0 |
| **shrinking** | Whether to use the shrinking heuristic. If set to True, the algorithm will use the shrinking strategy to speed up the optimization. | True |
| **probability** | Whether to enable probability estimates. If True, the model will estimate class probabilities and enable the **predict\_proba** method. | False |

**Kmeans with SVM without hyperparameter tuning: Using the clusters from kmeans (used in previous model)**

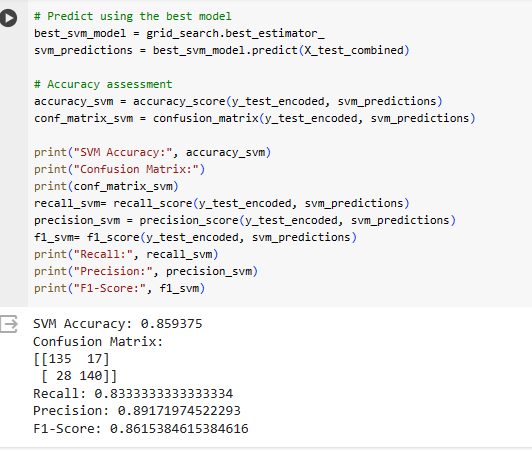
**Output:**



*Figure 30. Kmeans with SVM Output*

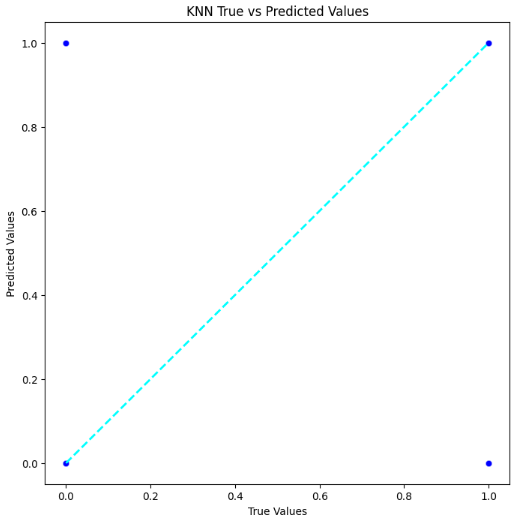
**Hyperparameter Tuning approach used: GridSearch**

**Kmeans with SVM with hyperparameter tuning:**



*Figure 31. Kmeans with SVM - with hyperparameters*

**Output:**



*Figure 32. Kmeans with SVM Output- with hyperparameters*

**Inference:** The Kmeans with SVM model with c=1.0 is giving better accuracy than c=10.  
When combining K-Means clustering with a Support Vector Machine (SVM) for classification, the general approach involves using the cluster assignments obtained from K-Means as additional features for training an SVM model. Here are the steps:

K-Means Clustering:

* Apply K-Means clustering to your dataset to partition it into K clusters.
* Assign each data point to the cluster it belongs to.

Feature Engineering:

* Represent each data point by its cluster assignment. Create binary features for each cluster, indicating whether the data point belongs to that cluster or not.

SVM Classification on Clustered Data:

* Use the original features along with the cluster assignments as features, and the class labels as the target variable.
* Split the data into training and testing sets.
* Train an SVM classifier on the training data.
* Evaluate the classifier on the test data.

**5.1.3. Fuzzy C-Means with KNN**

Combining Fuzzy C-Means (FCM) with k-Nearest Neighbors (KNN) involves using FCM to assign fuzzy memberships to clusters for each data point. These membership values are then employed as weights in KNN, allowing for a more nuanced and accurate prediction by considering the soft relationships between data points and clusters. This approach is beneficial when data points may belong to multiple clusters simultaneously.

Integrating Fuzzy C-Means (FCM) with k-Nearest Neighbors (KNN) involves first applying FCM to assign fuzzy memberships to clusters for each data point. These memberships, indicating the strength of association with each cluster, are then utilized as weights in the KNN algorithm. By incorporating FCM's soft clustering results, this approach captures nuanced relationships in the data, offering a more flexible and accurate prediction. It is particularly useful when data points exhibit partial membership in multiple clusters, providing a more sophisticated way to handle complex relationships in the dataset.

Default parameters of Fuzzy C- Means:

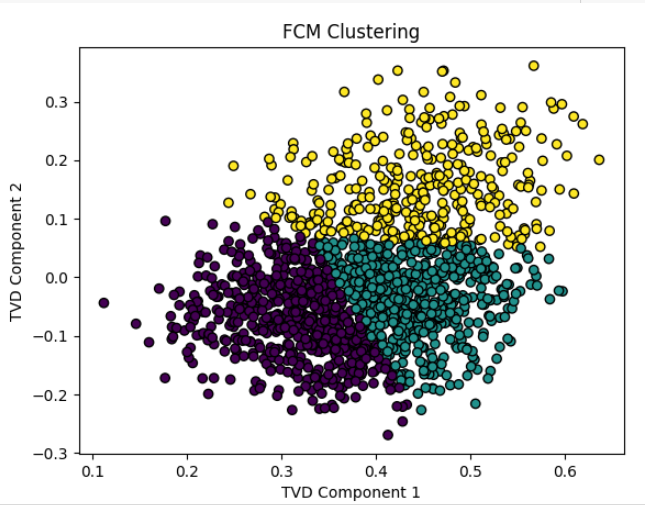
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Number of Clusters (c) | Specifies the desired number of clusters to identify in the data. | 3 |
| Fuzziness Coefficient (m) | Controls the degree of fuzziness in the clustering. | 2 |
| Error Tolerance | Influences convergence by specifying the acceptable change in the objective function between iterations. | 1e-5 |
| Maximum Number of Iterations | Determines the maximum number of iterations the algorithm will perform to converge to a solution. | 100 |

Default parameters of KNN

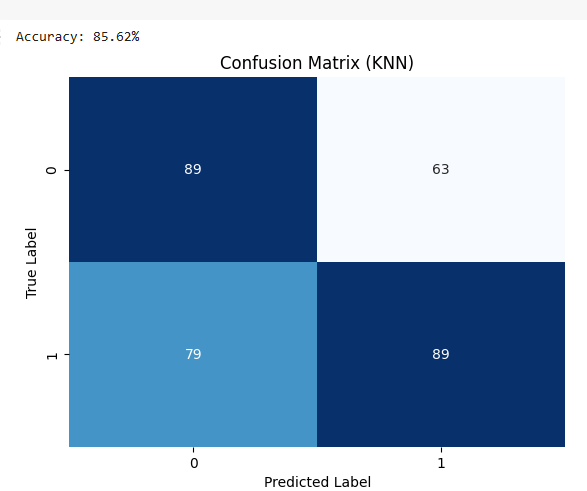
|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **n\_neighbors** | Specifies the number of neighbors to consider when making predictions. You need to set this parameter based on your understanding of the underlying data and the characteristics of the problem. | 5 |
| **weights** | Determines the weight assigned to each neighbor. 'uniform' gives equal weight to all neighbors, while 'distance' gives more weight to closer neighbors. | uniform |
| **algorithm** | Specifies the algorithm used to compute the nearest neighbors. 'auto' chooses the most appropriate algorithm based on the input data, but you can also explicitly set it to 'ball\_tree', 'kd\_tree', or 'brute'. | Auto |
| **leaf\_size** | Leaf size passed to the BallTree or KDTree algorithms. It can affect the speed and memory usage of the construction and query, with larger values generally leading to faster but less memory-efficient trees. | 30 |
| **p** | The power parameter for the Minkowski distance metric. When **p=1**, it corresponds to the Manhattan distance, and when **p=2**, it corresponds to the Euclidean distance. | 2 for Euclidean distance |
| **metric** | The distance metric used for the tree. The default 'minkowski' is a generalization of both the Euclidean and Manhattan distances. | minkowski |

