```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()
```

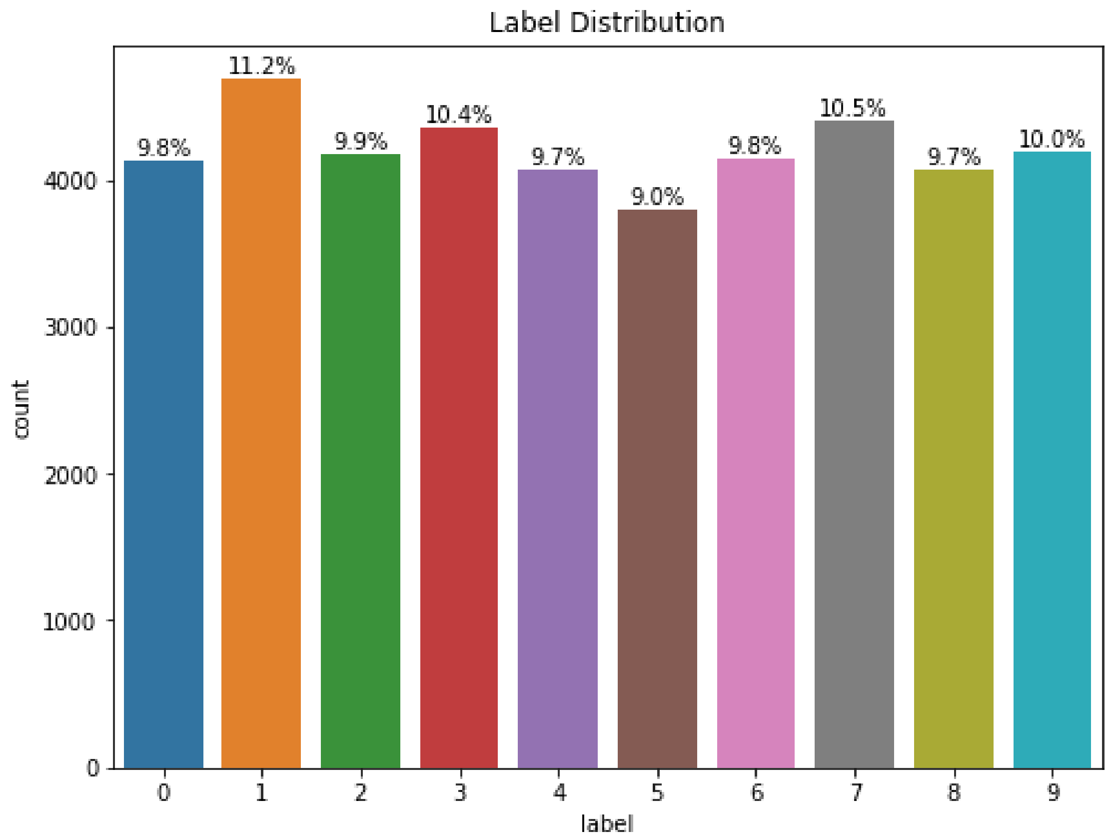
```
array([1, 0, 4, 7, 3, 5, 8, 9, 2, 6])
```

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)
```

```python
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')
```




```python
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)
```

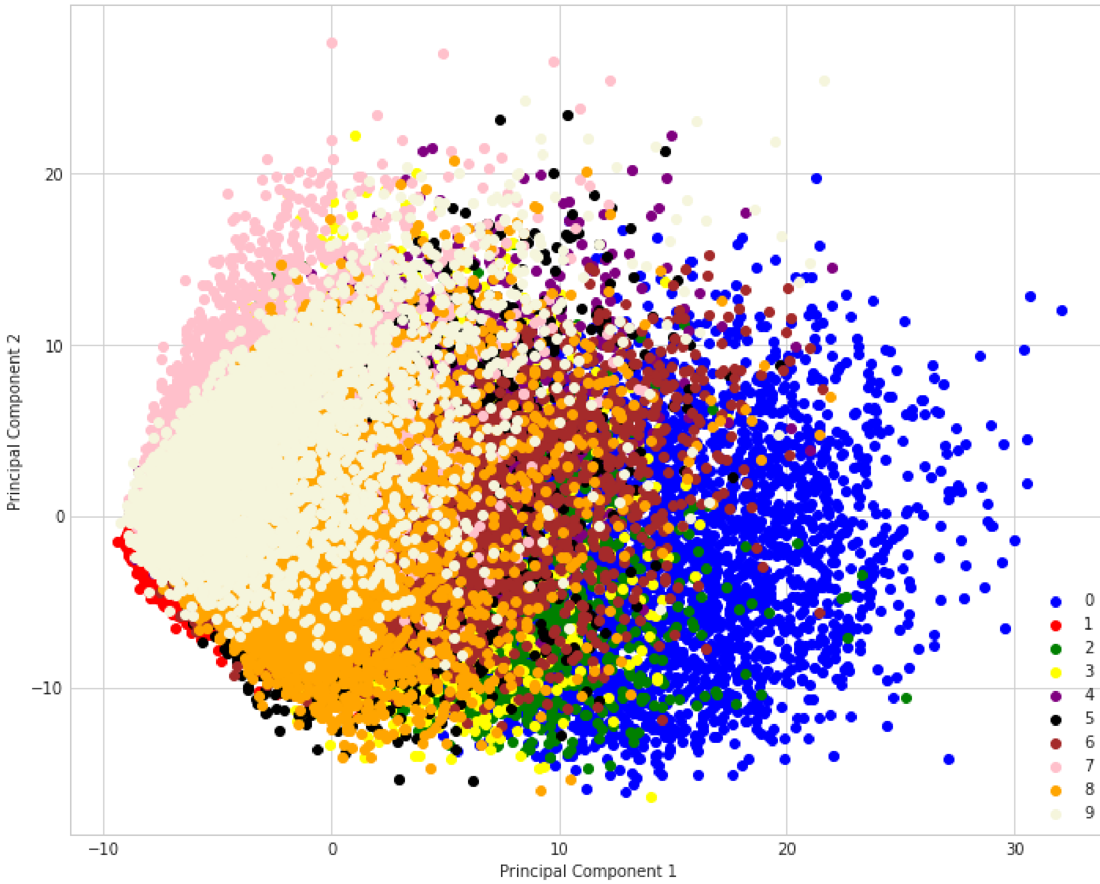
```
Y_sklearn
```

```
array([[-5.27223918, -5.22754114],
       [19.38074818,  6.06248827],
       [-7.83431909, -1.70797914],
       ...,
       [ 0.60961552,  7.06816955],
       [ 2.2599876 , -4.33652616],
       [-4.89814654,  1.55410133]])
```

```
#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	-5.272223	-5.226191	3.888754
1	19.380781	6.062378	1.339078
2	-7.834344	-1.708974	2.291150
3	-0.706285	5.847097	2.022486
4	26.648644	6.067427	0.982265
...	...	...	...
41995	13.527951	-1.320741	-3.915648
41996	-9.041416	-1.193018	2.320608
41997	0.609633	7.067946	-12.099625

```
41998   2.259998 -4.337004   0.713391
41999  -4.898139  1.555018  -2.501887

[42000 rows x 3 columns]
```

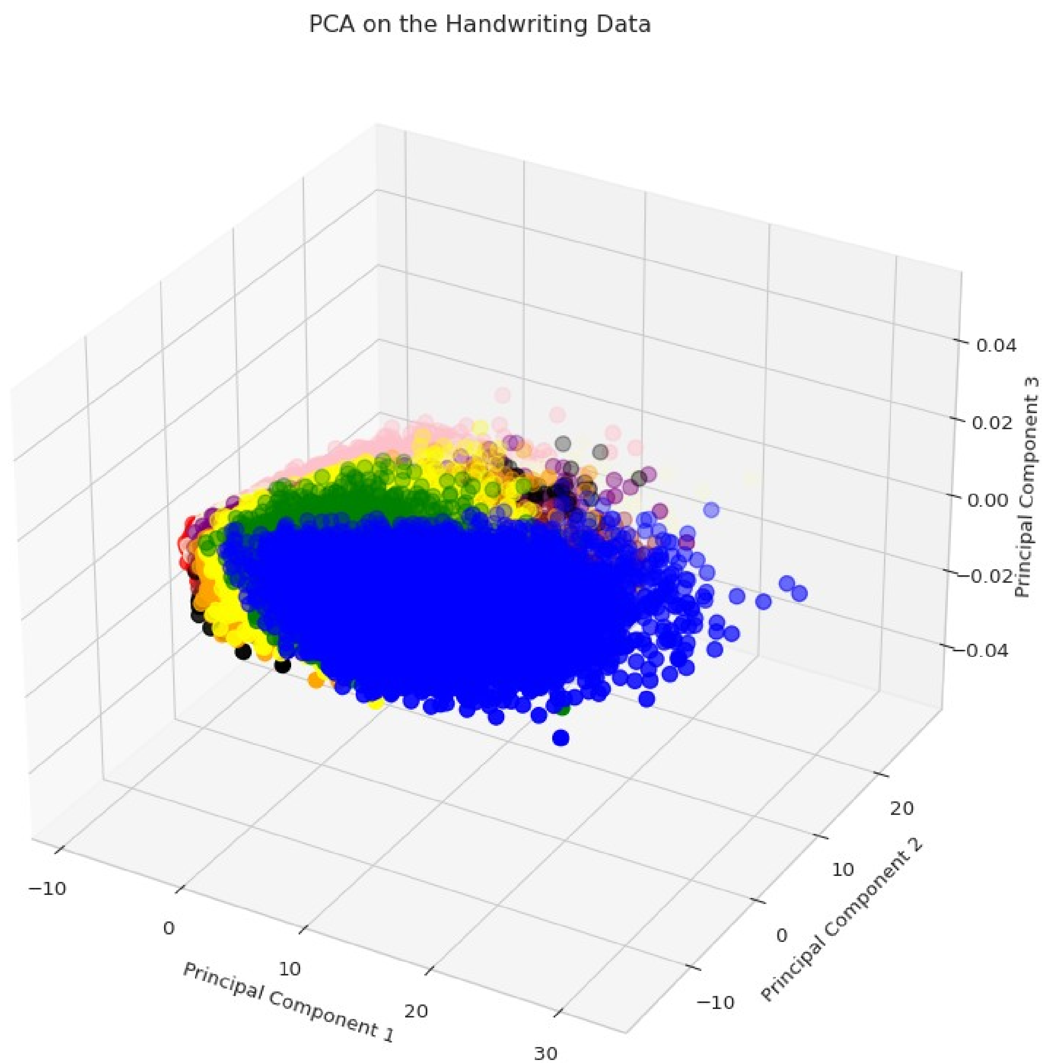
```
my_dpi=96
plt.figure(figsize=(480/my_dpi, 480/my_dpi), dpi=my_dpi)

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)}")
```

```
[09:16:58] 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.


