```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/digit-recognizer/sample_submission.csv
/kaggle/input/digit-recognizer/train.csv
/kaggle/input/digit-recognizer/test.csv
```

```python
df_train = pd.read_csv("/kaggle/input/digit-recognizer/train.csv")
df_test = pd.read_csv("/kaggle/input/digit-recognizer/test.csv")
```

```python
df_train
```

```
       label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6
pixel7  \
0          1       0       0       0       0       0       0       0
0
1          0       0       0       0       0       0       0       0
0
2          1       0       0       0       0       0       0       0
0
3          4       0       0       0       0       0       0       0
0
4          0       0       0       0       0       0       0       0
0
...      ...     ...     ...     ...     ...     ...     ...     ...
...
41995      0       0       0       0       0       0       0       0
0
41996      1       0       0       0       0       0       0       0
0
41997      7       0       0       0       0       0       0       0
0
41998      6       0       0       0       0       0       0       0
0
41999      9       0       0       0       0       0       0       0
0

       pixel8  ...  pixel774  pixel775  pixel776  pixel777
pixel778  \
```

```
0            0  ...         0         0         0         0         0

1            0  ...         0         0         0         0         0

2            0  ...         0         0         0         0         0

3            0  ...         0         0         0         0         0

4            0  ...         0         0         0         0         0

...        ...  ...       ...       ...       ...       ...       ...

41995        0  ...         0         0         0         0         0

41996        0  ...         0         0         0         0         0

41997        0  ...         0         0         0         0         0

41998        0  ...         0         0         0         0         0

41999        0  ...         0         0         0         0         0


       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
...         ...       ...       ...       ...       ...
41995         0         0         0         0         0
41996         0         0         0         0         0
41997         0         0         0         0         0
41998         0         0         0         0         0
41999         0         0         0         0         0

[42000 rows x 785 columns]
```

```
df_train.label.unique()
```

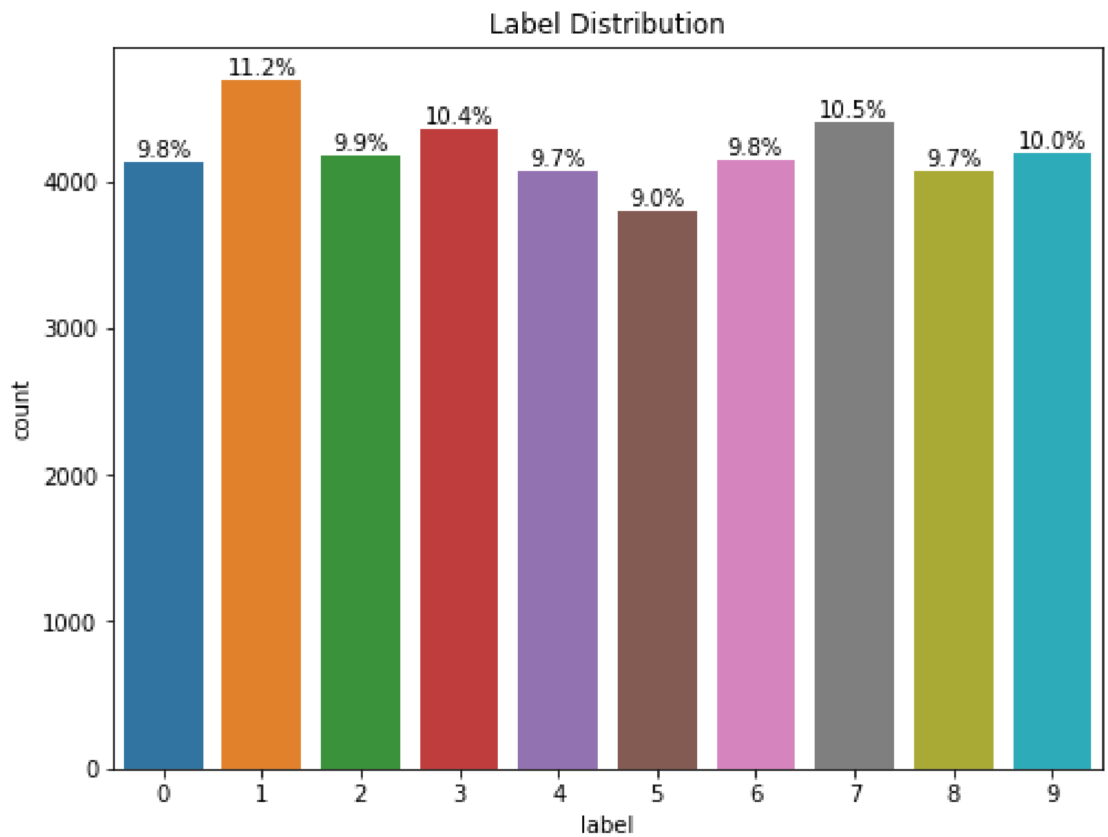
```
array([1, 0, 4, 7, 3, 5, 8, 9, 2, 6])
```

