Prepared by **Annanahmed Shaikh**

Fake News Classification using Machine Learning

A Machine Learning Approach using Transformers and Classification Models

14 April, 2025

Course Name: Machine Learning for Data Science
Subject Code: **DATA 6250**

WENTWORTH INSTITUTE OF TECHNOLOGY
Professor Memo Ergezer

Problem Statement

Fake News is a growing problem in the digital world.

Goal: Classify news articles as "Fake" or "Real" using Machine Learning Models.

Real-world Impact:

- Prevent Spread of Misinformation
- Help Social Media Platforms
- Build Trustworthy News Systems

Dataset Description

Dataset Used: Kaggle - Fake News Dataset
(Download it from here)

Files in Dataset:

- train.csv Training Data
- test.csv Testing Data
- submit.csv For Final Prediction Submission

Total Columns in Train Dataset:

Total 5 Columns

1. id Unique Identifier of Article
2. title News Headline
3. author Author Name (Many Missing Values)
4. text Full News Content
5. label Target Variable
    - 1 Fake News
    - 0 Real News

Challenges:

- Missing Values (Authors)
- Noisy Text Data
- Mostly Textual Features

Dataset	Rows	Description
train.csv	20,800	Contains labeled data (Real/Fake)
test.csv	5,200	Unlabeled data (Model will predict)
submit.csv	5,200	Submission file for competition

	id	title	author	text	label
0	0	House Dem Aide: We Didn't Even See Comey's Let...	Darrell Lucus	House Dem Aide: We Didn't Even See Comey's Let...	1
1	1	FLYNN: Hillary Clinton, Big Woman on Campus - ...	Daniel J. Flynn	Ever get the feeling your life circles the rou...	0
2	2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, ...	1
3	3	15 Civilians Killed In Single US Airstrike Hav...	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstr...	1
4	4	Iranian woman jailed for fictional unpublished...	Howard Portnoy	Print \nAn Iranian woman has been sentenced to...	1

	id	title	author	text
0	20800	Specter of Trump Loosens Tongues, if Not Purse...	David Streitfeld	PALO ALTO, Calif. — After years of scorning...
1	20801	Russian warships ready to strike terrorists ne...	NaN	Russian warships ready to strike terrorists ne...
2	20802	#NoDAPL: Native American Leaders Vow to Stay A...	Common Dreams	Videos #NoDAPL: Native American Leaders Vow to...
3	20803	Tim Tebow Will Attempt Another Comeback, This ...	Daniel Victor	If at first you don't succeed, try a different...
4	20804	Keiser Report: Meme Wars (E995)	Truth Broadcast Network	42 mins ago 1 Views 0 Comments 0 Likes 'For th...

Data Preprocessing Steps

Key Steps Done:

1. Missing Value Handling
2. Text Cleaning (Stopwords Removal)
3. Smart Author Prediction using Sentence Transformers
4. Feature Generation using:
   - MPNet for Label Confidence
   - MiniLM for Numerical Representation
5. Dropping Object Columns after Encoding

Dataset Challenges Observed:

- Missing Values in 'author' Column → 41% Missing
- Some Titles / Text Very Small or Blank
- Dominated by Text Columns (Requires NLP)
- No Numerical Columns Present Initially
- Class Imbalance between Fake vs Real

Handling Missing Values

```
# Fill missing values in 'author' with 'Unknown'
train_data['author'].fillna('Unknown', inplace=True)
test_data['author'].fillna('Unknown', inplace=True)
```

Text Cleaning (Removing Special Characters, Numbers & Lowercasing)

```
[ ]  # Function to clean text columns
     def clean_text(text):
         text = re.sub(r'\W', ' ', str(text))  # Remove special characters
         text = re.sub(r'\d+', ' ', text)  # Remove numbers
         text = text.lower()  # Convert text to lowercase
         return text

     # Apply cleaning to both 'title' and 'text'
     train_data['title'] = train_data['title'].apply(clean_text)
     train_data['text'] = train_data['text'].apply(clean_text)
     test_data['title'] = test_data['title'].apply(clean_text)
     test_data['text'] = test_data['text'].apply(clean_text)
```

