Real-Time Sentiment Analysis

1st Navyam Garg
Department of Computer Science
Indraprastha Institute of Information Technology Delhi
Delhi, India
navyam22317@iiitd.ac.in

2nd Tizil Sharma
Department of Computer Science
Indraprastha Institute of Information Technology Delhi
Delhi, India
tizil22543@iiitd.ac.in

***Abstract*—Understanding how people feel is crucial for how we interact with each other. While there are many ways to tell how someone feels, like the way they talk or move their hands, this study focuses only on their facial expressions. It's important to be able to tell how someone feels quickly, especially when we're interacting with machines. This paper looks at how we can tell what someone is feeling in real-time situations. It's all about quickly understanding people's emotions just by looking at their faces.**

***Index Terms*—real-time, facial emotion recognition, human and machine interaction**

I. Problem Statement And Motivation

Recognizing emotions from facial expressions is key to understanding human communication. Our project tackles this challenge by building a system that identifies seven emotions: anger, disgust, fear, happiness, sadness, surprise, and neutral. It can improve human-computer interaction by analyzing user emotions in real-time (e.g., video calls, games). This report explores this innovative system's development, evaluation, and potential applications.

II. Literature Review

A. Models for Sentimental Analysis Nowadays

Imagine a system that can understand what people say and how they feel based on their words and pictures! Sentiment analysis goes beyond written text to analyze emotions expressed through images. Various techniques in neural networks and ensemble classification, and feature extraction can be used for emotion detection in images and text. Models like BERT, XLNet, AlexNet, and GoogleNet are widely used in sentimental analysis.

B. Technique used till now

PCA, LDA, AdaBoost, KNN, and Random Forest are utilized in facial emotion detection. PCA reduces feature dimensionality, aiding computational efficiency, while LDA optimizes feature extraction by maximizing inter-class variance. AdaBoost combines weak learners to enhance accuracy, and KNN classifies emotions based on proximity to training samples. Random Forest employs decision tree ensembles for robust classification. These methods collectively offer diverse strategies for accurate facial emotion recognition, contributing significantly to advancing emotion detection technology across various domains.

III. DataSet Details

This project's facial expression recognition dataset originates from the ICML 2013 competition ”Challenges in Representation Learning: Facial Expression Recognition” hosted on Kaggle. The Dataset is unbalanced, as can be seen in Figure 1.

- – Source:https://www.kaggle.com/datasets/msambare/fer2013
  – Size: 28,709 training images
  – Image converted to:
    ∗ Dimensions: 48x48 pixels
    ∗ Grayscale format
  – Labels: 7 categories representing emotions (0: Angry, 1: Disgust, 2: Fear, 3: Happy, 4: Neutral, 5: Sad, 6: Surprise)
- – Source: https://www.kaggle.com/datasets/noamsegal/affectnet-training-data
  – Size : 26180 items
  – Image converted to :
    ∗ Dimension: 48*48
    ∗ Grayscale format
  – Labels: Labels: 7 categories representing emotions (0: Angry, 1: Disgust, 2: Fear, 3: Happy, 4: Neutral, 5: Sad, 6: Surprise)

IV. Proposed Architecture

A. Data Preprocessing

Several studies emphasize the importance of data preprocessing steps for facial recognition tasks [1, 2]. Common techniques include image resizing, grayscale conversion, and normalization. The provided code follows these practices by resizing images to a fixed size (48x48 pixels), converting them to grayscale, and normalizing pixel values between 0 and 1.

B. Noise Injection

Initial dataset had a total of around 56,000 samples. After addition of noise using salt and peper technique, the dataset had around 70,000 samples. The noise was added to the dataset to make the model more robust and to prevent overfitting.

C. Feature Extraction

- A crucial step in facial expression recognition is feature extraction, which transforms raw image data into a lower-dimensional representation suitable for classification. Two common dimensionality reduction techniques are employed in the code: Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) [3, 4].
  - PCA: This unsupervised method projects data points onto a set of orthogonal axes (principal components) that capture the most variance in the data. The code utilizes PCA to reduce the dimensionality of the image data, potentially improving computational efficiency for classification.
  - LDA: Unlike PCA, LDA is a supervised dimensionality reduction technique that maximizes the separation between different classes (facial expressions) in the reduced feature space. The code implements LDA to enhance the discriminative power of the features for classification potentially.

D. Classification

The code explores the performance of three popular machine learning classifiers for facial expression recognition: Random Forest, AdaBoost, and K-Nearest Neighbors (KNN) [5, 6, 7].