Explanatory Data Analysis

```
plt.figure(figsize=(8,6))
ax = sns.countplot(x='label',data=df_train)

plt.title("Label Distribution")
total= len(df_train.label)
```

```
for p in ax.patches:
    percentage = f'{100 * p.get_height() / total:.1f}%\n'
    x = p.get_x() + p.get_width() / 2
    y = p.get_height()
    ax.annotate(percentage, (x, y), ha='center', va='center')
```




```
df_train.describe()
```

```
               label   pixel0   pixel1   pixel2   pixel3   pixel4
pixel5  \
count  42000.000000  42000.0  42000.0  42000.0  42000.0  42000.0
42000.0
mean       4.456643      0.0      0.0      0.0      0.0      0.0
0.0
std        2.887730      0.0      0.0      0.0      0.0      0.0
0.0
min        0.000000      0.0      0.0      0.0      0.0      0.0
0.0
25%        2.000000      0.0      0.0      0.0      0.0      0.0
0.0
50%        4.000000      0.0      0.0      0.0      0.0      0.0
0.0
75%        7.000000      0.0      0.0      0.0      0.0      0.0
0.0
max        9.000000      0.0      0.0      0.0      0.0      0.0
```

```
0.0

        pixel6   pixel7   pixel8  ...      pixel774      pixel775  \
count  42000.0  42000.0  42000.0  ...  42000.000000  42000.000000
mean       0.0      0.0      0.0  ...      0.219286      0.117095
std        0.0      0.0      0.0  ...      6.312890      4.633819
min        0.0      0.0      0.0  ...      0.000000      0.000000
25%        0.0      0.0      0.0  ...      0.000000      0.000000
50%        0.0      0.0      0.0  ...      0.000000      0.000000
75%        0.0      0.0      0.0  ...      0.000000      0.000000
max        0.0      0.0      0.0  ...    254.000000    254.000000

           pixel776     pixel777      pixel778      pixel779  pixel780
\
count  42000.000000  42000.00000  42000.000000  42000.000000   42000.0

mean       0.059024      0.02019      0.017238      0.002857       0.0

std        3.274488      1.75987      1.894498      0.414264       0.0

min        0.000000      0.00000      0.000000      0.000000       0.0

25%        0.000000      0.00000      0.000000      0.000000       0.0

50%        0.000000      0.00000      0.000000      0.000000       0.0

75%        0.000000      0.00000      0.000000      0.000000       0.0

max      253.000000    253.00000    254.000000     62.000000       0.0


       pixel781  pixel782  pixel783
count   42000.0   42000.0   42000.0
mean        0.0       0.0       0.0
std         0.0       0.0       0.0
min         0.0       0.0       0.0
25%         0.0       0.0       0.0
50%         0.0       0.0       0.0
75%         0.0       0.0       0.0
max         0.0       0.0       0.0

[8 rows x 785 columns]
```

```
df_train.sum(axis=1)
```

```
0        16650
1        44609
2        13426
3        15029
```

```
4        51093
         ...
41995    29310
41996    13416
41997    31511
41998    26387
41999    18187
Length: 42000, dtype: int64
```

```
df_train.shape
```

```
(42000, 785)
```

```
pixels = df_train.columns.tolist()[1:]
df_train["sum"] = df_train[pixels].sum(axis=1)

df_test["sum"] = df_test[pixels].sum(axis=1)
```

```
df_train.groupby(['label'])['sum'].mean()
```

```
label
0    34632.407551
1    15188.466268
2    29871.099354
3    28320.188003
4    24232.722495
5    25835.920422
6    27734.917331
7    22931.244263
8    30184.148413
9    24553.750000
Name: sum, dtype: float64
```

```
# separate target values from df_train
targets = df_train.label
features = df_train.drop("label",axis=1)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
features[:] = scaler.fit_transform(features)
df_test[:] = scaler.transform(df_test)
```

```
del df_train
```

```
from sklearn.decomposition import PCA as sklearnPCA
sklearn_pca = sklearnPCA(n_components=2)
Y_sklearn = sklearn_pca.fit_transform(features)
```

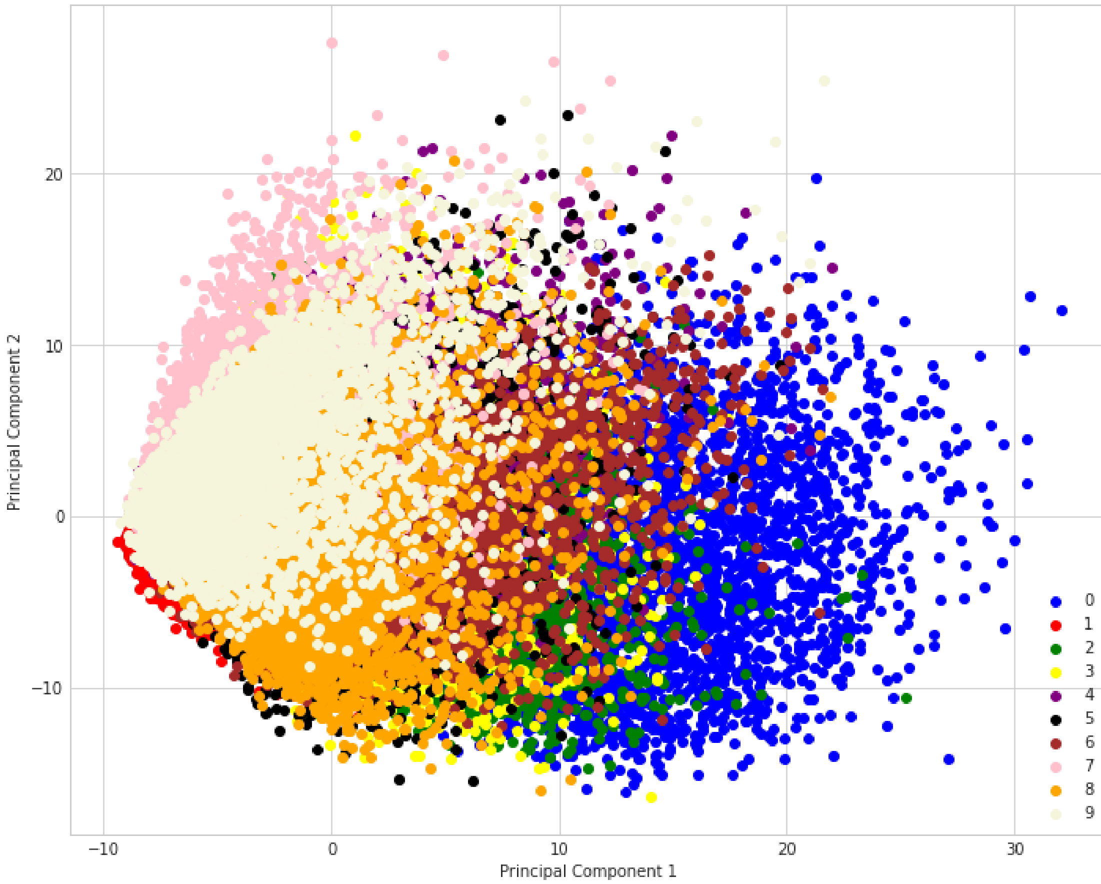
```
Y_sklearn
```

```
array([[-5.27222045, -5.22689222],
       [19.38082385,  6.06236423],
       [-7.83432902, -1.70820371],
       ...,
       [ 0.60967527,  7.06811022],
       [ 2.25995565, -4.33665466],
       [-4.89815874,  1.55445181]])
```

```
#referred to
https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html and
https://www.kaggle.com/arthurtok/interactive-intro-to-dimensionality-
reduction


with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(10, 8))
    for lab, col in zip((0,1,2,3,4,5,6,7,8,9),

('blue','red','green','yellow','purple','black','brown','pink','orange
','beige')):
        plt.scatter(Y_sklearn[targets==lab, 0],
                    Y_sklearn[targets==lab, 1],
                    label=lab,
                    c=col)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend(loc='lower right')
    plt.tight_layout()
    plt.show()
```




```
features.index
```

```
RangeIndex(start=0, stop=42000, step=1)
```

```
sklearn_pca_3 = sklearnPCA(n_components=3)
Y_sklearn_3 = sklearn_pca_3.fit_transform(features)
Y_sklearn_3_test = sklearn_pca_3.transform(df_test)
```

```
# Store results of PCA in a data frame
result=pd.DataFrame(Y_sklearn_3, columns=['PCA%i' % i for i in
range(3)], index=features.index)
```

```
result
```