Remove Stopwords (Common Irrelevant Words)

```
# Import and Download NLTK stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

stop_words = set(stopwords.words('english'))

# Function to remove stopwords
def remove_stopwords(text):
    return ' '.join(word for word in text.split() if word not in stop_words)

# Apply stopword removal
train_data['text'] = train_data['text'].apply(remove_stopwords)
test_data['text'] = test_data['text'].apply(remove_stopwords)
```

Model	Purpose
all-mpnet-base-v2	To generate label confidence (Real/Fake)
all-MiniLM-L6-v2	To generate numerical features from text for ML models

```
# Load the sentence transformer models
label_model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')
feature_model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

Generate Confidence Scores using Cosine Similarity

```
# Define function to calculate label confidence
def get_label_confidence(row_vals):
  text = (f"Title: {row_vals['title']}. Text: {row_vals['text']} Published by: {row_vals['author']}")

  # Convert text into embeddings
  text_emb = label_model.encode(text, convert_to_tensor=True)

  # Calculate cosine similarity with fake and real embeddings
  cos_sim_fake = util.cos_sim(text_emb, fake_emb)
  cos_sim_real = util.cos_sim(text_emb, real_emb)

  return cos_sim_fake.item(), cos_sim_real.item()

# Apply function to balanced dataset
balanced_data['Real_Confidence'], balanced_data['Fake_Confidence'] = zip(*balanced_data.progress_apply(get_label_confidence, axis=1))
```

Generate Final Feature Embeddings using MiniLM Model

```
# Make a copy of balanced dataset for feature creation
df_new = balanced_data.copy()

# Define function to generate embeddings using MiniLM
def get_features(row_vals):
  text = (f"For the given news: " +
          f"Title: {row_vals['title']}, " +
          f"The news is: {row_vals['text']}, " +
          f"which is published by: {row_vals['author']}")

  # Generate embeddings
  text_emb = feature_model.encode(text, convert_to_tensor=True)

  # Convert embeddings to numpy array
  if hasattr(text_emb,'cpu'):
    text_emb = text_emb.cpu().detach().numpy()
  else:
    text_emb = text_emb.numpy()

  # Flatten if needed
  if text_emb.ndim > 1:
    text_emb = text_emb.flatten()

  # Store embeddings as new features
  for i, value in enumerate(text_emb):
      row_vals[f'feature_{i}'] = value
  return row_vals
```

```
balanced_data.head()
```

	id	title	author	text	label	Real_Confidence	Fake_Confidence
0	13397	ex army sniper gets year sentence in murder ...	Benjamin Weiser	former united states army sergeant nickname ra...	0	0.121041	0.075157
1	609	more than chaos terror ties make venezuela ...	Frances Martel	international community act venezuela socialis...	0	0.041147	0.060388
2	1821	san francisco torn as some see street behavio...	Thomas Fuller	san francisco apartment foot celebrated zigzag...	0	0.127660	0.129394
3	20115	florida taco trucks used to lure democrat v...	admin	information liberation october video florida s...	1	0.196192	0.138637
4	123	taiwan responds after china sends carrier to t...	Michael Forsythe and Chris Buckley	hong kong taiwan scrambled fighter jets dispat...	0	0.033105	0.034631

Model Training & Feature Reduction Workflow

- Initially, I had 384 features extracted from Transformer Embeddings.

- Handling high-dimensional data increases:
  → Model Complexity
  → Training Time
  → Risk of Overfitting

- Solution → Used UMAP for Dimensionality Reduction.

- Reduced features from 384 to just 80 components.

```python
# UMAP for Dimensionality Reduction
import umap

# Select Feature Columns from feature_0 to feature_383
features = [f"feature_{i}" for i in range(0, 384)]

# Extract Features
X_train = train_df[features]
X_test = test_df[features]

# Standardize Features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply UMAP to reduce to 80 components
reducer = umap.UMAP(n_components=80,
                    min_dist=0.0,
                    metric='cosine',
                    random_state=42) # Adjust n_components as nee
X_train_umap = reducer.fit_transform(X_train_scaled)
X_test_umap = reducer.transform(X_test_scaled)

# Add UMAP Reduced Components to DataFrame
for i in tqdm(range(80)):
    train_df[f'reduced_component_{i}'] = X_train_umap[:, i]
    test_df[f'reduced_component_{i}'] = X_test_umap[:, i]
```