[09:16:59] 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:1.87622

/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)

[1]	validation_0-mlogloss:1.69484
[2]	validation_0-mlogloss:1.57399
[3]	validation_0-mlogloss:1.48991
[4]	validation_0-mlogloss:1.42447
[5]	validation_0-mlogloss:1.37521
[6]	validation_0-mlogloss:1.33515
[7]	validation_0-mlogloss:1.30526
[8]	validation_0-mlogloss:1.28139
[9]	validation_0-mlogloss:1.26046
[10]	validation_0-mlogloss:1.24381
[11]	validation_0-mlogloss:1.23053
[12]	validation_0-mlogloss:1.21937
[13]	validation_0-mlogloss:1.20992
[14]	validation_0-mlogloss:1.19970
[15]	validation_0-mlogloss:1.19256
[16]	validation_0-mlogloss:1.18690
[17]	validation_0-mlogloss:1.18108
[18]	validation_0-mlogloss:1.17713
[19]	validation_0-mlogloss:1.17401
```

```
[20]	validation_0-mlogloss:1.17057
[21]	validation_0-mlogloss:1.16799
[22]	validation_0-mlogloss:1.16562
[23]	validation_0-mlogloss:1.16270
[24]	validation_0-mlogloss:1.16080
[25]	validation_0-mlogloss:1.15969
[26]	validation_0-mlogloss:1.15850
[27]	validation_0-mlogloss:1.15729
[28]	validation_0-mlogloss:1.15640
[29]	validation_0-mlogloss:1.15501
[30]	validation_0-mlogloss:1.15377
[31]	validation_0-mlogloss:1.15319
[32]	validation_0-mlogloss:1.15247
[33]	validation_0-mlogloss:1.15210
[34]	validation_0-mlogloss:1.15143
