Dataset: <https://www.kaggle.com/c/fake-news/data?select=train.csv>

Base Paper: <https://dspace.mit.edu/handle/1721.1/120056>

Reference on same dataset: <https://arxiv.org/abs/1806.00749>

<https://ieeexplore.ieee.org/document/9312099>

<https://arxiv.org/abs/2203.09936>

Data Analysis:

1. It imports the necessary libraries and installs them if they are not already installed.
2. It reads the dataset **train\_news.csv** into a DataFrame.
3. It performs preliminary analysis on the dataset, including printing the shape and the first few rows of the DataFrame.
4. It removes any unnamed columns from the DataFrame.
5. It checks for missing data and prints the count of null values in each column.
6. It defines a function **show\_tf\_distribution** to display the distribution of null values based on the label (real or fake news).
7. It calls the **show\_tf\_distribution** function for the 'news', 'headline', and 'written\_by' columns to analyze the distribution of null values.
8. It investigates placeholder values and duplicates in the dataset, specifically focusing on the 'headline' column.
9. It drops rows with duplicate 'headline' and 'news' articles, as well as rows without 'headline' or 'news' values.
10. It replaces whitespace values with null in the DataFrame.
11. It shows the distribution of null values in the 'news' column based on the label.
12. It removes duplicated data by dropping rows with the same 'headline' and 'news' articles.
13. It explores the dataset by analyzing the distribution of labels (real or fake news) and the length of headlines and news articles.
14. It calculates the number of capital letters in the 'headline' and 'news' columns, as well as the percentage of capital letters in each.
15. It analyzes the presence of certain keywords ('via', 'image via', 'said', 'on', 'you') in the 'news' column and their relationship with the label.
16. Finally, it saves the cleaned dataset as 'train\_news\_cleaned.csv'.

Overall, this code performs data cleaning, handles missing values, removes duplicates, and explores various features of the news articles to gain insights into their characteristics.

Text Pre-processing / Feature extraction:

1. It imports the necessary libraries and modules such as pandas, nltk, and regex.
2. It reads the dataset from a CSV file using pandas.
3. It defines several helper functions for text preprocessing, such as concatenating lists of strings, finding URLs, creating frequency distributions, removing punctuation and single characters, and removing specific words.
4. It tokenizes the news articles using regular expressions and creates a frequency distribution of the tokens.
5. It identifies and replaces URLs in the news articles and headlines with a placeholder.
6. It identifies Twitter handles in the news articles and replaces them with a placeholder.
7. It performs capitalization normalization by making all words lowercase unless they are in all caps.
8. It replaces numbers with a space.
9. It tokenizes the clean news articles and headlines.
10. It removes punctuation and single-letter tokens from the clean headlines.
11. It removes possessive apostrophes from the clean headlines and news articles.
12. It removes date words from the clean headlines and news articles.
13. It removes stop words from the clean headlines and news articles.
14. It saves the preprocessed dataset to a new CSV file.

Execution:

It seems like you are implementing a machine learning model for text classification using various algorithms such as Naive Bayes, K Nearest Neighbors, Logistic Regression, Gradient Boosting, Random Forest, and Support Vector Machines. The code you provided is written in a Jupyter Notebook format.

Here is a breakdown of the code:

1. You import the necessary libraries, such as pandas, numpy, scikit-learn, and os.
2. You read the dataset from a CSV file using pandas, specifying the required columns.
3. You set some environment variables and random seeds for reproducibility.
4. You preprocess the data by extracting the features and the target variable from the dataset.
5. You split the data into training and validation sets using the train\_test\_split function from scikit-learn.
6. You perform feature extraction using TF-IDF (Term Frequency-Inverse Document Frequency) transformation on the training and validation data.
7. You define the parameters and hyperparameter search space for each algorithm.
8. You use the RandomizedSearchCV function from scikit-learn to perform hyperparameter tuning for each algorithm, using cross-validation and various scoring metrics.
9. You fit the models on the training data and evaluate them on the validation data.
10. You log the results, including the best parameters, training scores, and validation scores, for each algorithm.
11. You save the model parameters, training results, and validation classification report to separate CSV files and a text file.