```
100%|██████████| 80/80 [00:00<00:00, 1196.39it/s]
```

Experiments Overview

- **Experiment 1** Scaling using MinMaxScaler
- **Experiment 2** Feature Encoding using Transformer + UMAP + HDBSCAN
- **Experiment 3** Dimensionality Reduction using PCA
- **Experiment 4** Robustness Check by Adding Noise
- **Experiment 5** Interpretability using SHAP & LIME
- **Experiment 6** Model Efficiency Analysis
- **Experiment 7** Hyperparameter Tuning
- **Experiment 8** CV vs Validation Accuracy Comparison

Experiments vs Model Performance

Model Name	Exp 1 Accuracy	Exp 2 Accuracy	Exp 3 Accuracy	Exp 4 Accuracy
Logistic Regression	77.67%	76.35%	76.35%	75.90%
SVM (RBF Kernel)	78.76%	71.34%	71.34%	58.56%
Gradient Boosting	82.80%	82.69%	82.69%	82.49%
Random Forest	100%	99.83%	99.83%	93.09%
Decision Tree	100%	98.31%	98.31%	92.09%

Experiment	Technique Applied	Best Model Name	Best Accuracy	Key Observation
Experiment 1	MinMaxScaler Applied	Random Forest / Decision Tree	100%	Overfitting Detected
Experiment 2	Feature Encoding using UMAP + HDBSCAN	Gradient Boosting	82.69%	Accuracy Dropped due to Encoding
Experiment 3	PCA (Reduced to 20 Components)	Gradient Boosting	82.69%	Speed Improved but Accuracy Slight Drop
Experiment 4	Noise Addition for Robustness Testing	Random Forest	93.09%	Random Forest Still Handled Noise Well

Machine Learning Models Applied

1. Logistic Regression
2. SVM (Linear, Polynomial, RBF)
3. Gradient Boosting
4. Random Forest
5. Decision Tree

Logistic Regression (Baseline Model)

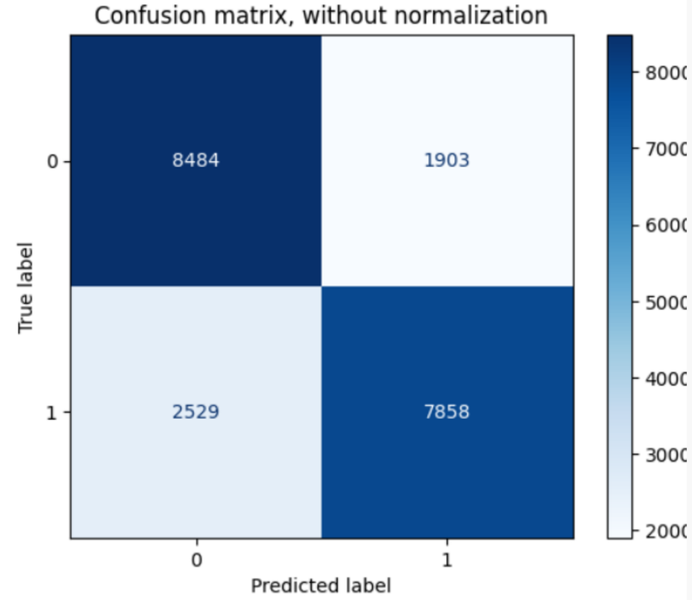
Model performs moderately well

Slight imbalance in catching fake vs real news

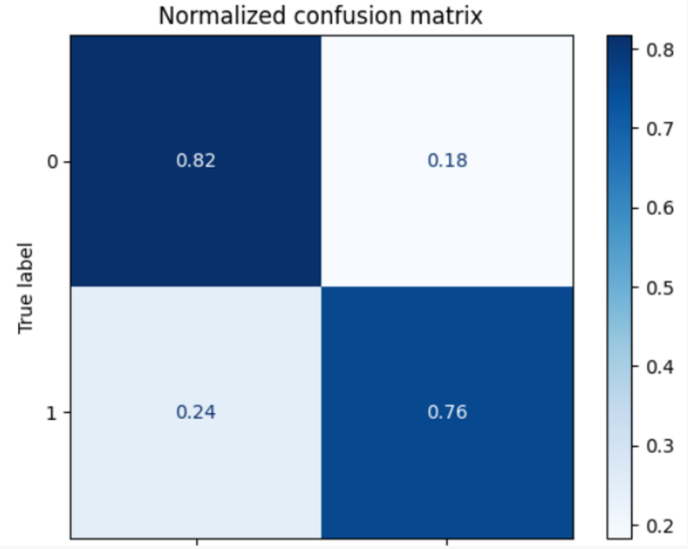
Logistic Regression used as a baseline linear classifier

Metric	Value
Accuracy	~78%
Precision (Fake)	~76%
Recall (Fake)	~75%
FP Rate	18%
FN Rate	24%