	PCA0	PCA1	PCA2
0	-5.272178	-5.227316	3.888809
1	19.380798	6.058249	1.341091
2	-7.834401	-1.709171	2.292029
3	-0.706265	5.845940	2.023307
4	26.648659	6.064500	0.983349
...	...	...	...
41995	13.527938	-1.322296	-3.913941
41996	-9.041446	-1.193596	2.321515
41997	0.609621	7.065237	-12.098722

```
41998   2.259975 -4.337381   0.714519
41999  -4.898080  1.555515  -2.502055

[42000 rows x 3 columns]
```

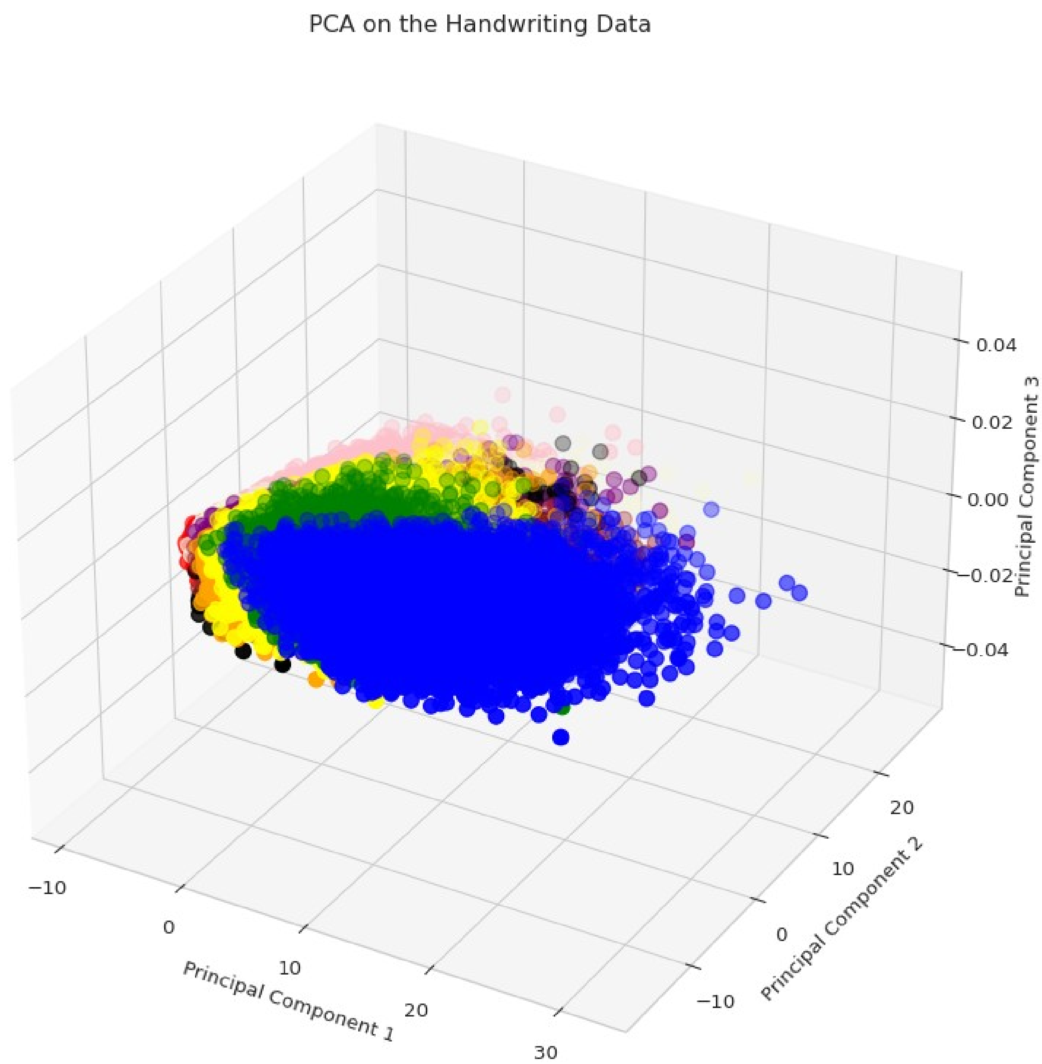
```
my_dpi=96
plt.figure(figsize=(480/my_dpi, 480/my_dpi), dpi=my_dpi)

with plt.style.context('seaborn-whitegrid'):
    my_dpi=96
    fig = plt.figure(figsize=(10, 10), dpi=my_dpi)
    ax = fig.add_subplot(111,projection ='3d')
    for lab, col in zip((0,1,2,3,4,5,6,7,8,9),

('blue','red','green','yellow','purple','black','brown','pink','orange
','beige')):
        plt.scatter(Y_sklearn[targets==lab, 0],
                    Y_sklearn[targets==lab, 1],
                    label=lab,
                    c=col,s =60)

    ax.set_xlabel('Principal Component 1')
    ax.set_ylabel('Principal Component 2')
    ax.set_zlabel('Principal Component 3')
    ax.set_title("PCA on the Handwriting Data")
    plt.show()
```

```
<Figure size 480x480 with 0 Axes>
```




```
encoder = LabelEncoder()
targets[:] = encoder.fit_transform(targets[:])

X_train,X_val, y_train,y_val =
train_test_split(result,targets,random_state=1)
```

