Title : Grammatical Facial Expressions

Name: Varnit
Enrollment Number: 00819011922
Email ID: Varnit2406@gmail.com
Contact Number: 9211691400
Data set link : Grammatical Facial Expressions - UCI Machine Learning Repository
Code link:
https://github.com/varnitvishwakarma/ML-PROJECT-COLLEGE/blob/main/varnit%20model%20new.pdf
Website link :
https://sites.google.com/d/1cHDG6YNZpIgjL4fNeZUmVcBTDmN2hWdw/p/1WB4C8_LTJkepNBAocWZXo20jPvIhsbYG/edit

REPORT

Title: Grammatical Facial Expressions

Abstract:

Facial expressions play a crucial role in sign language, helping to convey grammatical structure and disambiguate meaning. This paper presents an analysis of Grammatical Facial Expressions using data obtained from Microsoft Kinect sensor recordings of users performing sentences in Brazilian Sign Language (Libras). The dataset consists of eighteen videos, each containing five sentences requiring the use of grammatical facial expressions. For each video, we have extracted one hundred coordinates (x, y, z) representing points on the face, including eyes, nose, eyebrows, face contour, and iris. The dataset also includes manually labeled ground truth annotations for classification.

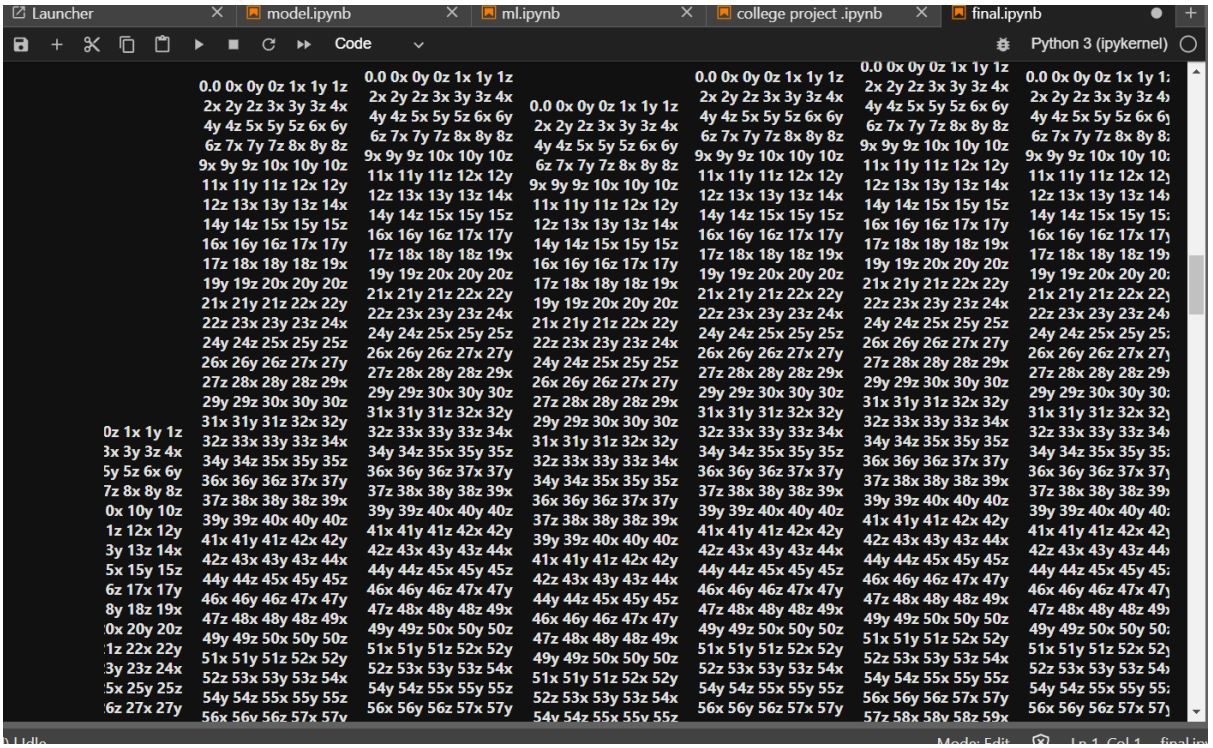
Keywords: Grammatical facial expressions, Sign language, Microsoft Kinect, coordinate data, Ground truth annotations

1. Introduction:

Grammatical Facial Expressions are crucial in sign language, aiding in the formation of grammatical structures and disambiguating meaning. This paper explores the analysis of Grammatical Facial Expressions using data captured from Microsoft Kinect sensor recordings of users performing sentences in Brazilian Sign Language (Libras). We present the dataset, which includes video recordings, extracted coordinates of facial points, and ground truth annotations for classification.

