PROJECT REPORT

Project Title - Deception Detection Model

Course Code: CS1138
Course Title: Machine Learning

Course Faculty: Dr. Arpan Gupta

Project By: Group 14

Group Members:-
Mayur Soni (2022Btech055)
Dev Rishi Verma (2022Btech028)
Karan Kumar (2022Btech046)
Abhinav Jajoo (2022Btech120)

TABLE OF CONTENTS

INTRODUCTION

Deception, the act of concealing or altering the truth, pervades all aspects of human interaction and has serious consequences. Detecting deception is critical for assuring security, avoiding fraud, and preserving confidence. Traditional methods of deception detection, such as polygraph testing, have limitations that reduce their usefulness. This research is motivated by the need for more accurate and reliable deception detection systems that can use new technologies to overcome these constraints and improve security measures.

The Significance of Deception Detection

Deception detection is critical in many fields, including law enforcement, business security, and personal interactions. According to statistics, deception costs the global economy billions of dollars each year due to fraud, espionage, and disinformation. For example, the Association of Certified Fraud Examiners (ACFE) discovered that fraud costs organisations around 5% of their yearly profits. Furthermore, fraudulent acts, such as fabricating credentials or presenting misleading information, can have major implications, including legal ramifications and reputational damage.

Challenges with Old Technologies

Traditional deception detection methods, like polygraph tests, have fundamental limitations that make them less useful in modern environments. Exams with polygraphs rely on physiological responses, including heart rate and sweating, which can be affected by non-deceptive factors like anxiety and lead to false positives or negatives. Because these tests necessitate precise calibration, experienced operators, and close interaction with the individual, they are time-consuming and prone to human error.

Motivation for Advanced Deception Detection

Our project is motivated by the shortcomings of conventional deception detection techniques, which often produce inconsistent outcomes and are not suited to today's complex security challenges. To address these limitations, we are integrating techs such as ML and multimodal data processing. Specifically, our approach involves analyzing facial features, vocal cues, and linguistic patterns extracted from transcripts. By combining these modalities, we aim to develop a more precise and effective fraud detection system. Our objective is to provide a robust framework capable of accurately distinguishing between dishonest and honest

behaviour, thereby enhancing security protocols and decision-making procedures in various contexts.

PROBLEM STATEMENT

The project aims to enhance deception detection by leveraging a multimodal approach that integrates various sources of information without explicitly focusing on individual modalities. The primary challenge is to develop a comprehensive deception detection system that effectively captures the complex nature of deceptive behaviour without being limited by the constraints of single-modal approaches.

The key challenges and objectives of this project include:

1. Developing deception detection models that can effectively analyze facial features, transcripts, and vocal features to identify patterns of deceptive behaviour.
2. Assessing the efficacy of the multimodal approach using the MU3D dataset, a well-established benchmark for deception detection research.
3. Identifying the most informative features and their relative importance in distinguishing between deceptive and truthful behaviours, provides insights into the underlying mechanisms of deception.
4. resolving potential dataset bias concerns and guaranteeing that the deception detection system can be applied to a variety of real-world situations.
5. Exploring the practical applications and implications of the developed deception detection system, particularly in domains such as law enforcement, security, and interpersonal communication.

The project's goal is to improve deception detection research and create more precise and trustworthy instruments for recognizing dishonest behavior in a variety of settings by tackling these issues.

LITERATURE REVIEW

Brief History Overview

The history of lie detection can be traced back to ancient civilizations, where various techniques were employed to detect deception. In ancient China around 1000 BC, the "Trial by Rice" was used, where a suspect's mouth was filled with uncooked rice, and if the rice remained dry after spitting, the suspect was considered guilty. Similarly, in ancient India, the "Trial by the Sacred Ass" was used, where a suspect's ability to pull the tail of a donkey without making a sound was seen as a sign of innocence.

Over the years, the Western world's idea of lie detection has evolved dramatically. In the nineteenth century, phrenology and graphology gained prominence, claiming to be able to discern someone's guilt based on the shapes of their skull and the characteristics of their writing. Cesare Lombroso devised the Volumetric Glove, the first scientific lie detector test, in 1895. It evaluated peripheral vasoconstriction by monitoring the amount of water displaced when a subject's hand was placed in a container of water.

- **The Era of Polygraph**

The modern polygraph apparatus was created in the early twentieth century, largely to help physicians monitor physiological changes in the body such as BP skin conductivity as a person answers a series of questions. John Larson constructed the first polygraph instrument, called as the device, while Leonard Keeler is widely regarded as the father of the contemporary polygraph. The technology detects dishonesty by analysing physiological reactions to queries, which are said to differ depending on whether a person is Truthful or lier

However, the polygraph is far from perfect and can be unreliable due to various factors. Anxiety or nervousness can cause physiological responses similar to those of a person lying, leading to false positive readings. Additionally, individuals who are aware of the polygraph's limitations can manipulate the test by using techniques to influence their physiological responses, making it difficult to accurately determine deception.

- **Recent Advancements**

Over the past several years, technological improvements have resulted in the creation of increasingly advanced lie detection tools, such as functional magnetic resonance imaging (fMRI), electroencephalography (EEG), and the facial action coding system (FACS). Furthermore, the advancement of artificial intelligence and deep learning algorithms has allowed for the creation of more precise and automatic lie detection systems capable of analysing fine details such as small variations in voice pitch, facial expressions, and body language.

Research Papers' Findings

1. **Deep Neural Networks for Lie Detection with Attention on Bio-signals**
   Authors: Ameya Rajendra Bhamare, Srinivas Katharguppe, Silviya Nancy J

Dtaa Processing

The data processing stage involved collecting and preparing the dataset for analysis. This included identifying facial landmarks in real-time during interviews, extracting Eyes, brows, lips, jawline, and nose are among the face traits examined, as with speech patterns such as pitch, frequency, and loudness.The dataset consists of 4000 data points from 25 volunteers, with a binary target variable indicating lie (0) or truth (1). MP4 files were separated into video and audio components. The video component was used to extract 136 facial landmark values per frame using dlib. The audio component yielded 20 Mel Frequency Cepstral Coefficients (MFCC) features. These video and audio features were then merged, resulting in a total of 156 features.

Neural Network Architecture

This project's neural net design consists of an input layer with dimensions (156, 1), followed by two intermediary dense layers that use a ReLU activation function. The output layer consists of a single neuron with a sigmoid activation function that provides a probabilistic score ranging from 0 to 1, signifying the possibility of deceit.
The model was trained with the Adam optimisation function and the binary cross-entropy loss function. The model was trained for 30 epochs with a batch size of 16.

Proposed Systems for Detecting Lie

The proposed lie detection system leverages two key components: facial landmarks and Mel frequency cepstral coefficients (MFCC).

Facial Landmarks: The system utilizes Python's dlib library to detect facial landmarks, specifically estimating the location of 68 (x, y) coordinates that map facial features onto the interviewee's face. A bounding box around the face tracks real-time changes in facial expressions. Humans often exhibit distinct facial cues when being deceitful, such as twitching of the nose or lip movements. The algorithm is designed to detect these subtle changes in expressions, which may indicate deceptive behavior.

Mel Frequency Cepstral Coefficients (MFCC): MFCC is a feature extraction technique that mimics the human auditory system's response to sound signals. It involves processing the logarithm of the signal's estimated spectrum using the Mel scale, which closely approximates the cochlea's frequency response. Recognizing MFCC's importance in speech analysis, the proposed system extracts 20 MFCC features using the librosa library, as most important information is contained within the first few coefficients. These features capture vocal cues associated with deception.

Workflow of Lie Detector

1.

Figure 2. Creating the training dataset.

2.

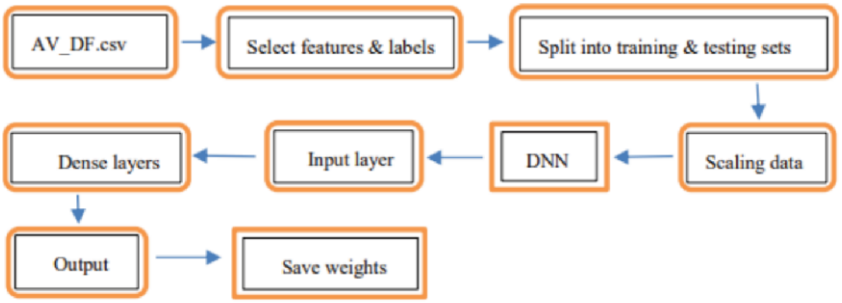


Figure 3. Training the lie detector.

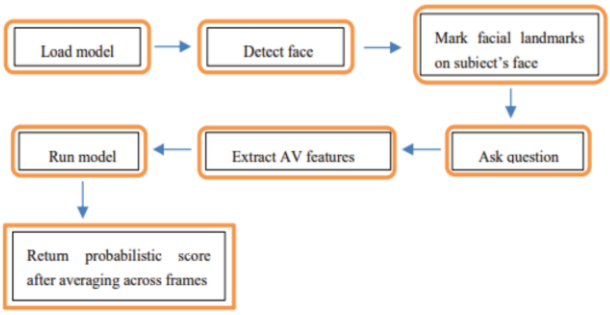


Figure 4. Testing the lie detector.

Evaluation Metrics

Model's performance in deception detection is evaluated using metrics such as accuracy, precision, recall, and F1-score: -

- Accuracy = 0.9845
- Precision = 1.0
- Recall = 0.9707
- F-1 score = 0.9853

TABLE I. CONFUSION MATRIX FOR THE LIE DETECTION SYSTEM

	Classified truth	*Classified lie*	*Total*
Actual truth	166	5	171
Actual lie	0	152	152
Total	166	157	323

2. **LieNet: A Deep Convolution Neural Network Framework for Detecting Deception** – Mohan Karnati, Ayan Seal, Anis Yazidi, Ondrej Krejcar

LieNet, a deep convolutional neural network (CNN) framework, has emerged as a powerful tool for deception detection across various data modalities. This review delves into LieNet's framework, experimental methodologies, and results, highlighting its contributions to the field of deception detection.

LieNet Framework

Architecture and Integration

LieNet operates within the Keras framework and is seamlessly integrated into the Anaconda development platform, leveraging Python's deep learning libraries. Its architecture is designed to process multimodal data, including audio, video, and EEG signals, making it versatile for capturing deceptive cues across different modalities.

Data Processing and Augmentation

Experimental settings often employ tenfold cross-validation approaches, ensuring robustness in model evaluation. Data augmentation techniques, such as horizontal flipping, Gaussian blur, and contrast normalization, enhance training data diversity, aiding LieNet in learning intricate patterns associated with deceptive behavior.

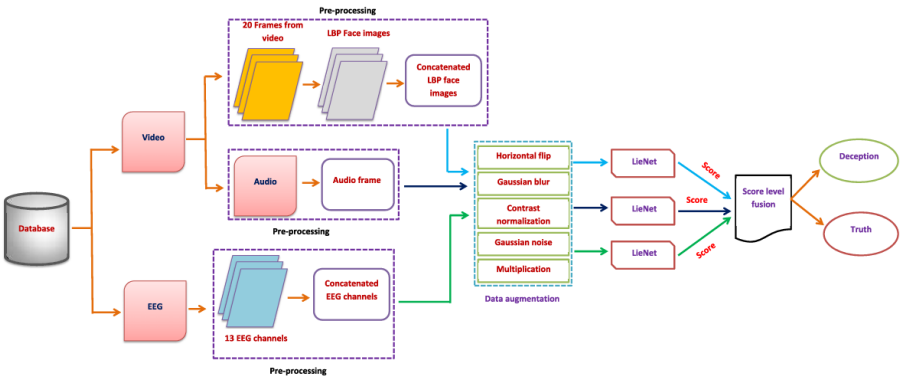


Fig 1: LieNet Architecture

Experimental Methodologies

Evaluation Metrics

LieNet's performance in deception detection is evaluated using metrics such as accuracy, precision, recall, and F1-score across visual, audio, and EEG modalities.

Classification report	BoL (Set-A)				BoL (Set-B)			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
Classification report on BoL (Set-A & Set-B)								
Deceptive	87.5	96	92	98	88	95	90	96
Truth	95	90	96	94	95	92	94	97
Average	91	96	91	97	91	95	91	95
Classification report on RL trail and MU3D								
Deceptive	97	97	97	97	98	98	98	98
Truth	97	97	97	97	98	98	98	98
Average	97	97	97	97	98	98	98	98

Fig 2: Accuracy Report

Results and Analysis

LieNet demonstrates impressive results in deception detection tasks, achieving high accuracy, precision, recall, and F1-score metrics across various modalities. Its utilization of Relu activation, Adam optimization, strategic network design, and pooling strategies significantly contributes to its success.

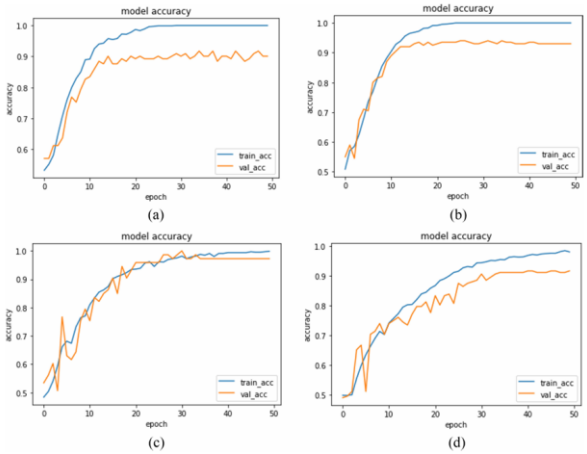


Fig 2: Accuracy on Various datasets

Conclusion

In conclusion, LieNet stands out as a robust and reliable framework for deception detection, offering a comprehensive approach to analyzing multimodal data effectively. Its framework, experimental methodologies, and results underscore its significance and advancements in the field, paving the way for more sophisticated deception detection systems.