Making a Model and Predictions

```
# 3 Principal Components
model = XGBClassifier(max_depth=5, objective='multi:softprob',
n_estimators=1000,
                        num_classes=10)

history = model.fit(X_train, y_train,eval_set
=[(X_val,y_val)],early_stopping_rounds =50)
acc = accuracy_score(y_val, model.predict(X_val))
print(f"Accuracy: , {round(acc,3)}")
```

```
/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1224:
UserWarning: The use of label encoder in XGBClassifier is deprecated
and will be removed in a future release. To remove this warning, do
the following: 1) Pass option use_label_encoder=False when
constructing XGBClassifier object; and 2) Encode your labels (y) as
integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[08:00:51] WARNING: ../src/learner.cc:576:
Parameters: { "num_classes" } might not be used.

  This could be a false alarm, with some parameters getting used by
language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter
actually being used
  but getting flagged wrongly here. Please open an issue if you find
any such cases.


[08:00:52] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'multi:softprob'
was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
[0]	validation_0-mlogloss:1.87608
[1]	validation_0-mlogloss:1.69503
[2]	validation_0-mlogloss:1.57346
[3]	validation_0-mlogloss:1.48884
[4]	validation_0-mlogloss:1.42423
[5]	validation_0-mlogloss:1.37535
[6]	validation_0-mlogloss:1.33578
[7]	validation_0-mlogloss:1.30672
[8]	validation_0-mlogloss:1.28205
[9]	validation_0-mlogloss:1.26075
[10]	validation_0-mlogloss:1.24447
[11]	validation_0-mlogloss:1.23037
[12]	validation_0-mlogloss:1.21939
[13]	validation_0-mlogloss:1.20901
[14]	validation_0-mlogloss:1.20089
[15]	validation_0-mlogloss:1.19374
[16]	validation_0-mlogloss:1.18798
[17]	validation_0-mlogloss:1.18268
[18]	validation_0-mlogloss:1.17749
[19]	validation_0-mlogloss:1.17383
[20]	validation_0-mlogloss:1.17062
```

```
[21]	validation_0-mlogloss:1.16804
[22]	validation_0-mlogloss:1.16525
[23]	validation_0-mlogloss:1.16314
[24]	validation_0-mlogloss:1.16106
[25]	validation_0-mlogloss:1.15924
[26]	validation_0-mlogloss:1.15819
[27]	validation_0-mlogloss:1.15731
[28]	validation_0-mlogloss:1.15630
[29]	validation_0-mlogloss:1.15525
[30]	validation_0-mlogloss:1.15423
[31]	validation_0-mlogloss:1.15340
[32]	validation_0-mlogloss:1.15286
[33]	validation_0-mlogloss:1.15257
[34]	validation_0-mlogloss:1.15231
[35]	validation_0-mlogloss:1.15145
[36]	validation_0-mlogloss:1.15127
[37]	validation_0-mlogloss:1.15126
[38]	validation_0-mlogloss:1.15117
[39]	validation_0-mlogloss:1.15091
[40]	validation_0-mlogloss:1.15039
[41]	validation_0-mlogloss:1.15000
[42]	validation_0-mlogloss:1.14966
[43]	validation_0-mlogloss:1.14940
[44]	validation_0-mlogloss:1.14938
[45]	validation_0-mlogloss:1.14853
[46]	validation_0-mlogloss:1.14836
[47]	validation_0-mlogloss:1.14829
[48]	validation_0-mlogloss:1.14846
[49]	validation_0-mlogloss:1.14822
[50]	validation_0-mlogloss:1.14838
[51]	validation_0-mlogloss:1.14809
[52]	validation_0-mlogloss:1.14829
[53]	validation_0-mlogloss:1.14834
[54]	validation_0-mlogloss:1.14837
[55]	validation_0-mlogloss:1.14860
[56]	validation_0-mlogloss:1.14841
[57]	validation_0-mlogloss:1.14832
[58]	validation_0-mlogloss:1.14805
[59]	validation_0-mlogloss:1.14828
[60]	validation_0-mlogloss:1.14846
[61]	validation_0-mlogloss:1.14868
[62]	validation_0-mlogloss:1.14870
[63]	validation_0-mlogloss:1.14880
[64]	validation_0-mlogloss:1.14906
[65]	validation_0-mlogloss:1.14901
[66]	validation_0-mlogloss:1.14936
[67]	validation_0-mlogloss:1.14949
[68]	validation_0-mlogloss:1.14994
[69]	validation_0-mlogloss:1.15010
[70]	validation_0-mlogloss:1.15031
```

```
[71]  validation_0-mlogloss:1.15055
[72]  validation_0-mlogloss:1.15054
[73]  validation_0-mlogloss:1.15017
[74]  validation_0-mlogloss:1.15029
[75]  validation_0-mlogloss:1.15039
[76]  validation_0-mlogloss:1.15051
[77]  validation_0-mlogloss:1.15062
[78]  validation_0-mlogloss:1.15097
[79]  validation_0-mlogloss:1.15134
[80]  validation_0-mlogloss:1.15156
[81]  validation_0-mlogloss:1.15201
[82]  validation_0-mlogloss:1.15221
[83]  validation_0-mlogloss:1.15239