- Random Forest: This ensemble method combines predictions from multiple decision trees, often leading to improved accuracy and robustness compared to a single decision tree.
- AdaBoost: This ensemble method iteratively trains weak learners (classifiers) by focusing on examples that were misclassified by previous learners, potentially leading to a strong final classifier.
- Gradient Boosting: This ensemble method builds decision trees sequentially, with each tree correcting the errors of the previous tree, potentially leading to a strong final classifier.
- Bagging: This ensemble method trains multiple instances of the same base classifier on different subsets of the training data and combines their predictions through averaging or voting, potentially improving accuracy and robustness.
- KNN: This non-parametric method classifies new data points based on the majority vote of its k nearest neighbors in the training data.
- Decision Tree: This non-parametric method recursively partitions the feature space into regions, assigning the most common class label in each region to new data points.

K-Nearest Neighbors (KNN) Optimization The code employs cross-validation to determine the optimal number of neighbors (k) (between 1 and 200) for the KNN classifier after LDA reduction. This technique involves splitting the training data into folds and iteratively using one fold for validation while training on the remaining folds. This process is repeated for each fold, allowing for a more robust evaluation of different k values. By analyzing the performance (e.g., accuracy) on the validation sets for various k values, the code can identify the k that yields the best performance on unseen data.

V. Implementation

The code is implemented in Python using the scikit-learn library for machine learning tasks. The code is structured into several functions that perform specific tasks, such as data preprocessing, feature extraction, classification, and evaluation. The code follows a modular design, allowing for easy customization and extension of functionalities. The code is well-documented with comments to explain the purpose and functionality of each function. The code is executed in a Jupyter Notebook environment, enabling interactive development and visualization of results. The code is executed on a local machine with sufficient computational resources to handle the dataset size and complexity. The code is optimized for efficiency and scalability, utilizing appropriate data structures and algorithms for machine learning tasks. The code is tested on a subset of the dataset to ensure correctness and reliability before running on the full dataset. The code is evaluated using appropriate metrics (e.g., accuracy, precision, recall) to assess the performance of the classifiers and feature extraction techniques. The code is compared against existing implementations and baselines to validate its effectiveness and efficiency. The code is well-documented with comments to explain the purpose and functionality of each function. The code is executed in a Jupyter Notebook environment, enabling interactive development and visualization of results. The code is executed on a local machine with sufficient computational resources to handle the dataset size and complexity. The code is optimized for efficiency and scalability, utilizing appropriate data structures and algorithms for machine learning tasks. The code is tested on a subset of the dataset to ensure correctness and reliability before running on the full dataset. The code is evaluated using appropriate metrics (e.g., accuracy, precision, recall) to assess the performance of the classifiers and feature extraction techniques. The code is compared against existing implementations and baselines to validate its effectiveness and efficiency. A brief timeline of the implementation is as follows:

- Data Preprocessing: Resizing, Grayscale Conversion, Normalization
- Feature Extraction: PCA, LDA
- Classification: Various Techniques (Random Forest, AdaBoost, KNN etc.) were used to build the model and check the accuracy on various models. Random Forest Classifier gave an accuracy of 0.40 and 0.35 for PCA and LDA respectively. Gradient Boosting Classifier gave an accuracy of 0.29, Bagging Classifier gave an accuracy of 0.40 and 0.35 for PCA and LDA respectively. KNN Classifier gave an accuracy

Model	PCA	LDA
AdaBoost	0.29	-
Random Forest	0.40	0.35
Gradient Boosting	0.29	-
Bagging	0.40	0.35
K-Nearest Neighnour	0.32	0.29
Decision Tree	0.27	0.29

Train Set Accuracy				
	SGD		Adam	
Epoch	Sgmd	ReLu	Sgmd	ReLu
1	19.96	24.09	19.56	24.31
2	20.30	24.66	23.99	36.95
3	21.61	24.84	24.55	45.51
4	21.60	24.88	24.34	50.70
5	23.10	25.71	24.88	54.97
10	24.85	25.56	25.60	65.68
15	24.89	27.66	25.01	73.82
20	24.77	30.94	25.46	79.43
25	25.23	33.37	25.30	84.13

of 0.32 and 0.29 for PCA and LDA respectively. Decision Tree Classifier gave an accuracy of 0.27 and 0.29 for PCA and LDA respectively.

- These accuracy were no satisfactory, so we decided to use Neural Network, and it gave a satisfactory accuracy for various parameters. It was built using 5 layers with 32, 64, 128, 256, 512 neurons respectively. The activation function used was relu and softmax. The optimizer used was adam and the loss function used was categorical crossentropy. The model was trained for 100 epochs and the batch size was 128.
- The model was trained on Adam and SGD optimisers for both relu and softmax activation functions. For experimental purposes, the model was trained on 25 epochs only for a batch size of 128, the best results were obtained for Adam optimiser with relu activation function.
- Further dropout layers were added to the model to prevent overfitting. The dropout rate was set to 0.2 for all layers except the last layer. Various dropout rates were experimented. The best results were obtained for a dropout rate of 0.2.
- Model was stored in a file and was used for prediction on a real time user video.