3.Explainable Enhanced Recurrent Neural Network for lie detection using voice stress analysis -Fatma M. Talaat

Automated lie detection systems face challenges due to limited datasets for real-world testing. Traditional methods like the polygraph, voice stress analysis (VSA), and pupil dilation analysis offer insights but have limitations.

This study introduces an Enhanced Recurrent Neural Network (ERNN) with Explainable AI capabilities for lie detection. The ERNN, based on LSTM architecture and optimized with

fuzzy logic, analyzes stress-related voice patterns. Training involved a dataset of audio recordings from interviews annotated with ground truth labels for deception or stress.

The ERNN achieved a significant 97.3% accuracy in detecting stress-related voice patterns linked to deception. This advancement showcases the potential of AI-driven approaches in overcoming lie detection challenges and providing transparent, interpretable results. The integration of Explainable AI ensures trust and understanding in the model's decision-making process, contributing to the evolution of reliable lie detection methods.

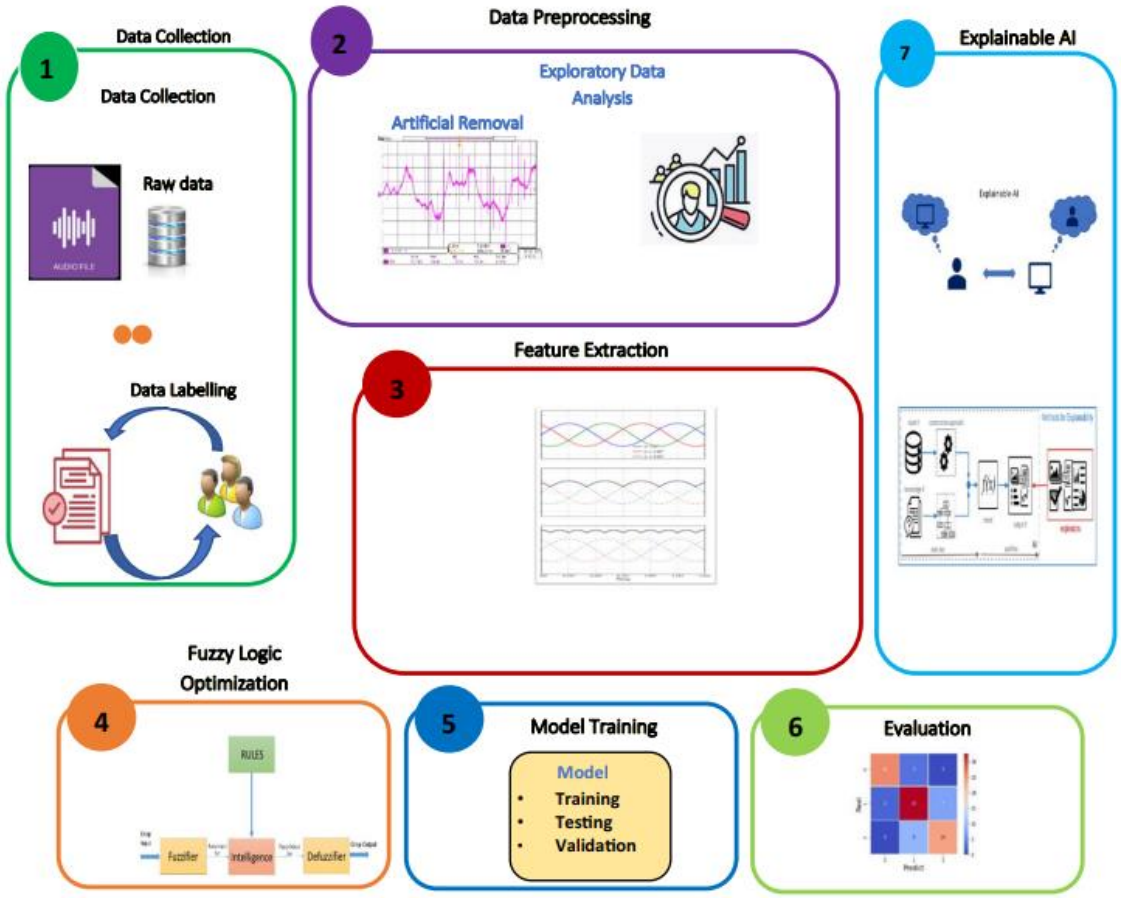


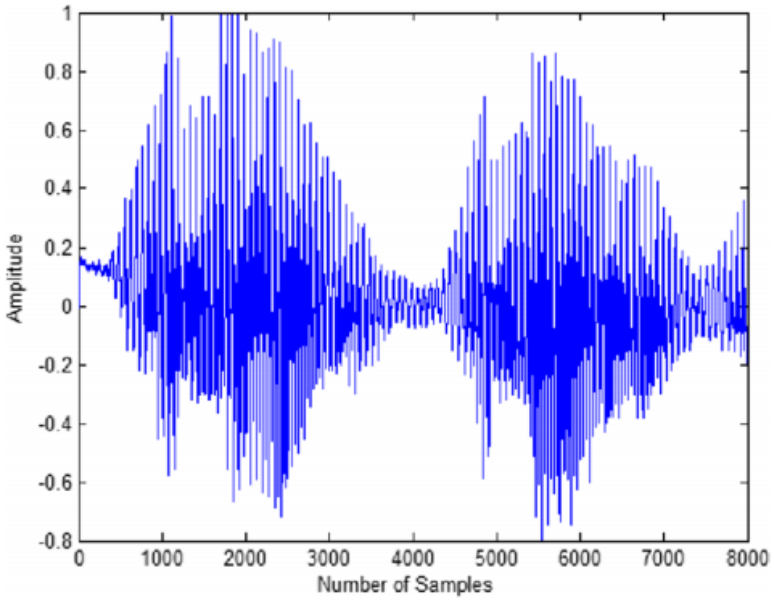
Fig. 1 The Proposed Enhanced Recurrent Neural Network (ERNN)

The **data preprocessing** for audio recordings involved using Audacity to divide interviews into individual files containing only the subject's replies. Silence at the beginning and end was removed, and each response was saved as a single file with assigned true or false labels. Algorithm 2 outlines the key steps:

1. Load audio recordings and ensure compatibility with the model's input requirements.
2. Apply a pre-emphasis filter to boost high-frequency components.
3. Divide audio into smaller segments, using a windowing function (e.g., Hamming) to reduce spectral leakage.
4. Compute the Fourier transform of each segment for spectral representation.
5. Convert spectral representation to Mel-frequency cepstral coefficient (MFCC) representation, a common feature for speech processing.
6. Normalize MFCCs for each feature across all segments.

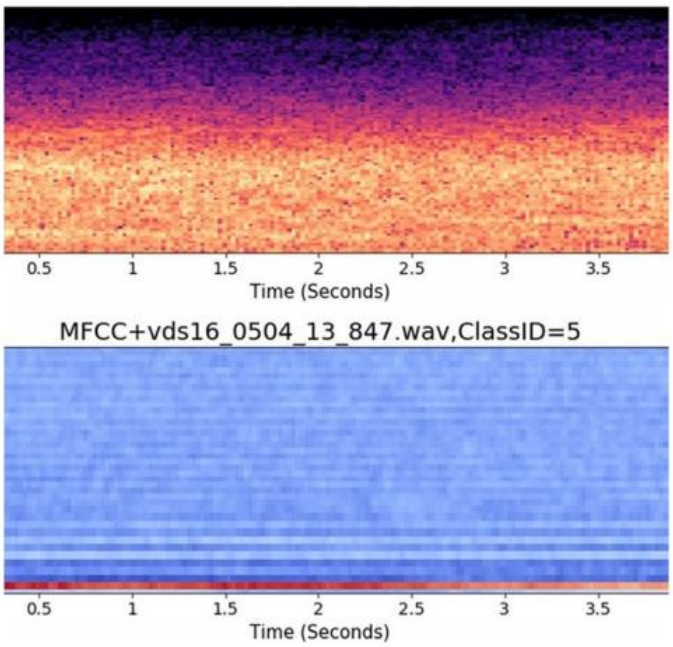
7. Save processed data in a suitable format (e.g., HDF5, TFRecord) for efficient model training.

Fig. 2 Graphical voice amplitude representation



In **audio feature extraction**, recordings are segmented and analyzed using windowing functions to reduce spectral leakage. The Fourier transform computes spectral representations, while pitch detection algorithms estimate fundamental frequencies. Additional features like pitch, intensity, duration, and formants are extracted for a comprehensive analysis. Normalization ensures consistency across segments by adjusting feature scales. These extracted and normalized features are then saved for further analysis and model training, facilitating tasks such as speech recognition and emotion detection.

MFCC Espectrogram



Result

The ERNN model exhibited superior performance with an accuracy of 97.3%, precision of 97.9%, recall of 98.1%, and an F1-score of 97.77%, showcasing its robustness in detecting stress patterns in voice recordings. Comparatively, the ANN model achieved 95.4% accuracy, the RNN model reached 96.4%, and the LSTM model achieved 96.9%. While these models demonstrated slightly lower accuracy than ERNN, they still exhibited high precision, recall, and F1-scores, validating their efficacy in stress pattern detection. Overall, these findings suggest the practical applicability of these models, particularly ERNN, in voice-based systems for stress detection.

Algorithm	Accuracy	Precision	Recall	F1-score
ERNN	97.3%	97.9%	98.1%	97.77%
ANN	95.4%	94.7%	95.2%	94.56%
RNN	96.4%	96.3%	97.1%	96.6%
LSTM	96.9%	97.0%	97.5%	96.63%

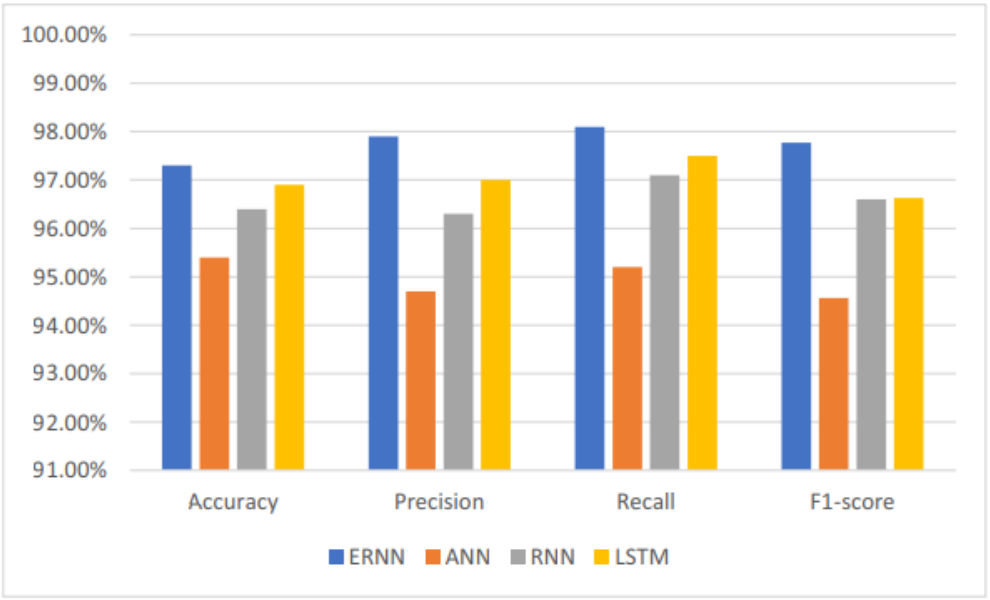


Fig. 6 Results of implementing the ERNN

Conclusion

This study introduces an ERNN-based lie detection algorithm that surpasses other neural network architectures like ANN, RNN, and LSTM with an accuracy of 97.3%. The ERNN's explainable AI features enhance transparency in decision-making, aiding stress pattern interpretation. These findings hold promise for law enforcement, intelligence agencies, and related sectors in stress detection from voice data. Future research can validate the model in practical scenarios, explore scalability with larger datasets, and integrate with OCNN, Resnet, attention mechanisms, and correlation algorithms for enhanced functionality in stress detection and related applications.

4.DeepLie: Detect Lies with Facial Expression (Computer Vision)

Kai (Jiabo) Feng jiabo@stanford.edu

Lie detection is a crucial topic in various sectors, prompting the proposal of a computer vision-based approach called DeepLie. This algorithm leverages deep learning to detect lies in video streams by analyzing universal micro-expressions associated with human emotions.

• There are various approaches to lie detection from the videos exist, including uni-modal methods like audio, text, and video (micro expressions), as well as multimodal fusion combining audio, text, and video data.

• While multimodal methods offer comprehensive information for detecting deception, real world scenarios often necessitate real-time lie detection. Hence, this paper mainly focus on micro-expression-based visual-only methods .

Methodology Used

Data Pre-processing:

• Read video data frame by frame and detect human faces using OpenCV.