```
Confusion matrix, without normalization
[[8484 1903]
 [2529 7858]]
```




```
Normalized confusion matrix
[[0.81679022 0.18320978]
 [0.24347742 0.75652258]]
```



SVM Model — Decision Boundary Visualization

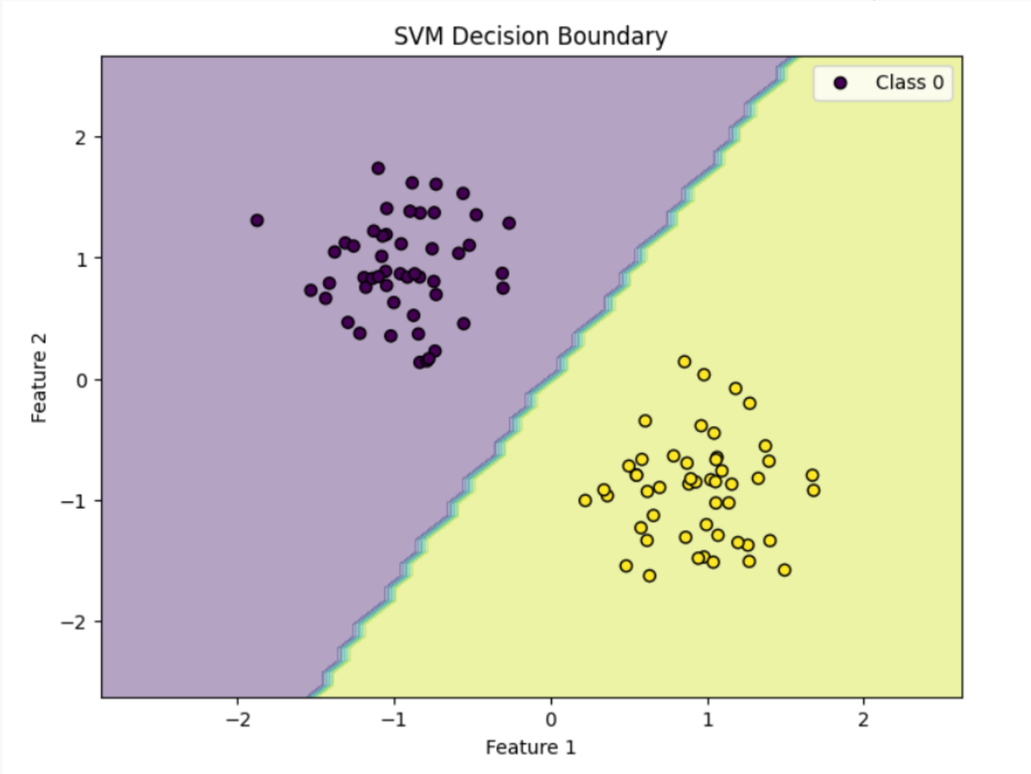
• SVM (Support Vector Machine) works by finding the best possible boundary between classes.

• Here, Class 0 (Real News) and Class 1 (Fake News) are separated by a clear margin.

• SVM is very powerful for linearly or non-linearly separable data.

