nizer-simple-xgb-classifier-6ed5c1

April 7, 2023

[1]:

```
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
```

[2]:

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

[3]:

```
df_train
```

[3]:

```
       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]
```

[4]:

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

[4]:

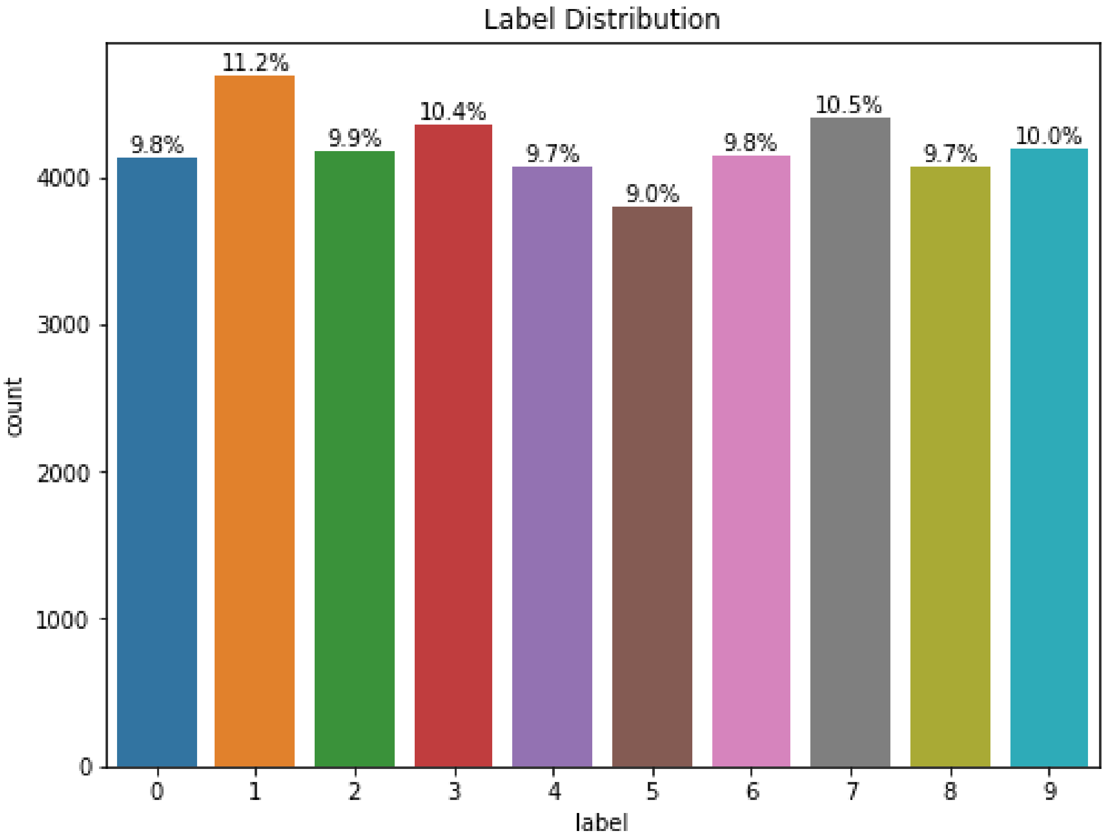
```
array([1, 0, 4, 7, 3, 5, 8, 9, 2, 6])
```

1 Explanatory Data Analysis

[5]:

```
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')
```



```
[6]: df_train.describe()
```

```
[6]:               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]
```

```
[7]: df_train.sum(axis=1)
```

```
[7]: 0        16650
     1        44609
     2        13426
     3        15029
     4        51093
              …
     41995    29310
     41996    13416
     41997    31511
     41998    26387
     41999    18187
     Length: 42000, dtype: int64
```

```
[8]: df_train.shape
```

```
[8]: (42000, 785)
```

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

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

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

```
[10]: 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
```

