**Code Report:**

**Libraries & Tools Used:**

* Data Manipulation: pandas, numpy
* Visualization: matplotlib, seaborn
* Text Preprocessing: nltk (lemmatization, stemming, stopword removal)
* Feature Extraction: TfidfVectorizer
* Machine Learning:
  + Classifiers: MultinomialNB, SVC, RandomForestClassifier, XGBClassifier, LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier, AdaBoostClassifier, GradientBoostingClassifier
  + Utilities: train\_test\_split, LabelEncoder, cross\_val\_score
* Evaluation Metrics: accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_matrix

**Dataset Used:**

The Spam-ham dataset provided from Kaggle was used for this project (spam.csv), <https://www.kaggle.com/datasets/tmehul/spamcsv?resource=download>.

**Data Preprocessing:**

* Dataset was loaded using pandas
* We used various ways to clean up the text, like
  + Removing non-alphabetical characters.
  + Converting to lowercase.
  + Removing stopwords.
  + Stemming and Lemmatizing tokens.
* Dataset was shrunk, the remove any parts that were only ham. Like, removing all data that was over 350 characters, over 50 words, and over 10 sentences because all of them were ham.

**Models Trained & Classifiers used:**

MultinomialNB: Simple and fast but struggles with non-linear relationships.

LogisticRegression: Average performance, but limited to complex patterns.

SVC & Linear SVC: Good for smaller datasets, but long loading times because its computationally expensive.

RandomForest: Good for handling overfitting, but it works slower on big datasets.

XGBClassifier: Gives high accuracy but takes longer to train.

DecisionTree: Easy to understand and interpret, but runs into issues like overfitting a lot.

KNeighbours: Simple but sensitive to irrelevant features.

Adaboost & GradientBoosting: Good but usually requires more tuning for higher accuracy.

**Models were evaluated using:**

* Accuracy
* Precision
* Recall
* F1 Score
* Confusion Matrix
* Cross-validation Scores

**Findings:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Precision | Recall | F1 Score | Test Accuracy | Train Accuracy |
| Naive Bayes | 1.000000 | 0.787671 | 0.881226 | 0.977148 | 0.992911 |
| Random Forest | 1.000000 | 0.835616 | 0.910448 | 0.977148 | 0.992911 |
| K-Nearest Neighbors | 1.000000 | 0.390411 | 0.561576 | 0.977148 | 0.992911 |
| SVC | 0.992063 | 0.856164 | 0.919118 | 0.977148 | 0.992911 |
| Linear SVC | 0.984615 | 0.876712 | 0.927536 | 0.977148 | 0.992911 |
| Logistic Regression | 0.982456 | 0.767123 | 0.861538 | 0.977148 | 0.992911 |
| Gradient Boosting | 0.974790 | 0.794521 | 0.875472 | 0.977148 | 0.992911 |
| AdaBoost | 0.864583 | 0.568493 | 0.685950 | 0.977148 | 0.992911 |
| Decision Tree | 0.866197 | 0.842466 | 0.854167 | 0.977148 | 0.992911 |
| XGBoost | 0.961832 | 0.863014 | 0.909747 | 0.977148 | 0.992911 |

Best Model Overall: LinearSVC had the highest F1 Score (0.9275) and strong balance between precision and recall.

Most Precise Models: Naive Bayes, Random Forest, and KNN achieved perfect precision (1.0), meaning no false positives, but Naive Bayes and KNN struggled with recall.

Worst Performing: K-Nearest Neighbors had very poor recall (0.39) making it unreliable despite perfect precision.

XGBoost & Random Forest: Both performed excellently with a good balance of precision and recall.

AdaBoost & Decision Tree: Underperformed compared to other ensemble models, especially on recall.