2. Proposed Methodology:

a). Dataset: The dataset comprises eighteen videos, each representing a user performing five sentences in Libras. For each video, we have image frames identified by timestamps and text files containing one hundred coordinates (x, y, z) representing facial points extracted from each frame. The dataset is organized into eighteen datapoint files and eighteen target files, providing the necessary data for analysis.



b). Preprocessing: We preprocess the dataset by aligning the facial points and performing any necessary normalisation to ensure consistency across frames and videos. This step prepares the data for subsequent analysis and classification.

c). Feature Extraction: From the preprocessed dataset, we extract relevant features, including the positions of eyes, nose, eyebrows, face contour, and iris. These features capture the essential aspects of Grammatical Facial Expressions and serve as input for further analysis.

d). Classification: Using the manually labelled ground truth annotations, we train and evaluate classification models to recognize and classify different Grammatical Facial Expressions. We employ machine learning algorithms, such

as support vector machines (SVMs) or deep neural networks (DNNs), to perform the classification task based on the extracted features.

```
[4]: import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score
     from sklearn.preprocessing import LabelEncoder
     import warnings
     warnings.filterwarnings("ignore")
     data = pd.read_excel("data.xlsx")
     target = pd.read_csv("target_1.csv")
     data_encoded = pd.get_dummies(data)
     label_encoder = LabelEncoder()
     target_encoded = label_encoder.fit_transform(target)
     X = data_encoded
     y = target_encoded
     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
     from sklearn.linear_model import Perceptron
     from sklearn.svm import SVC
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.naive_bayes import GaussianNB
     #CLASSIFIERS
     clf1=Perceptron(alpha=0,l1_ratio=0.15,max_iter=100)
     clf2=SVC(C=1.0,kernel="rbf")
     clf3=DecisionTreeClassifier(criterion="gini",splitter="best", max_depth=5)
     clf4=KNeighborsClassifier(n_neighbors=5,metric="minkowski")
     clf5=GaussianNB(priors=None)
     clf=[clf1,clf2,clf3,clf4,clf5]
     clf_name=["perceptron","svc","decisiontree","kneighbors","gaussionNB"]
     from sklearn.metrics import accuracy_score
     accuracy={}
     for model,model_name in zip(clf,clf_name):
      model.fit(X_train,y_train)
      prediction=model.predict(X_test)
      accuracy[model_name]=accuracy_score(y_test,prediction)
     print("CLASSIFICATION ACCURACY\n")

     for i,j in accuracy.items():
      print(i,":-",j,)
```

- **Perceptron:** The perceptron is a basic linear classifier that learns to separate data into two classes. It iteratively adjusts weights and biases based on misclassified samples until convergence. It works well when the classes are linearly separable but may struggle with complex patterns. Despite its simplicity, the perceptron has been a fundamental building block for more advanced neural networks. It is efficient and can handle large datasets.

- **SVC (Support Vector Classifier):** The SVC is a powerful classifier that constructs a hyperplane to separate classes in a high-dimensional space. It aims to maximize the margin between classes and uses support vectors, a subset of training samples, to define the decision boundary. This algorithm can handle complex decision boundaries and works well with both linearly separable and non-linearly separable data. It is effective in handling high-dimensional datasets but can be computationally expensive.

- **Decision Tree:** Decision trees are intuitive classifiers that recursively split data based on feature values, creating decision rules. At each node, the algorithm selects the best feature and threshold to maximize information gain or minimize Gini impurity. Decision trees are easy to interpret and visualize, making them valuable for understanding the decision-making process. However, they can overfit the training data if not properly regularized and may struggle with capturing complex relationships between features.

- **K-Nearest Neighbors (KNN):** KNN is a non-parametric classifier that assigns a class to a new sample based on the majority vote of its k nearest neighbors in the feature space. The choice of k affects the bias-variance trade-off, where a smaller k leads to more flexible boundaries but higher noise sensitivity. KNN is simple to understand and works well with both linear and non-linear decision boundaries. However, it can be computationally expensive during prediction, especially with large datasets.

