Hand Gesture Recognition Database

Author: Irfan Ullah Khan

GITHUB PROFILE KAGGLE PROFILE LINKEDIN PROFILE

YOUTUBE PROFILE EMAIL CONTACT ME WEBSITE CONTACT ME

Objectives

- View the data as an image
- Train different classifiers
- Compare performance for different classifiers using various metrics

Dataset

Dataset description

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Exploration

```
#reading csv file
df=pd.read_csv('/content/sign_mnist_train.csv')
```

```
df.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775
0	3	107	118	127	134	139	143	146	150	153	...	207.0
1	6	155	157	156	156	156	157	156	158	158	...	69.0
2	2	187	188	188	187	187	186	187	188	187	...	202.0
3	2	211	211	212	212	211	210	211	210	210	...	235.0
4	13	164	167	170	172	176	179	180	184	185	...	92.0

5 rows × 785 columns

```
#shape of the data
df.shape
```

```
(1380, 785)
```

```
df.columns
```

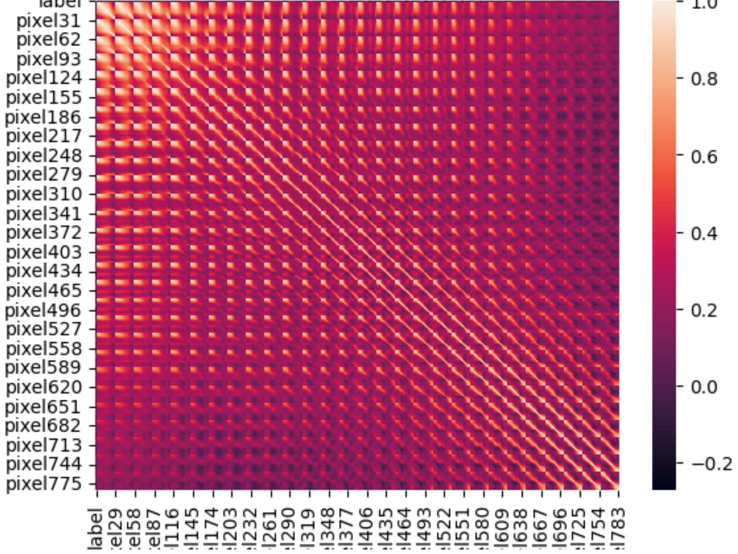
```
Index(['label', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6',
       'pixel7', 'pixel8', 'pixel9',
       ...
       'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779', 'pixel780',
       'pixel781', 'pixel782', 'pixel783', 'pixel784'],
      dtype='object', length=785)
```

```
df.isnull().values.any()#finding null values
```

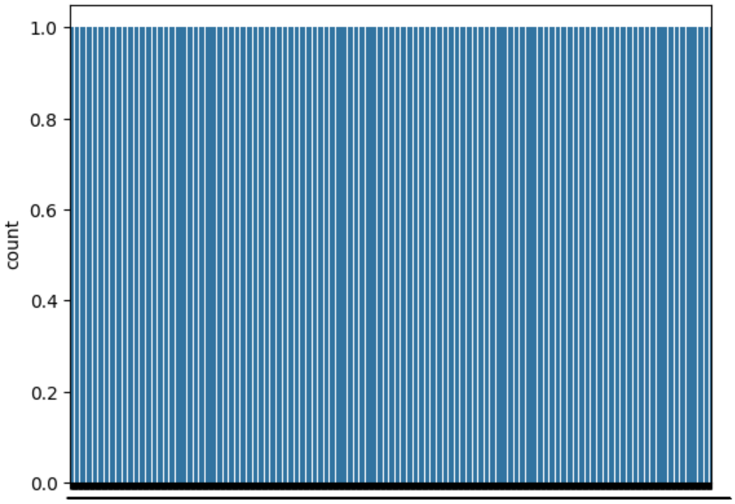
```
True
```

```
#defining corelation using heat map
corr_m = df.corr()
sns.heatmap(corr_m)
```