[84]  validation_0-mlogloss:1.15283
[85]  validation_0-mlogloss:1.15290
[86]  validation_0-mlogloss:1.15311
[87]  validation_0-mlogloss:1.15353
[88]  validation_0-mlogloss:1.15377
[89]  validation_0-mlogloss:1.15395
[90]  validation_0-mlogloss:1.15379
[91]  validation_0-mlogloss:1.15396
[92]  validation_0-mlogloss:1.15382
[93]  validation_0-mlogloss:1.15385
[94]  validation_0-mlogloss:1.15376
[95]  validation_0-mlogloss:1.15400
[96]  validation_0-mlogloss:1.15424
[97]  validation_0-mlogloss:1.15452
[98]  validation_0-mlogloss:1.15481
[99]  validation_0-mlogloss:1.15491
[100] validation_0-mlogloss:1.15539
[101] validation_0-mlogloss:1.15544
[102] validation_0-mlogloss:1.15564
[103] validation_0-mlogloss:1.15568
[104] validation_0-mlogloss:1.15574
[105] validation_0-mlogloss:1.15594
[106] validation_0-mlogloss:1.15630
[107] validation_0-mlogloss:1.15649
[108] validation_0-mlogloss:1.15664
Accuracy: , 0.561
```

```
X_train,X_val, y_train,y_val =
train_test_split(features,targets,random_state=1)


model = XGBClassifier(max_depth=5, objective='multi:softprob',
n_estimators=1000,
                        num_classes=10)

history = model.fit(X_train, y_train,eval_set =[(X_train,y_train),
(X_val,y_val)],early_stopping_rounds =5)
```

```python
acc = accuracy_score(y_val, model.predict(X_val))
print(f"Accuracy: , {round(acc,3)}")
```

```
[08:01:10] WARNING: ../src/learner.cc:576:
Parameters: { "num_classes" } might not be used.

  This could be a false alarm, with some parameters getting used by
language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter
actually being used
  but getting flagged wrongly here. Please open an issue if you find
any such cases.


[08:01:15] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'multi:softprob'
was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
[0]	validation_0-mlogloss:1.42839	validation_1-mlogloss:1.44561
[1]	validation_0-mlogloss:1.09931	validation_1-mlogloss:1.13044
[2]	validation_0-mlogloss:0.88151	validation_1-mlogloss:0.92129
[3]	validation_0-mlogloss:0.72542	validation_1-mlogloss:0.77069
[4]	validation_0-mlogloss:0.60957	validation_1-mlogloss:0.65913
[5]	validation_0-mlogloss:0.51863	validation_1-mlogloss:0.57078
[6]	validation_0-mlogloss:0.44818	validation_1-mlogloss:0.50382
[7]	validation_0-mlogloss:0.38978	validation_1-mlogloss:0.44809
[8]	validation_0-mlogloss:0.34231	validation_1-mlogloss:0.40327
[9]	validation_0-mlogloss:0.30416	validation_1-mlogloss:0.36713
[10]	validation_0-mlogloss:0.27249	validation_1-mlogloss:0.33768
[11]	validation_0-mlogloss:0.24479	validation_1-mlogloss:0.31212
[12]	validation_0-mlogloss:0.22142	validation_1-mlogloss:0.29098
[13]	validation_0-mlogloss:0.20190	validation_1-mlogloss:0.27290
[14]	validation_0-mlogloss:0.18458	validation_1-mlogloss:0.25708
[15]	validation_0-mlogloss:0.16888	validation_1-mlogloss:0.24343
[16]	validation_0-mlogloss:0.15593	validation_1-mlogloss:0.23174
[17]	validation_0-mlogloss:0.14441	validation_1-mlogloss:0.22166
[18]	validation_0-mlogloss:0.13361	validation_1-mlogloss:0.21180
[19]	validation_0-mlogloss:0.12413	validation_1-mlogloss:0.20371
[20]	validation_0-mlogloss:0.11625	validation_1-mlogloss:0.19627
[21]	validation_0-mlogloss:0.10815	validation_1-mlogloss:0.18900
[22]	validation_0-mlogloss:0.10031	validation_1-mlogloss:0.18205
[23]	validation_0-mlogloss:0.09473	validation_1-mlogloss:0.17735
[24]	validation_0-mlogloss:0.08789	validation_1-mlogloss:0.17083
[25]	validation_0-mlogloss:0.08249	validation_1-mlogloss:0.16605
[26]	validation_0-mlogloss:0.07738	validation_1-mlogloss:0.16129
[27]	validation_0-mlogloss:0.07189	validation_1-mlogloss:0.15697
[28]	validation_0-mlogloss:0.06801	validation_1-mlogloss:0.15350
[29]	validation_0-mlogloss:0.06414	validation_1-mlogloss:0.14999
[30]	validation_0-mlogloss:0.06057	validation_1-mlogloss:0.14722
[31]	validation_0-mlogloss:0.05713	validation_1-mlogloss:0.14373
```

```
[32]	validation_0-mlogloss:0.05349	validation_1-mlogloss:0.14027
[33]	validation_0-mlogloss:0.05094	validation_1-mlogloss:0.13833
[34]	validation_0-mlogloss:0.04831	validation_1-mlogloss:0.13593
[35]	validation_0-mlogloss:0.04560	validation_1-mlogloss:0.13382
