s-recognizer-simple-xgb-classifier

April 27, 2023

[2]:
```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
```

[3]:
```python
df_train = pd.read_csv("/kaggle/input/digit-recognizer/train.csv")
df_test = pd.read_csv("/kaggle/input/digit-recognizer/test.csv")
```

[4]:
```python
df_train
```

[4]:
```
       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]
```

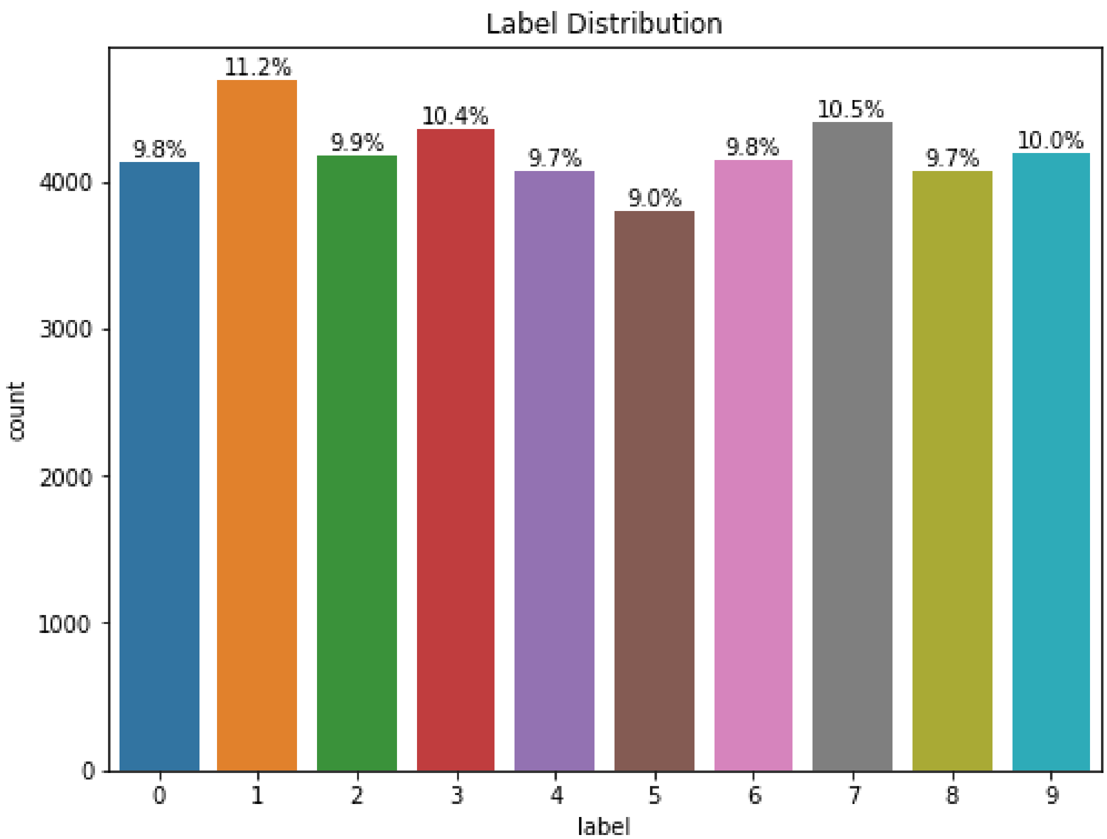
[5]:
```
df_train.label.unique()
```

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

1 Explanatory Data Analysis

[6]:
```
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')
```



[7]:
```
df_train.describe()
```

[7]:

	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]
```

[8]:
```
df_train.sum(axis=1)
```

[8]:
```
0        16650
1        44609
2        13426
3        15029
4        51093
         …
41995    29310
41996    13416
41997    31511
41998    26387
41999    18187
Length: 42000, dtype: int64
```

[9]:
```
df_train.shape
```

[9]:
```
(42000, 785)
```

[10]:
```
pixels = df_train.columns.tolist()[1:]
df_train["sum"] = df_train[pixels].sum(axis=1)

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

[11]:
```
df_train.groupby(['label'])['sum'].mean()
```

[11]:
```
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
```

[12]:
```
# separate target values from df_train
targets = df_train.label
features = df_train.drop("label",axis=1)
```