```
<Axes: >
```



```
#plotting he total number of each type of label in data
sns.countplot(df['label'])
plt.show()
```



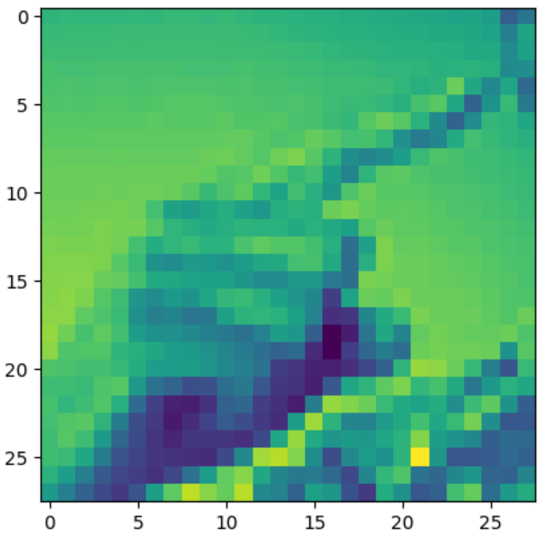
```
X = df.iloc[:,1:]
Y = df.iloc[:,0]
```

```
# print(X)
print(Y)
```

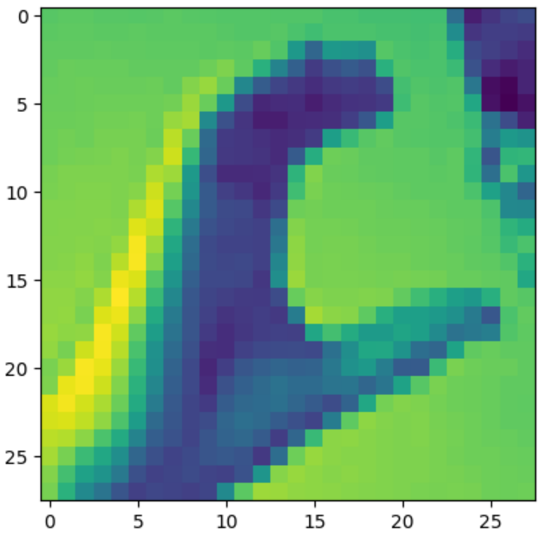
```
     0        3
     1        6
     2        2
     3        2
     4       13
             ..
     1375    23
     1376    22
     1377    20
     1378     7
     1379    22
     Name: label, Length: 1380, dtype: int64
```

Forming pictures from pixels

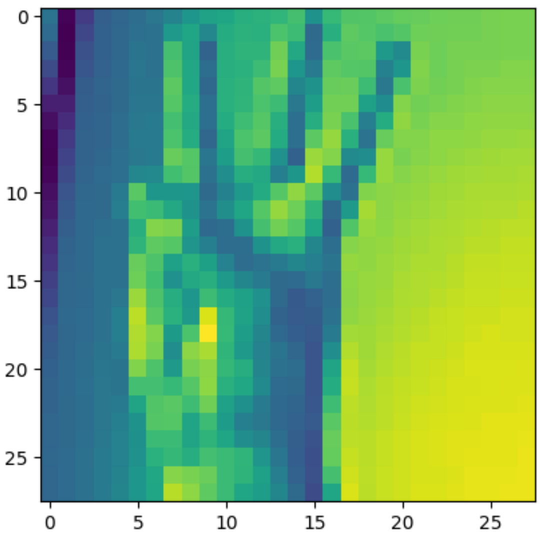
```
first = X.iloc[1,:]
# print(first)
first = np.array(first , dtype='float')
pixel = first.reshape((28,28))
plt.imshow(pixel)
plt.show()
```



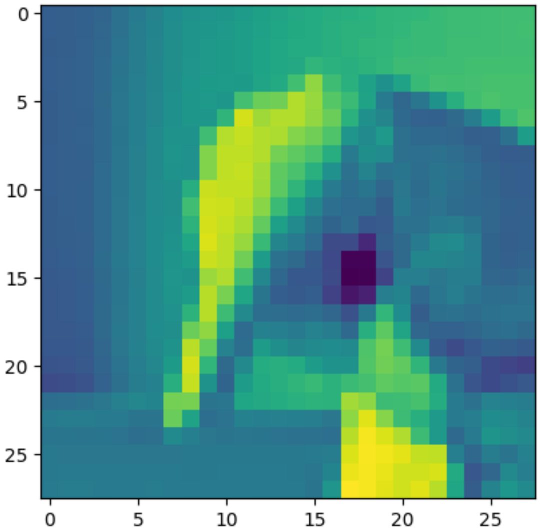
```
second = X.iloc[2,:]
second = np.array(second , dtype='float')
pixel2 = second.reshape((28,28))
plt.imshow(pixel2)
plt.show()
```



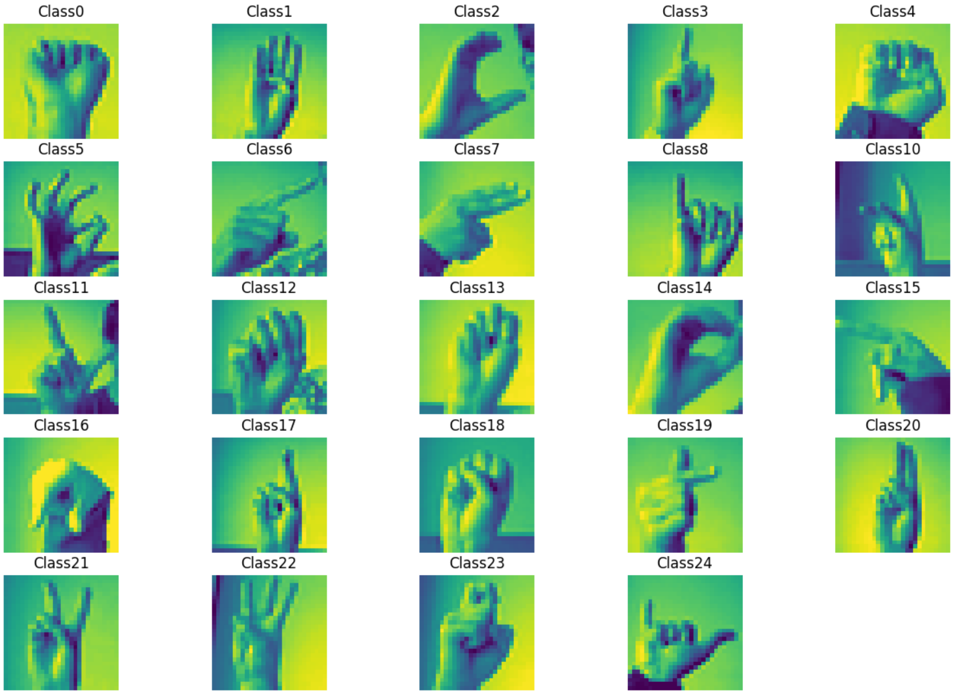
```
third = X.iloc[7,:]
third = np.array(third , dtype='float')
pixel3 = third.reshape((28,28))
plt.imshow(pixel3)
plt.show()
```



```
fourth = X.iloc[15,:]
fourth = np.array(fourth , dtype='float')
pixel4 = fourth.reshape((28,28))
plt.imshow(pixel4)
plt.show()
```



```
plt.figure(figsize=(15,10))
k = 0
for i in range(26):
    if(i==9 or i==25):
        continue
    else:
        plt.subplot(5,5,k+1)
        img=df[df.label==i].iloc[0,1:].values
        img=img.reshape((28,28))
        plt.imshow(img)
        plt.title("Class" + str(i))
        plt.axis('off')
        k=k+1
plt.show()
```



Splitting the Data

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.2,random_state = 0)
```

1.KNN

```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
```

```
from sklearn.neighbors import KNeighborsClassifier
#instantiate
classifier = KNeighborsClassifier()
#fitting the data
classifier.fit(X_train,Y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
#predict
Y_pred=classifier.predict(X_test)
```