The code appears to be a comprehensive implementation of several machine learning algorithms for text classification.

Result Aggregation:

1. The code begins by importing the necessary libraries such as pandas, numpy, regex, matplotlib, and seaborn.
2. It sets up the directory paths for the dataset and results.
3. It defines a list of model names and initializes empty lists for storing the results.
4. The code then processes the results for the base models. It reads the result files, drops unnecessary columns, and formats the data.
5. It calculates the evaluation metrics (accuracy, precision, recall, F1-score) for the base models on the validation set and stores the results.
6. The code processes the results for the hyper-tuned models. It reads the result files, selects the top ten models based on accuracy, and formats the data.
7. It calculates the evaluation metrics for the hyper-tuned models on the validation set and stores the results.
8. The code generates dataframes for the test, train, and validation results for both base and hyper-tuned models.
9. It saves the results as CSV files.
10. Finally, the code generates graphs to visualize the accuracy of the models on the test, train, and validation sets using seaborn and matplotlib.

Add results in this format: for each parameter accuracy, precision, recall and f1 score

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Base Algorithms** | | | **After Hyperparameter Tuning** | | |
| **Model** | **Train** | **Test** | **Validation** | **Train** | **Test** | **Validation** |
| GBC |  |  |  |  |  |  |
| LR |  |  |  |  |  |  |
| MNB |  |  |  |  |  |  |
| RFC |  |  |  |  |  |  |
| SVM |  |  |  |  |  |  |

Base Train Tes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1-Score** | **Recall** | **Precision** | **Model** |
| **0** | 0.938617 | 0.936771 | 0.950572 | 0.923365 | GBC |
| **1** | 0.965926 | 0.964425 | 0.965535 | 0.963319 | LR |
| **2** | 0.824251 | 0.775571 | 0.634866 | 0.996439 | MNB |
| **3** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | RFC |
| **4** | 0.997829 | 0.997733 | 0.998619 | 0.996849 | SVM |

Base Test Res

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1-Score** | **Recall** | **Precision** | **Model** |
| **0** | 0.923610 | 0.921226 | 0.933703 | 0.909124 | GBC |
| **1** | 0.938271 | 0.935821 | 0.940805 | 0.930910 | LR |
| **2** | 0.779387 | 0.701553 | 0.542360 | 0.993514 | MNB |
| **3** | 0.901397 | 0.893358 | 0.863458 | 0.925446 | RFC |
| **4** | 0.955449 | 0.953688 | 0.958958 | 0.948509 | SVM |

Base Validation res

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1-Score** | **Recall** | **Precision** | **Model** |
| **0** | 0.92776 | 0.92761 | 0.92809 | 0.92731 | GBC |
| **1** | 0.93657 | 0.93633 | 0.93604 | 0.93669 | LR |
| **2** | 0.80695 | 0.79455 | 0.79574 | 0.86391 | MNB |
| **3** | 0.90813 | 0.90745 | 0.90617 | 0.91039 | RFC |
| **4** | 0.95620 | 0.95604 | 0.95579 | 0.95635 | SVM |

Tuned Test Res

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1-Score** | **Recall** | **Precision** | **Model** |
| **0** | 0.962094 | 0.960448 | 0.962260 | 0.958699 | GBC |
| **1** | 0.957860 | 0.956082 | 0.958827 | 0.953369 | LR |
| **2** | 0.896055 | 0.881730 | 0.811406 | 0.966009 | MNB |
| **3** | 0.874686 | 0.864276 | 0.834794 | 0.896019 | RFC |
| **4** | 0.957948 | 0.956163 | 0.958695 | 0.953675 | SVM |