• Crop the faces to include only the primary face in each frame.

• Convert the cropped facial images to grayscale and resize them to a standard 48x48 pixel format, storing them as numpy arrays.

• Split the dataset into development (dev) and test sets in a 90-10 ratio. Further split the dev set into training and validation sets using another 90–10 ratio.

Facial Expression Recognition:

• Use a pre-trained CNN model trained on the FER-2013 dataset to convert facial images into encoding vectors.

• The CNN model consists of 8 convolutional layers, intermediate layers for max-pooling, batch normalization, dropouts, and 4 fully connected layers with softmax activation at the output.

• Initially, a pre-built model with 66.4% accuracy on the test dataset is used, which is considered a good baseline performance.

Embedding Model and Triplet Loss:

• Construct an embedding model using a two-layer gated recurrent unit (GRU) network to capture complex patterns of facial expressions and micro-movements.

• Utilize a Siamese network architecture with triplet loss to optimize the embedding model weights.

• Triplet loss function ensures that embedding vectors of the same category (truth/lie) videos are closer together while different categories are farther apart in the embedding space.

• The triplet loss formula ensures this distance optimization with a margin α.

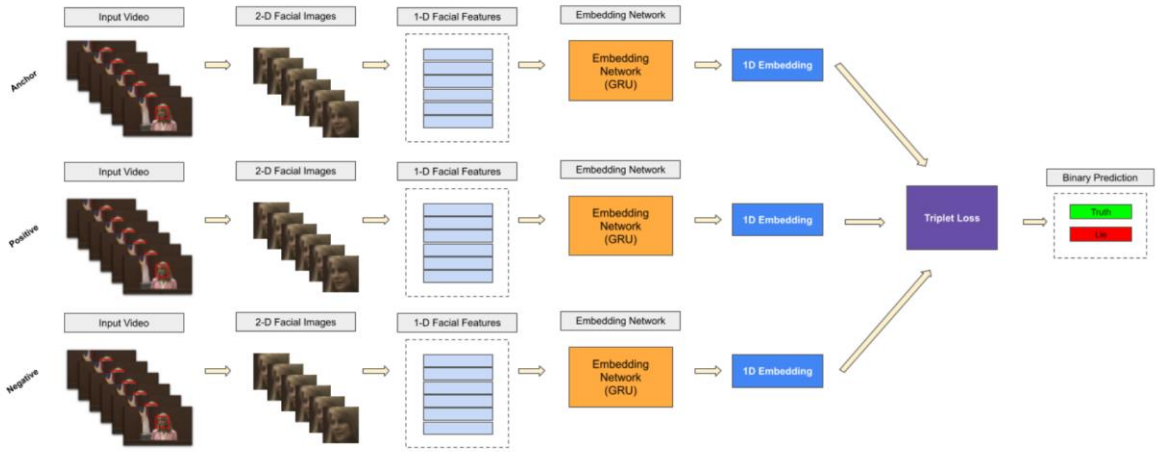
• The overall loss function is the sum of triplet losses over the training examples.

Training and Optimization:

• Use early stopping with a patience of 10 epochs to avoid overfitting and restore the best weights during training.

• Efficiently train the model without iterating through all possible triplets by leveraging the advantages of the Siamese network and GRU-based RNN's ability to detect recurring patterns.

Figure 1: Model architecture



Experiments/Results/Discussion

1. Baseline Models:

• Two initial classification networks were trained:

• A basic 5-layer deep neural network achieved 72.73% validation accuracy.

• A 3-dimensional CNN model did not significantly improve performance over the linear model, reaching around 70% validation accuracy.

2. GRU/RNN Model:

• A GRU/RNN model with 2 layers achieved 81.82% CCR on the validation set and 94.31% on the training set, showing promise in capturing temporal patterns in the data.

3. DeepLie Model with Siamese Network:

• Addressing the limited data issue, the DeepLie model using a Siamese network with triplet loss achieved 81.82% accuracy on the validation set and 100% CCR with leave-one-out cross-validation.

• The training time was efficient, stopping automatically after around 50 epochs using early stopping mechanics.

4. Discussion on Performance:

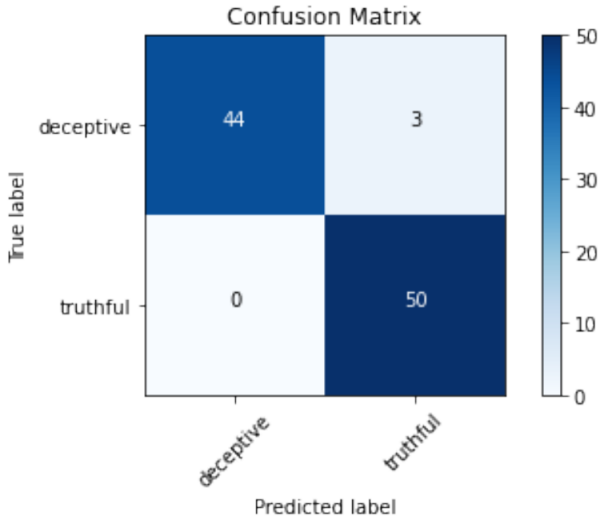
• The discussion acknowledges the challenge of limited training data for high-stake scenarios like courtrooms.

• The DeepLie model showed higher precision in detecting lies but slightly lower recall, indicating it's less likely to misclassify truthful statements as lies.

5. Hyperparameter Tuning:

• The experiment explored different values of α for triplet loss but settled on α = 0.2, which balanced model performance without risking overfitting.

Figure 4: Model architecture



Conclusion

In conclusion, the DeepLie algorithm presented an effective solution for lie detection in videos, leveraging facial expressions and micro-movements via a 2-layer GRU network and a Siamese network with triplet loss. Achieving high accuracy and avoiding overfitting, DeepLie prioritizes precision to minimize false accusations while missing some lies. Future work involves expanding the dataset to diverse scenarios, incorporating multi-modal inputs, and conducting rigorous fairness testing before real-world deployment.

METHODOLOGIES

Bidirectional Encoder Representation from Transformer (BERT)

For textual analysis tasks, we used sophisticated BERT (Bidirectional Encoder Representations from Transformers) models: DistilBERT, BERT Base, and BERT Large. Because these models were pre-trained on large text corpora, they are well known for their profound comprehension of linguistic subtleties.

Working of BERT:

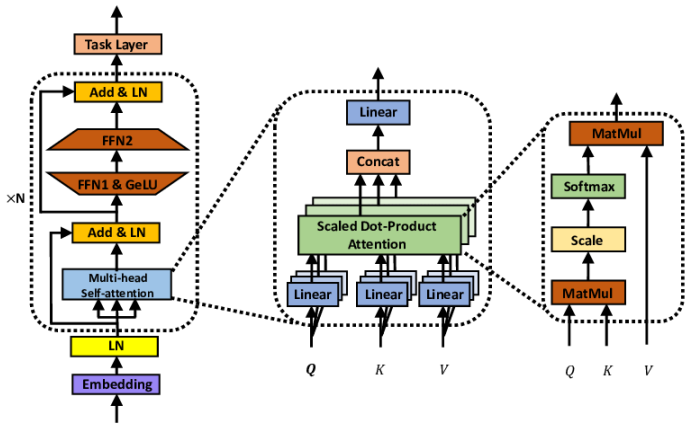


Fig: BERT Architecture

Liu, Zejian & Li, Gang & Cheng, Jian. (2021). Hardware Acceleration of Fully Quantized BERT for Efficient Natural Language Processing.

I. **Bidirectional Context:** BERT reads and comprehends text in both left and right contexts, taking into account the context of each word. This aids in its understanding of each word's complete meaning in a sentence.
II. **Attention Mechanism:** During encoding, BERT concentrates on pertinent portions of the text via an attention mechanism. It is able to assign greater weight to significant words and their connections because to this technique.
III. **Transformer Architecture:** The self-attention layers of the Transformer architecture serve as the foundation for BERT. These layers allow BERT to efficiently capture word dependencies and their contextual meanings.
IV. **Pretraining and Fine-Tuning:** To acquire general language representations, BERT is pretrained on a sizable corpus of textual data. By adjusting its parameters in response to task-specific data, it can be refined on certain tasks following pretraining.

Support Vector Machine

The support vector machine (SVM) is a machine learning algorithm that determines boundaries between data points based on predefined classes, labels, or outputs. It uses supervised learning models to solve complex problems related to classification, regression, and outlier detection. SVMs are widely used in many sectors, including speech and image recognition, natural language processing, healthcare, and signal processing applications.

Working of Support Vector Machine:

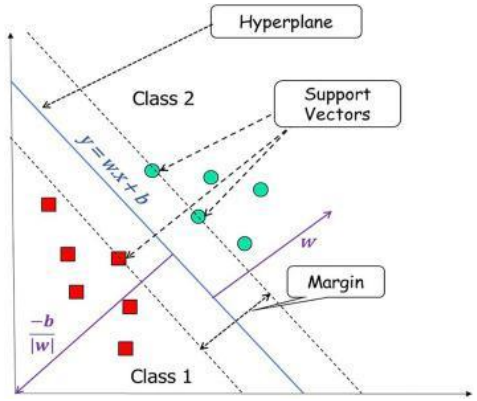


Fig SVM Algorithm

Alka Rani, Nishant K. Sinha, in Deep Learning for Sustainable Agriculture, 2022

I. **Margin Maximization**: SVM seeks the optimal hyperplane to separate different class data points with the maximum margin.
II. **Kernel Trick**: SVM handles non-linearly separable data by transforming it into a higher-dimensional space using functions like linear, polynomial, RBF, and sigmoid kernels.
III. **Decision Boundary**: SVM's hyperplane defines the decision boundary, classifying new data points based on their position relative to it in binary classification tasks.
IV. **Margin and Regularization**: SVM balances margin size and classification error using a regularization parameter (C), with smaller C values favoring wider margins but potentially more misclassifications and vice versa.
V. **Support Vectors**: Closest to the hyperplane, support vectors determine margin width and are essential in defining the decision boundary and making predictions for new data points.

Logistic Regression

Logistic regression assumes a linear relationship between features and log-odds, uses the sigmoid function to model the probability of belonging to a class, optimises using the logistic loss function, and offers a probabilistic interpretation of predictions based on threshold-based binary predictions. Because of its efficacy, simplicity, and interpretability, it is frequently utilised for binary classification jobs in a variety of domains.

Working of Logistic Regression

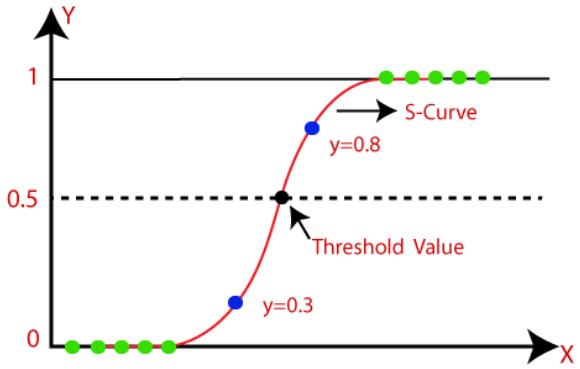


Fig. Logistic Regression curve

Javatpoint

I. **Sigmoid Function**: Logistic Regression employs the sigmoid function to model the probability of class membership, mapping real values to probabilities between 0 and 1.
II. **Binary Classification:** In binary classification, Logistic Regression predicts the probability of an input belonging to a specific class (0 or 1), with a threshold (often 0.5) determining class assignment.
III. **Linear Decision Boundary:** Logistic Regression assumes a linear relationship between input features and the log-odds, using a weighted sum plus bias and sigmoid function for probability estimation.
IV. **Cost Function and Optimization:** Logistic Regression minimizes the logistic loss or cross-entropy loss during training using optimization methods like gradient descent, aligning predicted probabilities with actual labels.
V. Probabilistic Interpretation: Unlike linear regression, Logistic Regression provides probabilistic outputs suitable for binary problems, aiding in understanding class probabilities and decision-making.

Random Forest:

Using random feature and data subsets, the Random Forest ensemble learning method creates several decision trees. It uses ensemble voting, bagging, and random feature selection to build a reliable and accurate model that can handle challenging regression and classification tasks. Its popularity in machine learning applications stems from its capacity to reduce overfitting, manage high-dimensional data, and offer feature importance.

Working of Random Forest:

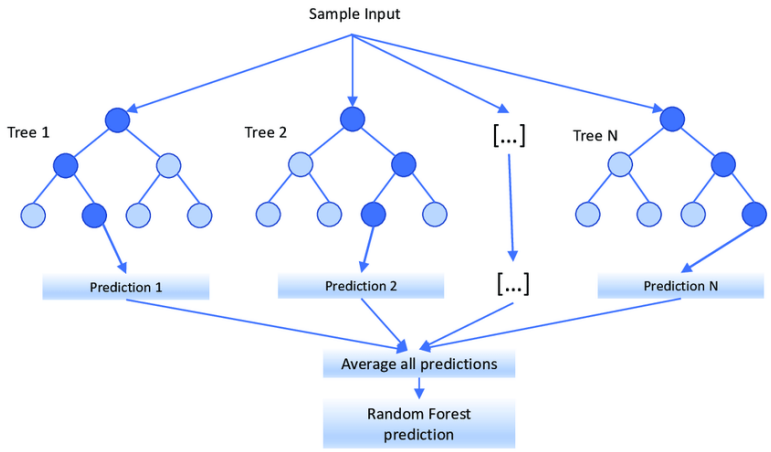


Fig Random Forest Structure

Segura, David & Khatib, Emil & Barco, Raquel. (2022). Dynamic Packet Duplication for Industrial URLLC. Sensors. 22. 587. 10.3390/s22020587.