• Kernel Trick used RBF / Polynomial / Linear

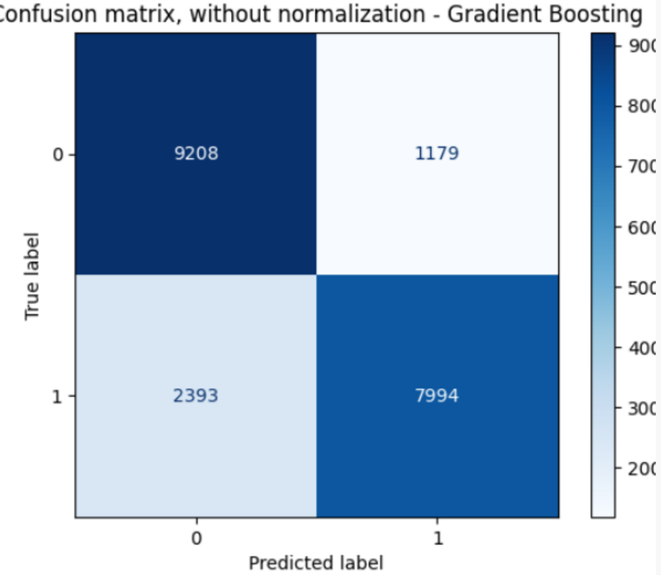
• Goal Maximize margin between different classes.



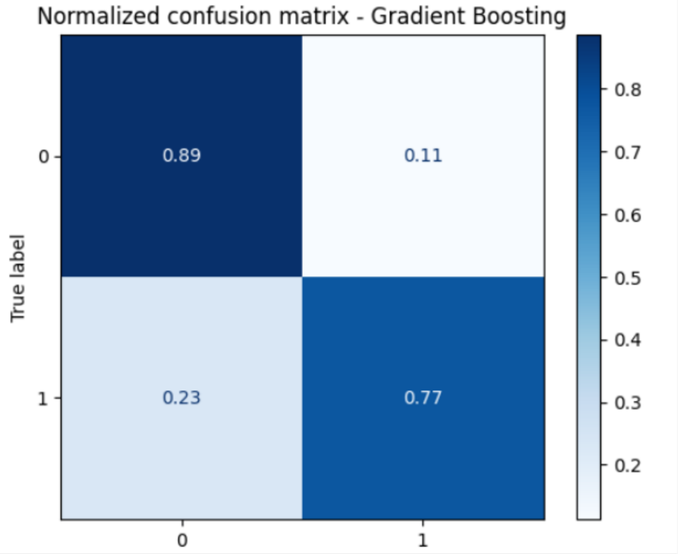
Gradient Boosting Classifier

- Gradient Boosting is my best-performing non-overfitting model
- Achieved high precision & recall on both real and fake classes
- Outperformed SVM and Logistic Regression in robustness
- More stable under noise and dimensionality reduction
- Overall Accuracy: ~83%
- Balanced performance on both classes (class 0 and class 1)