[35]	validation_0-mlogloss:1.15104
[36]	validation_0-mlogloss:1.15016
[37]	validation_0-mlogloss:1.14995
[38]	validation_0-mlogloss:1.14983
[39]	validation_0-mlogloss:1.14938
[40]	validation_0-mlogloss:1.14853
[41]	validation_0-mlogloss:1.14870
[42]	validation_0-mlogloss:1.14831
[43]	validation_0-mlogloss:1.14805
[44]	validation_0-mlogloss:1.14786
[45]	validation_0-mlogloss:1.14812
[46]	validation_0-mlogloss:1.14789
[47]	validation_0-mlogloss:1.14742
[48]	validation_0-mlogloss:1.14783
[49]	validation_0-mlogloss:1.14800
[50]	validation_0-mlogloss:1.14786
[51]	validation_0-mlogloss:1.14770
[52]	validation_0-mlogloss:1.14778
[53]	validation_0-mlogloss:1.14775
[54]	validation_0-mlogloss:1.14739
[55]	validation_0-mlogloss:1.14736
[56]	validation_0-mlogloss:1.14730
[57]	validation_0-mlogloss:1.14744
[58]	validation_0-mlogloss:1.14776
[59]	validation_0-mlogloss:1.14778
[60]	validation_0-mlogloss:1.14793
[61]	validation_0-mlogloss:1.14795
[62]	validation_0-mlogloss:1.14827
[63]	validation_0-mlogloss:1.14861
[64]	validation_0-mlogloss:1.14852
[65]	validation_0-mlogloss:1.14890
[66]	validation_0-mlogloss:1.14898
[67]	validation_0-mlogloss:1.14913
[68]	validation_0-mlogloss:1.14896
[69]	validation_0-mlogloss:1.14941
```

```
[70]	validation_0-mlogloss:1.14926
[71]	validation_0-mlogloss:1.14952