I. **Ensemble Learning:** Multiple decision trees are combined in Random Forest, an ensemble learning technique, to increase generalisation and forecast accuracy.

II. **Random Feature Selection:** To improve model robustness and decrease correlations across trees, each decision tree in a Random Forest is trained on a random selection of features.

III. **Bootstrap Aggregating (Bagging):** Random Forest has bootstrap sampling. To ensure variety and minimise overfitting, each tree is trained on a bootstrapped sample of the data with replacement.

IV. **Voting/Averaging:** To generate the final output during prediction, Random Forest averages (for regression) or uses majority voting (for classification) to combine the predictions of individual trees.

K Nearest Neighbour:

The KNN algorithm is a supervised learning classifier that operates in a non-parametric manner. It employs proximity to classify or anticipate how a single data point will be grouped. This is a widely used and straightforward classifier for regression and classification in modern machine learning.

KNN working

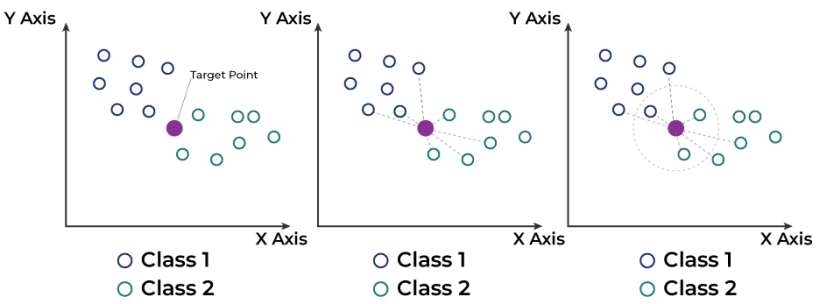


Fig KNN classifier

I. **Instance-Based Learning:** KNN is an instance-based or lazy learning algorithm that stores instances of training data and makes predictions based on similarity to new data points.

II. **Distance Calculation:** It calculates the distance between the new data point and all existing data points in the training set using metrics like Euclidean distance.

III. **Neighbour Selection:** It selects the k nearest neighbours to the new data point based on calculated distances.

IV. **Classification or Regression:** For classification, KNN uses majority voting among the k neighbours to determine the class of the new point. For regression, it averages the target values of the k neighbours to predict the target value of the new point.

V. **Choice of k:** The choice of k affects the algorithm's performance, with smaller k values leading to more complex decision boundaries and larger k values resulting in smoother boundaries.

Dataset Description

The MU3D (Miami University 3-Dimensional Deception) dataset is a collection of 320 videos created to study deception detection, social perceptions, and communication dynamics. It is a valuable resource for researchers in fields such as psychology, linguistics, computer science, and communication studies.

Stimulus Generation Wave:

1. **Participant Recruitment:** The dataset includes contributions from 112 participants recruited from Miami University's campus. The participants were diverse in terms of race (Black vs. White) and sex (male vs. female), ensuring a varied sample for analysis.
2. **Recording Environment:** Videos were recorded in controlled settings using a c525 Logitech HD Webcam. This maintained consistency in video quality, resolution (1,280 × 720), and frame rate (30 fps) across all recordings, minimizing confounding factors in analysis.
3. **Task Design:** Participants were instructed to describe individuals they liked or disliked, creating scenarios of positive truths, negative truths, positive lies, and negative lies. This task design aimed to elicit naturalistic responses, enhancing the authenticity of the dataset.
4. **Data Collection:** The recorded videos were transcribed by trained undergraduate research assistants. Transcriptions included verbal content, word-like vocalizations, and non-verbal cues, providing a rich dataset for linguistic and behavioural analysis.

Stimulus Rating Wave:

1. **Rater Recruitment:** A separate group of 405 raters from Amazon's Mechanical Turk evaluated the videos. Raters represented diverse demographics and were compensated for their participation, ensuring a broad range of perspectives in the dataset.
2. **Evaluation Tasks:** Raters performed two main tasks: discerning truths from lies in each video and providing subjective ratings of targets' attractiveness, trustworthiness, and anxiety levels. This dual-task setup allowed for the assessment of both objective deception detection and subjective social judgments.
3. **Dataset Structure:** The dataset follows a fully factorial mixed design, incorporating factors such as race, sex, valence (positive vs. negative), and veracity (honest vs. dishonest). This design enables researchers to analyze interactions between these factors in deception detection and social perception processes.

Ethical Considerations: The dataset was collected following ethical guidelines, including obtaining informed consent from participants and ensuring data privacy and confidentiality.

Experiments

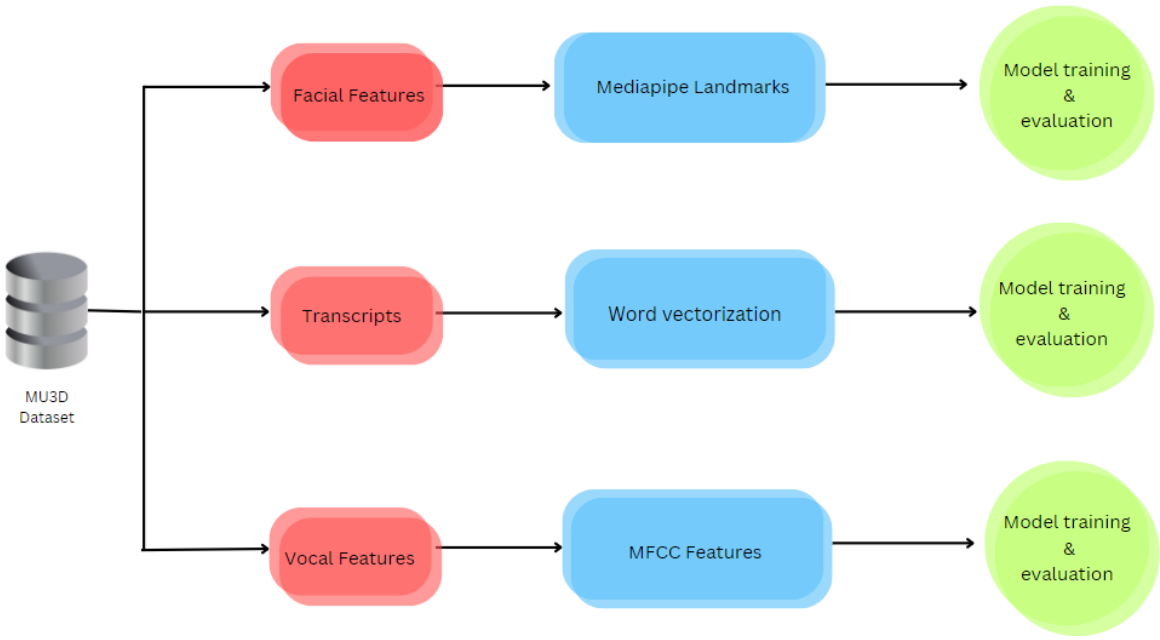


Fig Model Architecture

In the experiment section, our focus was on exploring three key features instrumental in deception detection: Facial Features, Transcripts, and Vocal Features.

I. **Transcripts:** We extracted significant information from textual transcripts by using sophisticated word vectorization techniques. Then, the extracted traits were put to the test in different models to see how well they worked at identifying dishonesty.

II. **Face Features:** We extracted features from face data using the Mediapipe framework. After that, a variety of model tests were conducted on these attributes to determine how well they detected dishonesty.

III. **Vocal Features:** We extracted Mel-frequency cepstral coefficients (MFCC) from vocal data by utilising the Librosa library's capabilities. These features were analysed with several models to determine their effectiveness in detecting misleading patterns.

Experiments

Transcripts Analysis Workflow

Experiment 1:

1. **Data Preprocessing:**
    - **Text Normalization:** Initially, the transcript data undergoes normalization to ensure consistency in the text format. This involves converting the text to lowercase, removing extra whitespaces, and handling accented characters to maintain data integrity.
    - **Bag-of-Words Encoding:** The normalized text is then transformed into a bag-of-words representation using techniques like Count Vectorizer. This step converts the text into a numerical format suitable for machine learning algorithms.
    - **Train-Test Split:** The pre-processed data is split into training and testing sets using an 80/20 ratio. This separation allows for model training on the training set and subsequent evaluation on the unseen testing set.

2. **Model Training:**
    - **Logistic Regression:** A Logistic Regression model is trained on the bag-of-words encoded data. This model learns to predict outcomes based on the features extracted from the transcripts.
    - **Support Vector Machine (SVM):** Another model such as Support Vector Machines (SVM) may also be trained simultaneously to explore different algorithm performances.
    - **Random Forest:** Additionally, a Random Forest classifier is trained on the pre-processed transcript data. This ensemble learning method can capture complex relationships in the data.

3. **Evaluation Metrics:**
    - **Accuracy:** The accuracy of each model is computed to measure its overall performance in correctly classifying transcripts.
    - **Precision and Recall:** Precision and recall metrics are evaluated to assess the model's ability to correctly identify true positives while minimizing false positives and false negatives.
    - **F1 Score:** The F1 score, which considers both precision and recall, provides a balanced measure of a model's performance, especially in scenarios with imbalanced classes.

For the transcript analysis, the first models Logistic Regression, SVM, and Random Forest did not produce accuracy that was good enough. In an effort to improve accuracy and performance, I made the decision to move to BERT, a more sophisticated model renowned for its capacity to comprehend context and subtleties in text input.

Experiment 2:

Data Preparation and Preprocessing:

I. **Train-Test Split:** The dataset undergoes an 90/10 train-test split to ensure an adequate amount of data for training while reserving a separate portion for testing model generalization.
II. **Text Preprocessing:** Text preprocessing is crucial for BERT-based models. The 'Transcription' data is tokenized, normalized, and cleaned to handle variations in text formats. Tokenization converts text into tokens (words or sub words) that BERT can understand. Padding and truncation are applied to ensure all sequences have a consistent length, typically set to a maximum of 512 tokens, which is a standard for BERT models.

Model Training:

I. **BERT-based Text Classifier:** The core of Experiment 2 involves training a BERT-based text classifier. BERT (Bidirectional Encoder Representations from Transformers) is a powerful language model that captures contextual information effectively, making it ideal for text classification tasks.
II. **Optimization Strategy:** The model is optimized using the Adam optimizer with a learning rate of 1e-5 and a decay rate of 1e-5. This learning rate is chosen to balance between learning speed and stability, and decay helps prevent overfitting by gradually reducing the learning rate over time.
III. **Loss Function:** Binary cross-entropy is used as the loss function, suitable for binary classification tasks where each transcript is labelled as either true or false.

Evaluation:

I. **One-Cycle Policy:** Training follows the One-Cycle Policy, an approach that varies the learning rate cyclically during training. This technique enhances model convergence and generalization by exploring a range of learning rates.
II. **Validation and Testing Metrics:** Throughout training, the model's performance is monitored using validation metrics such as accuracy. After training, the model's effectiveness is evaluated using standard evaluation metrics including accuracy, precision, recall, and F1 score.
III. **Batch Size and Epochs:** The model is trained with a batch size of 3, a commonly used value in BERT-based models, and for a total of 20 epochs, ensuring sufficient iterations for the model to learn complex patterns in the data.

The very short dataset size of 320 rows caused the BERT model to show evidence of overfitting despite its advanced design and capabilities, which resulted in subpar performance when compared to simpler models. Due to the restricted amount of available data, BERT was unable to generalise as well as it could have, memorising noise or certain patterns in the training data instead of learning meaningful representations. As a result, the model's performance declined, underscoring the difficulty of using intricate models like BERT on

tiny datasets where more straightforward models might generalise more effectively and produce more trustworthy outcomes.

Experiment 3:

Data Preparation:

I. Utilized the same text normalization techniques as in Experiment 1 to maintain consistency and data integrity.

II. Employed an 80/20 train-test split to separate the dataset into training and testing sets.

Feature Extraction and Selection:

I. Applied Count Vectorizer with an n-gram range of (1, 2) to capture both single words (unigrams) and word pairs (bigrams) in the transcript data.

II. Implemented a feature selection method using Select Percentile with a percentile value of 70 and chi-squared feature selection, following insights from a relevant research paper.

III. The feature selection process aimed to retain the most informative features for classification, enhancing the model's ability to learn relevant patterns.

Methodological Inspiration:

I. The approach was inspired by the paper "Increasing Accuracy of Support Vector Machine (SVM) By Applying N-Gram and Chi-Square Feature Selection for Text Classification."

II. Leveraged insights from the paper to improve feature selection and subsequently enhance the performance of the classification model.

Experimental Goal:

I. The objective of Experiment 3 was to explore advanced feature extraction and selection techniques to improve the model's accuracy and predictive power.

II. By incorporating n-grams and chi-squared feature selection, the experiment aimed to capture nuanced linguistic patterns and reduce noise in the data, leading to more effective classification. (Highest accuracy)

Facial Features Workflow

Data Preprocessing

I. **Data Splitting:** Videos folder was split into train, validation, and test sets. First, videos were split into an 80-20 ratio for the train-test split. Then, further, the train set was divided into a 75-25 ratio for the train-validation split.