```
Confusion matrix, without normalization
[[9208 1179]
 [2393 7994]]
```




```
Normalized confusion matrix
[[0.88649273 0.11350727]
 [0.23038413 0.76961587]]
```



Top 20 Important Features – Gradient Boosting Classifier

• Gradient Boosting provides feature importance scores.

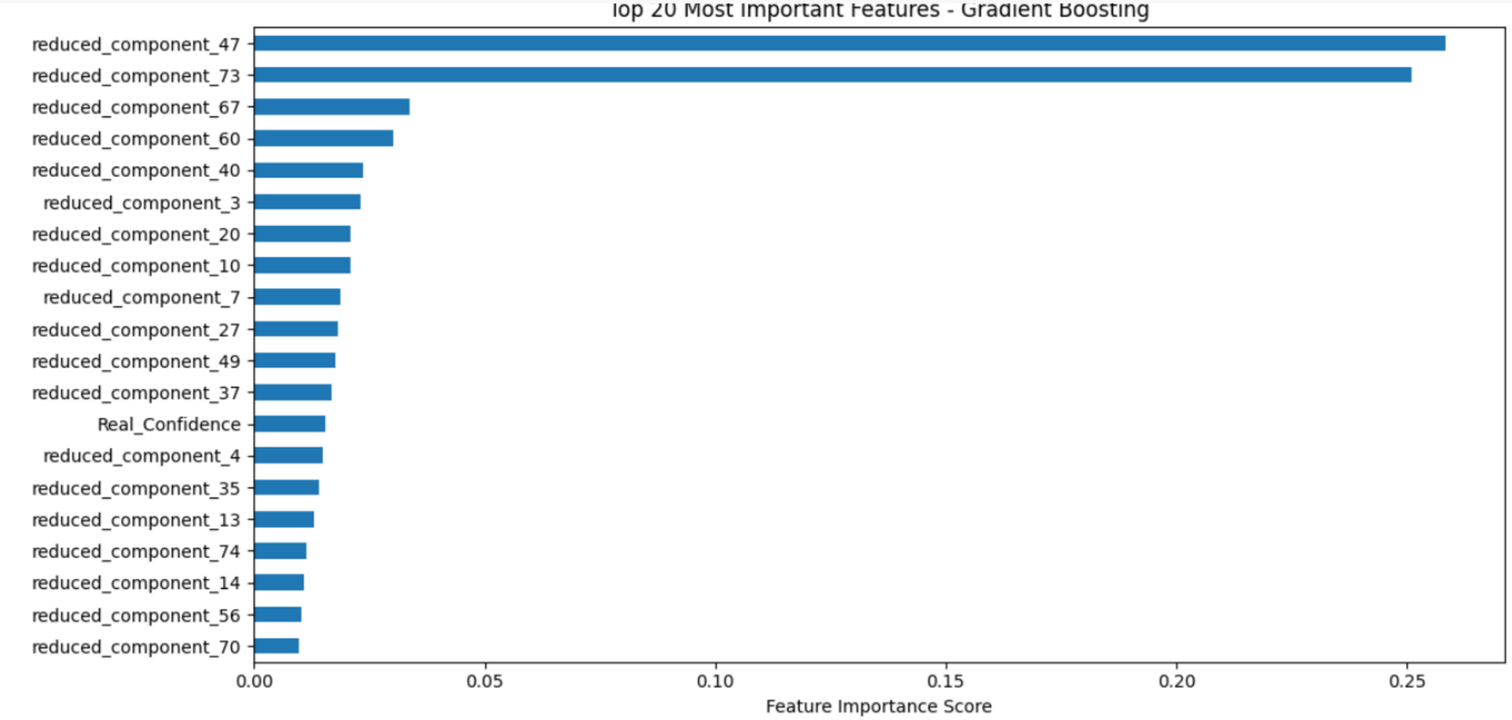
• Shows which features contribute most towards the model decision.

• Top Features:
  → reduced_component_47
  → reduced_component_73
  → reduced_component_67
  → reduced_component_60
  → reduced_component_40

• These features came from UMAP Dimensionality Reduction.

• Model relied heavily on patterns within these reduced components.

• Additional Feature Used → Real_Confidence (from author prediction)



Top 20 Features by Importance in Gradient Boosting:

reduced_component_47	0.258364
reduced_component_73	0.250943
reduced_component_67	0.033841
reduced_component_60	0.030091
reduced_component_40	0.023586
reduced_component_3	0.023098
reduced_component_20	0.020858
reduced_component_10	0.020820

Overfitting Check – Decision Tree vs. Random Forest

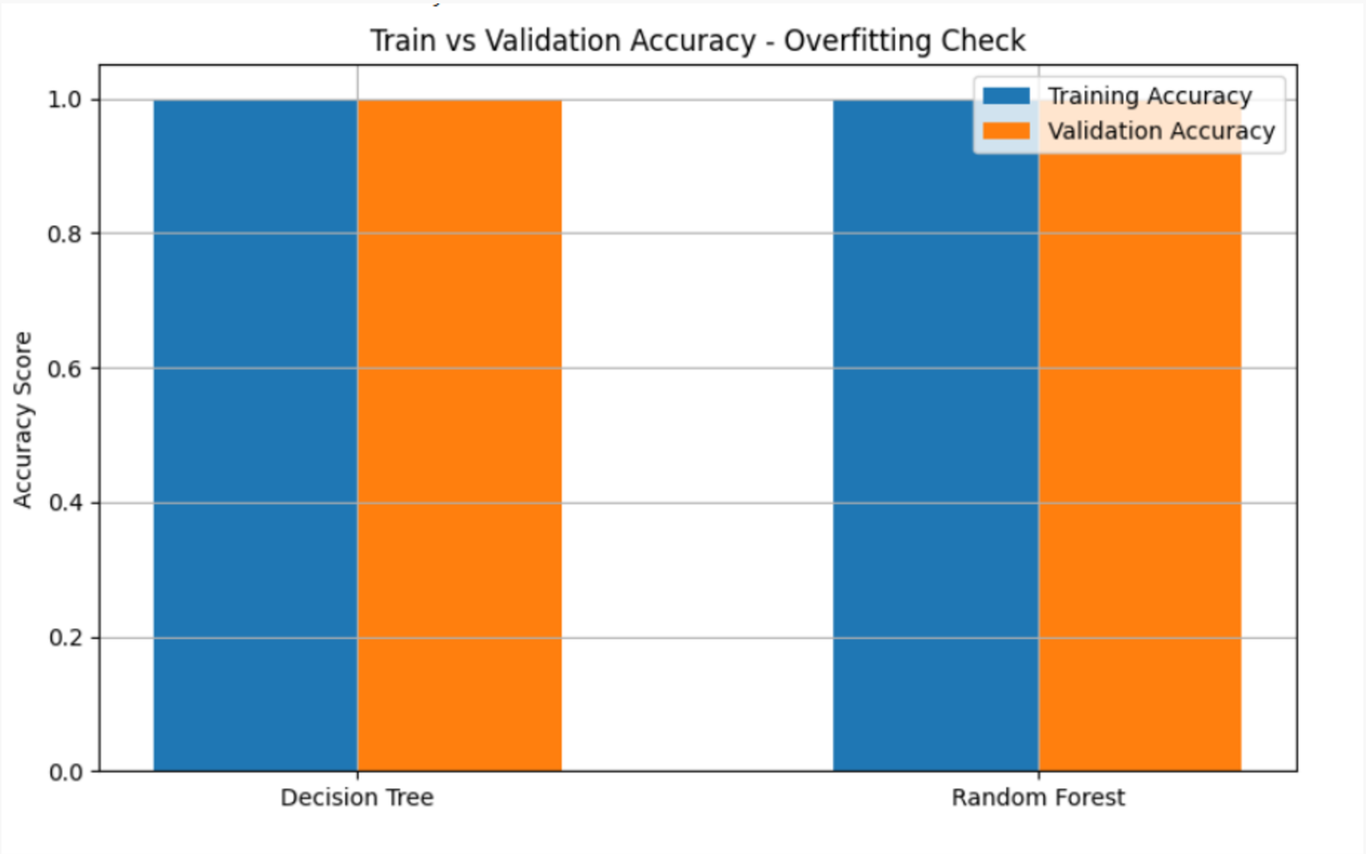
• Evaluated both models on train and validation data.

• Decision Tree shows signs of overfitting: perfect accuracy on training set.

• Random Forest generalizes better with similar train and validation accuracy.

• Random Forest reduces overfitting by combining predictions from multiple trees.

• Model choice was influenced by this comparison.