[36]	validation_0-mlogloss:0.04306	validation_1-mlogloss:0.13161
[37]	validation_0-mlogloss:0.04033	validation_1-mlogloss:0.12903
[38]	validation_0-mlogloss:0.03796	validation_1-mlogloss:0.12721
[39]	validation_0-mlogloss:0.03598	validation_1-mlogloss:0.12509
[40]	validation_0-mlogloss:0.03401	validation_1-mlogloss:0.12340
[41]	validation_0-mlogloss:0.03242	validation_1-mlogloss:0.12218
[42]	validation_0-mlogloss:0.03080	validation_1-mlogloss:0.12055
[43]	validation_0-mlogloss:0.02905	validation_1-mlogloss:0.11884
[44]	validation_0-mlogloss:0.02748	validation_1-mlogloss:0.11706
[45]	validation_0-mlogloss:0.02617	validation_1-mlogloss:0.11563
[46]	validation_0-mlogloss:0.02485	validation_1-mlogloss:0.11447
[47]	validation_0-mlogloss:0.02354	validation_1-mlogloss:0.11336
[48]	validation_0-mlogloss:0.02231	validation_1-mlogloss:0.11205
[49]	validation_0-mlogloss:0.02120	validation_1-mlogloss:0.11124
[50]	validation_0-mlogloss:0.02023	validation_1-mlogloss:0.10998
[51]	validation_0-mlogloss:0.01928	validation_1-mlogloss:0.10883
[52]	validation_0-mlogloss:0.01829	validation_1-mlogloss:0.10787
[53]	validation_0-mlogloss:0.01723	validation_1-mlogloss:0.10640
[54]	validation_0-mlogloss:0.01643	validation_1-mlogloss:0.10563
[55]	validation_0-mlogloss:0.01576	validation_1-mlogloss:0.10474
[56]	validation_0-mlogloss:0.01502	validation_1-mlogloss:0.10384
[57]	validation_0-mlogloss:0.01445	validation_1-mlogloss:0.10332
[58]	validation_0-mlogloss:0.01378	validation_1-mlogloss:0.10245
[59]	validation_0-mlogloss:0.01308	validation_1-mlogloss:0.10127
[60]	validation_0-mlogloss:0.01244	validation_1-mlogloss:0.10036
[61]	validation_0-mlogloss:0.01200	validation_1-mlogloss:0.09975
[62]	validation_0-mlogloss:0.01149	validation_1-mlogloss:0.09906
[63]	validation_0-mlogloss:0.01097	validation_1-mlogloss:0.09831
[64]	validation_0-mlogloss:0.01044	validation_1-mlogloss:0.09763
[65]	validation_0-mlogloss:0.01000	validation_1-mlogloss:0.09712
[66]	validation_0-mlogloss:0.00961	validation_1-mlogloss:0.09666
[67]	validation_0-mlogloss:0.00912	validation_1-mlogloss:0.09616
[68]	validation_0-mlogloss:0.00872	validation_1-mlogloss:0.09550
[69]	validation_0-mlogloss:0.00829	validation_1-mlogloss:0.09468
[70]	validation_0-mlogloss:0.00795	validation_1-mlogloss:0.09430
[71]	validation_0-mlogloss:0.00766	validation_1-mlogloss:0.09384
[72]	validation_0-mlogloss:0.00734	validation_1-mlogloss:0.09339
[73]	validation_0-mlogloss:0.00706	validation_1-mlogloss:0.09305
[74]	validation_0-mlogloss:0.00676	validation_1-mlogloss:0.09246
[75]	validation_0-mlogloss:0.00646	validation_1-mlogloss:0.09196
[76]	validation_0-mlogloss:0.00620	validation_1-mlogloss:0.09154
[77]	validation_0-mlogloss:0.00600	validation_1-mlogloss:0.09107
[78]	validation_0-mlogloss:0.00578	validation_1-mlogloss:0.09086
[79]	validation_0-mlogloss:0.00557	validation_1-mlogloss:0.09031
[80]	validation_0-mlogloss:0.00537	validation_1-mlogloss:0.09002
[81]	validation_0-mlogloss:0.00517	validation_1-mlogloss:0.08971
```

```
[82]	validation_0-mlogloss:0.00500	validation_1-mlogloss:0.08952
[83]	validation_0-mlogloss:0.00484	validation_1-mlogloss:0.08928
[84]	validation_0-mlogloss:0.00464	validation_1-mlogloss:0.08888
[85]	validation_0-mlogloss:0.00451	validation_1-mlogloss:0.08865
[86]	validation_0-mlogloss:0.00435	validation_1-mlogloss:0.08854
[87]	validation_0-mlogloss:0.00424	validation_1-mlogloss:0.08818
[88]	validation_0-mlogloss:0.00410	validation_1-mlogloss:0.08801
[89]	validation_0-mlogloss:0.00398	validation_1-mlogloss:0.08777
[90]	validation_0-mlogloss:0.00383	validation_1-mlogloss:0.08748
[91]	validation_0-mlogloss:0.00371	validation_1-mlogloss:0.08738
[92]	validation_0-mlogloss:0.00359	validation_1-mlogloss:0.08702
[93]	validation_0-mlogloss:0.00348	validation_1-mlogloss:0.08697
[94]	validation_0-mlogloss:0.00338	validation_1-mlogloss:0.08668
[95]	validation_0-mlogloss:0.00327	validation_1-mlogloss:0.08649
[96]	validation_0-mlogloss:0.00318	validation_1-mlogloss:0.08631
[97]	validation_0-mlogloss:0.00309	validation_1-mlogloss:0.08618
[98]	validation_0-mlogloss:0.00301	validation_1-mlogloss:0.08595
[99]	validation_0-mlogloss:0.00294	validation_1-mlogloss:0.08581