II. **Video Frames Extraction:** The MediaPipe Face Mesh model was initialized to detect facial landmarks in each frame of the videos. Then Each frame of the input videos was read using OpenCV's VideoCapture functionality.
Different frame counts were generated in the following manner for all the data splits: 20 Frames/Video, 50 Frames/Video, 100 Frames/Video and 200 Frames/Video.

III. **Frames Preprocessing:** The frames were converted from BGR to RGB colour space using OpenCV's cvtColor function. This step was crucial because MediaPipe Face Mesh expects input frames in RGB format.

IV. **Data Extraction:** The MediaPipe Face Mesh model was applied to each frame to detect facial landmarks, including points such as the eyes, nose, mouth, and chin and then for each detected face in the frame, the x and y coordinates of the facial landmarks were extracted. Then the extracted coordinates were stored in a numpy array for further processing and analysis.

V. **Data Reshaping:** In some cases, the data was reshaped to a specific format required by machine learning models. For example, the 3D array of coordinates was flattened into a 2D array to fit logistic regression.

Model Training

After preprocessing the facial feature data extracted from the videos, various machine learning models were trained and evaluated on the dataset. The following models were applied:

1. **Logistic Regression:** Logistic regression is a linear classification algorithm used for binary classification tasks. It models the probability of the input belonging to a particular class using a logistic function.
   The logistic regression model was trained using the facial feature data flattened into a 2D array.`

2. **Support Vector Machine (SVM)**: SVM is a supervised learning algorithm used for classification tasks. It finds the hyperplane that best separates the classes in the feature space.

   The SVM model was trained on the flattened facial feature data.
3. **Random Forest**: Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees.

   The Random Forest model was trained on the flattened facial feature data.
4. **K-Nearest Neighbors (KNN)**: KNN is a non-parametric and lazy learning algorithm used for classification tasks. It assigns a class label to an input based on the majority class of its k nearest neighbours.

   The KNN classifier was trained using the facial feature data flattened into a 2D array.

Experiment 1:

In the initial experiment, we generated the first 20 frames per video to extract facial features. However, this approach resulted in a very low accuracy, approximately 0.25. Due to this poor performance, we decided to discard the data obtained from these 20 frames per video.

Experiment 2:

In the subsequent experiment, we increased the number of frames per video to 50. This led to an improvement in accuracy compared to using only 20 frames. Encouraged by this result, we further experimented with extracting 100 frames per video. However, despite the increased number of frames, the accuracies did not show a significant improvement compared to using 50 frames. As a result, we discarded the data obtained from 100 frames per video.

Experiment 3:

Continuing our exploration, we increased the number of frames per video to 200. This adjustment resulted in a notable increase in accuracy compared to using 50 frames. The improved performance suggested that extracting more frames per video provided richer information for our models, leading to better classification results.

Vocal Feature (Audio) workflow Analysis

Data Collection:

- Using the MU3D (Miami University 3-Dimensional Deception) dataset which is a collection of 320 videos created to study deception detection, social perceptions, and communication dynamics.

Preprocessing:

- Convert all the video file into audio file such as MP4 to WAV or MP3.

Feature Engineering:

- Extract relevant features from the audio files. Common features for voice analysis include pitch, intensity, duration, and formants.
- Normalise the features to ensure that they are on a similar scale, which can improve the performance of the machine learning model.
- Using techniques such as Mel-Frequency Cepstral Coefficients (MFCCs) to extract more complex features from the audio data.

Model Selection:

- Choosing a machine learning model suitable for classification tasks. Popular choices for audio analysis include Support Vector Machines (SVMs), Random Forests, or deep learning models such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs).
- Splitting the dataset into training and testing sets to evaluate the performance of our model.

Model Training:

- Train our chosen model using the training data. Adjust hyperparameters as needed to optimize performance.
- We used SVM linear (Support Vector Machine) ,SVM with RBF kernel,Random forest,KNN,XGboost.

Evaluation:

- Evaluate the performance of your model using the testing data. Common evaluation metrics for classification tasks include accuracy, precision, recall, and F1 score.
- Analyze the results to understand how well your model is performing at detecting lies based on voice analysis.

RESULTS AND DISCUSSION

Textual Analysis

Logistic Regression, SVM and Random Forest

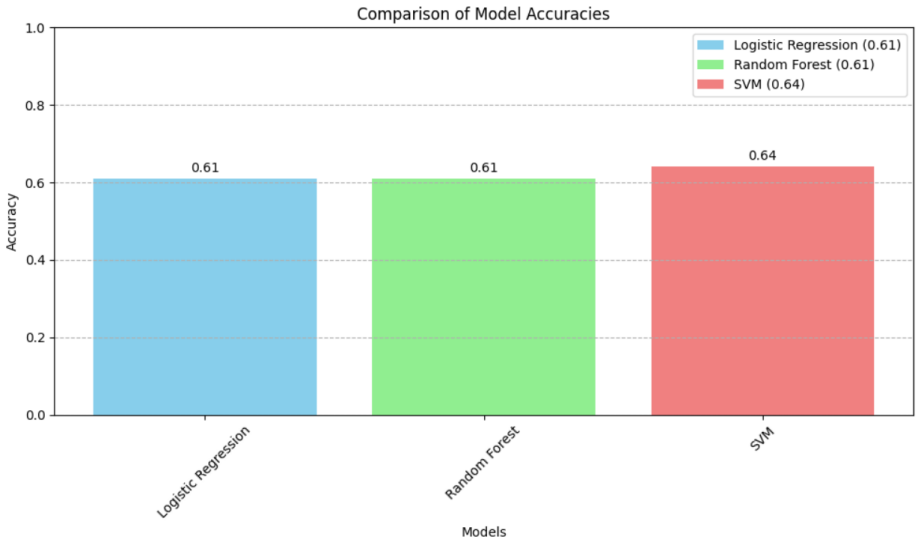


Fig. Comparison of models

```
Accuracy: 0.609375
Confusion Matrix:
[[17 14]
 [11 22]]
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.55      0.58        31
           1       0.61      0.67      0.64        33

    accuracy                           0.61        64
   macro avg       0.61      0.61      0.61        64
weighted avg       0.61      0.61      0.61        64
```

```
Random Forest Accuracy: 0.609375
Confusion Matrix:
[[20 11]
 [14 19]]
Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.65      0.62        31
           1       0.63      0.58      0.60        33

    accuracy                           0.61        64
   macro avg       0.61      0.61      0.61        64
weighted avg       0.61      0.61      0.61        64
```

```
SVM Accuracy: 0.640625
Confusion Matrix:
[[19 12]
 [11 22]]
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.61      0.62        31
           1       0.65      0.67      0.66        33

    accuracy                           0.64        64
   macro avg       0.64      0.64      0.64        64
weighted avg       0.64      0.64      0.64        64
```

Fig. Classification report and confusion matrices of models

It can be seen that the highest accuracy achieved was by SVM while Logistic Regression and Random Forest performed almost the same

BERT:

```
begin training using onecycle policy with max lr of 1e-05...
Epoch 1/20
96/96 [==============================] - 344s 3s/step - loss: 0.7058 - accuracy: 0.5174 - val_loss: 0.6977 - val_accuracy: 0.5312
Epoch 2/20
96/96 [==============================] - 323s 3s/step - loss: 0.6850 - accuracy: 0.5729 - val_loss: 0.6903 - val_accuracy: 0.5000
Epoch 3/20
96/96 [==============================] - 323s 3s/step - loss: 0.6700 - accuracy: 0.5694 - val_loss: 0.6852 - val_accuracy: 0.5312
Epoch 4/20
96/96 [==============================] - 326s 3s/step - loss: 0.6329 - accuracy: 0.6736 - val_loss: 0.6776 - val_accuracy: 0.5000
Epoch 5/20
96/96 [==============================] - 319s 3s/step - loss: 0.5635 - accuracy: 0.7604 - val_loss: 0.7258 - val_accuracy: 0.5312
Epoch 6/20
96/96 [==============================] - 319s 3s/step - loss: 0.4489 - accuracy: 0.8646 - val_loss: 0.8054 - val_accuracy: 0.5000
Epoch 7/20
96/96 [==============================] - 318s 3s/step - loss: 0.2680 - accuracy: 0.9410 - val_loss: 0.7821 - val_accuracy: 0.6250
Epoch 8/20
96/96 [==============================] - 318s 3s/step - loss: 0.1682 - accuracy: 0.9688 - val_loss: 0.9395 - val_accuracy: 0.5312
Epoch 9/20
96/96 [==============================] - 319s 3s/step - loss: 0.1062 - accuracy: 0.9792 - val_loss: 0.9195 - val_accuracy: 0.6250
Epoch 10/20
96/96 [==============================] - 317s 3s/step - loss: 0.0598 - accuracy: 0.9965 - val_loss: 1.1738 - val_accuracy: 0.5938
Epoch 11/20
96/96 [==============================] - 316s 3s/step - loss: 0.0223 - accuracy: 1.0000 - val_loss: 1.4089 - val_accuracy: 0.5000
...
Epoch 19/20
96/96 [==============================] - 316s 3s/step - loss: 0.0040 - accuracy: 1.0000 - val_loss: 1.7297 - val_accuracy: 0.5312
Epoch 20/20
96/96 [==============================] - 317s 3s/step - loss: 0.0038 - accuracy: 1.0000 - val_loss: 1.7409 - val_accuracy: 0.5312
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Fig. Bert model results

The BERT model struggled due to overfitting on the small dataset of 320 rows, leading to poorer performance than simpler models. Its highest accuracy on the validation set was just 62%, underscoring the challenges of complex models with limited data. This highlights the importance of dataset size and diversity for building robust, generalizable models, as larger datasets can mitigate overfitting and better capture real-world variations.

Performance Comparison of SVM, Random Forest, and Logistic Regression with N-gram and Chi-square Feature Selection

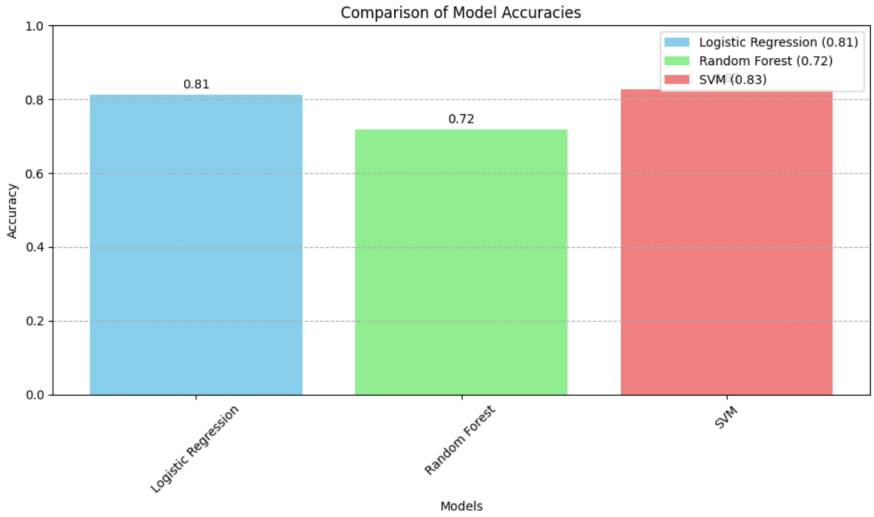


Fig. Model comparison

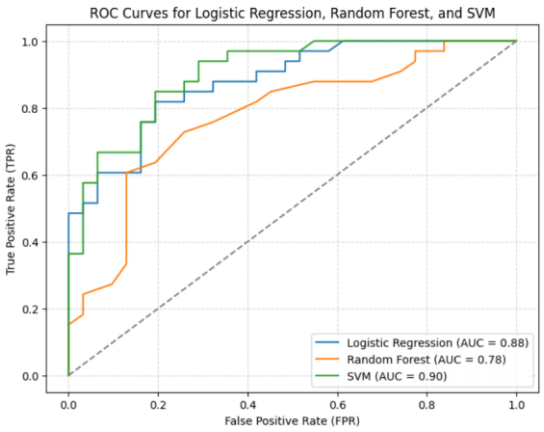


Fig. Combined ROC curve of all models

```
Logistic Regression Accuracy: 0.8125
Confusion Matrix (Logistic Regression):
[[25  6]
 [ 6 27]]
Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        31
           1       0.82      0.82      0.82        33

    accuracy                           0.81        64
   macro avg       0.81      0.81      0.81        64
weighted avg       0.81      0.81      0.81        64
```

```
Random Forest Accuracy: 0.71875
Confusion Matrix (Random Forest):
[[21 10]
 [ 8 25]]
Classification Report (Random Forest):
              precision    recall  f1-score   support

           0       0.72      0.68      0.70        31
           1       0.71      0.76      0.74        33

    accuracy                           0.72        64
   macro avg       0.72      0.72      0.72        64
weighted avg       0.72      0.72      0.72        64
```