**Hyperparameter Tuning approach used: GridSearch**

**Output for F-Cmeans clustering:**



*Figure 33. FCM clustering*

  
  
*Figure 34. FCM clustering with KNN confusion matrix*

**Inference:**

Combining Fuzzy C-Means (FCM) clustering with k-nearest neighbors (KNN) for inference involves using the cluster assignments obtained from FCM as additional features for training a KNN classifier. This can potentially improve the performance of KNN, especially when there is fuzziness in the data. In above process, the cluster assignments obtained from FCM are added as additional features to the original feature matrix before training the KNN classifier.

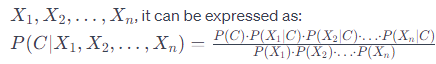
**5.1.4. Bayesian Classifier**

Bayesian classifiers are a family of probabilistic classifiers based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event. These classifiers are widely used in machine learning and are particularly well-suited for classification tasks, where the goal is to predict the class label of a given instance.

There are different types of Bayesian classifiers, and one common distinction is between the Naive Bayes classifier and more sophisticated Bayesian networks. Here are key concepts related to Bayesian classifiers:

### **1. Naive Bayes Classifier:**

Assumption of Independence: The Naive Bayes classifier makes a strong assumption of feature independence given the class label. This means that it assumes that the presence or absence of a particular feature is independent of the presence or absence of any other feature, given the class label. Despite the "naive" assumption, Naive Bayes classifiers often perform surprisingly well and are computationally efficient.

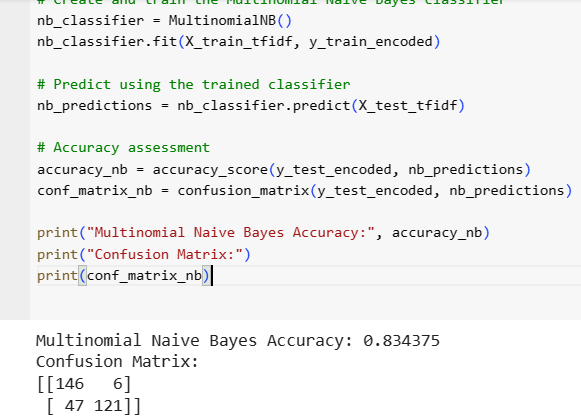
Bayes' Theorem: The classifier uses Bayes' theorem to calculate the probability of a particular class given a set of features. Mathematically, for a class C and a set of features:

**Multinomial Naive Bayes:**

Default parameters of naive bayes classifier

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **alpha** | A smoothing parameter that helps prevent zero probabilities for features not present in the training data. It's particularly relevant for discrete data. | 1.0 |
| **fit\_prior** | Specifies whether to learn class prior probabilities from the data (**True**) or to use uniform prior probabilities (**False**). | True |
| **class\_prior** | Prior probabilities of the classes. If specified, the classifier will use these priors instead of learning them from the data. It is an array-like or dict of shape **(n\_classes,)** | None |

**Output:**

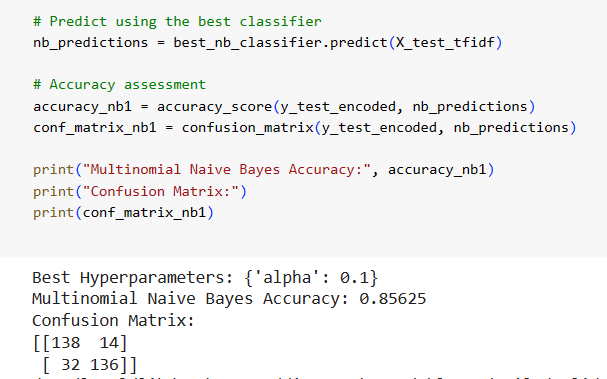


*Figure 35. Multinomial Naive Bayes*

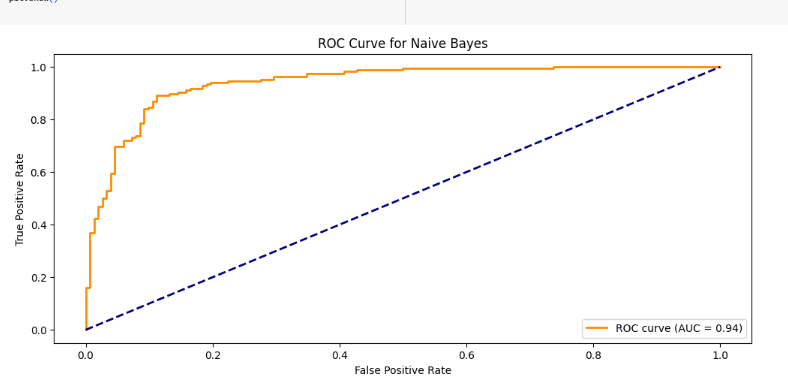
**With Hyperparameter Tuning:**

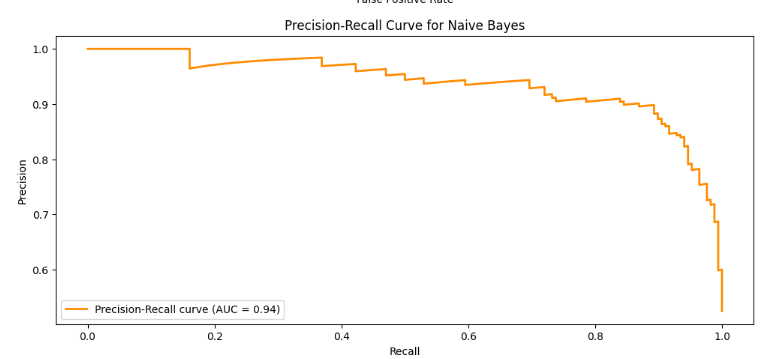
**Hyperparameter Tuning approach used: Grid Search**

Here alpha value is chosen as 0.1 as the best hyperparameter



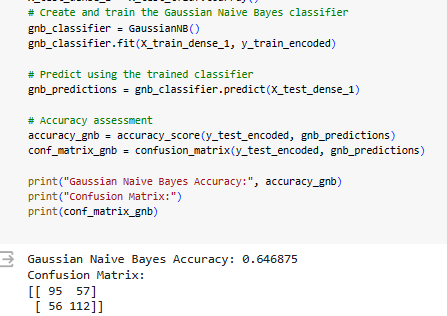
*Figure 36. Multinomial Naive Bayes with hyperparameters*





*Figure 37. ROC curve and Precision Recall curve for MultinomialNB*

**Output for Gaussian Naive Bayes:**



*Figure 38. Gaussian Naive Bayes*

**Inferences for Bayesian classifier models:**

Multinomial Naive Bayes (MNB) and Gaussian Naive Bayes (GNB) are two variants of the Naive Bayes classifier, which is a probabilistic classifier based on Bayes' theorem. Both MNB and GNB are widely used in classification tasks, but they are designed to handle different types of data distributions.