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

```
[12]: from sklearn.preprocessing import StandardScaler

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

```
[13]: del df_train
```

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

```
[15]: Y_sklearn
```

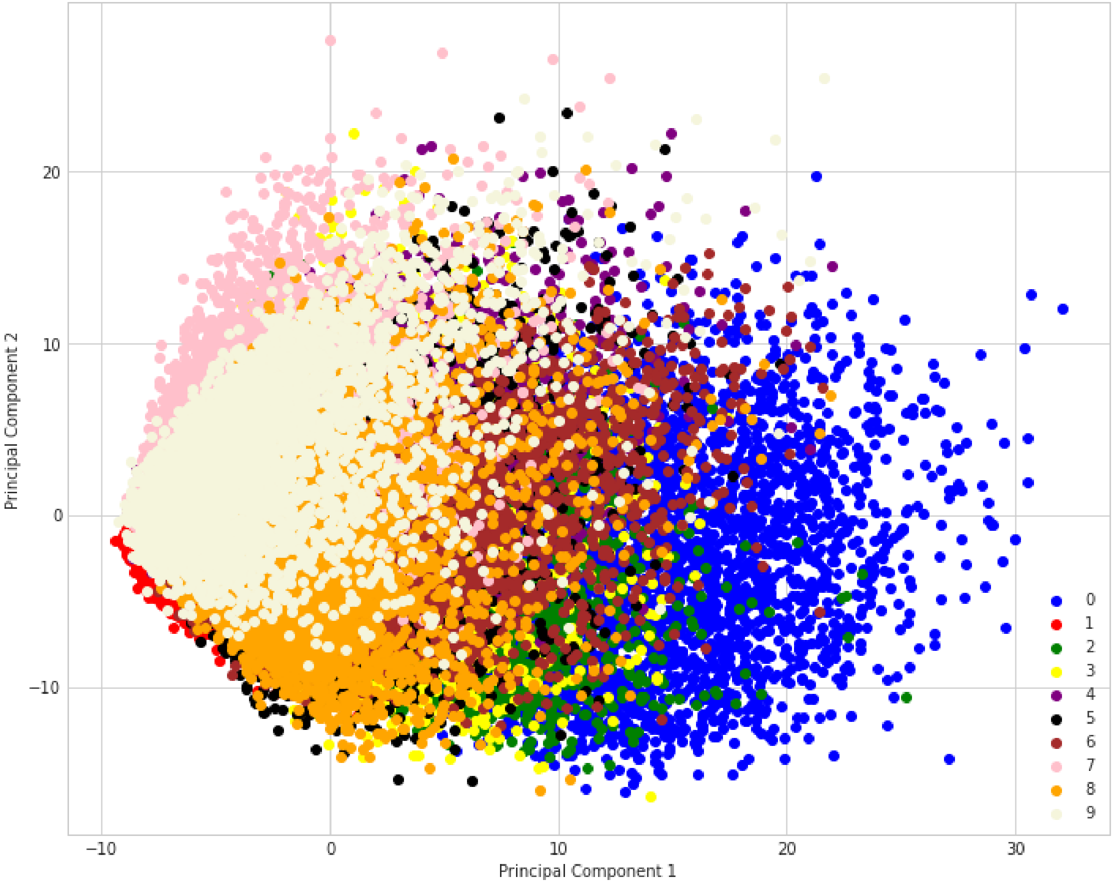
```
[15]: array([[-5.27218318, -5.22774493],
             [19.38080907,  6.06213098],
             [-7.83436529, -1.70777172],
             …,
             [ 0.60965064,  7.06767455],
             [ 2.25996494, -4.33643289],
             [-4.89810691,  1.55410746]])
```

```
[16]: #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()
```



[17]:
```
features.index
```

[17]:
```
RangeIndex(start=0, stop=42000, step=1)
```

[18]:

```
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)
```

[19]:

```
# 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)
```

[20]:

```
result
```

[20]:

```
            PCA0      PCA1       PCA2
0      -5.272200 -5.227576   3.888980
1      19.380803  6.062140   1.339064
2      -7.834355 -1.707918   2.291291
3      -0.706307  5.845857   2.023411
4      26.648676  6.067638   0.981800
…            …         …         …
41995  13.527937 -1.320166  -3.914860
41996  -9.041423 -1.192330   2.320770
41997   0.609672  7.067932 -12.100110
41998   2.259955 -4.337248   0.714577
41999  -4.898145  1.554266  -2.501381

[42000 rows x 3 columns]
```

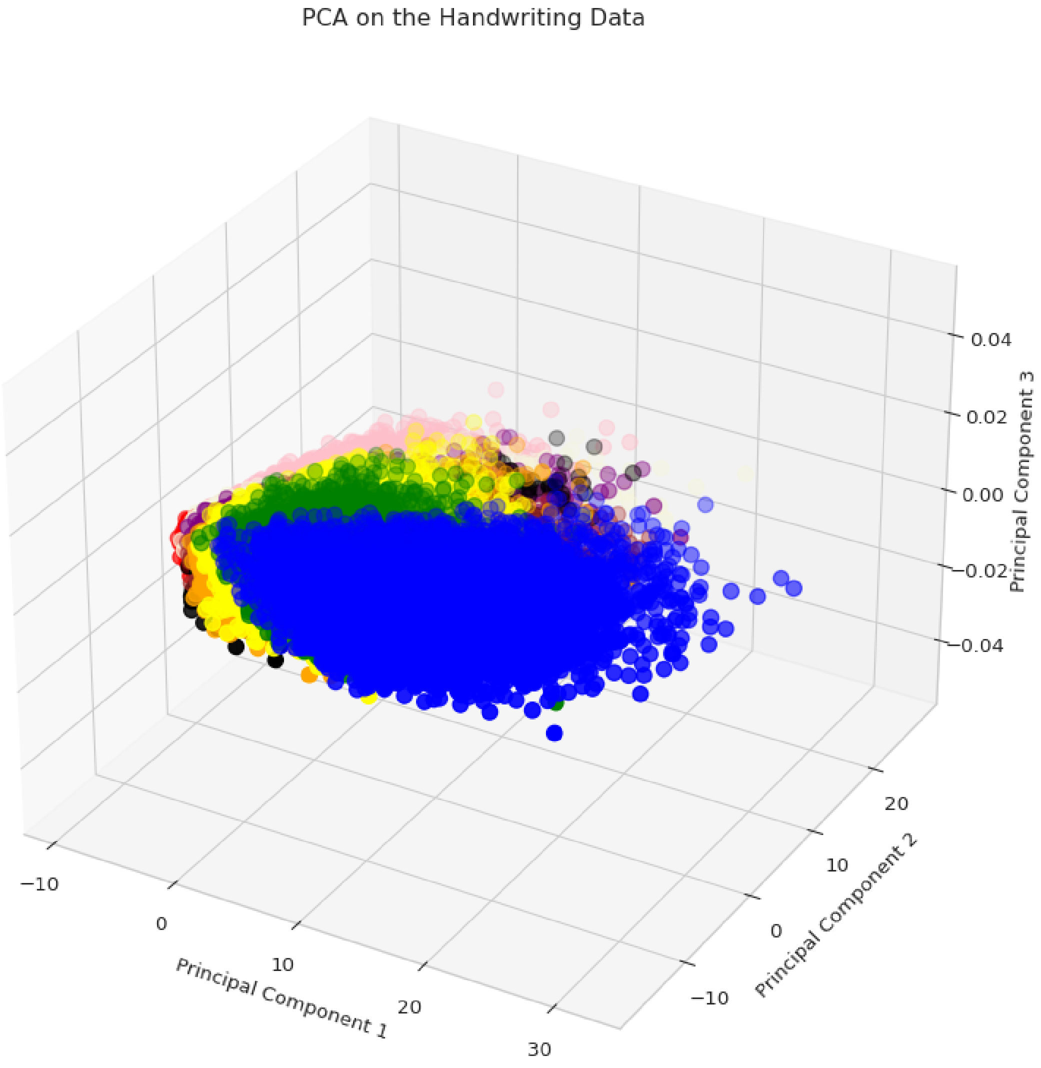
[21]:

```
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>
```



[22]:

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

[23]:

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

2 Making a Model and Predictions

[24]:

```
# 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)
```

```
[11:37:36] 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.


[11:37:36] 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:1.87620
[1]     validation_0-mlogloss:1.69532
[2]     validation_0-mlogloss:1.57450
[3]     validation_0-mlogloss:1.49041
[4]     validation_0-mlogloss:1.42499
[5]     validation_0-mlogloss:1.37516
[6]     validation_0-mlogloss:1.33594
[7]     validation_0-mlogloss:1.30528
[8]     validation_0-mlogloss:1.28138
[9]     validation_0-mlogloss:1.26042
[10]    validation_0-mlogloss:1.24410
[11]    validation_0-mlogloss:1.23011
[12]    validation_0-mlogloss:1.21839
[13]    validation_0-mlogloss:1.20846
[14]    validation_0-mlogloss:1.20028
[15]    validation_0-mlogloss:1.19254
[16]    validation_0-mlogloss:1.18694
[17]    validation_0-mlogloss:1.18069
[18]    validation_0-mlogloss:1.17630
```

```
[19]    validation_0-mlogloss:1.17323
[20]    validation_0-mlogloss:1.17057
[21]    validation_0-mlogloss:1.16672
[22]    validation_0-mlogloss:1.16426
[23]    validation_0-mlogloss:1.16236
[24]    validation_0-mlogloss:1.16045
[25]    validation_0-mlogloss:1.15898
[26]    validation_0-mlogloss:1.15801
[27]    validation_0-mlogloss:1.15613
[28]    validation_0-mlogloss:1.15503
[29]    validation_0-mlogloss:1.15392
[30]    validation_0-mlogloss:1.15284
[31]    validation_0-mlogloss:1.15198
[32]    validation_0-mlogloss:1.15184
[33]    validation_0-mlogloss:1.15075
[34]    validation_0-mlogloss:1.15035
[35]    validation_0-mlogloss:1.14970
[36]    validation_0-mlogloss:1.14949
[37]    validation_0-mlogloss:1.14931
[38]    validation_0-mlogloss:1.14892
[39]    validation_0-mlogloss:1.14844
[40]    validation_0-mlogloss:1.14843
[41]    validation_0-mlogloss:1.14848
[42]    validation_0-mlogloss:1.14835
[43]    validation_0-mlogloss:1.14812
[44]    validation_0-mlogloss:1.14821
[45]    validation_0-mlogloss:1.14781
[46]    validation_0-mlogloss:1.14776
[47]    validation_0-mlogloss:1.14804
[48]    validation_0-mlogloss:1.14821
[49]    validation_0-mlogloss:1.14820
[50]    validation_0-mlogloss:1.14821
[51]    validation_0-mlogloss:1.14821
[52]    validation_0-mlogloss:1.14837
[53]    validation_0-mlogloss:1.14841
[54]    validation_0-mlogloss:1.14838
[55]    validation_0-mlogloss:1.14846
[56]    validation_0-mlogloss:1.14864
[57]    validation_0-mlogloss:1.14880
[58]    validation_0-mlogloss:1.14898
[59]    validation_0-mlogloss:1.14932
[60]    validation_0-mlogloss:1.14910
[61]    validation_0-mlogloss:1.14920
[62]    validation_0-mlogloss:1.14941
[63]    validation_0-mlogloss:1.14962
[64]    validation_0-mlogloss:1.14978
[65]    validation_0-mlogloss:1.14941
[66]    validation_0-mlogloss:1.14915
```