```
SVM Accuracy: 0.828125
Confusion Matrix:
[[25  6]
 [ 5 28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.81      0.82        31
           1       0.82      0.85      0.84        33

    accuracy                           0.83        64
   macro avg       0.83      0.83      0.83        64
weighted avg       0.83      0.83      0.83        64
```

Fig Classification Report of model

The N-gram and chi-square feature selection enhanced SVM, Random Forest, and Logistic Regression models, leading to a significant improvement in performance with SVM achieving the highest accuracy of 83%.

Final Results:

MODEL	ACCURACY
BERT	9 EPOCH VAL ACC 62%
SVM	64%
RANDOM FOREST	61%
LOGISTIC REGRESSION	61%
SVM (CHI & NGRAM)	83%
RANDOM FOREST (CHI & NGRAM)	72%
LOGISTIC REGRESSION (CHI & NGRAM)	81%

TABLE: Final Accuracy Comparison

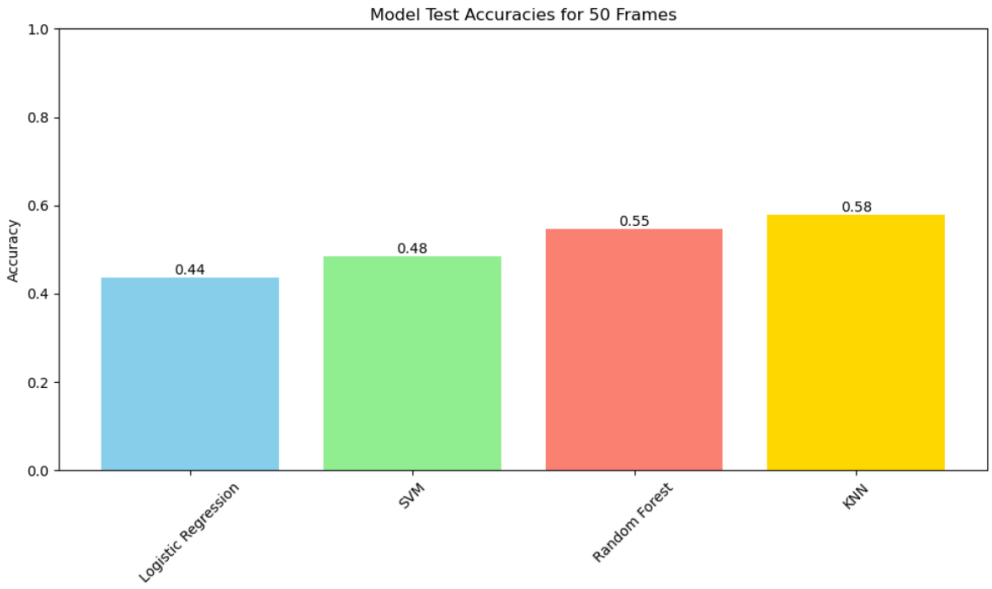
The experiment incorporating SVM, Random Forest, and Logistic Regression with N-gram and Chi-square feature selection performed better than the other two trials. This result can be explained by the feature selection algorithms' ability to extract pertinent information from the textual material. In particular, the SVM model's greatest accuracy of 83% on the validation set demonstrated its strong generalisation and ability to produce precise predictions on untested data. In comparison to the other tests, the models performed better overall because of the large improvement in predictive power that came from the combination of
N-gram features and chi-square feature selection.

Facial Analysis

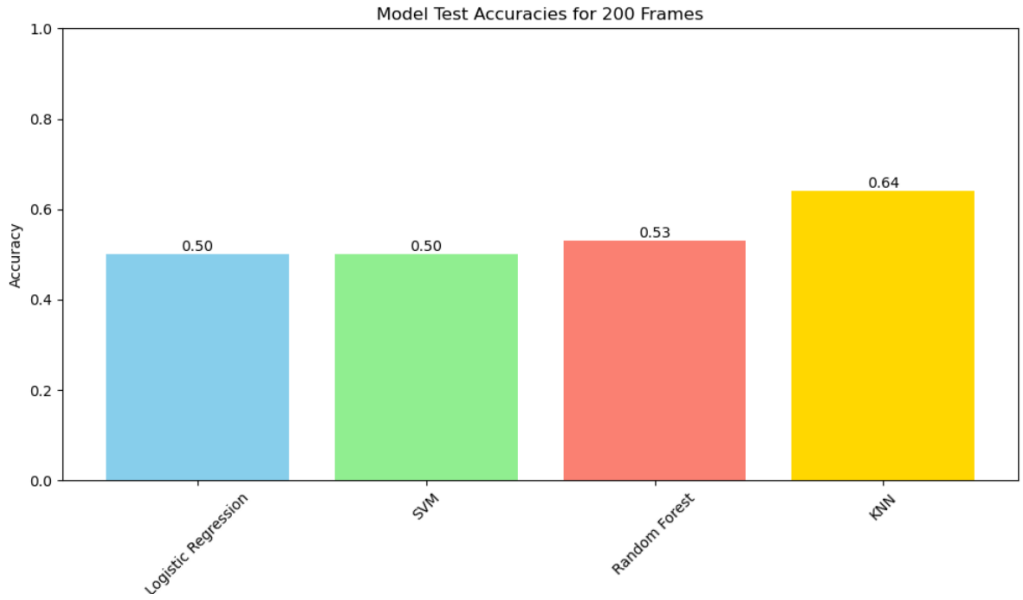
Models Used: Logistic Reg, SVM, Random Forest and KNN

Below are the generated graphs for test set accuracies: -

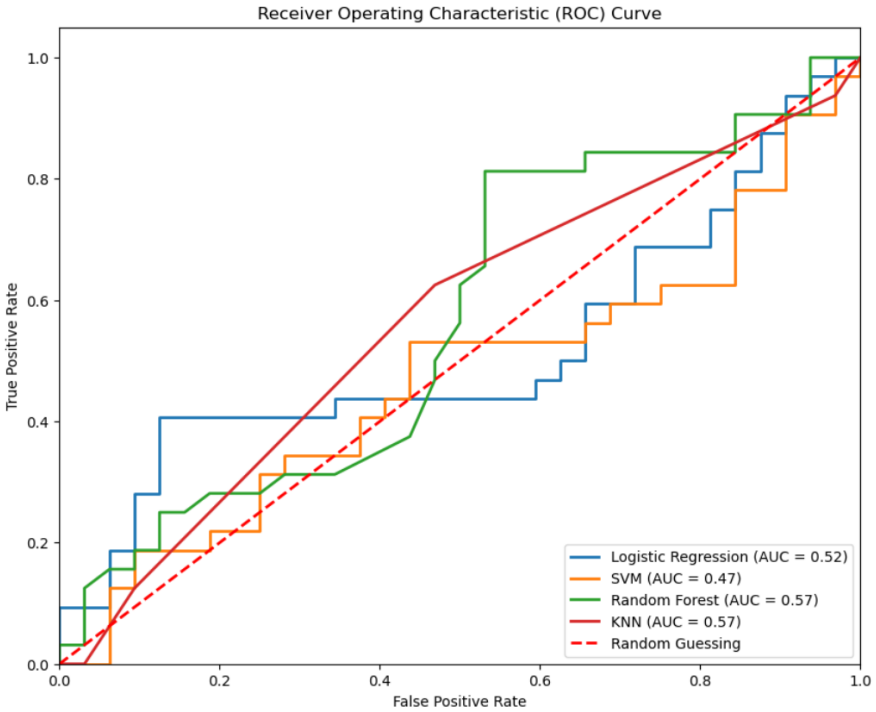
1. For 50 Frames



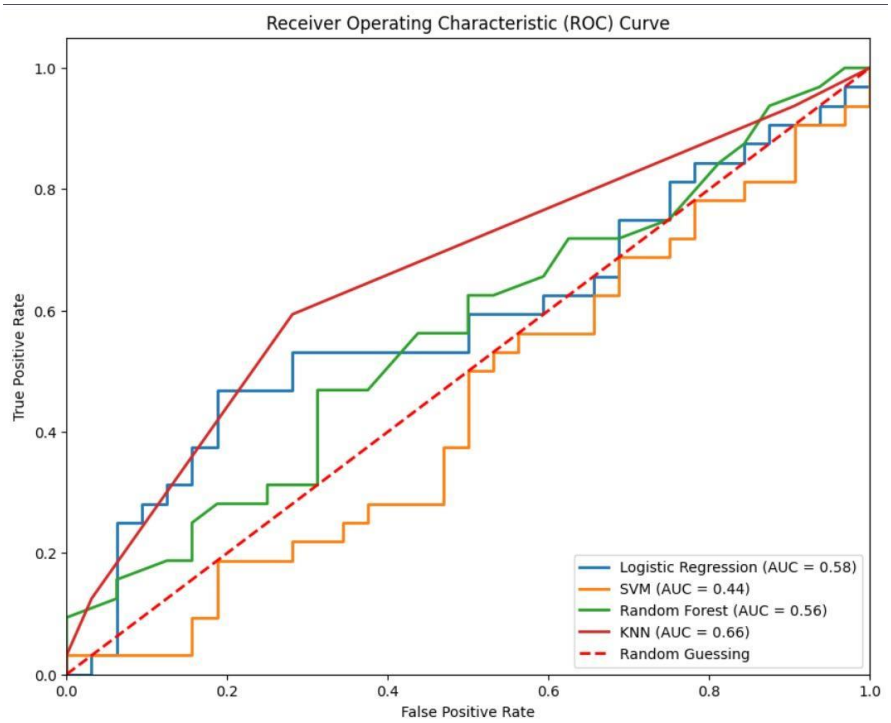
For 200 Frames



ROC Curve for 50 frames per video:



ROC Curve for 200 frames per video:



Classification report comparisons:-

1. For 50 Frames: -

Logistic Regression Report:

	precision	recall	f1-score	support
0	0.44	0.44	0.44	32
1	0.44	0.44	0.44	32
accuracy			0.44	64
macro avg	0.44	0.44	0.44	64
weighted avg	0.44	0.44	0.44	64

SVM Report:

	precision	recall	f1-score	support
0	0.45	0.16	0.23	32
1	0.49	0.81	0.61	32
accuracy			0.48	64
macro avg	0.47	0.48	0.42	64
weighted avg	0.47	0.48	0.42	64

Random Forest Report:

	precision	recall	f1-score	support
0	0.52	0.47	0.49	32
1	0.51	0.56	0.54	32
accuracy			0.52	64
macro avg	0.52	0.52	0.51	64
weighted avg	0.52	0.52	0.51	64

KNN Report:

	precision	recall	f1-score	support
0	0.59	0.53	0.56	32
1	0.57	0.62	0.60	32
accuracy			0.58	64

2. For 200 Frames: -

Logistic Regression Report:

	precision	recall	f1-score	support
0	0.50	0.50	0.50	32
1	0.50	0.50	0.50	32
accuracy			0.50	64
macro avg	0.50	0.50	0.50	64
weighted avg	0.50	0.50	0.50	64

SVM Report:

	precision	recall	f1-score	support
0	0.50	0.25	0.33	32
1	0.50	0.75	0.60	32
accuracy			0.50	64
macro avg	0.50	0.50	0.47	64
weighted avg	0.50	0.50	0.47	64

Random Forest Report:

	precision	recall	f1-score	support
0	0.58	0.56	0.57	32
1	0.58	0.59	0.58	32
accuracy			0.58	64
macro avg	0.58	0.58	0.58	64
weighted avg	0.58	0.58	0.58	64

KNN Report:

	precision	recall	f1-score	support
0	0.66	0.59	0.62	32
1	0.63	0.69	0.66	32
accuracy			0.64	64
macro avg	0.64	0.64	0.64	64
weighted avg	0.64	0.64	0.64	64

Final Results (for 200 Frames/video as they gave the best accuracies): -

Model	**Accuracy**
Logistic Regression	50%
Support Vector Machine	50%
Random Forest	53%
KNN	64% (Best)

Conclusion for Facial Analysis:

The highest accuracy was achieved when utilizing 200 frames per video in conjunction with the K-Nearest Neighbors (KNN) model. This suggests that increasing the number of frames extracted from each video significantly improved the classification performance. The KNN algorithm, which relies on the similarity of feature vectors, likely benefited from the richer information obtained from the larger number of frames.

This finding underscores the importance of data quantity in improving model performance, particularly in tasks such as facial feature extraction and classification. It also highlights the effectiveness of the KNN algorithm in leveraging this increased data richness to achieve superior accuracy.

Audio Analysis

Performance comparison between SVM (Linear) ,XGBoost,SVM with RBF

SVM Accuracy: 0.546875
Classification Report:

	precision	recall	f1-score	support
0	0.55	0.53	0.54	32
1	0.55	0.56	0.55	32
accuracy			0.55	64
macro avg	0.55	0.55	0.55	64
weighted avg	0.55	0.55	0.55	64

XGBoost Accuracy: 0.609375
XGBoost Classification Report:

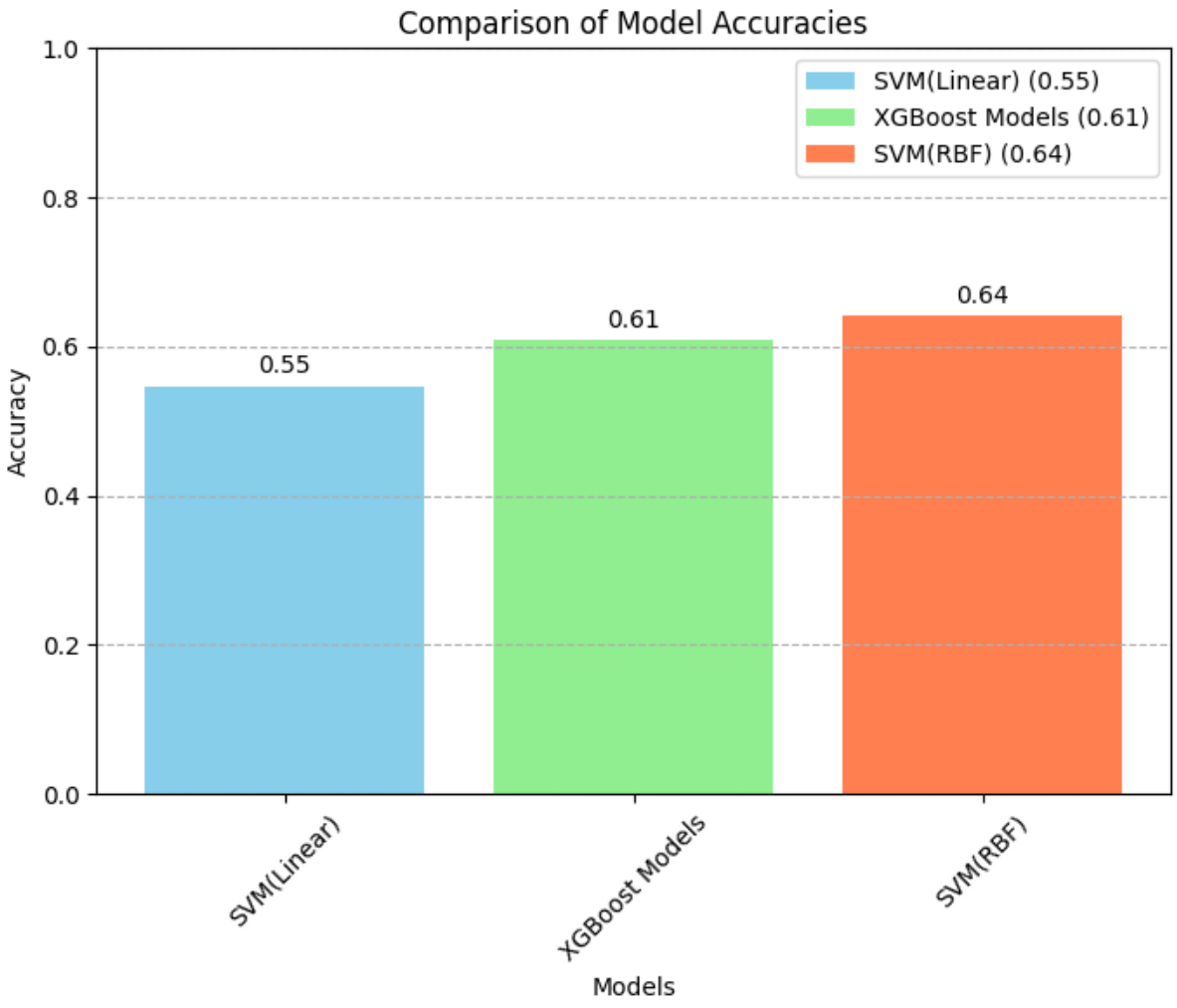
	precision	recall	f1-score	support
0	0.63	0.53	0.58	32
1	0.59	0.69	0.64	32
accuracy			0.61	64
macro avg	0.61	0.61	0.61	64
weighted avg	0.61	0.61	0.61	64

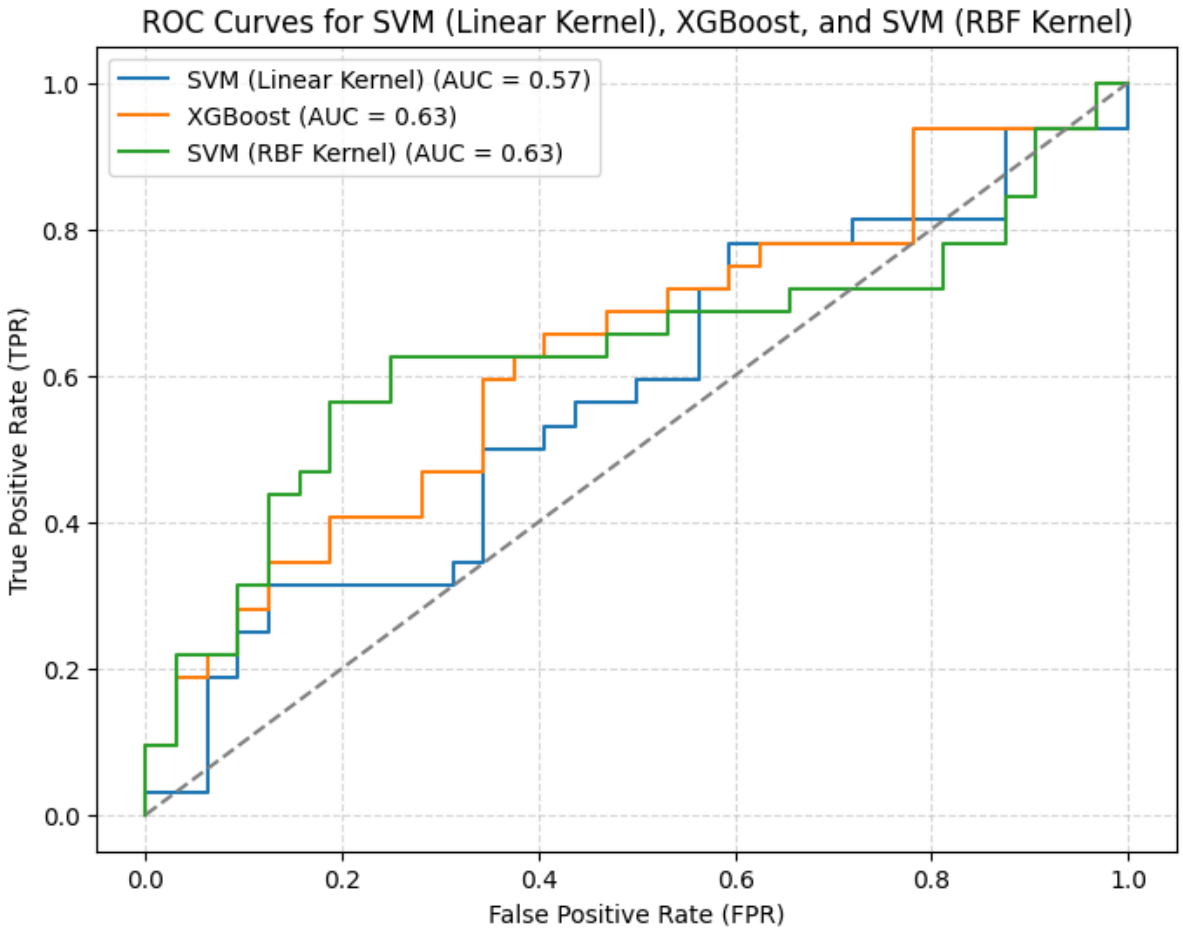
Random Forest Accuracy: 0.59375
Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.61	0.53	0.57	32
1	0.58	0.66	0.62	32
accuracy			0.59	64
macro avg	0.60	0.59	0.59	64
weighted avg	0.60	0.59	0.59	64

SVM with RBF Kernel Accuracy: 0.640625
SVM with RBF Kernel Classification Report:

	precision	recall	f1-score	support
0	0.64	0.66	0.65	32
1	0.65	0.62	0.63	32
accuracy			0.64	64
macro avg	0.64	0.64	0.64	64
weighted avg	0.64	0.64	0.64	64





MODEL	ACCURACY
Support vector machine(SVM)	0.546875
RANDOM FOREST	0.59375
XGBoost	0.609375
KNN	0.546875
SVM with RBF kernel	0.640625 (best)

TABLE: Final Accuracy Comparison

CONCLUSION

Our findings demonstrate the potential of utilizing different modalities for deception detection, with notable achievements in transcript analysis. The highest accuracy of 83% achieved by SVM on transcript analysis underscores the effectiveness of textual cues in identifying deceptive behavior. This result highlights the importance of linguistic patterns and semantic content in discerning deceitful statements.

Furthermore, our project has shed light on the challenges and complexities inherent in deception detection, such as the need for nuanced feature extraction and the limitations of existing methodologies like the polygraph. By leveraging advanced techniques like deep learning algorithms, we have made progress in overcoming these challenges and enhancing the accuracy and reliability of deception detection systems.

Moving forward, our project lays the groundwork for further research and development in automatic deception detection. Future endeavors may explore additional modalities, refine existing methodologies, and address remaining challenges to achieve even greater accuracy and scalability in real-world applications.

Overall, our project contributes to the ongoing evolution of deception detection techniques and holds promise for improving security measures, enhancing investigative procedures, and advancing our understanding of human communication and behaviour. Through continued innovation and collaboration, we can continue to push the boundaries of deception detection and pave the way for more effective and reliable systems in the future.

FUTURE SCOPE

The future scope of this study encompasses several avenues for further exploration and enhancement:

Incorporation of Additional Features: The current analysis focused primarily on facial features, transcripts and vocal features extracted from videos. However, many other features, such as trustworthiness, attractiveness, word count, race, sex, and anxiousness, were not considered. Integrating these features into the classification model could provide valuable insights and potentially improve accuracy.

Time-Series Analysis: Time-series analysis was not conducted in the current study. Incorporating temporal information from video sequences could capture dynamic patterns and dependencies over time, leading to more robust classification models. Exploring time-series techniques such as recurrent neural networks (RNNs) or convolutional neural networks (CNNs) could yield significant improvements in accuracy.

Development of a Mega Dataset: Integrating multiple datasets and features into a comprehensive "mega dataset" represents a promising direction for future research. By aggregating diverse sources of information, including facial features, transcripts, vocal characteristics, and potentially additional modalities, such a mega dataset could provide a rich and comprehensive input for unified modelling. Leveraging advanced techniques such as ensemble learning or meta-learning, this unified model could effectively harness the collective information from the mega dataset to make more informed veracity predictions.

Exploration of Advanced Machine Learning Techniques: Future investigations could explore advanced machine learning techniques beyond the traditional models employed in this study. Deep learning architectures, such as deep neural networks (DNNs) or transformers, have demonstrated remarkable capabilities in handling complex data and could offer superior performance in veracity detection tasks.

Overall, the future scope of this study involves leveraging a broader range of features, incorporating advanced machine learning techniques, and exploring innovative model architectures to advance the state-of-the-art in veracity detection from facial features.

REFERENCES

Karnati, M., Seal, A., Yazidi, A., & Krejcar, O. (2021). LieNet: a deep convolution neural network framework for detecting deception. *IEEE transactions on cognitive and developmental systems*, *14*(3), 971-984.

Fachrurrozi, S., Shidik, G. F., Fanani, A. Z., & Al Zami, F. (2021, September). Increasing Accuracy of Support Vector Machine (SVM) By Applying N-Gram and Chi-Square Feature Selection for Text Classification. In *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)* (pp. 42-47). IEEE.

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune bert for text classification?. In *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18* (pp. 194-206). Springer International Publishing.

Wahba, Y., Madhavji, N., & Steinbacher, J. (2022, September). A comparison of svm against pre-trained language models (plms) for text classification tasks. In *International Conference on Machine Learning, Optimization, and Data Science* (pp. 304-313). Cham: Springer Nature Switzerland.

Venkatesh, S. & Ramachandra, R. & Bours, P., (2019) Robust algorithm for multimodal deception detection. In: 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pp. 534–537, March 2019

Zhe Wu & Bharat Singh & Larry S. Davis & V. S. Subrahmanian, (2018) Deception Detection in Videos. AAAI.

Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., ... & Grundmann, M. (2019, June). Mediapipe: A framework for perceiving and processing reality. In *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)* (Vol. 2019).

Frank, M. G., & Feeley, T. H. (2003). To catch a liar: Challenges for research in lie detection training. *Journal of Applied Communication Research*, *31*(1), 58-75.

APPENDIX

Codes for facial features