[72]	validation_0-mlogloss:1.14966
[73]	validation_0-mlogloss:1.14994
[74]	validation_0-mlogloss:1.14992
[75]	validation_0-mlogloss:1.15017
[76]	validation_0-mlogloss:1.15039
[77]	validation_0-mlogloss:1.15057
[78]	validation_0-mlogloss:1.15048
[79]	validation_0-mlogloss:1.15070
[80]	validation_0-mlogloss:1.15094
[81]	validation_0-mlogloss:1.15094
[82]	validation_0-mlogloss:1.15101
[83]	validation_0-mlogloss:1.15133
[84]	validation_0-mlogloss:1.15142
[85]	validation_0-mlogloss:1.15176
[86]	validation_0-mlogloss:1.15191
[87]	validation_0-mlogloss:1.15203
[88]	validation_0-mlogloss:1.15237
[89]	validation_0-mlogloss:1.15281
[90]	validation_0-mlogloss:1.15300
[91]	validation_0-mlogloss:1.15314
[92]	validation_0-mlogloss:1.15324
[93]	validation_0-mlogloss:1.15337
[94]	validation_0-mlogloss:1.15346
[95]	validation_0-mlogloss:1.15352
[96]	validation_0-mlogloss:1.15381
[97]	validation_0-mlogloss:1.15392
[98]	validation_0-mlogloss:1.15433
[99]	validation_0-mlogloss:1.15445
[100]	validation_0-mlogloss:1.15477
[101]	validation_0-mlogloss:1.15484
[102]	validation_0-mlogloss:1.15494
[103]	validation_0-mlogloss:1.15514
[104]	validation_0-mlogloss:1.15515
[105]	validation_0-mlogloss:1.15543
Accuracy: , 0.56
```

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

```
[09:17:17] 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.