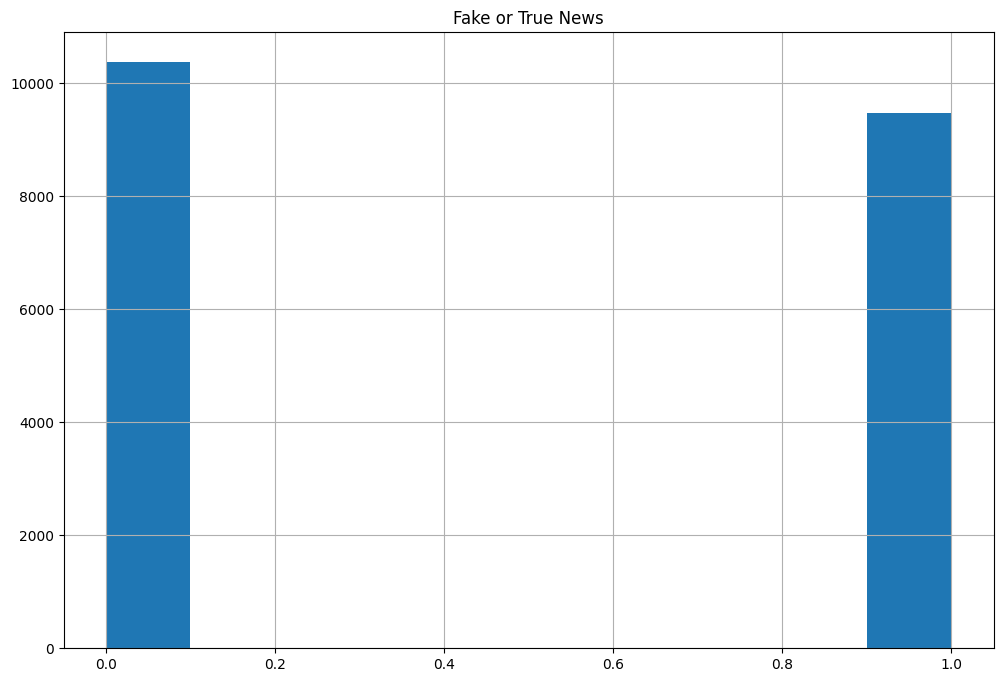
Tuned Train Res

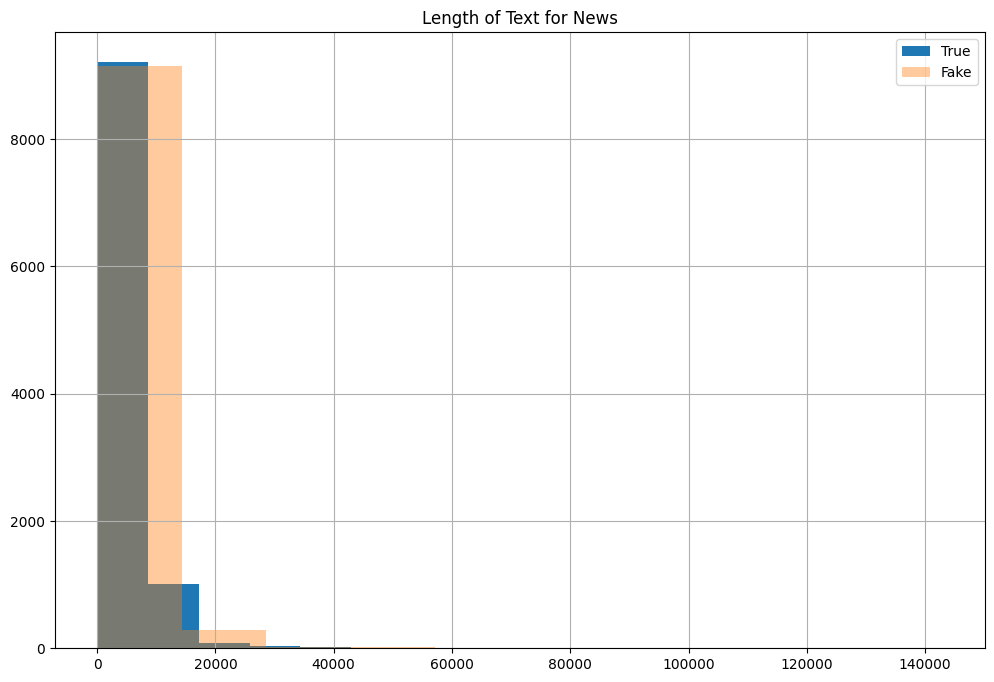
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1-Score** | **Recall** | **Precision** | **Model** |
| **0** | 0.999637 | 0.999620 | 0.999980 | 0.999261 | GBC |
| **1** | 0.999998 | 0.999998 | 0.999997 | 1.000000 | LR |
| **2** | 0.958594 | 0.954961 | 0.920886 | 0.991983 | MNB |
| **3** | 0.934303 | 0.930666 | 0.926309 | 0.935240 | RFC |
| **4** | 0.999580 | 0.999561 | 0.999549 | 0.999573 | SVM |