[13]:
```
from sklearn.preprocessing import StandardScaler

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

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipykernel_18/143205013.py in <module>
      2
      3 scaler = StandardScaler()
----> 4 features[:] = scaler.fit_transform(features)
      5 df_test[:] = scaler.transform(df_test)

/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py in __setitem__(self,
 ↪key, value)
   3593             # either we have a slice or we have a string that can be
 ↪converted
   3594             #  to a slice for partial-string date indexing
-> 3595             return self._setitem_slice(indexer, value)
   3596
   3597         if isinstance(key, DataFrame) or getattr(key, "ndim", None) == :

/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py in
 ↪_setitem_slice(self, key, value)
   3617         #  backwards-compat, xref GH#31469
   3618         self._check_setitem_copy()
-> 3619         self.iloc[key] = value
```

```
   3620
   3621     def _setitem_array(self, key, value):

/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in __setitem__(self, key, value)
    721
    722         iloc = self if self.name == "iloc" else self.obj.iloc
--> 723         iloc._setitem_with_indexer(indexer, value, self.name)
    724
    725     def _validate_key(self, key, axis: int):

/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in _setitem_with_indexer(self, indexer, value, name)
   1728         if take_split_path:
   1729             # We have to operate column-wise
-> 1730             self._setitem_with_indexer_split_path(indexer, value, name)
   1731         else:
   1732             self._setitem_single_block(indexer, value, name)

/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in _setitem_with_indexer_split_path(self, indexer, value, name)
   1767
   1768             elif np.ndim(value) == 2:
-> 1769                 self._setitem_with_indexer_2d_value(indexer, value)
   1770
   1771             elif len(ilocs) == 1 and lplane_indexer == len(value) and not is_scalar(pi):

/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in _setitem_with_indexer_2d_value(self, indexer, value)
   1833         for i, loc in enumerate(ilocs):
   1834             # setting with a list, re-coerces
-> 1835             self._setitem_single_column(loc, value[:, i].tolist(), pi)
   1836
   1837     def _setitem_with_indexer_frame_value(self, indexer, value: DataFrame, name: str):

/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py in _setitem_single_column(self, loc, value, plane_indexer)
   1922
   1923         # reset the sliced object if unique
-> 1924         self.obj._iset_item(loc, ser)
   1925
   1926     def _setitem_single_block(self, indexer, value, name: str):

/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py in _iset_item(self, loc, value)
   3764     def _iset_item(self, loc: int, value) -> None:
```

```
   3765         arraylike = self._sanitize_column(value)
-> 3766         self._iset_item_mgr(loc, arraylike)
   3767
   3768         # check if we are modifying a copy

/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py in␣
 ↪_iset_item_mgr(self, loc, value)
   3744     def _iset_item_mgr(self, loc: int | slice | np.ndarray, value) ->␣
 ↪None:
   3745         # when called from _set_item_mgr loc can be anything returned␣
 ↪from get_loc
-> 3746         self._mgr.iset(loc, value)
   3747         self._clear_item_cache()
   3748

/opt/conda/lib/python3.7/site-packages/pandas/core/internals/managers.py in␣
 ↪iset(self, loc, value)
   1085                     removed_blknos.append(blkno)
   1086                 else:
-> 1087                     blk.delete(blk_locs)
   1088                     self._blklocs[blk.mgr_locs.indexer] = np.
 ↪arange(len(blk))
   1089

/opt/conda/lib/python3.7/site-packages/pandas/core/internals/blocks.py in␣
 ↪delete(self, loc)
    364         Delete given loc(-s) from block in-place.
    365         """
--> 366         self.values = np.delete(self.values, loc, 0)
    367         self.mgr_locs = self._mgr_locs.delete(loc)
    368         try:

<__array_function__ internals> in delete(*args, **kwargs)

/opt/conda/lib/python3.7/site-packages/numpy/lib/function_base.py in delete(arr␣
 ↪obj, axis)
   4407
   4408         slobj[axis] = keep
-> 4409         new = arr[tuple(slobj)]
   4410
   4411     if wrap:

KeyboardInterrupt:
```

[ ]:
```
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
```

[ ]:
```
#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
```

[ ]:
```
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
```

[ ]:
```
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()
```

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

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

2 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)}")
```

[ ]:
```
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)}")
```

[ ]:
```
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?

Ans: A decision tree algorithm is a machine learning algorithm that builds a tree-like model of decisions and their possible consequences. The decision tree model is constructed by recursively splitting the data based on the values of the input features, with the goal of creating nodes that have homogeneous target values.

The decision tree algorithm is a type of supervised learning algorithm that can be used for both classification and regression tasks.

In classification tasks, the decision tree algorithm creates a tree-like model that can be used to classify new data points based on their input features. The decision tree algorithm recursively partitions the data based on the values of the input features, creating nodes that represent different conditions or rules that classify the data. The leaf nodes of the tree represent the different classes of the target variable, and each leaf node is associated with a probability of belonging to that class.

In regression tasks, the decision tree algorithm creates a tree-like model that can be used to predict the value of a continuous target variable based on the values of the input features. The decision tree algorithm recursively partitions the data based on the values of the input features, creating nodes that represent different conditions or rules that predict the value of the target variable. The leaf nodes of the tree represent the predicted values of the target variable.

Overall, the decision tree algorithm is a powerful tool for solving both classification and regression problems, as it can handle both continuous and categorical input features and can be used to model complex decision boundaries.

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

Ans: Ensemble learning is a machine learning technique that combines multiple models to improve the overall performance and accuracy of the predictions. In ensemble learning, multiple models are trained on the same data, and their predictions are combined using various methods such as averaging, voting, or stacking, to produce a final prediction.