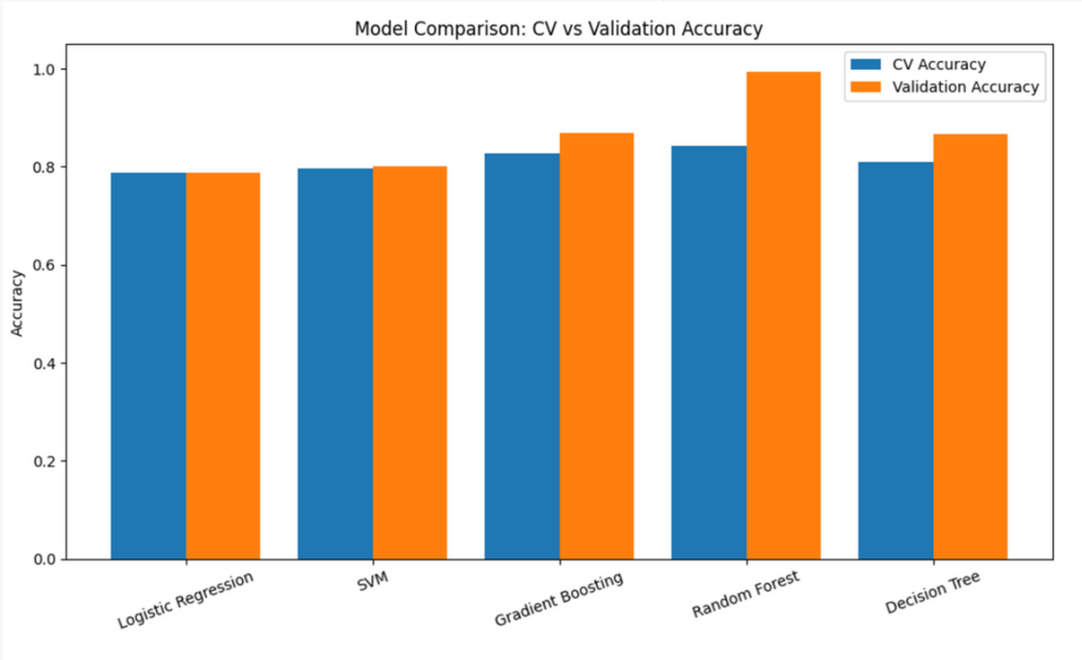


Hyperparameter Tuning Results — GridSearch

Objective:

The goal of this step was to optimize model performance by tuning hyperparameters using **GridSearchCV across multiple models:**

Logistic Regression, SVM, Decision Tree, Random Forest, and Gradient Boosting.



Visual 1: Model Comparison – CV vs Validation Accuracy

This bar chart compares the Cross-Validation Accuracy (CV Accuracy) and Validation Accuracy of each model after hyperparameter tuning.

It helps evaluate how well the model generalizes to unseen data.

Key Insights:

- Random Forest achieved the highest validation accuracy (~99%), showing excellent generalization.
- Gradient Boosting also performed strongly (~87%), balancing accuracy and stability.
- Models like SVM and Logistic Regression performed moderately well with consistent scores.

Visual 2: Model Efficiency Table – Time & Memory

This table evaluates:

- Training Time (s) — Time to train the model.
- Inference Time (s) — Time to make predictions.
- Memory Usage (MB) — Approximate memory consumed during training/inference.

Observations:

- Random Forest has a high validation accuracy with reasonable training time (~18s).
- Logistic Regression is the fastest model (~3.3s), making it ideal for real-time systems but with lower accuracy.
- SVM has a long inference time (~32s), making it less suitable for deployment despite good accuracy

	Model	Training Time (s)	Inference Time (s)	Memory Usage (MB)