Tuned Validation Res

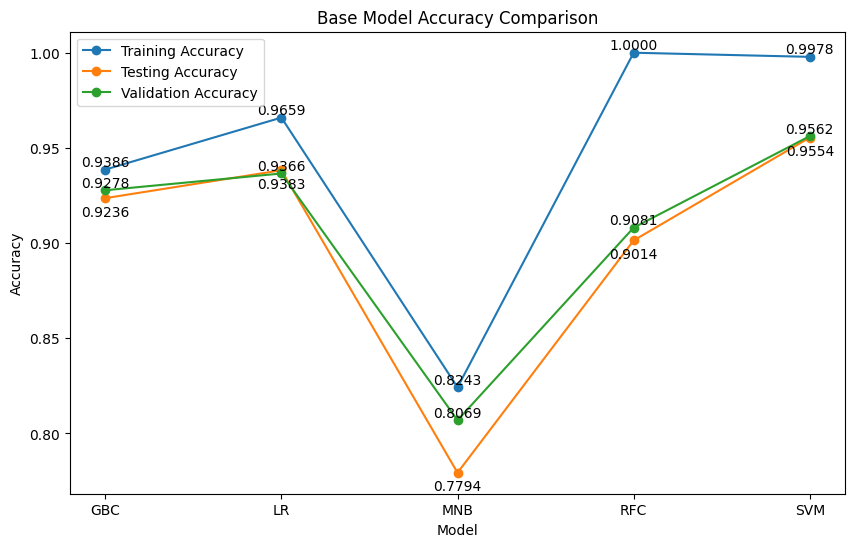
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Accuracy** | **F1-Score** | **Recall** | **Precision** | **Model** |
| **0** | 0.95872 | 0.95858 | 0.95848 | 0.95869 | GBC |
| **1** | 0.95469 | 0.95455 | 0.95447 | 0.95463 | LR |
| **2** | 0.91140 | 0.91021 | 0.90781 | 0.91931 | MNB |
| **3** | 0.91040 | 0.90980 | 0.90868 | 0.91214 | RFC |
| **4** | 0.95998 | 0.95984 | 0.95970 | 0.96000 | SVM |