```
[67]	validation_0-mlogloss:1.14919
[68]	validation_0-mlogloss:1.14951
[69]	validation_0-mlogloss:1.14995
[70]	validation_0-mlogloss:1.15012
[71]	validation_0-mlogloss:1.15042
[72]	validation_0-mlogloss:1.15072
[73]	validation_0-mlogloss:1.15102
[74]	validation_0-mlogloss:1.15121
[75]	validation_0-mlogloss:1.15103
[76]	validation_0-mlogloss:1.15137
[77]	validation_0-mlogloss:1.15157
[78]	validation_0-mlogloss:1.15196
[79]	validation_0-mlogloss:1.15205
[80]	validation_0-mlogloss:1.15232
[81]	validation_0-mlogloss:1.15233
[82]	validation_0-mlogloss:1.15241
[83]	validation_0-mlogloss:1.15252
[84]	validation_0-mlogloss:1.15278
[85]	validation_0-mlogloss:1.15281
[86]	validation_0-mlogloss:1.15290
[87]	validation_0-mlogloss:1.15296
[88]	validation_0-mlogloss:1.15307
[89]	validation_0-mlogloss:1.15325
[90]	validation_0-mlogloss:1.15331
[91]	validation_0-mlogloss:1.15385
[92]	validation_0-mlogloss:1.15409
[93]	validation_0-mlogloss:1.15446
[94]	validation_0-mlogloss:1.15469
[95]	validation_0-mlogloss:1.15475
Accuracy:  , 0.559
```

[25]:

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

[26]:

```python
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)}")
```

```
[11:37:52] 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.