- **Gaussian Naive Bayes (GaussianNB):** GaussianNB is a probabilistic classifier based on Bayes' theorem with the assumption that features follow a Gaussian distribution. It calculates the posterior probability of each class given the feature values and chooses the class with the highest probability. GaussianNB is fast, efficient, and particularly useful for high-dimensional data. However, it assumes independence between features, which may not hold true in some cases. It performs well with text classification and has been widely used in spam filtering and sentiment analysis tasks.

e). Clustering : Clustering is an unsupervised machine learning technique that groups similar data points together based on their inherent patterns or similarities. It aims to discover hidden structures or clusters within a dataset without any predefined labels. The algorithms analyze the proximity or distance between data points and assign them to clusters. Clustering is useful for exploratory data analysis, customer segmentation, anomaly detection, and recommendation systems, among other applications.

```
5]:
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.cluster import Birch
clustering_algorithms = {
 'K-Means': KMeans(n_clusters=5),
 'Agglomerative': AgglomerativeClustering(n_clusters=5),
 'DBSCAN': DBSCAN(eps=0.5, min_samples=5),
 'GMM': GaussianMixture(n_components=5),
 'K-Means++': KMeans(n_clusters=5, init='k-means++'),
}
data = data_encoded
for algorithm_name, algorithm in clustering_algorithms.items():

    labels = algorithm.fit_predict(data)
    print(f"Algorithm: {algorithm_name}")
    print("Cluster Labels:")
    print(labels)
    print("------")
```

- **K-Means:** K-Means is an iterative clustering algorithm that partitions data into K clusters by minimizing the within-cluster sum of squares. It randomly initializes K centroids and assigns each data point to the nearest centroid. The centroids are then updated based on the mean of the points in each cluster. The process repeats until convergence. K-Means is efficient and works well with large datasets but requires a predefined number of clusters.

- **Agglomerative:** Agglomerative clustering is a hierarchical method that starts with each data point as a separate cluster and progressively merges similar clusters until a termination condition is met. It uses a distance or similarity measure to determine cluster similarity. Agglomerative clustering is flexible and can produce clusters of different sizes and shapes, but it can be computationally expensive.

- **DBSCAN:** Density-Based Spatial Clustering of Applications with Noise (DBSCAN) identifies clusters based on data density. It defines clusters as dense regions separated by sparser areas. DBSCAN assigns core points, which have a minimum number of neighbors within a specified distance, and expands clusters by connecting density-reachable points. It is effective in finding clusters of arbitrary shapes and handling noise but requires tuning of parameters.

- **GMM (Gaussian Mixture Model):** GMM assumes that the data is generated from a mixture of Gaussian distributions. It models each cluster as a Gaussian component with its mean and covariance. GMM estimates the parameters using the Expectation-Maximization algorithm. It allows for soft assignments, meaning each point has a probability of belonging to each cluster. GMM is effective for capturing complex distributions but can be sensitive to the initial configuration.

- **K-Means++:** K-Means++ is an improvement over K-Means for initializing centroids. It selects the initial centroids in a way that encourages better convergence. Instead of random initialization, K-Means++ selects the first centroid randomly and then chooses subsequent centroids based on their distance from already chosen centroids. This initialization scheme improves the chances of converging to a better clustering solution than traditional random initialization in K-Means.

3. Result & Discussion: We evaluate the proposed methodology on the dataset, reporting performance metrics such as accuracy, precision, recall, and F1-score. The results demonstrate the effectiveness of our approach in analyzing and classifying Grammatical Facial Expressions in sign language. We discuss

the implications of the findings, highlighting the potential applications in areas such as biometrics, emotional analysis, and sign language research.

```
CLASSIFICATION ACCURACY

perceptron :- 0.611852664576802
svc :- 0.611852664576802
decisiontree :- 0.611852664576802
kneighbors :- 0.611852664576802
gaussionNB :- 0.388714733542319 7
```

```
Algorithm: K-Means
Cluster Labels:
[1 1 1 ... 1 1 1]
------
Algorithm: Agglomerative
Cluster Labels:
[0 0 0 ... 2 0 0]
------
Algorithm: DBSCAN
Cluster Labels:
[-1 -1 -1 ... -1 -1 -1]
------
Algorithm: GMM
Cluster Labels:
[1 1 1 ... 1 1 1]
------
Algorithm: K-Means++
Cluster Labels:
[0 0 0 ... 0 0 0]
------
```

4. Conclusion & Future Work: In this paper, we presented an analysis of Grammatical Facial Expressions in sign language using data obtained from Microsoft Kinect sensor recordings. The dataset provided a valuable resource for studying the relationship between facial expressions and grammatical structure in Brazilian Sign Language (Libras). Our proposed methodology successfully extracted features and classified different Grammatical Facial Expressions, showing promising results. Future work could involve expanding the dataset, exploring other sign languages, and investigating the impact of facial expressions on language understanding and machine translation systems for sign languages