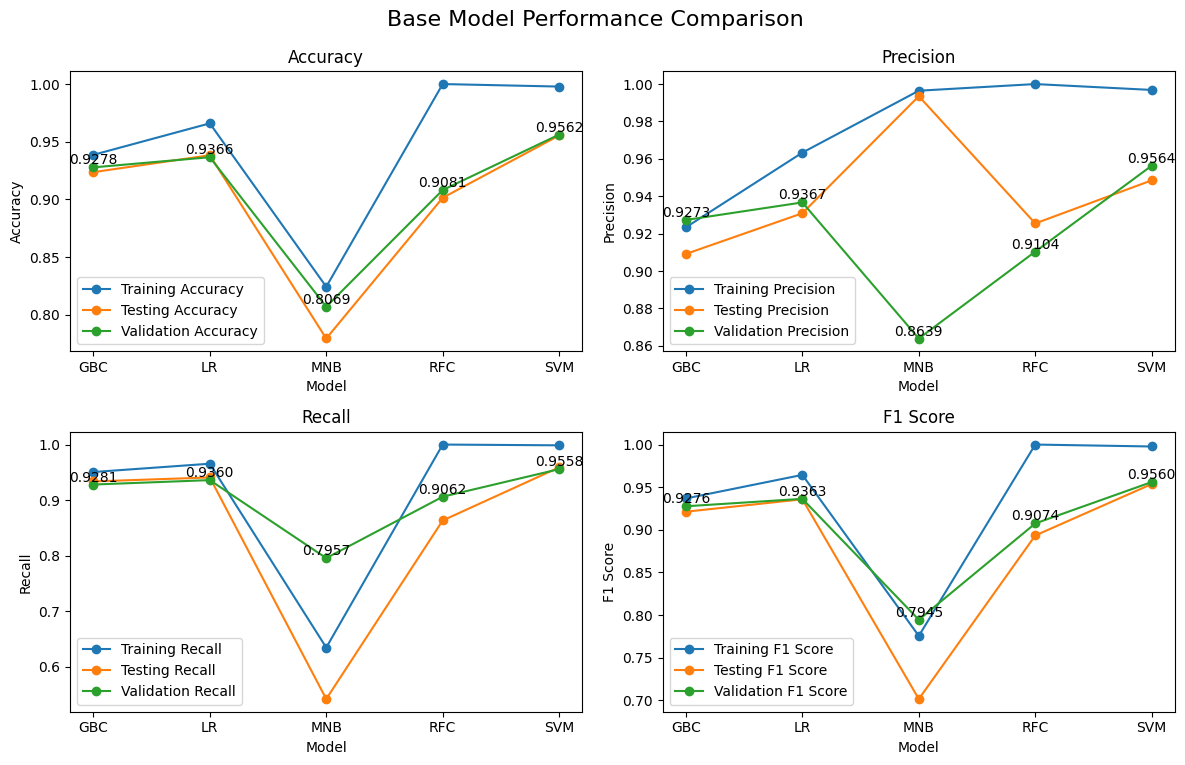
Dataset Info



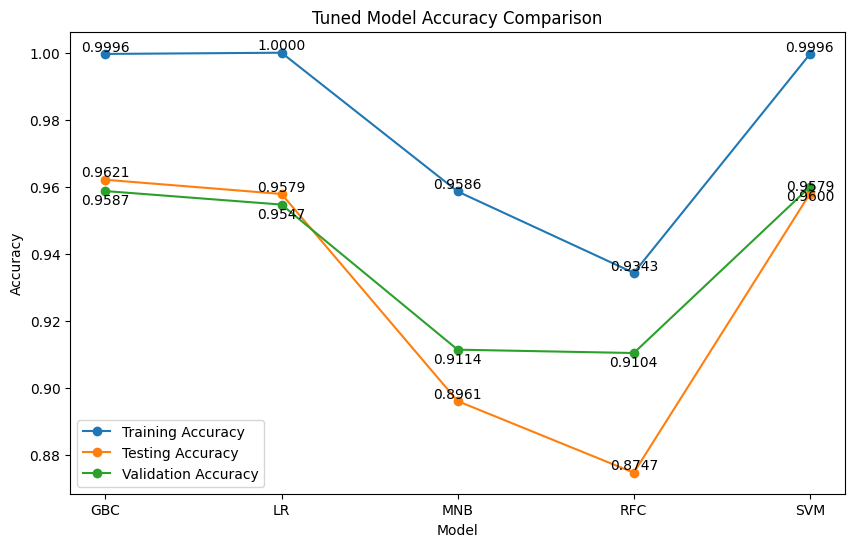