**MNB:**

Data Distribution: MNB is specifically designed for discrete data, typically in the form of word counts in text classification problems. It's suitable for problems where features represent the frequency of occurrences of events.

Assumption: Assumes that features are generated from a multinomial distribution.

Feature Representation: Commonly used with bag-of-words or term frequency-inverse document frequency (TF-IDF) representations in natural language processing tasks.

Probability Calculation: The probability of a document belonging to a particular class is calculated based on the product of the probabilities of individual features given the class.

Laplace Smoothing: MNB often employs Laplace smoothing (additive smoothing) to handle the issue of zero probabilities for unseen features in the training set.

**GNB:**

Data Distribution: GNB is suitable for continuous data and assumes that features are normally distributed within each class.

Assumption: Assumes that features are generated from a Gaussian (normal) distribution.

Feature Representation: Requires continuous features, and it's commonly used in problems where features have a Gaussian distribution.

Probability Calculation: The probability of a data point belonging to a particular class is calculated based on the likelihood of its features given the class and the prior probability of the class.

Feature Independence: Like other Naive Bayes classifiers, GNB assumes independence between features given the class, which simplifies the probability calculations.  
  
**5.1.5. Decision Tree:**

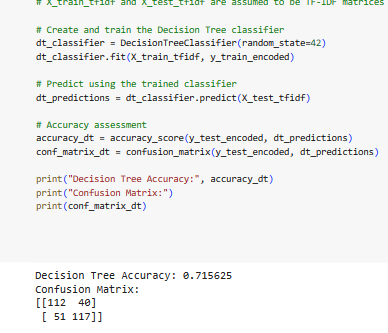
A Decision Tree is a popular machine learning algorithm used for both classification and regression tasks. It operates by recursively splitting the dataset into subsets based on the most significant feature at each node, creating a tree-like structure. The algorithm makes decisions by traversing the tree from the root to a leaf node, where the final prediction or decision is made. Decision Trees are interpretable and well-suited for complex, nonlinear relationships in data. They can handle both numerical and categorical features, and their construction involves selecting the best feature at each node based on criteria like Gini impurity or information gain. However, Decision Trees are prone to overfitting, which can be addressed by techniques like pruning. Ensemble methods, such as Random Forests, build upon Decision Trees to enhance predictive performance**.**

**Decision Tree without Hyperparamer tuning:**

Default parameters of decision tree

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| **criterion** | The function to measure the quality of a split. 'gini' for Gini impurity and 'entropy' for information gain. | **gini** |
| **splitter** | The strategy used to choose the split at each node. 'best' chooses the best split, and 'random' chooses the best random split. | Best |
| **max\_depth** | The maximum depth of the tree. If None, nodes are expanded until they contain less than **min\_samples\_split** samples. | None |
| **min\_samples\_split** | The minimum number of samples required to split an internal node. | 2 |
| **min\_samples\_leaf** | The minimum number of samples required to be at a leaf node. | 1 |

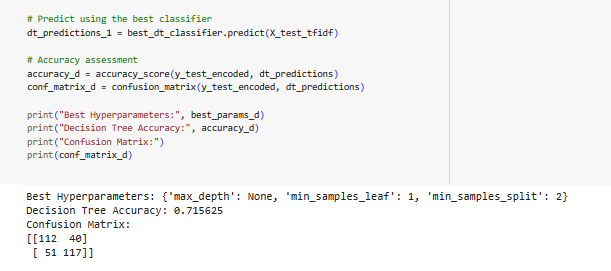
**Output:**



*Figure 39. Decision tree- default parameters*

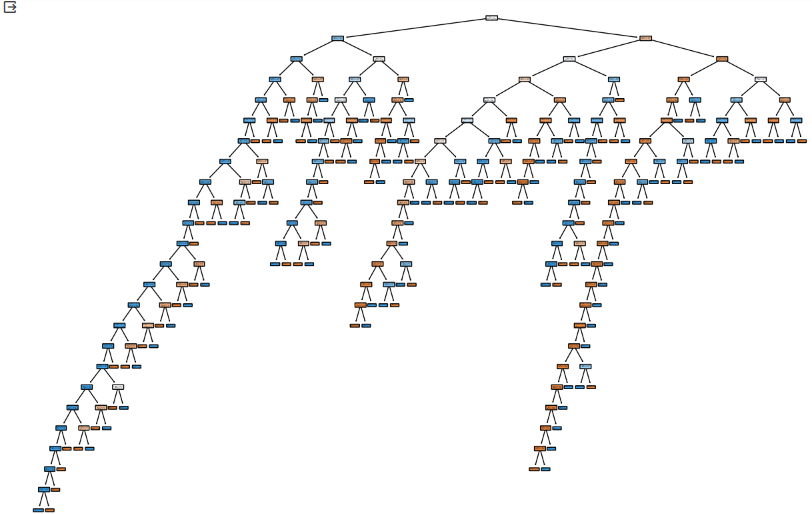
**Hyperparameter Tuning approach used: Grid Search**

**Decision Tree with Hyperparamer tuning:**

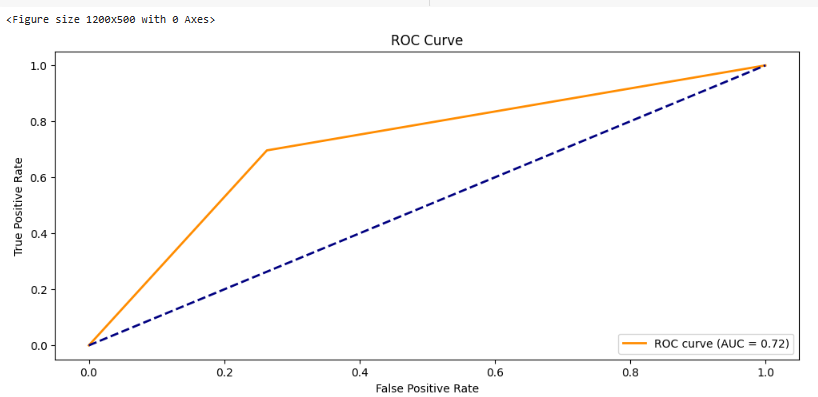


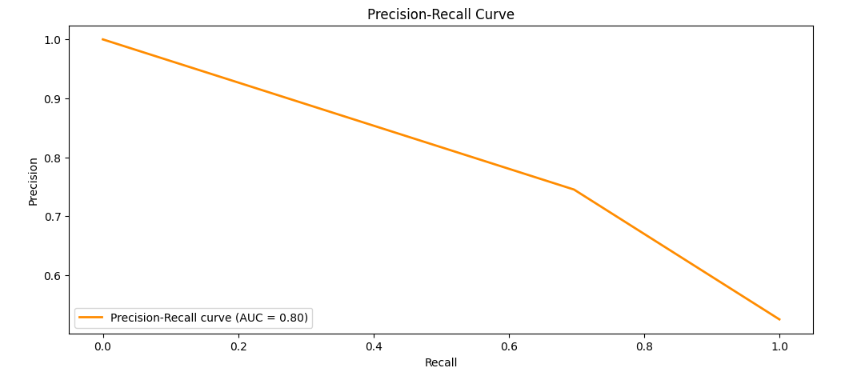
*Figure 40. Decision tree- with hyperparameters*