[11:37:56] 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
```

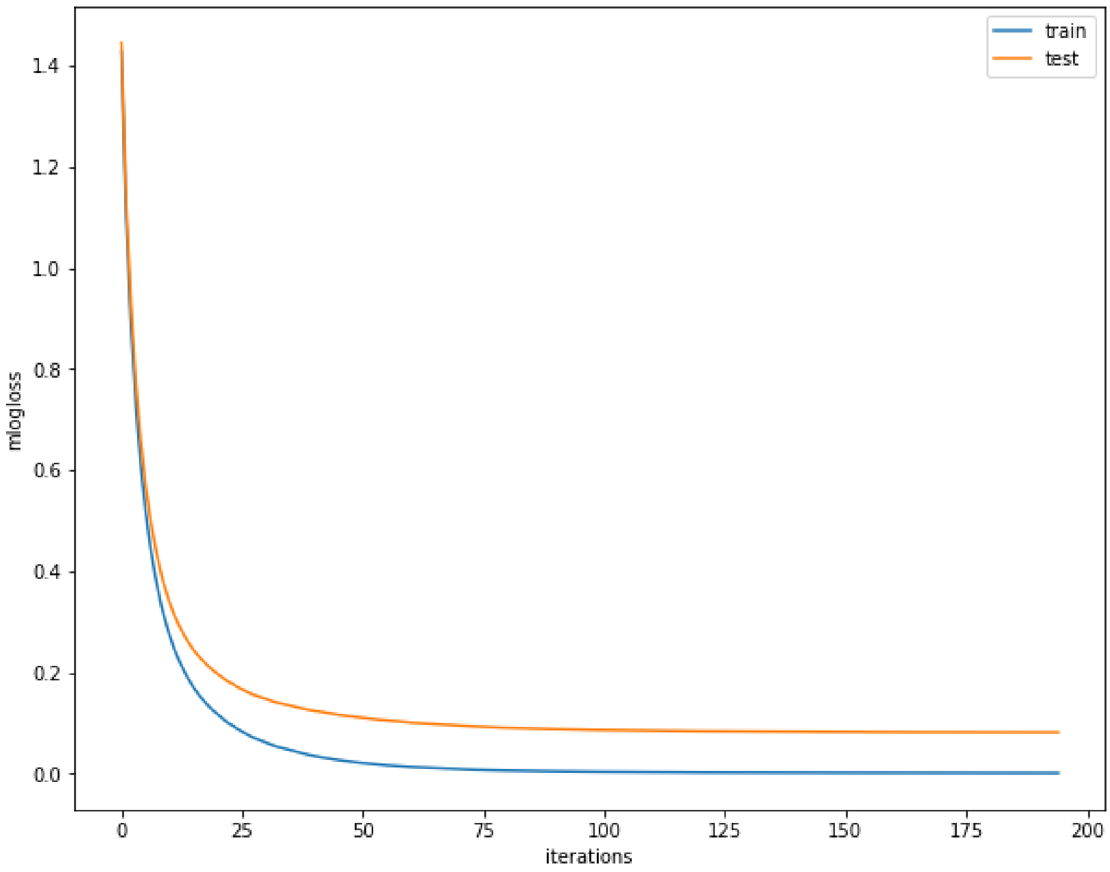
```
[183]	validation_0-mlogloss:0.00081	validation_1-mlogloss:0.08103
[184]	validation_0-mlogloss:0.00081	validation_1-mlogloss:0.08102
[185]	validation_0-mlogloss:0.00080	validation_1-mlogloss:0.08100
[186]	validation_0-mlogloss:0.00080	validation_1-mlogloss:0.08102
[187]	validation_0-mlogloss:0.00079	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
```

[27]:

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

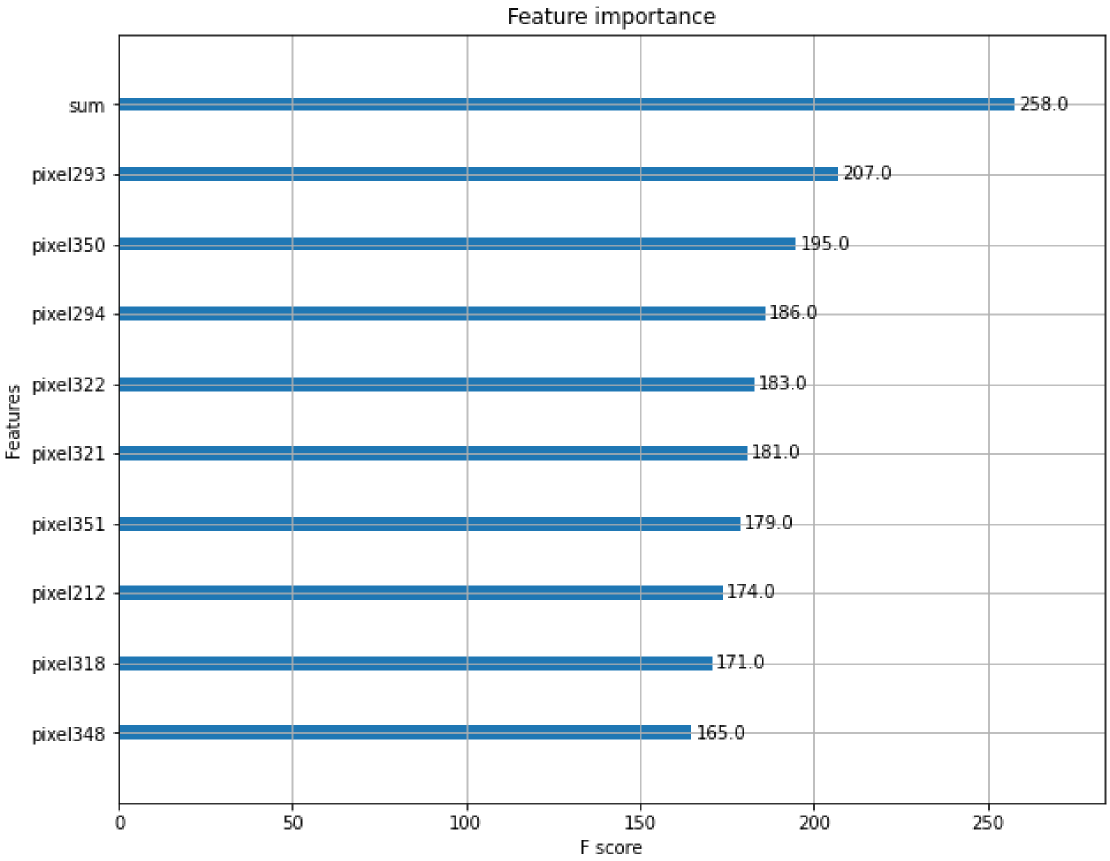
[28]:

```
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()
```



[29]:

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



[30]:
```
predictions = model.predict(df_test)
```

[31]:
```
output = pd.read_csv("../input/digit-recognizer/sample_submission.csv")
output['Label'] = predictions
output.to_csv('submission.csv',index=False)
```

##After Learning the above Notebook one has to take print of it and answer following questions in writing and collectively complete write up.

#Q.1 What is Decision Tree Algorithm ? Which type of ML we can solve using Decision Tree?

#ANSWER:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Decision trees are a popular supervised learning method that like many other learning methods we've seen, can be used for both regression and classification. Decision trees in machine learning (ML) are used to structure algorithms. A decision tree algorithm helps split dataset features with a cost function. Through a process called pruning, the trees are grown before being optimized to remove branches that use irrelevant features.

#Q.2 What do you mean by ensemble learning ? Does XGBoost support ensemble learning ?

#ANSWER:

Ensemble learning is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models. Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc. There are two main reasons to use an ensemble over a single model, and they are related; they are: Performance: An ensemble can make better predictions and achieve better performance than any single contributing model. Robustness: An ensemble reduces the spread or dispersion of the predictions and model performance.

XGBoost, that is, Extreme Gradient Boosting, is a very popular machine learning ensemble technique that has helped data scientists across the globe to achieve great results with phenomenal accuracy. XGBoost is built on the principles of ensemble modeling and is an improved version of the Gradient Boosted Machine algorithm. In general, the XgBoost algorithm creates multiple classifiers that are weak learners, which means a model that gives a bit better accuracy than just a random guess. The learner in the ensemble model can be a linear or tree model that is built iteratively with random sampling along with an added weight from the learnings of the previously built model. At each step, a tree is built and the cases where the tree has failed to classify an outcome correctly is assigned a corresponding weight. The next iteration of model building learns from the mistakes of the previous model. Gradient boosting refers to a class of ensemble machine learning algorithms that can be used for classification or regression predictive modeling problems.

Ensembles are constructed from decision tree models. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. This is a type of ensemble machine learning model referred to as boosting.

#Q.3 What is Principal Component Analysis ? Why do we use PCA in our notebook?

#ANSWER:

PCA is a widely covered machine learning method on the web, and there are some great articles about it, but many spend too much time in the weeds on the topic, when most of us just want to know how it works in a simplified way.

Principal component analysis can be broken down into five steps. I'll go through each step, providing logical explanations of what PCA is doing and simplifying mathematical concepts such as standardization, covariance, eigenvectors and eigenvalues without focusing on how to compute them.

PCA helps you interpret your data, but it will not always find the important patterns. Principal component analysis (PCA) simplifies the complexity in high-dimensional data while retaining trends and patterns. It does this by transforming the data into fewer dimensions, which act as summaries of features. PCA is used to visualize multidimensional data. It is used to reduce the number of dimensions in healthcare data. PCA can help resize an image. When PCA is used as part of preprocessing, the algorithm is applied to: Reduce the number of dimensions in the training dataset. De-noise the data. Because PCA is computed by finding the components which explain the greatest amount of variance, it captures the signal in the data and omits the noise.

#Q.4 Check use of “StandardScalar” class from sklearn in notebook. What do you think is this API used for?

#ANSWER:

Python sklearn library offers us with StandardScaler() function to standardize the data values into a standard format. According to the above syntax, we initially create an object of the StandardScaler() function. Further, we use fit_transform() along with the assigned object to transform the data and standardize it. StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way. StandardScaler can be influenced by outliers (if they exist in the dataset) since it involves the estimation of the empirical mean and standard deviation of each feature. When the features of the given dataset fluctuate significantly within their ranges or are recorded in various units of measurement, StandardScaler enters the picture.

The data are scaled to a variance of 1 after the mean is reduced to 0 via StandardScaler. But when determining the empirical mean of the data and standard deviation, outliers present in data have a significant impact that reduces the spectrum of characteristic values.

Many machine learning algorithms may encounter issues due to these variations in the starting features. For algorithms that calculate distance, for instance, if any of the dataset’s features have values having large or completely different ranges, that particular feature of the dataset will control the distance calculation.

The StandardScaler function of sklearn is based on the theory that the dataset’s variables whose values lie in different ranges do not have an equal contribution to the model’s fit parameters and training function and may even lead to bias in the predictions made with that model.

#Q.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?

#ANSWER:

#“model = XGBClassifier(max_depth=5, objective=‘multi:softprob’, n_estimators=1000, num_classes=10)” This statement is used for model training and model fitting it determines and gives the parameter such as the maximum depth of the tree and no. of trees, categorize the class_id and also it gives the objective parameter essentially gives a fuzzy clustering for the distinct probability of belonging to each class.

#max_depth=5: This determines the maximum depth of the tree. In our case, we use a depth of two to make our decision tree. The default value is set to none. This will often result in over-fitted decision trees.

#n_estimators=1000 n_estimators: The n_estimators parameter specifies the number of trees in the forest of the model. The default value for this parameter is 1000, which means that 10 different decision trees will be constructed in the random forest

#objective='multi:softprob': The multi:softprob objective parameter essentially gives us a fuzzy clustering in which each observation is given a distinct probability of belonging to each class. In order to use these probabilities for classification, we will have to determine the max probability for each observation and assign a class.

#num_classes=10 the num_classes still means the maximum category_id and this setting does not mean to include the background class. I checked the dateset/create_pascal_tfrecord.py and found the class label is the category_id (for pascal it ranges in [1, 20]). Setting num_classes to 21 merely adds an extra non-sense class, but does not influence the original 20 classes. The maximum category_id is 21 now, but there is no data of this class. Maybe this is a minor mistake. I suggest using the last setting num_classes: 20.

#What step in ML pipeline fit fuction carries out?

#ANSWER:

The fit() method takes the training data as arguments, which can be one array in the case of unsupervised learning, or two arrays in the case of supervised learning. Note that the model is fitted using X and y , but the object holds no reference to X and y . Pipelines allow us to streamline this process by compiling the preparation steps while easing the task of model tuning and monitoring. Scikit-Learn's Pipeline class provides a structure for applying a series of data transformations followed by an estimator. Step 1: Data Preprocessing. The first step in any pipeline is data preprocessing. Step 2: Data Cleaning. Next, this data flows to the cleaning step. Step 3: Feature Engineering. Step 4: Model Selection. Step 5: Prediction Generation. One definition of an ML pipeline is a means of automating the machine learning workflow by enabling data to be transformed and correlated into a model that can then be analyzed to achieve outputs. This type of ML pipeline makes the process of inputting data into the ML model fully automated. Before training a model, you should split your data into a training set and a test set. Each dataset will go through the data cleaning and preprocessing steps before you put it in a machine learning model.

It's not efficient to write repetitive code for the training set and the test set. This is when the scikit-learn pipeline comes into play.

Scikit-learn pipeline is an elegant way to create a machine learning model training workflow.