[09:17:21] 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
```

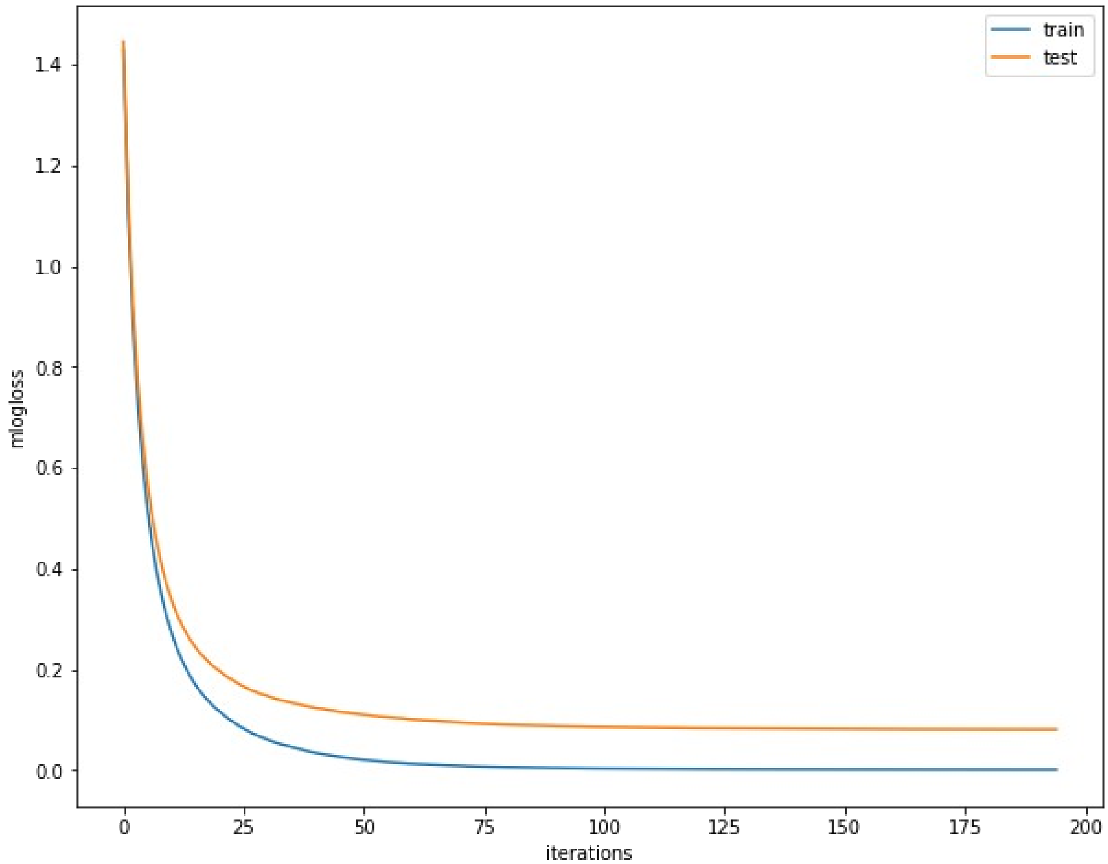
```
[85]	validation_0-mlogloss:0.00451	validation_1-mlogloss:0.08865
[86]	validation_0-mlogloss:0.00435	validation_1-mlogloss:0.08854
[87]	validation_0-mlogloss:0.00424	validation_1-mlogloss:0.08818
[88]	validation_0-mlogloss:0.00410	validation_1-mlogloss:0.08801
[89]	validation_0-mlogloss:0.00398	validation_1-mlogloss:0.08777
[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
[151]	validation_0-mlogloss:0.00113	validation_1-mlogloss:0.08187
[152]	validation_0-mlogloss:0.00111	validation_1-mlogloss:0.08185
[153]	validation_0-mlogloss:0.00110	validation_1-mlogloss:0.08190
[154]	validation_0-mlogloss:0.00109	validation_1-mlogloss:0.08171
[155]	validation_0-mlogloss:0.00107	validation_1-mlogloss:0.08170
[156]	validation_0-mlogloss:0.00106	validation_1-mlogloss:0.08162
[157]	validation_0-mlogloss:0.00105	validation_1-mlogloss:0.08158
[158]	validation_0-mlogloss:0.00104	validation_1-mlogloss:0.08154
[159]	validation_0-mlogloss:0.00103	validation_1-mlogloss:0.08149
[160]	validation_0-mlogloss:0.00101	validation_1-mlogloss:0.08153
[161]	validation_0-mlogloss:0.00100	validation_1-mlogloss:0.08147
[162]	validation_0-mlogloss:0.00099	validation_1-mlogloss:0.08142
[163]	validation_0-mlogloss:0.00098	validation_1-mlogloss:0.08138
[164]	validation_0-mlogloss:0.00097	validation_1-mlogloss:0.08144
[165]	validation_0-mlogloss:0.00096	validation_1-mlogloss:0.08140
[166]	validation_0-mlogloss:0.00095	validation_1-mlogloss:0.08142
[167]	validation_0-mlogloss:0.00094	validation_1-mlogloss:0.08137
[168]	validation_0-mlogloss:0.00093	validation_1-mlogloss:0.08130
[169]	validation_0-mlogloss:0.00092	validation_1-mlogloss:0.08129
[170]	validation_0-mlogloss:0.00091	validation_1-mlogloss:0.08131
[171]	validation_0-mlogloss:0.00090	validation_1-mlogloss:0.08130
[172]	validation_0-mlogloss:0.00090	validation_1-mlogloss:0.08127
[173]	validation_0-mlogloss:0.00089	validation_1-mlogloss:0.08126
[174]	validation_0-mlogloss:0.00088	validation_1-mlogloss:0.08131
[175]	validation_0-mlogloss:0.00087	validation_1-mlogloss:0.08117
[176]	validation_0-mlogloss:0.00086	validation_1-mlogloss:0.08117