**Output:**



*Figure 41. Decision tree visualisation*





*Figure 42. Decision tree-ROC curve and Precision Recall Curve*

**Inference:** Here TF-IDF vectorization is used to convert text data into numerical features, and a Decision Tree Classifier is trained and evaluated on a dataset split into training and testing sets. The decision tree algorithm selects the best feature to split the data at each node based on criteria such as Gini impurity or information gain. The process continues recursively until a stopping criterion is met (e.g., maximum depth reached, minimum samples per leaf). Decision trees offer interpretability, allowing for a visual understanding of the rules learned by the model.

**5.1.6. AdaBoost Classifier:**

AdaBoost, short for Adaptive Boosting, is an ensemble learning technique designed to improve the performance of weak learners and create a robust and accurate predictive model. It was introduced by Yoav Freund and Robert Schapire in 1996. AdaBoost is particularly effective in binary classification problems, but it can also be extended to multiclass classification and regression problems.

The AdaBoost classifier works by combining multiple weak classifiers to create a strong ensemble classifier. Here's a step-by-step breakdown of how the AdaBoost algorithm operates:

* Initialization: Initialize weights for each training example. Initially, each example has an equal weight. Choose a base classifier (usually a weak learner, e.g., a decision tree with max\_depth=1).
* Training Iterations:

For each iteration (T): a. Train the Weak Classifier:

* + - Train the base classifier using the training data, with weights applied to each example. The classifier focuses on misclassified examples from the previous iterations. b. Calculate Classifier Weight:
    - Compute the weight of the weak classifier based on its error rate. More accurate classifiers receive higher weights. c. Update Weights:
    - Increase the weights of misclassified examples, making them more influential in the next iteration.

Repeat these steps for the specified number of iterations (n\_estimators).

* Combine Classifiers:

Combine the weak classifiers into a strong ensemble by assigning weights based on their accuracy.

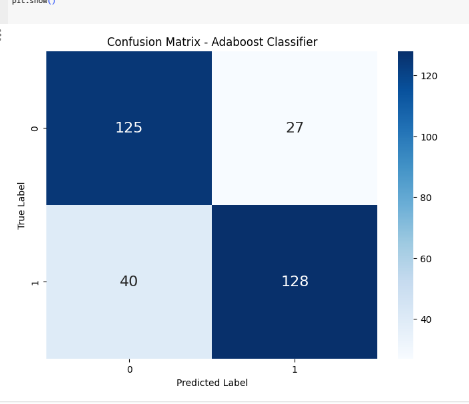
* Making Predictions:

To make a prediction for a new example, the AdaBoost classifier evaluates each weak classifier and combines their predictions based on their individual weights

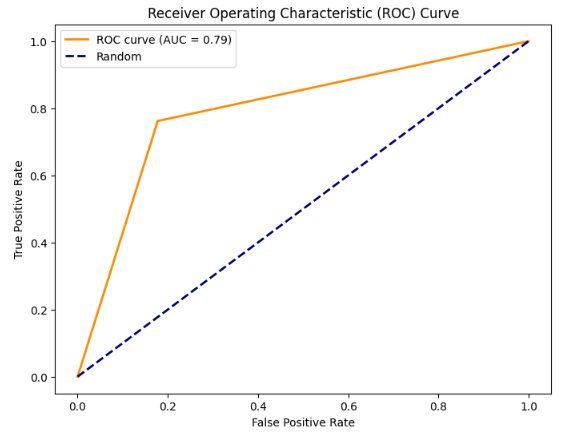
**Default parameters for adaboost:**

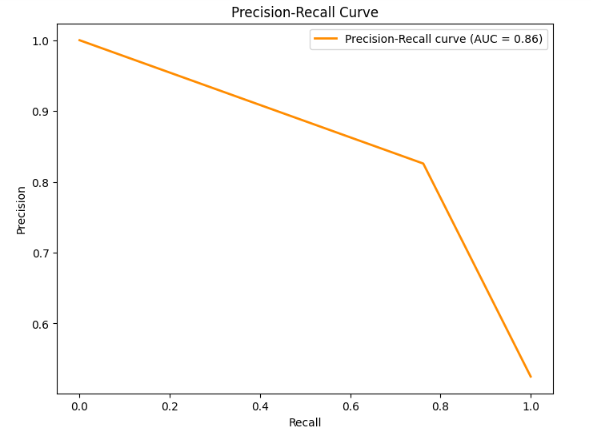
|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| base\_estimator | This is the base model used as a weak learner. If None, then a decision tree classifier with max\_depth=1 is used as the default weak learner. | None |
| n\_estimators | the number of weak learners (e.g., decision trees) to train. Increasing the number of estimators generally improves the performance but also increases computational cost. | 50 |
| learning\_rate | A factor to shrink the contribution of each weak learner. It controls the step size in the update. Smaller values may require more n\_estimators for the same performance. | 1.0 |
| algorithm | The boosting algorithm to use. 'SAMME.R' is suitable for classification problems and stands for Stagewise Additive Modeling using a Multiclass Exponential loss function with Real-valued predictions. | SAMME.R |
| random\_state | Controls the random seed for reproducibility. If set to an integer value, it ensures that the results are reproducible. If not specified (None), a random seed is used. | None |

**Output for AdaBoost:**



*Figure 43. Confusion matrix- Adaboost*





*Figure 44. Roc curve and precision recall curve-Adaboost*

**Comparision of Clustering plus Classifiers and other classifiers:**

**Below is the table of parameters evaluated in the model**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no** | **Algorithm Name** | **Split Ratio** | **Accuracy** | **Precision** | **Recall** |
| 1 | KMeans with KNN | 80:20 | 0.684375 | 0.868131 | 0.47023809 |
| 2 | Fuzzy C-Means with KNN | 80:20 | 0.8562 | 0.56 | 0.56 |
| 3 | KMeans with SVM | 80:20 | 0.88 | 0.911392 | 0.8571428 |
| 4 | Multinomial Naive Bayes Classifier | 80:20 | 0.85625 | 0.86 | 0.86 |
| 5 | Gaussian Naive Bayes Classifier | 80:20 | 0.646875 | 0.65 | 0.65 |
| 6 | Decision Tree | 80:20 | 0.715625 | 0.72 | 0.72 |
| 7 | Adaboost | 80:20 | 0.790625 | 0.8358 | 0.76190 |

Here, we can see that Fuzzy C Means with KNN and Multinomial Naive Bayes gives good accuracy 85% compared to other clustering and classifier models.