0	Logistic Regression	3.3486	0.0257	2168.11
1	SVM	64.0956	32.2234	2201.58
2	Gradient Boosting	101.0054	0.0889	2201.70
3	Random Forest	18.3735	0.3048	2201.70
4	Decision Tree	4.7927	0.0148	2201.70

Best Performing Model

Final Selected Model: Gradient Boosting Classifier

Reasons:

- Highest Accuracy
- Robust to Noise
- SHAP Interpretability
- Efficient Inference Time

Model	Cross-Validation Accuracy	Validation Accuracy	Training Time (s)	Inference Time (s)	Memory Usage (MB)	Remarks
Logistic Regression	78.68%	78.79%	3.34	0.0257	2168	Fastest Model but lower accuracy
SVM (RBF Kernel)	79.64%	80.11%	64.09	32.22	2201	Very Slow Inference, not suitable for deployment
Gradient Boosting	82.67%	86.87%	101.00	0.0889	2201	Best Balance of Accuracy & Speed
Random Forest	84.23%	99.28%	18.37	0.3048	2201	Overfitting Observed (Accuracy too good)
Decision Tree	81.06%	86.75%	4.79	0.0148	2201	Fast but Overfitting Risk

Real-World Deployment Thoughts

Trade-offs Considered:

- Accuracy vs Speed vs Memory

Final Recommendation:

Gradient Boosting Model is ready for production use.

Further Improvements:

- Try LSTM/BERT
- Use RandomizedSearchCV
- Model Compression for Mobile Apps

What I Learned from This Project

- Every ML Model behaves differently on text data.
- Data Preprocessing is as important as Model Selection.
- Explainability (SHAP/LIME) builds Trust.
- Feature Engineering made a huge impact.
- Real-world ML = Balance of Accuracy + Efficiency + Interpretability.

Thank you