[177]	validation_0-mlogloss:0.00086	validation_1-mlogloss:0.08115
[178]	validation_0-mlogloss:0.00085	validation_1-mlogloss:0.08114
[179]	validation_0-mlogloss:0.00084	validation_1-mlogloss:0.08113
[180]	validation_0-mlogloss:0.00084	validation_1-mlogloss:0.08112
[181]	validation_0-mlogloss:0.00083	validation_1-mlogloss:0.08109
[182]	validation_0-mlogloss:0.00082	validation_1-mlogloss:0.08103
[183]	validation_0-mlogloss:0.00081	validation_1-mlogloss:0.08103
[184]	validation_0-mlogloss:0.00081	validation_1-mlogloss:0.08102
```

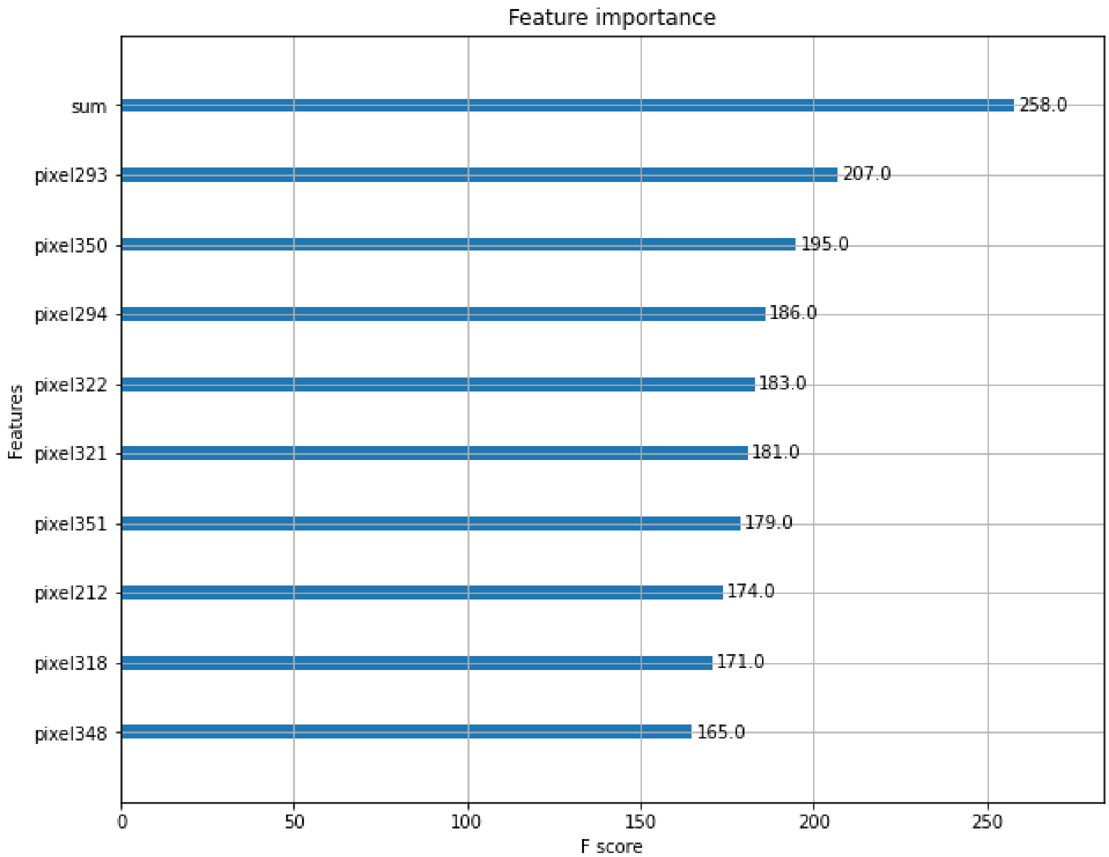
```
[185] validation_0-mlogloss:0.00080	validation_1-mlogloss:0.08100
[186] validation_0-mlogloss:0.00080	validation_1-mlogloss:0.08102
[187] validation_0-mlogloss:0.00079	validation_1-mlogloss:0.08093
[188] validation_0-mlogloss:0.00078	validation_1-mlogloss:0.08083
[189] validation_0-mlogloss:0.00078	validation_1-mlogloss:0.08081
[190] validation_0-mlogloss:0.00077	validation_1-mlogloss:0.08086
[191] validation_0-mlogloss:0.00076	validation_1-mlogloss:0.08087
[192] validation_0-mlogloss:0.00076	validation_1-mlogloss:0.08091
[193] validation_0-mlogloss:0.00075	validation_1-mlogloss:0.08097
[194] validation_0-mlogloss:0.00075	validation_1-mlogloss:0.08099
Accuracy: , 0.976
```

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

```python
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?

The decision tree algorithm is a supervised machine learning algorithm that is used for both classification and regression tasks. The algorithm works by recursively splitting the data into subsets based on the most significant attributes or features until a target variable or decision is reached.
The algorithm builds a tree-like structure where each internal node represents a feature, and each leaf node represents a decision. To classify a new data point, the algorithm starts at the root of the tree and traverses the branches until it reaches a leaf node, which contains the predicted decision or class.
The decision tree algorithm can handle both categorical and numerical data types and is relatively easy to interpret and explain, making it a popular choice for decision-making applications.
Some of the applications of decision trees include:
Classification problems: Decision trees can be used to classify data into different categories based on a set of features or attributes. Examples of classification problems include spam filtering, sentiment analysis, and disease diagnosis.
Regression problems: Decision trees can also be used for regression tasks, such as predicting the value of a continuous variable based on a set of features or attributes. Examples of regression problems include stock price prediction, housing price prediction, and demand forecasting.
Feature selection: Decision trees can be used to identify the most important features or attributes that contribute the most to the decision or class. This can help to reduce the dimensionality of the data and improve the performance of other machine learning algorithms.

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

Ensemble learning is a technique in machine learning that combines multiple individual models to improve the overall predictive power and reduce the risk of overfitting. Instead of relying on a single model, ensemble learning methods create an ensemble of models that work together to make predictions.
There are several ensemble learning techniques, such as:
Bagging: A technique that trains multiple independent models on different random subsets of the training data and averages their predictions.
Boosting: A technique that trains multiple weak models sequentially, where each new model tries to correct the errors of the previous models.
Stacking: A technique that combines the predictions of multiple models by training a meta-model on the outputs of the base models.
XGBoost is a popular gradient boosting library that supports ensemble learning. XGBoost uses a combination of bagging and boosting techniques to improve the performance of individual decision trees. It also has several built-in features to prevent overfitting, such as regularization and early stopping. Additionally, XGBoost allows users to customize the loss function and evaluation metric to fit their specific problem. Overall, XGBoost is a powerful tool for ensemble learning and is widely used in various machine learning applications, such as classification, regression, and ranking.