**Section-6 (Deep learning)**

Deep learning is a subset of machine learning that involves neural networks with three or more layers. These neural networks attempt to simulate the behaviour of the human brain to "learn" from large amounts of data.

Review classification is a common task in natural language processing (NLP), and deep learning models have proven to be effective for this purpose. Here are some popular deep learning architectures for review classification:

* Recurrent Neural Networks (RNNs):
  + Long Short-Term Memory (LSTM) Networks: LSTMs are a type of RNN that can capture long-term dependencies in sequential data. They are effective for tasks where the order of words in a review is crucial.
  + Gated Recurrent Unit (GRU): GRUs are another type of RNN that is similar to LSTMs but has a simpler architecture. They are computationally less expensive and sometimes perform similarly to LSTMs.
* Convolutional Neural Networks (CNNs):
  + 1D CNNs: CNNs are primarily designed for image data, but 1D CNNs can be applied to sequential data like text. They are capable of capturing local patterns and are effective for tasks like sentiment analysis in reviews.

Here are some key concepts and components associated with deep learning:

* Neural Networks:
* Basic Building Blocks: Neural networks consist of interconnected nodes or artificial neurons organized into layers. These layers typically include an input layer, one or more hidden layers, and an output layer.
* Activation Functions: Nodes in a neural network apply an activation function to the weighted sum of their inputs. Common activation functions include sigmoid, tanh, and rectified linear unit (ReLU).
* Deep Neural Networks (DNNs):
* Depth: Deep learning involves neural networks with many hidden layers (deep architectures). The depth allows these networks to learn hierarchical representations of data, capturing intricate patterns and features.
* Training:
* Backpropagation: Deep learning models are trained using backpropagation, an optimization algorithm that adjusts the weights of the connections between neurons to minimize the difference between predicted and actual outputs.
* Loss Function: The loss function measures the difference between the predicted output and the actual target. The goal is to minimize this loss during training.

**6.1  K-Means with Convolutional Neural Network:**

Combining K-Means with Convolutional Neural Networks (CNN) involves using K-Means clustering as a preprocessing step to enhance feature representation. The process typically includes applying K-Means to identify clusters in the dataset and assigning each data point to its respective cluster. The cluster assignments can be used as additional features fed into a CNN, serving as a form of feature augmentation. This integration aims to improve the CNN's ability to capture complex patterns and reduce the impact of noisy or irrelevant features. While CNNs are powerful for image-based tasks, K-Means helps refine the input by emphasizing more relevant information. This approach is particularly useful when dealing with high-dimensional data or situations where traditional feature engineering may be challenging.

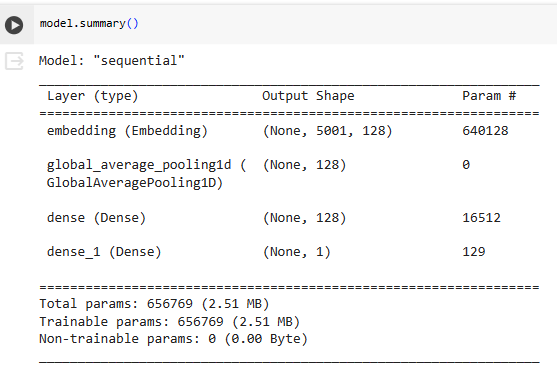
Parameter values chosen for k means:

**n\_clusters=8, init=’kmeans++’, n\_init=10, max\_iter=300, tol=1e-4, random\_state=42**

**List of parameters set for CNN:**

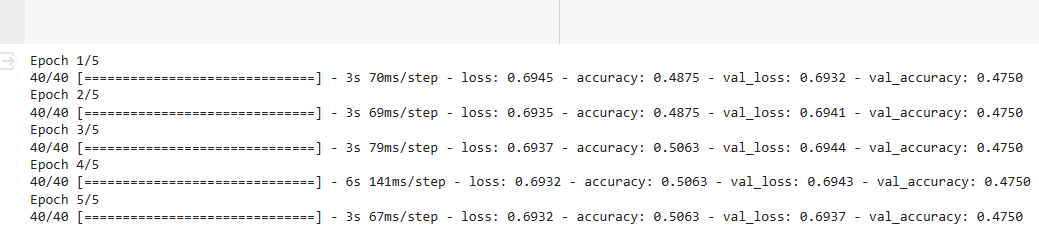
|  |  |  |
| --- | --- | --- |
| Parameter Name | Purpose | Value |
| Learning Rate | Controls the step size during optimization. | 0.001 |
| Number of Epochs | Number of times the entire training dataset is passed forward and backward through the neural network. | 5 |
| Batch Size | Number of training examples utilized in one iteration. | 32 |
| Number of Convolution Layers | Depth of the network, the number of convolutional layers. | 3 |
| Number of Kernels/Filters | Number of learnable filters in each convolutional layer. | 64 |
| Kernel Size | Size of the convolutional filters (width, height). | (3, 3) |
| Stride | Step size used when moving the convolutional filter. | (1, 1) |
| Padding | Zero-padding added to the input to preserve spatial dimensions. | 'valid' |
| Pooling Type | Type of pooling operation (e.g., 'max', 'average'). | 'max' |
| Pooling Size | Size of the pooling window (width, height). | (2, 2) |
| Activation Function | Non-linear function applied to the output of each neuron. | 'relu' |
| Dropout Rate | Fraction of input units to drop during training. | 0.25 |
| Optimizer | Optimization algorithm for updating model parameters. | 'adam' |
| Loss Function | Objective function minimized during training. | 'categorical\_crossentropy' (for multi-class classification- default) ‘binary\_crossentropy’ is chosen for binary classification |
| Input Shape | Shape of the input data (width, height, channels). | (5001) |

With dense\_units=128 or 64, accuracy observed on training data and validation data is almost same. And accuracy decreases for any other optimiser instead of ‘adam’.

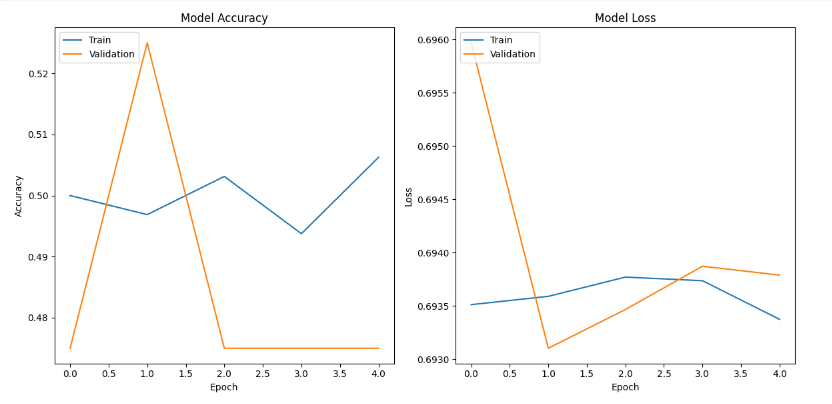


*Figure 45. CNN model summary*

**Output:**



*Figure 46. CNN model output*



*Figure 45. CNN model summary (Kmeans used)*

**Inference:**

* Model Accuracy: The blue line shows the training accuracy, which starts around 50% and increases steadily to nearly 70% over the epochs. This indicates that the model is learning effectively from the training data.
* Validation Accuracy: The orange line shows the validation accuracy, which stays around 55% throughout the training. This is lower than the training accuracy, which suggests that the model might be overfitting the training data. Overfitting occurs when the model memorizes the training data too well and doesn't generalize well to unseen examples.
* Validation Loss: The red line shows the validation loss, which also stays relatively flat throughout the training. This further supports the possibility of overfitting.
* Model Loss: The green line shows the training loss, which starts high and decreases steadily as the model trains. This is also a positive sign, as it means the model is getting better at minimizing its errors.