[100]	validation_0-mlogloss:0.00287	validation_1-mlogloss:0.08575
[101]	validation_0-mlogloss:0.00279	validation_1-mlogloss:0.08545
[102]	validation_0-mlogloss:0.00271	validation_1-mlogloss:0.08522
[103]	validation_0-mlogloss:0.00264	validation_1-mlogloss:0.08503
[104]	validation_0-mlogloss:0.00258	validation_1-mlogloss:0.08492
[105]	validation_0-mlogloss:0.00252	validation_1-mlogloss:0.08472
[106]	validation_0-mlogloss:0.00246	validation_1-mlogloss:0.08459
[107]	validation_0-mlogloss:0.00240	validation_1-mlogloss:0.08452
[108]	validation_0-mlogloss:0.00235	validation_1-mlogloss:0.08442
[109]	validation_0-mlogloss:0.00230	validation_1-mlogloss:0.08439
[110]	validation_0-mlogloss:0.00225	validation_1-mlogloss:0.08441
[111]	validation_0-mlogloss:0.00220	validation_1-mlogloss:0.08430
[112]	validation_0-mlogloss:0.00215	validation_1-mlogloss:0.08411
[113]	validation_0-mlogloss:0.00210	validation_1-mlogloss:0.08391
[114]	validation_0-mlogloss:0.00205	validation_1-mlogloss:0.08373
[115]	validation_0-mlogloss:0.00202	validation_1-mlogloss:0.08373
[116]	validation_0-mlogloss:0.00197	validation_1-mlogloss:0.08364
[117]	validation_0-mlogloss:0.00194	validation_1-mlogloss:0.08346
[118]	validation_0-mlogloss:0.00190	validation_1-mlogloss:0.08344
[119]	validation_0-mlogloss:0.00187	validation_1-mlogloss:0.08342
[120]	validation_0-mlogloss:0.00183	validation_1-mlogloss:0.08336
[121]	validation_0-mlogloss:0.00179	validation_1-mlogloss:0.08324
[122]	validation_0-mlogloss:0.00176	validation_1-mlogloss:0.08306
[123]	validation_0-mlogloss:0.00172	validation_1-mlogloss:0.08294
[124]	validation_0-mlogloss:0.00169	validation_1-mlogloss:0.08292
[125]	validation_0-mlogloss:0.00166	validation_1-mlogloss:0.08289
[126]	validation_0-mlogloss:0.00164	validation_1-mlogloss:0.08286
[127]	validation_0-mlogloss:0.00161	validation_1-mlogloss:0.08281
[128]	validation_0-mlogloss:0.00158	validation_1-mlogloss:0.08274
[129]	validation_0-mlogloss:0.00155	validation_1-mlogloss:0.08273
[130]	validation_0-mlogloss:0.00153	validation_1-mlogloss:0.08260
[131]	validation_0-mlogloss:0.00150	validation_1-mlogloss:0.08258
```

```
[132] validation_0-mlogloss:0.00148    validation_1-mlogloss:0.08249
[133] validation_0-mlogloss:0.00145    validation_1-mlogloss:0.08250
[134] validation_0-mlogloss:0.00143    validation_1-mlogloss:0.08239
[135] validation_0-mlogloss:0.00141    validation_1-mlogloss:0.08235
[136] validation_0-mlogloss:0.00138    validation_1-mlogloss:0.08235
[137] validation_0-mlogloss:0.00136    validation_1-mlogloss:0.08233
[138] validation_0-mlogloss:0.00134    validation_1-mlogloss:0.08227
[139] validation_0-mlogloss:0.00132    validation_1-mlogloss:0.08229
[140] validation_0-mlogloss:0.00130    validation_1-mlogloss:0.08223
[141] validation_0-mlogloss:0.00129    validation_1-mlogloss:0.08227
[142] validation_0-mlogloss:0.00127    validation_1-mlogloss:0.08219
[143] validation_0-mlogloss:0.00125    validation_1-mlogloss:0.08212
[144] validation_0-mlogloss:0.00123    validation_1-mlogloss:0.08204
[145] validation_0-mlogloss:0.00122    validation_1-mlogloss:0.08200
[146] validation_0-mlogloss:0.00120    validation_1-mlogloss:0.08204
[147] validation_0-mlogloss:0.00119    validation_1-mlogloss:0.08193
[148] validation_0-mlogloss:0.00117    validation_1-mlogloss:0.08193
[149] validation_0-mlogloss:0.00116    validation_1-mlogloss:0.08186
[150] validation_0-mlogloss:0.00114    validation_1-mlogloss:0.08187
```

```
results = model.evals_result()
```

```
from matplotlib import pyplot
# plot learning curves
plt.figure(figsize=(10, 8))
pyplot.plot(results['validation_0']['mlogloss'], label='train')
pyplot.plot(results['validation_1']['mlogloss'], label='test')
# show the legend
pyplot.legend()
plt.xlabel('iterations')
plt.ylabel('mlogloss')
# show the plot
pyplot.show()
```

```
from xgboost import plot_importance
ax = plot_importance(model,max_num_features=10)
fig = ax.figure
fig.set_size_inches(10,8)
plt.show()
```

```
predictions = model.predict(df_test)
```

```
output = 
pd.read_csv("../input/digit-recognizer/sample_submission.csv")
output['Label'] = predictions
output.to_csv('submission.csv',index=False)
```