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

Principal Component Analysis (PCA) is a popular unsupervised machine learning technique used for dimensionality reduction. The main goal of PCA is to identify the most important patterns in the data by reducing the number of variables while retaining the majority of the original information.

PCA works by identifying the principal components in the data, which are the linear combinations of the original variables that explain the maximum amount of variance in the data. These components are then used to transform the original data into a lower-dimensional space.
PCA has several applications in machine learning, including:
Dimensionality Reduction: PCA is used to reduce the dimensionality of the data by eliminating the redundant or noisy features. This can help to simplify the data and improve the performance of other machine learning algorithms.
Visualization: PCA can be used to visualize high-dimensional data in 2D or 3D space by projecting the data onto the principal components. This can help to identify clusters or patterns in the data.
Data Compression: PCA can be used to compress the data by reducing the number of variables while retaining most of the original information. This can help to reduce storage requirements and improve the efficiency of machine learning algorithms.
In our notebook, we may use PCA for dimensionality reduction, visualization, or data compression, depending on the specific problem and dataset. PCA can help us to identify the most important features and reduce the noise in the data, which can lead to better performance of other machine learning algorithms. Additionally, PCA can help us to visualize the data and gain insights into its underlying structure, which can aid in exploratory data analysis and model interpretation.

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

The `StandardScaler` class from sklearn is used for feature scaling or normalization. Feature scaling is a technique used in machine learning to standardize the range of features or variables so that they have similar scales. This is important because machine learning algorithms often perform better when the input features are normalized, as it can prevent some features from dominating others and making the algorithm biased towards those features.
The `StandardScaler` class scales the features such that they have zero mean and unit variance. Specifically, it subtracts the mean of each feature and divides it by its standard deviation. This transformation ensures that the resulting features have a mean of zero and a standard deviation of one.
In our notebook, we may use the `StandardScaler` class to preprocess our data before training a machine learning model. We can apply the `fit_transform` method of the `StandardScaler` class to our training data to learn the mean and standard deviation of each feature and transform the data accordingly. We can then apply the `transform` method of the `StandardScaler` class to our test data to ensure that both the training and test data are normalized using the same mean and standard deviation values.
Overall, the `StandardScaler` class is a useful API for normalizing the features of our dataset and improving the performance of our machine learning models

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?

The statement `model = XGBClassifier(max_depth=5, objective='multi:softprob', n_estimators=1000, num_classes=10)` is defining a model with specific parameters.

Here is a brief explanation of each parameter:
`max_depth`: This parameter specifies the maximum depth of each decision tree in the ensemble. It controls the level of complexity of the model and helps to prevent overfitting. A lower value of `max_depth` can lead to underfitting, while a higher value can lead to overfitting.
`objective`: This parameter specifies the loss function to be optimized during the training process. In this case, `multi:softprob` indicates that we are using a multi-class version of the softmax loss function. This is appropriate for classification problems with more than two classes.
`n_estimators`: This parameter specifies the number of decision trees to be used in the ensemble. Increasing this value can improve the performance of the model, but can also increase the training time and memory requirements.
`num_classes`: This parameter specifies the number of classes in the classification problem. This is required for multi-class classification problems, where each instance can belong to one of several classes.
Overall, the statement is defining an instance of the `XGBClassifier` class, which is a gradient boosting algorithm that uses decision trees as the base learners. The specific parameter values chosen in this statement will be used during the training process to construct the ensemble of decision trees.
Note that this statement does not train the model yet. We will need to provide training data to the `fit` method of the `XGBClassifier` object to train the model using the specified parameters.

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

In the machine learning pipeline, the `fit` function is used to train the model on the training data. Specifically, the `fit` function applies an algorithm to the training data to learn the patterns and relationships between the input features and the target variable.
During the training process, the model adjusts its internal parameters to minimize a chosen loss function or objective function. This is done through an iterative process, where the algorithm makes predictions on the training data and updates the model parameters based on the error between the predicted values and the true values.
After the training process is complete, the resulting model can be used to make predictions on new, unseen data.
In summary, the `fit` function is a crucial step in the machine learning pipeline, as it allows us to train the model on the available data and prepare it for making predictions on new data. Without this step, we would not be able to take advantage of the predictive power of machine learning algorithms.