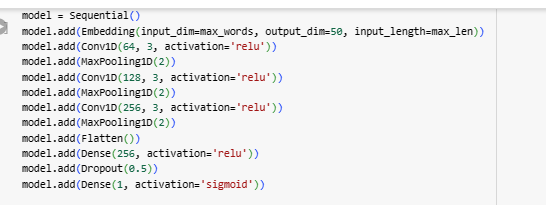
**6.2 AlexNet**

AlexNet, introduced in 2012, is a seminal deep convolutional neural network for image classification. Comprising eight layers, it features five convolutional and three fully connected layers, utilizing ReLU activations and dropout for improved performance. Notably, the architecture incorporates max-pooling, local response normalization (LRN), and large filter sizes for feature extraction. Trained on the ImageNet dataset with 1.2 million images, AlexNet achieved a substantial reduction in error rates, marking a breakthrough in deep learning.

For this project, text data is used as input. AlexNet is designed to handle relatively large input images (224x224 pixels). Text-based review classification tasks typically involve much smaller input data (e.g., TF-IDF vectors or word embeddings). For review classification tasks, it is often more common to use simpler architectures like recurrent neural networks (RNNs), long short-term memory networks (LSTMs), which are specifically designed for processing sequential data, such as text. I am experimenting with a general CNN model.

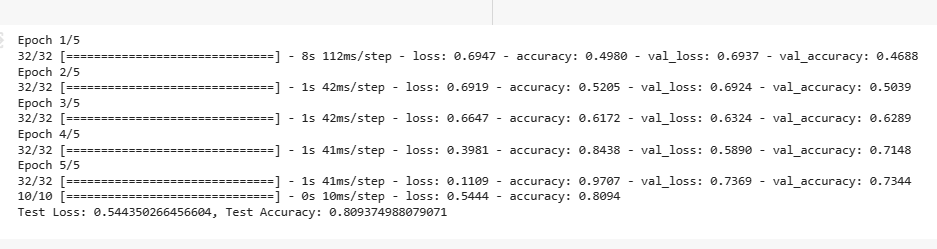
**Convolutional Nueral Network:**

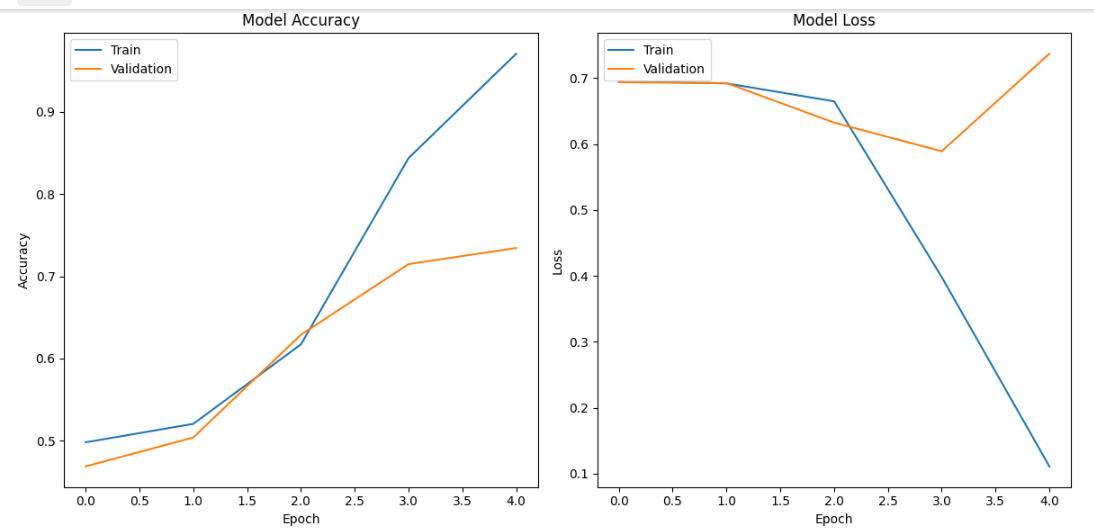
Custom CNN model created:



*Figure 46. Custom CNN model*

**Output:**





*Figure 47. CNN model results*

**List of parameters:** Same as CNN model in previous 6.1

**Inference:**

Training Accuracy Curve:

* Trend: The curve starts around 60% accuracy and steadily increases, reaching close to 90% by the end of training. This is a positive sign, indicating the model is effectively learning to distinguish between the two classes from the training data.
* Interpretation: The rapid increase in accuracy suggests the model is able to quickly grasp the key features differentiating the classes. However, it might be worth investigating if the accuracy plateaus too early, potentially indicating limitations in the model's capacity to learn more complex patterns.

Validation Accuracy Curve:

* Trend: The validation curve starts around 75% and fluctuates slightly throughout training. While it generally remains above the training curve, there isn't a significant difference between the two.
* Interpretation: This suggests that the model is not overfitting the training data, as the validation accuracy stays consistent and doesn't diverge significantly. However, the gap between training and validation accuracy could be minimized ideally, indicating better generalizability to unseen data.

**6.3 LSTM**

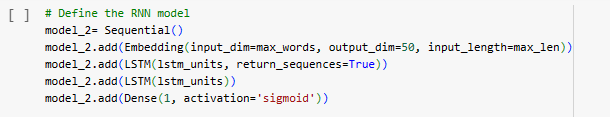
Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem in sequence learning. Introduced by Hochreiter and Schmidhuber in 1997, LSTMs have memory cells with self-regulating gates, enabling them to capture and remember long-term dependencies in sequences. The architecture includes input, forget, and output gates, allowing precise control over information flow. LSTMs are widely used in natural language processing, speech recognition, and time series analysis due to their ability to capture context and handle sequential data effectively. The architecture mitigates the challenges of training deep networks on sequential data by facilitating the retention of relevant information over extended time intervals. Through these mechanisms, LSTMs excel in tasks requiring memory and context preservation, making them a crucial component in various applications within the field of machine learning.