1) What is Decision Tree Algorithm ? Which type of ML we can solve using Decision Tree?

-> A Decision Tree algorithm is a type of supervised learning algorithm used for classification and regression tasks. It is a non-parametric model that works by recursively splitting the data into subsets based on the values of the input features, until a leaf node is reached that corresponds to the predicted output.

In a Decision Tree, each internal node represents a test on a feature of the input data, and each branch represents the outcome of the test. The leaves of the tree represent the predicted output or class label.

Decision Trees can be used for both classification and regression tasks. In classification, the output is a categorical variable, while in regression, the output is a continuous variable.

Decision Trees are particularly useful when the data has a hierarchical structure or when the decision-making process can be represented as a series of if-then-else statements. They are also useful for handling both numerical and categorical data, and can be applied to both binary and multi-class classification problems.

Some common applications of Decision Trees include:

Predicting customer churn or credit risk in finance Identifying diseases based on medical symptoms Recommending products or services based on user behavior Predicting the success of a marketing campaign based on customer demographics Overall, Decision Trees are a versatile and widely-used machine learning algorithm that can be used for a variety of classification and regression tasks. They are relatively easy to interpret and can provide insights into the underlying structure of the data, making them a valuable tool for data analysis and decision-making.

2) What do you mean by ensemble learning ? Does XGBoost support ensemble learning ?

-> Ensemble learning is a machine learning technique that combines the predictions of multiple individual models to improve the accuracy and robustness of the overall prediction. The idea behind ensemble learning is that by combining the predictions of multiple models, the weaknesses of any one model can be offset by the strengths of the other models.

There are different types of ensemble learning methods, such as bagging, boosting, and stacking. Bagging and boosting are the most common techniques, with boosting being particularly popular due to its ability to iteratively improve the model's accuracy.

XGBoost (Extreme Gradient Boosting) is a popular machine learning library that supports ensemble learning. In fact, XGBoost is a type of boosting algorithm that uses an ensemble of decision trees to make predictions. It works by iteratively adding decision trees to the model, with each new tree focusing on the samples that the previous trees have struggled to classify correctly. XGBoost also includes regularization techniques to prevent overfitting and improve the model's generalization ability.

XGBoost has been widely used in various machine learning competitions and is known for its speed and accuracy. Its support for ensemble learning makes it particularly effective for handling large datasets with complex relationships between the input variables and output variables.

3) What is Principal Component Analysis ? Why do we use PCA in our notebook?

-> Principal Component Analysis (PCA) is a dimensionality reduction technique that is commonly used in machine learning and data analysis. PCA works by transforming a high-dimensional dataset into a lower-dimensional representation that retains as much of the original information as possible. The transformed features are known as principal components, which are linear combinations of the original features.

PCA is useful in reducing the dimensionality of a dataset while retaining most of the important information. This can lead to faster and more efficient machine learning algorithms, especially when dealing with high-dimensional datasets. PCA is also used for data visualization, where it can be used to visualize the data in two or three dimensions.

In our notebook, PCA is used for feature extraction and visualization. The dataset we are working with has 30 input features, which can make it difficult to visualize and analyze. By applying PCA to the dataset, we can reduce the number of features to a smaller set of principal components that capture the most important information. We can then visualize the data in two or three dimensions to gain insights into the underlying structure of the data.

In addition, PCA can also help with reducing the impact of multicollinearity, where two or more input features are highly correlated. By using PCA to reduce the number of input features, we can remove the redundant information and improve the performance of our machine learning algorithms.

4) Check use of "StandardScalar" class from sklearn in notebook. What do you think is this API used for?

-> In the notebook, the "StandardScaler" class from the "sklearn.preprocessing" module is used to scale the input features of the dataset to have zero mean and unit variance. This is a common preprocessing step in machine learning, where the input features are often of different scales and ranges.

Scaling the features using StandardScaler helps to ensure that all the features are on a similar scale, which can improve the performance of some machine learning algorithms. For example, many linear regression and logistic regression models assume that the input features are normally distributed with mean zero and unit variance. Scaling the features to have these properties can help to ensure that the models perform optimally.

The "StandardScaler" class from the "sklearn.preprocessing" module provides a simple way to standardize the input features by subtracting the mean and dividing by the standard deviation. This is achieved by first computing the mean and standard deviation of each feature in the training set and then using these values to scale the features in both the training and test sets.

Overall, the "StandardScaler" class is a useful API in the "sklearn.preprocessing" module for preprocessing numerical data in machine learning.

5) Consider statement "model = XGBClassifier(max_depth=5, objective='multi:softprob', n_estimators=1000, num_classes=10) " in the notebook explain purpose of each parameter of this constructor. What are we doing here defining a model with specific parameters or training the model?

-> Here's a breakdown of the parameters in this constructor:

max_depth: This parameter sets the maximum depth of each decision tree in the ensemble. A higher value of max_depth can lead to overfitting, while a lower value can lead to underfitting.

objective: This parameter specifies the loss function to be optimized during the training of the model. In this case, 'multi:softprob' is used for multiclass classification, and it optimizes the softmax loss function.

n_estimators: This parameter sets the number of decision trees in the ensemble. A higher value of n_estimators can lead to better performance, but it can also increase the training time and memory usage.

num_classes: This parameter sets the number of classes in the multiclass classification problem. In this case, there are 10 classes.

After defining the model instance, we can then train the model using the "fit" method and evaluate its performance on the test set.

Overall, the purpose of defining a model with specific parameters is to tune the model to achieve the best performance on the given task. By adjusting the hyperparameters of the model, we can balance the bias-variance tradeoff and improve the model's generalization ability.

6) What step in ML pipeline fit fuction carries out?

-> The "fit" function in machine learning pipelines is used to train a model on a given dataset. Specifically, the "fit" function adjusts the model parameters to minimize the difference between the predicted outputs and the true outputs in the training dataset. This process is also known as model training or model fitting.

During the model training process, the "fit" function takes in the input features and the corresponding target labels as arguments. The function then adjusts the parameters of the model using an optimization algorithm such as gradient descent, and iteratively updates the model until the difference between the predicted outputs and the true outputs is minimized.

The "fit" function typically involves a number of steps, including pre-processing the input data, initializing the model parameters, iterating through multiple epochs, computing the loss function, and updating the model parameters. The number of epochs and the optimization algorithm used in the "fit" function can be specified by the user.

Overall, the "fit" function is a critical step in the machine learning pipeline, as it is responsible for training the model and optimizing its parameters to make accurate predictions on new, unseen data.