```
Click here to ask Blackbox to help you code faster
import cv2
import mediapipe as mp
import numpy as np
import os

mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(static_image_mode=False, max_num_faces=1, min_detection_confidence=0.5)

folder_path = '/home/mayur/Desktop/Machine Learning/ML Project (Lie Detection)/Facemesh/Model Prediction/Train_val_test Split/val_25'

video_files = [f for f in os.listdir(folder_path) if f.endswith('.wmv')]

num_videos = len(video_files)
all_coordinates = np.zeros((num_videos,50, 468, 2))

# Process each video
for i, video_file in enumerate(video_files):
    video_path = os.path.join(folder_path, video_file)

    print("Processing video:", video_file)
    cap = cv2.VideoCapture(video_path)

    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    interval = total_frames // 50

    frame_count = 0
    frame_index = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_count += 1
        if frame_count % interval == 0:
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            results = face_mesh.process(frame_rgb)
            if results.multi_face_landmarks:
                landmarks = results.multi_face_landmarks[0].landmark
                for j, landmark in enumerate(landmarks):
                    all_coordinates[i, frame_index, j] = [landmark.x, landmark.y]

                frame_index += 1
                if frame_index == 50:
                    break

    cap.release()

np.save('val_facemesh_coordinates.npy', all_coordinates)
```

```
Click here to ask Blackbox to help you code faster
### LOGISTIC REGRESSION

import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load data
train_X = np.load('train_facemesh_coordinates.npy')
val_X = np.load('val_facemesh_coordinates.npy')
test_X = np.load('test_facemesh_coordinates.npy')

train_df = pd.read_csv('train_75.csv')
val_df = pd.read_csv('val_25.csv')
test_df = pd.read_csv('test_videos.csv')

train_y = train_df['Veracity'].values
val_y = val_df['Veracity'].values
test_y = test_df['Veracity'].values