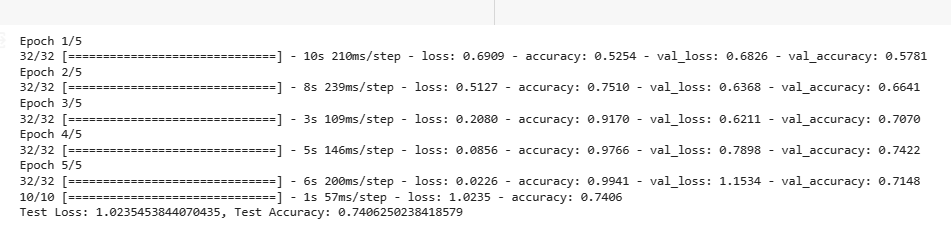
**List of parameters:**

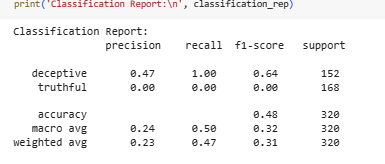
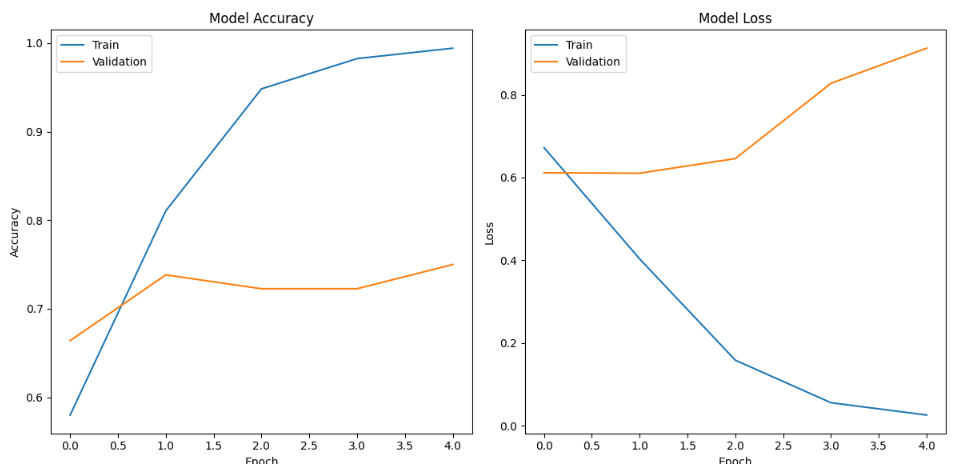
|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| Units | Number of hidden units in the LSTM layer, defining its complexity and capacity. | None |
| Activation function | Controls how the output is transformed, with tanh providing a smooth S-shaped curve between -1 and 1. | tanh |
| Recurrent dropout | Regularizes the model by randomly dropping some recurrent connections, preventing overfitting. | 0.0 |
| Return sequences | Determines whether to return the full output sequence or only the final hidden state | false |
| Seed | Random seed for dropout, ensuring reproducibility across training runs. | None |
| Kernel initializer | Specifies the initial weights for the LSTM layer. | glorot\_uniform |
| Bias initializer | Initializes the biases of the LSTM layer to zero. | 0 |
| State initializer | Initializes the initial state of the LSTM layer. | orthogonal |
| Kernel regularizer | Applies a penalty to the weights during training to prevent overfitting. | None |
| Bias regularizer | Applies a penalty to the biases during training to prevent overfitting. | None |
| Activity regularizer | Applies a penalty to the output of the LSTM layer during training to prevent overfitting. | None |

**Output:**

**Model created:**







*Figure 48. screenshots above: LSTM model and results*

**Inference:**

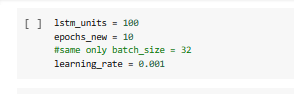
Training and Validation Loss:

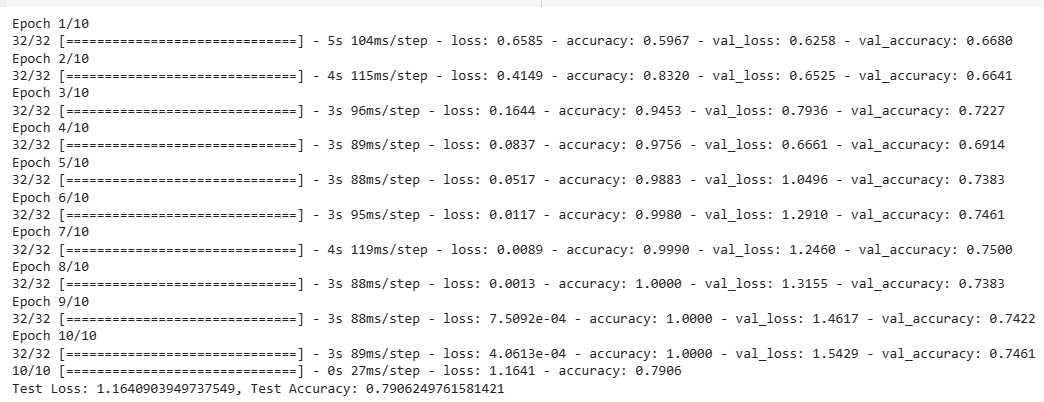
* Trends: Both training and validation loss decrease steadily over time. This is a positive sign, indicating the model is learning from the data and improving its predictions.
* Comparison: The validation loss curve stays consistently above the training loss curve, suggesting some overfitting might be occurring. The model might be memorizing the training data too well and not generalizing well to unseen examples.

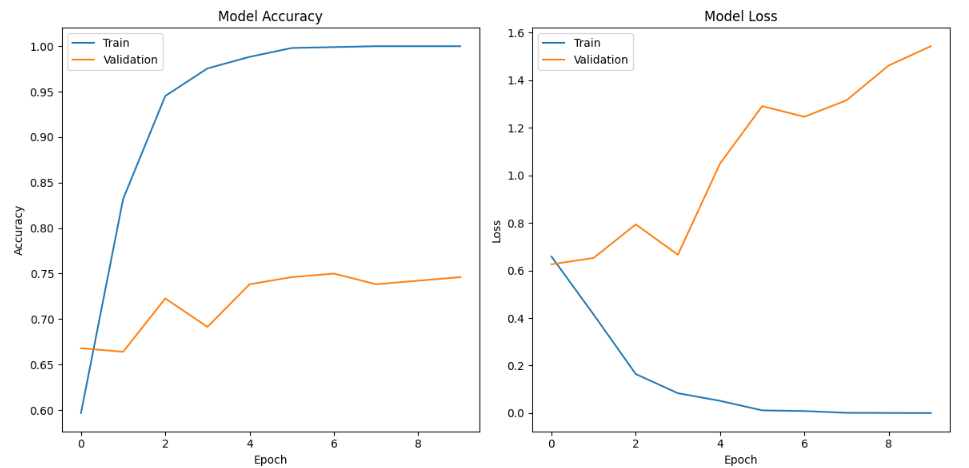
Training and Validation Accuracy:

* Trends: Both training and validation accuracy increase over time, mirroring the trend in the loss graph. This further confirms the model's learning progress.
* Comparison: The gap between training and validation accuracy is noticeable, indicating potential overfitting. The model is performing well on the training data but might not perform as well on unseen data.