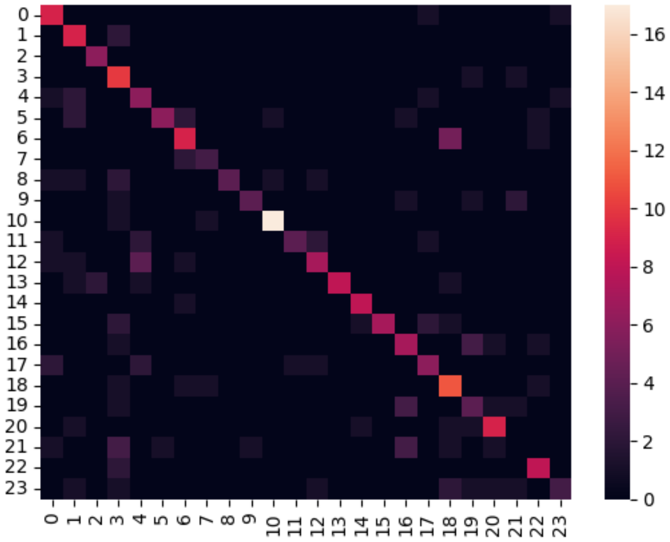
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but KNeighborsClassifier was fitted without featur
  warnings.warn(
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(Y_test,Y_pred)
```

```
sns.heatmap(cm)
```

```
<Axes: >
```



```
from sklearn.metrics import accuracy_score
```

```
#accuracy score
ascore=accuracy_score(Y_test , Y_pred , normalize=True)
print(ascore)
```

```
0.5978260869565217
```

```
from sklearn.metrics import f1_score
#f1_score
score=f1_score(Y_pred,Y_test,average='micro')
print(score)
```

```
0.5978260869565217
```

2.Logistic Regression

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown ir
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
  n_iter_i = _check_optimize_result(
```

```
▾ LogisticRegression
LogisticRegression()
```

```
Y_pred = classifier.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature
  warnings.warn(
```

```
ascore1=accuracy_score(Y_test , Y_pred , normalize=True)
print(ascore1)
```

```
0.782608695652174
```

```
score1=f1_score(Y_pred,Y_test,average='micro')
print(score1)
```

```
0.782608695652174
```

3.SVM

```
from sklearn.svm import SVC
#instantiate
svc = SVC()
#fiting the data
svc.fit(X_train , Y_train)
```

```
▾ SVC
SVC()
```

```
#predict
sv_pred = svc.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SVC was fitted without feature names
  warnings.warn(
```

```
ascore3=accuracy_score(Y_test , sv_pred, normalize=True)
print(ascore3)
```

```
0.7608695652173914
```

```
score3=f1_score(Y_pred,sv_pred,average='weighted')
print(score3)
```

```
0.7682431925529398
```

4.Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
```

```
#instantiate
obj = GaussianNB()
```

```
#fitting the data
obj.fit(X_train,Y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
#predict
Y_pred = obj.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but GaussianNB was fitted without feature names
  warnings.warn(
```

```
ascore4=accuracy_score(Y_test,Y_pred, normalize=True)
```

```
ascore4=accuracy_score(Y_test,Y_pred, normalize=True)
print(ascore4)
```

```
0.37318840579710144
```

```
score4=f1_score(Y_pred,Y_test,average='micro')
print(score4)
```

```
0.37318840579710144
```

5.MultinomialNB

```
from sklearn.naive_bayes import MultinomialNB
#instantiate
ob = MultinomialNB()
#fitting the data
ob.fit(X_train,Y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
#predict
Y_pred = ob.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but MultinomialNB was fitted without feature names
  warnings.warn(
```

```
ascore5=accuracy_score(Y_test,Y_pred, normalize=True)
print(ascore5)
```

```
0.5471014492753623
```

```
score5=f1_score(Y_pred,Y_test,average='micro')
print(score5)
```

```
0.5471014492753623
```

6.Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
# instantiate
dtc = DecisionTreeClassifier()
```

```
# fitting the data
dtc.fit(X_train, Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# predict
Y_pred = dtc.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but DecisionTreeClassifier was fitted without feat
  warnings.warn(
```

```
#accuracy
ascore6=accuracy_score(Y_test,Y_pred)
print(ascore6)
```

```
0.391304347826087
```

```
# f1 score
score6 = f1_score(Y_pred, Y_test,average='weighted')
print(score6)
```

```
0.40224762262081304
```