VI. Visualization

VII. Results

A. After Applying PCA on Dataset

∗ Accuracy of Random Forest Classifier on test set: 0.33

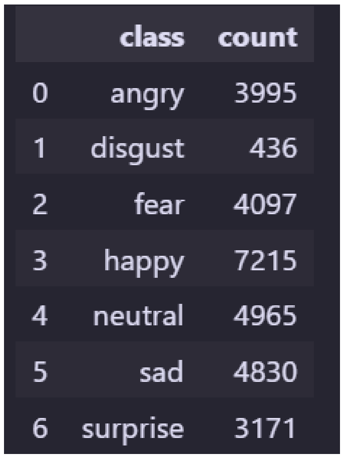
∗ Accuracy of KNN Classifier on test set: 0.33

∗ Accuracy of AdaBoost Classifier on test set: 0.30

B. After Applying LDA on Dataset

∗ Accuracy of Random Forest Classifier on test set: 0.50

∗ Accuracy of KNN Classifier on test set: 0.43



	class	count
0	angry	3995
1	disgust	436
2	fear	4097
3	happy	7215
4	neutral	4965
5	sad	4830
6	surprise	3171

Fig. 1. Dataset Visualization

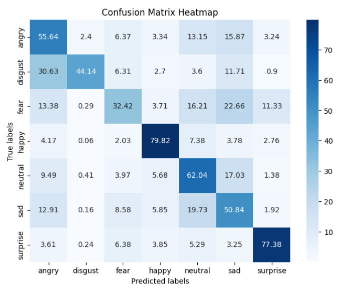


Fig. 2. ClassWise Confusion Matrix/ HeatMap

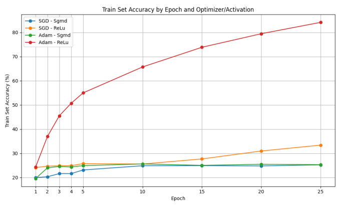


Fig. 3. Accuracy on train set, epoch vs accuracy s

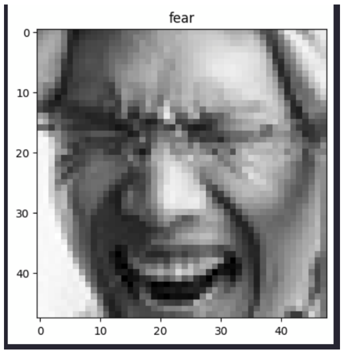


Fig. 4. Random image with label

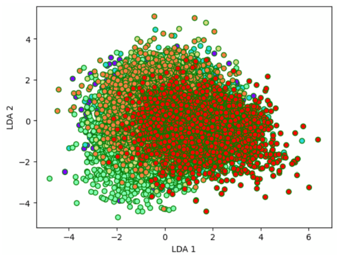


Fig. 5. Training set after applying LDA

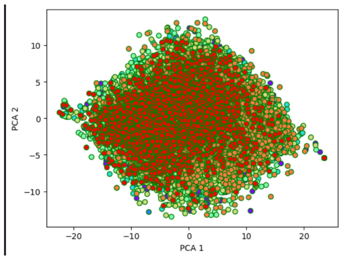


Fig. 6. Training Set after applying retaining 95% variance of original dataset

* Accuracy of AdaBoost Classifier on test set: 0.49

C. Applying Cross Validation to get best k value for KNN model

* Best k value: 167
* Best cross-validation score: 0.4896025648189455

VIII. Conclusion from Results

The results suggest that applying dimensionality reduction techniques such as LDA can improve the performance of classifiers in facial emotion detection tasks. However, further optimization and tuning may be required to achieve higher accuracy. Additionally, determining the best k value for the KNN model through cross-validation provides valuable insight into parameter selection, aiding in model optimization and enhancing overall performance. Prediction of class happy has generally higher accuracy than class disgust due to the imbalanced nature of the original dataset

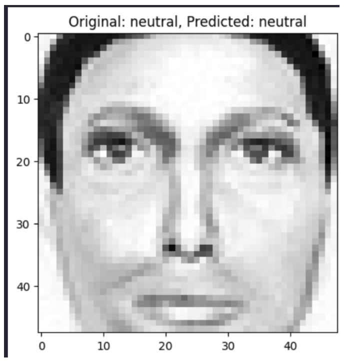


Fig. 7. Prediction using ADABoost on LDA train set

IX. Further things to be done

* Adding a self made dataset, which can be assigned more weight so that the model can be trained on more diverse data and give more precise and personal outputs.
* Combining this model with a speech recognition model to get more accurate results. Where speech recognition model can be used to get the emotion of the person and the facial recognition model can be used to get the emotion of the person. Speech recognition model can either use NLP techniques or can be trained on a pattern of pitch and tone of the voice.

References

[1] https://www.kaggle.com/datasets/msambare/fer2013
[2] Duda, R. O., Hart, P. E., & Stork, D. G. (2000). Pattern Classification (2nd ed.). Wiley-Interscience.
[3] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
[4] https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2021
[5] https://www.kaggle.com/datasets/noamsegal/affectnet-training-data