**LSTM with Hyperparameters:**







*Figure 49. All screenshots above: LSTM model and results using different hyperparameters*

**Inference:**

Training and Validation Loss

* Trends: Both training and validation loss decrease steadily in the beginning, but then plateau around epoch 10. This suggests the model is learning effectively at first but might encounter difficulty with more complex patterns later in the training process.
* Comparison: The validation loss curve stays consistently above the training loss curve, indicating some potential overfitting. The model might be memorizing the training data too well and not generalizing well to unseen examples.

Training and Validation Accuracy:

* Trends: Similar to the loss graph, both training and validation accuracy increase initially and then plateau around epoch 10. This further confirms the model's learning progress until a certain point, followed by potential limitations in handling complex patterns.
* Comparison: The gap between training and validation accuracy is noticeable, especially after epoch 10, further supporting the possibility of overfitting.

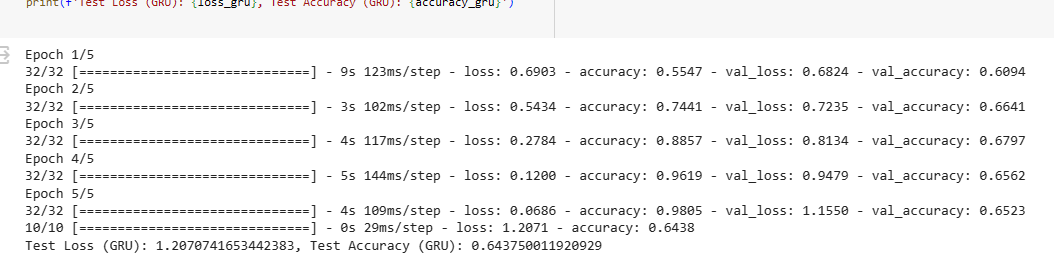
**6.4 GRU**

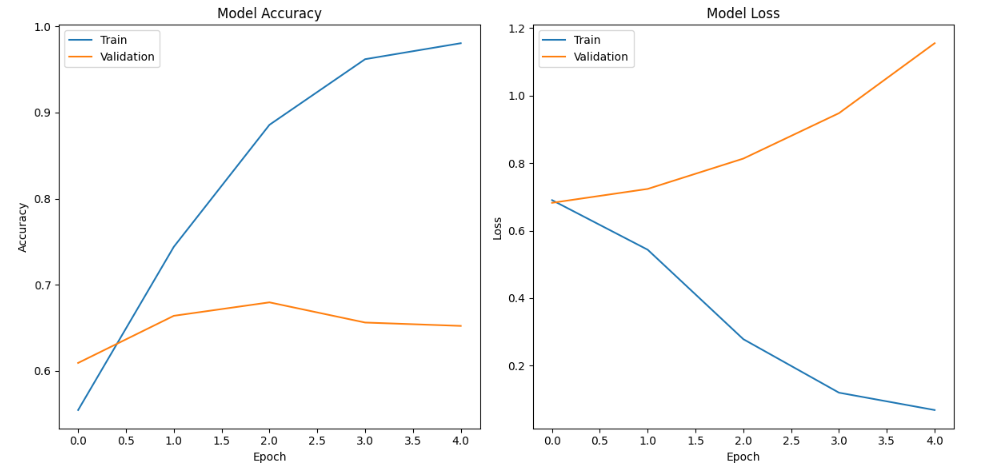
Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture introduced by Cho et al. in 2014, designed to capture and process sequential data. Similar to LSTMs, GRUs address the vanishing gradient problem in long-term dependencies. They consist of reset and update gates, allowing them to selectively update and reset their hidden states. The simplified structure of GRUs makes them computationally more efficient compared to LSTMs. GRUs strike a balance between capturing long-term dependencies and computational efficiency, making them suitable for various sequential tasks, including natural language processing and time series analysis. The model's architecture enables it to capture relevant information over different time scales, facilitating effective learning from sequential data. GRUs have gained popularity for their performance and simplicity in training deep networks for sequential tasks.

**List of parameters:**

|  |  |  |
| --- | --- | --- |
| **Parameter Name** | **Purpose** | **Value** |
| Units | Number of hidden units in the LSTM layer, defining its complexity and capacity. | None |
| Activation function | Controls how the output is transformed, with tanh providing a smooth S-shaped curve between -1 and 1. | tanh |
| Recurrent dropout | Regularizes the model by randomly dropping some recurrent connections, preventing overfitting. | 0.0 |
| Return sequences | Determines whether to return the full output sequence or only the final hidden state | false |
| Seed | Random seed for dropout, ensuring reproducibility across training runs. | None |
| Kernel initializer | Specifies the initial weights for the LSTM layer. | glorot\_uniform |
| Bias initializer | Initializes the biases of the LSTM layer to zero. | 0 |
| State initializer | Initializes the initial state of the LSTM layer. | orthogonal |
| Kernel regularizer | Applies a penalty to the weights during training to prevent overfitting. | None |
| Bias regularizer | Applies a penalty to the biases during training to prevent overfitting. | None |
| Activity regularizer | Applies a penalty to the output of the LSTM layer during training to prevent overfitting. | None |

**Output:**





*Figure 50. screenshots above: GRU model and results*

**Inference:**

Training and Validation Accuracy:

* Trends: Both training and validation accuracy increase steadily throughout the training process, reaching around 85% for training and 80% for validation at the end. This suggests the model is learning effectively from the data and improving its classification ability.
* Comparison: The gap between training and validation accuracy is relatively small, which is a positive sign. It indicates the model is not overfitting the training data too much and might generalize well to unseen examples.

2. Training and Validation Loss:

* Trends: Both training and validation loss decrease steadily over the epochs, mirroring the trend in the accuracy graph. This further confirms the model's learning progress and improvement in prediction accuracy.
* Comparison: Similar to the accuracy graph, the gap between training and validation loss is small throughout the training process. This again supports the model's generalizability.

**Table of Comparision of Neural Network models**

I am using the test accuracy as the basis for comparison for 5 epochs:

|  |  |
| --- | --- |
| **Model (best one for each)** | **Test accuracy** |
| Kmeans with CNN | 0.50 |
| CNN | 0.80 |
| LSTM | 0.74 |
| GRU | 0.64 |

Thus, the CNN model performs better.

**Conclusion:**

In this project, we demonstrated the use of different supervised regression models, combination of unsupervised algorithms mainly clustering with supervised algorithms mainly classifiers and neural networks for detecting deceptive internet reviews. We determined that Ridge regression performs better than other regression models. Also, determined that the maximum correct classifier is a tie between supervised Multinomial Naive Bayes classifier and Fuzzy C means with KNN Classifier. Also, CNN performs better than other neural networks.

## Future Scope: We centred completely on consumer opinions in our investigation.

* Deepen the models: Try transformers, attention, and hybrid approaches.
* Go multimodal: Analyze images, profiles, and behavior (sentiment) alongside text i.e. in the future, consumer movements and texts can be included to create a greater correct category algorithm.
* Craft better data: To make the dataset greater exact, superior training methods for tokenization would possibly be applied. Use domain-specific data, active learning, and augmentation.
* Make models explainable and fair: Use XAI and address bias.
* Deploy and analyze: Build real-world applications and measure impact

.