7.RandomForest

```
from sklearn.ensemble import RandomForestClassifier
```

```
#instantiate
rc = RandomForestClassifier()
#fitting the data
rc.fit(X_train , Y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
#predict
rc_pred = rc.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fitted without feat
  warnings.warn(
```

```
ascore2=accuracy_score(Y_test , rc_pred)
print(ascore2)
```

```
0.7463768115942029
```

```
score2=f1_score(Y_pred,Y_test,average='micro')
print(score2)
```

```
0.391304347826087
```

Conclusion

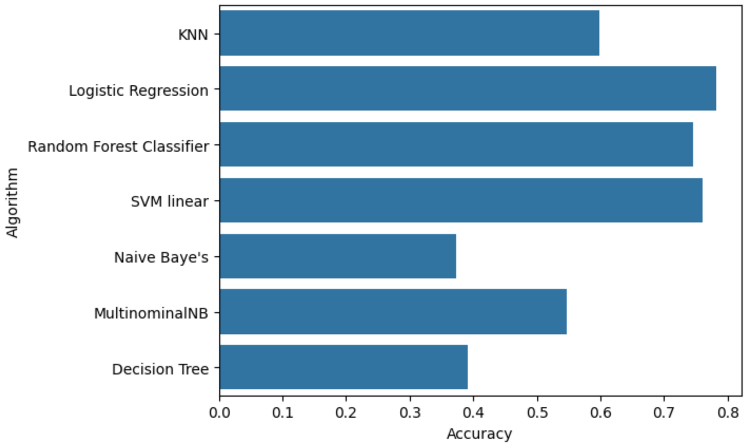
By the Implemented of 6 Algorithms ,now we can compare the performance of them.

```
Accuracy = [ascore,ascore1,ascore2,ascore3,ascore4,ascore5,ascore6]
data1 = {
    'Accuracy':Accuracy,
    'Algorithm': ['KNN','Logistic Regression','Random Forest Classifier','SVM linear',"Naive Baye's","MultinominalNB",'Decision Tree']}

df1 = pd.DataFrame(data1)
```

```
F1_score = [score,score1,score2,score3,score4,score5,score6]
data2 = {
    'F1_score':F1_score,
    'Algorithm': ['KNN','Logistic Regrss,ion','Random Forest Classifier','SVM linear',"Naive Baye's","MultinominalNB",'Decision Tree']}

df2 = pd.DataFrame(data2)
```

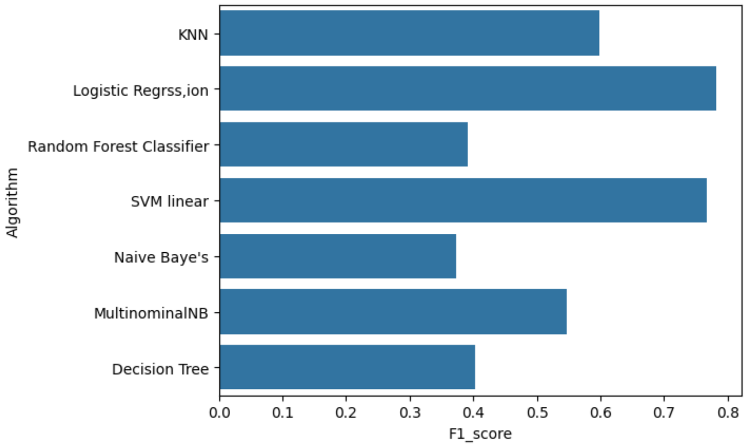
```
sns.barplot(x = df1.Accuracy, y = df1.Algorithm)
```

```
<Axes: xlabel='Accuracy', ylabel='Algorithm'>
```



```
sns.barplot(x = df2.F1_score, y = df2.Algorithm)
```

```
<Axes: xlabel='F1_score', ylabel='Algorithm'>
```



Now we can say that the best model in the above 6 models on the basis of accuracy is the logistic regression model