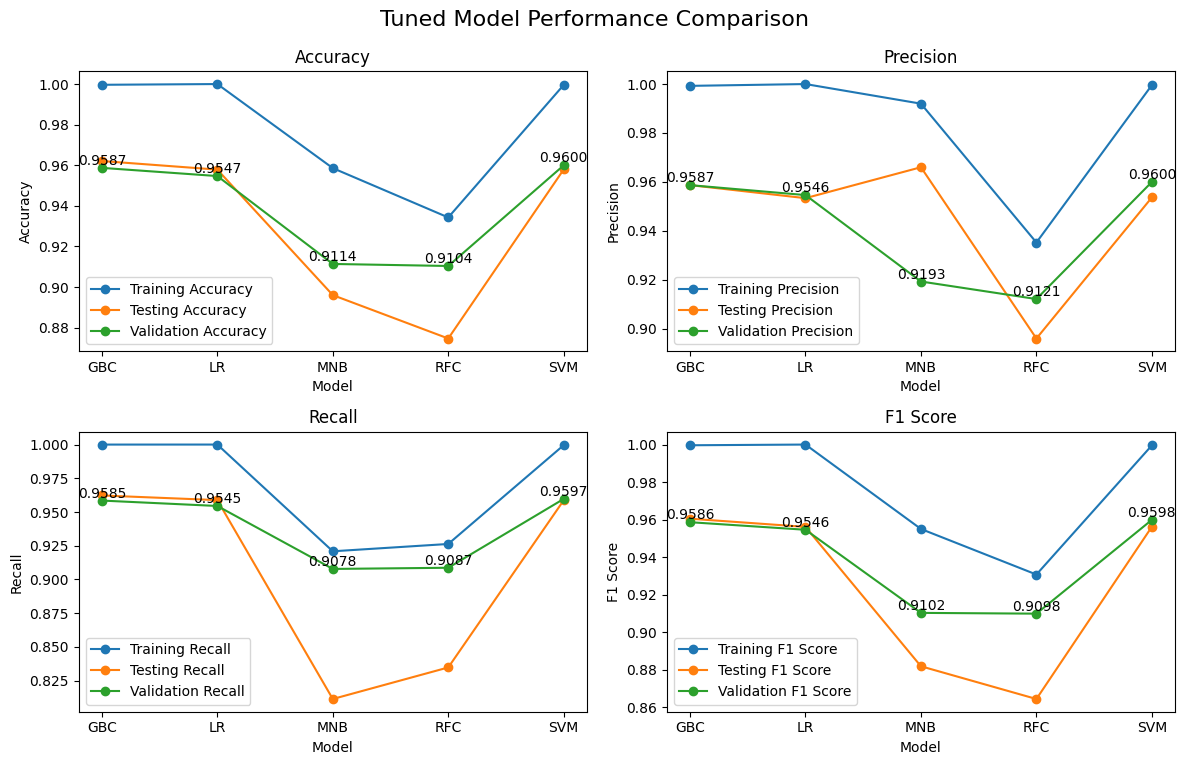
Base Models:



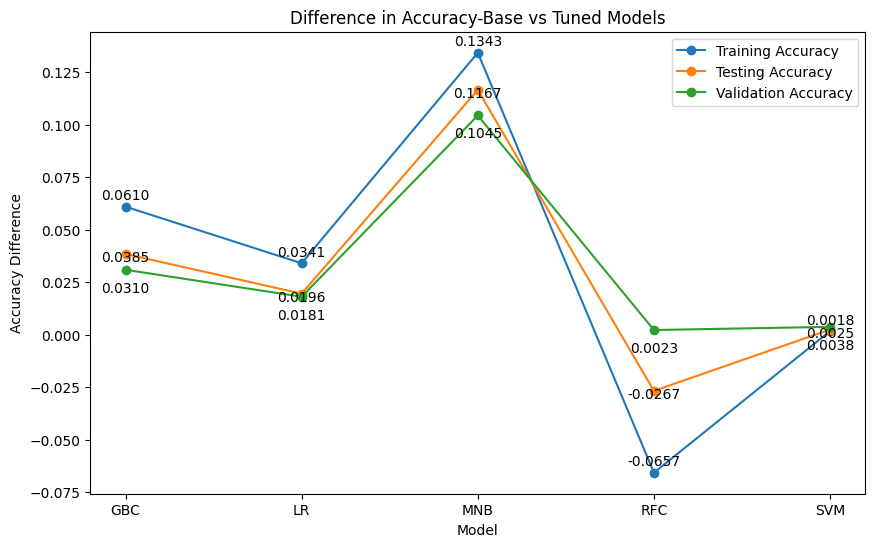


Tuned Model:





Difference in Accuracy Base Vs Tuned Model



Basic Summary of these results:

Basis of tuned vs base

Based on the provided results for the base model and tuned model, with a focus on the validation set, here are some observations:

1. Accuracy: The tuned model shows higher accuracy than the base model on the validation set. This indicates that the tuned model performs better in correctly classifying fake and real news compared to the base model.
2. F1-Score: The F1-score considers both precision and recall and provides a balanced measure of model performance. The tuned model demonstrates higher F1-scores compared to the base model on the validation set. This suggests that the tuned model achieves better overall performance in identifying both fake and real news instances.
3. Recall: Recall represents the ability of the model to correctly identify positive instances (in this case, correctly identifying fake news). The tuned model shows higher recall values than the base model on the validation set. This means that the tuned model has a better ability to detect fake news instances.
4. Precision: Precision represents the ability of the model to correctly classify instances as positive (correctly identifying fake news) out of the total instances predicted as positive. The tuned model demonstrates higher precision values compared to the base model on the validation set. This indicates that the tuned model has a lower rate of falsely classifying real news as fake news.

Considering these results, the tuned model outperforms the base model in terms of accuracy, F1-score, recall, and precision on the validation set. This suggests that the tuned model is more effective in detecting fake news, as it achieves higher accuracy while maintaining a balance between identifying fake news instances and minimizing false positives (misclassifying real news as fake news). Therefore, the tuned model can be considered more suitable for fake news detection tasks compared to the base model.

Based on results of validation on tuned model:

Based on the validation results for the tuned models, the following observations can be made regarding the performance of the models for fake news detection:

1. The Support Vector Classifier (SVM) model achieved the highest accuracy of approximately 0.95998, indicating that it correctly classified 95.998% of the fake news instances in the validation set.
2. The Gradient Boosting Classifier (GBC) model demonstrates a slightly lower accuracy of approximately 0.95872 compared to SVM.
3. In terms of F1-Score, which considers both precision and recall, the SVM model achieves an F1-Score of approximately 0.95984, outperforming GBC, which has an F1-Score of approximately 0.95858.
4. The Random Forest Classifier (RFC) model exhibits an accuracy of approximately 0.91040 and an F1-Score of approximately 0.90980, indicating relatively lower performance compared to SVM and GBC.
5. The Multinomial Naive Bayes (MNB) model, outperformed by RFC in terms of both accuracy and F1-Score. MNB has an accuracy of approximately 0.91140 and an F1-Score of approximately 0.91021.

To summarize, based on the provided validation results, the SVM model demonstrates the highest accuracy and a strong F1-Score, indicating its effectiveness in detecting fake news. The GBC model follows closely in terms of accuracy but has a slightly lower F1-Score. The RFC model performs relatively poorly compared to the other models, including MNB, which shows a comparable performance to RFC.

Overall:

Overall, your machine learning-based fake news detection models show promising results. They are able to achieve reasonably high accuracy, F1-scores, recall, and precision on the validation set. This suggests that your models have the potential to effectively distinguish between fake and real news articles. However, it is important to note that the performance of your models may vary depending on the specific dataset and the chosen machine learning algorithms and techniques. Further evaluation and testing on different datasets would be beneficial to validate the generalizability and robustness of your models in the field of machine learning-based fake news detection.