# Flatten the features to fit logistic regression
train_X_flat = train_X.reshape(train_X.shape[0], -1)
val_X_flat = val_X.reshape(val_X.shape[0], -1)
test_X_flat = test_X.reshape(test_X.shape[0], -1)

```

```
Click here to ask Blackbox to help you code faster
### Logistic Regression

logreg = LogisticRegression()
logreg.fit(train_X_flat, train_y)

val_preds = logreg.predict(val_X_flat)

val_accuracy = accuracy_score(val_y, val_preds)
print("Validation Accuracy:", val_accuracy)

test_preds = logreg.predict(test_X_flat)

```

```
Click here to ask Blackbox to help you code faster
#### STATE VECTOR MACHINE (SVM)

from sklearn.svm import SVC

svm = SVC()
svm.fit(train_X_flat, train_y)

val_preds = svm.predict(val_X_flat)

val_accuracy = accuracy_score(val_y, val_preds)
print("Validation Accuracy:", val_accuracy)

test_preds = svm.predict(test_X_flat)

test_accuracy = accuracy_score(test_y, test_preds)
print("Test Accuracy:", test_accuracy)
```

```
Validation Accuracy: 0.47692307692307695
Test Accuracy: 0.484375
```

```
Click here to ask Blackbox to help you code faster
#### RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(train_X_flat, train_y)

val_preds = rf.predict(val_X_flat)

val_accuracy = accuracy_score(val_y, val_preds)
print("Validation Accuracy:", val_accuracy)

test_preds = rf.predict(test_X_flat)

test_accuracy = accuracy_score(test_y, test_preds)
print("Test Accuracy:", test_accuracy)
```

```
Validation Accuracy: 0.4307692307692308
Test Accuracy: 0.515625
```

```
Click here to ask Blackbox to help you code faster
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


models = {
    "Logistic Regression": LogisticRegression(),
    "SVM": SVC(probability=True),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(n_neighbors=5)
}

plt.figure(figsize=(10, 8))
for name, model in models.items():
    model.fit(train_X_flat, train_y)
    if hasattr(model, "predict_proba"):
        probas = model.predict_proba(test_X_flat)[:, 1]
    else:
        probas = model.decision_function(test_X_flat)
    fpr, tpr, _ = roc_curve(test_y, probas)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label='%s (AUC = %0.2f)' % (name, roc_auc))

plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random Guessing')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
💡 Click here to ask Blackbox to help you code faster
#### Confusion Matrices

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def plot_confusion_matrix(model_name, cm):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()

## Logistic Regression
logreg_cm = confusion_matrix(test_y, logreg.predict(test_X_flat))
plot_confusion_matrix('Logistic Regression', logreg_cm)

## SVM
svm_cm = confusion_matrix(test_y, svm.predict(test_X_flat))
plot_confusion_matrix('SVM', svm_cm)

## Random Forest
rf_cm = confusion_matrix(test_y, rf.predict(test_X_flat))
plot_confusion_matrix('Random Forest', rf_cm)

## KNN
knn_cm = confusion_matrix(test_y, knn.predict(test_X_flat))
plot_confusion_matrix('KNN', knn_cm)

```

```
💡 Click here to ask Blackbox to help you code faster
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt


test_accuracies = {}
for model_name, model in models.items():
    model.fit(train_X_flat, train_y)
    test_preds = model.predict(test_X_flat)
    test_accuracy = accuracy_score(test_y, test_preds)
    test_accuracies[model_name] = test_accuracy

plt.figure(figsize=(10, 6))
bars = plt.bar(test_accuracies.keys(), test_accuracies.values(), color=['skyblue', 'lightgreen', 'salmon', 'gold'])
plt.title('Model Test Accuracies for 50 Frames')
plt.ylabel('Accuracy')
plt.ylim(0, 1)

for bar in bars:
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height(),
             f'{bar.get_height():.2f}', ha='center', va='bottom')

plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

Codes for textual analysis

```
import pandas as pd
import numpy as np
import re
import unicodedata
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
df = pd.read_csv("binaryclass.csv", index_col=0)
df.head(5)
df.reset_index(inplace=True)
df.info()
df.head(5)
```

```
def remove_accented_characters(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text

def normalized(text):
    text = text.lower()
    text = re.sub(r'[\r|\n|\r\n]+', ' ', text)
    text = remove_accented_characters(text)
    return text

df["normalization"] = df.Transcription.apply(lambda x : normalized(x))
```

```
ngram_range = (1, 2)
percentile = 70
```

```
vectorizer = CountVectorizer(ngram_range=ngram_range)
X = vectorizer.fit_transform(df['normalization'])
```

```
from sklearn.feature_selection import SelectPercentile, chi2
y = df['Veracity']
selector = SelectPercentile(chi2, percentile=percentile)
X_selected = selector.fit_transform(X, y)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

```
import matplotlib.pyplot as plt


models = ['Logistic Regression', 'Random Forest', 'SVM']
accuracies = [logistic_accuracy, rf_accuracy, svm_accuracy]
colors = ['skyblue', 'lightgreen', 'lightcoral']


plt.figure(figsize=(10, 6))
bars = plt.bar(models, accuracies, color=colors)

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.01, round(yval, 2), ha='center', va='bottom', fontsize=10)

plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Comparison of Model Accuracies')
plt.ylim(0.0, 1.0)
plt.xticks(rotation=45)
plt.tight_layout()


legend_labels = [f'{model} ({round(acc, 2)})' for model, acc in zip(models, accuracies)]
plt.legend(bars, legend_labels, loc='upper right')

plt.grid(axis='y', linestyle='--')

plt.show()
```

```
Click here to ask Blackbox to help you code faster
import pandas as pd
import numpy as np
import spacy
import re
import unicodedata

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


```

```
Click here to ask Blackbox to help you code faster
# to see all the cells and data in the dataframes
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 300)
pd.set_option('display.max_colwidth', 300)

# to hide warnings about train/test size for train_test_split
import warnings
warnings.filterwarnings('ignore')
```

```
Click here to ask Blackbox to help you code faster
df = pd.read_csv("binaryclass.csv", index_col=0)
df.head(5)
```

```
Click here to ask Blackbox to help you code faster
def remove_accented_characters(text):
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8', 'ignore')
    return text
```

```
Click here to ask Blackbox to help you code faster
def normalized(text):
    text = text.lower()
    text = re.sub(r'[\r|\n|\r\n]+', ' ', text)
    text = remove_accented_characters(text)
    return text
```

```
Click here to ask Blackbox to help you code faster
string ="My best friend is a really nice person. Um. She's always kind to everyone. She continues to just be herself around everyone. Um. She has taught me so much throughout, like I've known he
```

```
Click here to ask Blackbox to help you code faster
normalized(string)
```

Codes for Vocal analysis

```
Click here to ask Blackbox to help you code faster
import os
from moviepy.editor import VideoFileClip

def convert_videos_to_audio(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)
    video_files = [file for file in os.listdir(input_folder) if file.endswith('.wmv')]

    for video_file in video_files:
        video_path = os.path.join(input_folder, video_file)
        audio_output_path = os.path.join(output_folder, f"{os.path.splitext(video_file)[0]}.mp3")
        video = VideoFileClip(video_path)
        audio = video.audio
        audio.write_audiofile(audio_output_path)
        print(f"Converted {video_file} to {os.path.basename(audio_output_path)}")

input_video_folder = r'E:\study material\Karan ML\voice\val_25'  # Path to video folder
output_audio_folder = r'E:\study material\Karan ML\voice\Train_val_test Split\val_audio'  # Output audio folder

convert_videos_to_audio(input_video_folder, output_audio_folder)
```

,

```
Click here to ask Blackbox to help you code faster
import os
from moviepy.editor import VideoFileClip

def convert_videos_to_audio(input_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)

    video_files = [file for file in os.listdir(input_folder) if file.endswith('.wmv')]

    for video_file in video_files:
        video_path = os.path.join(input_folder, video_file)
        audio_output_path = os.path.join(output_folder, f"{os.path.splitext(video_file)[0]}.mp3")
        video = VideoFileClip(video_path)
        audio = video.audio
        audio.write_audiofile(audio_output_path)
        print(f"Converted {video_file} to {os.path.basename(audio_output_path)}")
input_video_folder = r'E:\study material\Karan ML\voice\test_videos'  # Path to video folder
output_audio_folder = r'E:\study material\Karan ML\voice\Train_val_test Split\test_audio'  # Output audio folder

convert_videos_to_audio(input_video_folder, output_audio_folder)
```

Feature extraction using Librosa (MFCC)

```
Click here to ask Blackbox to help you code faster
import os
import numpy as np
import librosa

def extract_features(audio_file, max_length=100):
    audio, sample_rate = librosa.load(audio_file)

    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate)

    if mfccs.shape[1] < max_length:
        pad_width = max_length - mfccs.shape[1]
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant', constant_values=0)
    elif mfccs.shape[1] > max_length:
        mfccs = mfccs[:, :max_length]

    return mfccs

folder_path = r'E:\study material\Karan ML\voice\Train_val_test Split\test_audio'

all_features = []
```

```
Click here to ask Blackbox to help you code faster
# for excel file

import os
import numpy as np
import pandas as pd
import librosa

def extract_features(audio_file, max_length=100):

    audio, sample_rate = librosa.load(audio_file)

    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate)

    if mfccs.shape[1] < max_length:
        pad_width = max_length - mfccs.shape[1]
        mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)), mode='constant', constant_values=0)
    elif mfccs.shape[1] > max_length:
        mfccs = mfccs[:, :max_length]

    return mfccs

folder_path = r'E:\study material\Karan ML\voice\Train_val_test Split\train_audio'

all_features = []
all_labels = []

for root, dirs, files in os.walk(folder_path):
    for file in files:
        if file.endswith('.mp3'):  # Process only MP3 files
            audio_file = os.path.join(root, file)
            label = os.path.basename(root)  # Get label from folder name
            features = extract_features(audio_file, max_length=100)  # Adjust max_length as needed
            all_features.append(features.flatten())  # Flatten features
            all_labels.append(label)

all_features = np.array(all_features)
all_labels = np.array(all_labels)

# Create a pandas DataFrame for features and labels
df = pd.DataFrame(all_features)
df['Label'] = all_labels

# Save DataFrame to Excel file
df.to_excel('audio_features.xlsx', index=False)

```

```
Click here to ask Blackbox to help you code faster
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

train_label_file_path = r'E:\study material\Karan ML\voice\train_75.csv'
train_label_data = pd.read_csv(train_label_file_path)

test_label_file_path = r'E:\study material\Karan ML\voice\test_videos.csv'
test_label_data = pd.read_csv(test_label_file_path)

train_labels = train_label_data['Veracity'].values
test_labels = test_label_data['Veracity'].values

features = np.load(r'E:\study material\Karan ML\voice\train_features.npy')

samples, time_steps, num_features = features.shape
X_train = features.reshape(samples, -1)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

svm_clf = SVC(kernel='linear', random_state=42)

svm_clf.fit(X_train_scaled, train_labels)

X_test = np.load(r'E:\study material\Karan ML\voice\test_features.npy')
X_test = X_test.reshape(X_test.shape[0], -1)  # Flatten to (samples, time_steps * num_features)
X_test_scaled = scaler.transform(X_test)

y_pred = svm_clf.predict(X_test_scaled)

SVMaccuracy = accuracy_score(test_labels, y_pred)
report = classification_report(test_labels, y_pred)

print("SVM Accuracy:", SVMaccuracy)
print("Classification Report:")
print(report)
```

```
Click here to ask Blackbox to help you code faster
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the label file for training data
train_label_file_path = r'E:\study material\Karan ML\voice\train_75.csv'
train_label_data = pd.read_csv(train_label_file_path)

# Load the label file for test data
test_label_file_path = r'E:\study material\Karan ML\voice\test_videos.csv'
test_label_data = pd.read_csv(test_label_file_path)

# Assuming 'Veracity' column contains the labels (0 for truth, 1 for lie) in both files
train_labels = train_label_data['Veracity'].values
test_labels = test_label_data['Veracity'].values

# Load the pre-extracted features
features = np.load(r'E:\study material\Karan ML\voice\train_features.npy')


samples, time_steps, num_features = features.shape
X_train = features.reshape(samples, -1)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

knn_clf = KNeighborsClassifier(n_neighbors=5)

knn_clf.fit(X_train_scaled, train_labels)

# Prepare test features
test_features = np.load(r'E:\study material\Karan ML\voice\test_features.npy')
X_test = test_features.reshape(test_features.shape[0], -1)
X_test_scaled = scaler.transform(X_test)

# Make predictions on the test set
y_pred_knn = knn_clf.predict(X_test_scaled)

accuracy_knn = accuracy_score(test_labels, y_pred_knn)
report_knn = classification_report(test_labels, y_pred_knn)

print("KNN Accuracy:", accuracy_knn)
print("KNN Classification Report:")
print(report_knn)
```