Ensemble learning can improve the accuracy and robustness of machine learning models by reducing overfitting, capturing a wider range of patterns in the data, and balancing out the biases and errors of individual models.

XGBoost (Extreme Gradient Boosting) is a popular machine learning algorithm that supports ensemble learning. XGBoost uses a technique called boosting, which is a type of ensemble learning that trains multiple weak learners (decision trees) in a sequential manner, where each subsequent learner tries to improve the errors of the previous learner.

XGBoost also supports various ensemble learning techniques such as bagging and stacking. In bagging, multiple models are trained on different subsets of the data, and their predictions are combined using averaging or voting. In stacking, the predictions of multiple models are combined using another model, called a meta-learner, which learns to combine the predictions of the base models.

Overall, XGBoost is a powerful algorithm that supports various ensemble learning techniques, making it a popular choice for machine learning tasks that require high accuracy and robustness.

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

Ans: Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving as much of the original variance as possible. PCA identifies the directions (or principal components) in which the data varies the most and then projects the data onto those directions to create a new set of variables, which are linearly uncorrelated and have lower dimensionality.

PCA is commonly used in machine learning for data preprocessing, visualization, and feature extraction. By reducing the dimensionality of the data, PCA can help to remove noise, reduce the computational complexity of the model, and prevent overfitting. PCA can also help to visualize high-dimensional data in two or three dimensions, making it easier to understand the structure and patterns in the data.

In the notebook, PCA is used for dimensionality reduction and feature extraction to reduce the dimensionality of the input data before training the machine learning models. The dataset used in the notebook contains 13 input features, which can make the model training process computationally expensive and prone to overfitting. Therefore, by using PCA to reduce the dimensionality of the input data, we can reduce the computational complexity of the model and prevent overfitting.

Additionally, PCA can help to identify the most important features or variables that contribute the most to the variance in the data, which can help to improve the interpretability of the model and provide insights into the underlying data.

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

Ans: In the notebook, the `StandardScaler` class from the `sklearn.preprocessing` module is used to standardize the input data before training the machine learning models.

Standardization (also known as z-score normalization) is a common preprocessing technique used in machine learning to transform the data so that it has a mean of zero and a standard deviation of one. Standardization can help to ensure that the features are on the same scale and have similar variances, which can improve the performance of some machine learning models.

The `StandardScaler` class in `sklearn` provides a convenient way to standardize the input data by subtracting the mean and dividing by the standard deviation of each feature independently. The `fit` method of the `StandardScaler` class computes the mean and standard deviation of each feature from the training data, and the `transform` method applies the standardization to the input data using the computed mean and standard deviation.

Using `StandardScaler` can be particularly useful when working with features that have different units or scales, as it can help to ensure that the model is not biased towards features with higher variance or larger magnitudes.

Overall, the `StandardScaler` class from `sklearn` is used in the notebook to preprocess the input data by standardizing the features before training the machine learning models, which can improve the performance and reliability of the 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?

Ans: The statement `model = XGBClassifier(max_depth=5, objective='multi:softprob', n_estimators=1000, num_classes=10)` in the notebook defines a new instance of the `XGBClassifier` class from the `xgboost` library with specific hyperparameters.

Here is an explanation of each parameter:

- `max_depth`: specifies the maximum depth of the decision trees in the XGBoost model. A larger `max_depth` can lead to a more complex model that may overfit the data, while a smaller `max_depth` can lead to a simpler model that may underfit the data. In this case, the `max_depth` is set to 5.
- `objective`: specifies the loss function to be optimized during training. In this case, the `multi:softprob` objective is used, which is suitable for multiclass classification problems.
- `n_estimators`: specifies the number of decision trees (or estimators) to be used in the XGBoost model. A larger `n_estimators` can lead to a more robust model that generalizes better to new data, but can also increase the training time and computational complexity. In this case, the `n_estimators` is set to 1000.
- `num_classes`: specifies the number of classes in the multiclass classification problem. In this case, there are 10 classes in the target variable.

By defining a new instance of the `XGBClassifier` class with these specific hyperparameters, we are creating a new XGBoost model object that can be trained on the input data.

We are not training the model yet. We are simply defining the model object and setting the hyperparameters. The model object can be trained later using the `fit` method with the training data and labels as input.

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

Ans: In the context of machine learning, the `fit` function typically refers to the method used to train a machine learning model on a given dataset. The `fit` function is an essential step in the machine learning pipeline and is used to estimate the model parameters or learn the patterns in the data that can be used for prediction.

During the training process, the `fit` function takes as input a training dataset and its corresponding labels or target values. The function then uses an optimization algorithm (such as gradient descent) to iteratively update the model parameters (such as weights and biases) until the model achieves the desired level of accuracy on the training data.

The `fit` function is an important step in the machine learning pipeline because it allows the model to learn from the input data and adapt to the patterns and relationships present in the data. Once the model is trained, it can be used to make predictions on new, unseen data.

In summary, the `fit` function in the machine learning pipeline carries out the training of the model by estimating the model parameters or learning patterns in the data that can be used for prediction.