lasso-and-rigde

May 8, 2023

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
df=pd.read_csv(r"\\itserver1\ML LAB\50_Startups.csv")
df.head(3)
```

[2]:
```
   R&D Spend  Administration  Marketing Spend       State     Profit
0  165349.20       136897.80        471784.10    New York  192261.83
1  162597.70       151377.59        443898.53  California  191792.06
2  153441.51       101145.55        407934.54     Florida  191050.39
```

[3]:
```
x=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

[4]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[("encoder",␣
 ↪OneHotEncoder(),[3])],remainder='passthrough')
x=np.array(ct.fit_transform(x))
```

[5]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=0)
```

[6]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
```

[7]:
```
LR = LinearRegression()
LS = Lasso()
RR = Ridge()
```

[8]:
```
LR.fit(x_train, y_train)
```

[8]:
```
LinearRegression()
```

[9]:
```
LS.fit(x_train, y_train)
```

```
[9]: Lasso()
```

```
[10]: RR.fit(x_train,y_train)
```

```
[10]: Ridge()
```

```
[11]: y_plr = LR.predict(x_test)
      y_pls = LS.predict(x_test)
      y_pred = RR.predict(x_test)
```

```
[12]: data={"LR_pred":y_plr,"LS_pred":y_pls,"RR_pred":y_pred,"test":y_test}
      df1=pd.DataFrame(data)
      df1.head(3)
```

```
[12]:          LR_pred        LS_pred        RR_pred       test
      0  103015.201598  103019.161042  103094.496229  103282.38
      1  132582.277608  132583.065601  132592.106767  144259.40
      2  132447.738452  132452.734811  132546.032224  146121.95
```

```
[ ]:
```

neural-networks-2

May 8, 2023

[2]:
```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
```

[3]:
```
# Define the model architecture
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=10))
model.add(Dense(1, activation='sigmoid'))
```

[4]:
```
# Compile the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy',␣
  ↪metrics=['accuracy'])
```

[5]:
```
# Generate some random data for training
data = np.random.random((1000, 10))
labels = np.random.randint(2, size=(1000, 1))
```

[6]:
```
# Train the model
model.fit(data, labels, epochs=10, batch_size=32)
```

```
Epoch 1/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6985 - accuracy:
0.5010
Epoch 2/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6963 - accuracy:
0.5020
Epoch 3/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6947 - accuracy:
0.5230
Epoch 4/10
32/32 [==============================] - 0s 1ms/step - loss: 0.6937 - accuracy:
0.5300
Epoch 5/10
32/32 [==============================] - 0s 1ms/step - loss: 0.6929 - accuracy:
0.5220
Epoch 6/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6917 - accuracy:
0.5270
```

```
Epoch 7/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6910 - accuracy:
0.5390
Epoch 8/10
32/32 [==============================] - 0s 1ms/step - loss: 0.6906 - accuracy:
0.5220
Epoch 9/10
32/32 [==============================] - 0s 2ms/step - loss: 0.6903 - accuracy:
0.5250
Epoch 10/10
32/32 [==============================] - 0s 1ms/step - loss: 0.6892 - accuracy:
0.5260
```

[6]:

```
<keras.callbacks.History at 0x23eb4a9ef70>
```

[7]:

```python
# Evaluate the model
test_data = np.random.random((100, 10))
test_labels = np.random.randint(2, size=(100, 1))
score = model.evaluate(test_data, test_labels, batch_size=32)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.6907 - accuracy:
0.4800
Test loss: 0.6906949877738953
Test accuracy: 0.47999998927116394
```

[ ]:

```

```

neural-networks

May 8, 2023

[35]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()

# Split the data into labeled and unlabeled sets
X_labeled, X_unlabeled, y_labeled, _ = train_test_split(iris.data, iris.target,␣
 ↪test_size=0.3,random_state=42, stratify=iris.target)

# Train a logistic regression model on the labeled data
clf = LogisticRegression(max_iter=1000)
clf.fit(X_labeled, y_labeled)


kmeans = KMeans(n_clusters=3)
kmeans.fit(X_labeled,y_labeled)

# Assign pseudo-labels to the unlabeled data based on the cluster assignments
pseudo_labels1 = kmeans.predict(X_unlabeled)
#unlabeled_data['class'] = pseudo_labels

# Generate pseudo-labels for the unlabeled data using the trained model
pseudo_labels = clf.predict(X_unlabeled)

# Combine the labeled and pseudo-labeled data
X_combined = np.concatenate((X_labeled, X_unlabeled), axis=0)
y_combined = np.concatenate((y_labeled, pseudo_labels1), axis=0)

# Train a new model on the combined data
clf_pseudo = LogisticRegression(max_iter=1000)
clf_pseudo.fit(X_combined, y_combined)

# Evaluate the performance of the model on the test set
```

```
X_test, y_test = iris.data, iris.target
score = clf_pseudo.score(X_test, y_test)
print(f"Accuracy: {score}")
```

```
C:\Users\UDAY\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

```
Accuracy: 0.9466666666666667
```

```
[ ]:
```

In [ ]:

```
#1)a.Implementation of Find-S algorithm

import pandas as pd
import numpy as np

d = pd.read_csv("dataset.csv")

print(d)

a = np.array(d)[:,:-1]
print(" The attributes are: ",a)

t = np.array(d)[:,-1]
print("The target is: ",t)

def fun(c,t):
     for i, val in enumerate(t):
            if val == "Yes":
                specific_hypothesis = c[i].copy()
                break

     for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
        return specific_hypothesis
print(" The final hypothesis is:",train(a,t))
```

In [ ]:

```
#Decison Tree Exercise-3
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)

from sklearn.preprocessing import StandardScaler

s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)

from sklearn.tree import DecisionTreeClassifier

k=DecisionTreeClassifier()
k.fit(X_train,Y_train)
y_pred=k.predict(X_test)

from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test,y_pred)*100)#
```

In [ ]:

```
#4)a.Implementation of Simple Linear Regression Algorithm using Python
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

data_set= pd.read_csv('Salary_Data.csv')

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_st

#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)

#Prediction of Test and Training set result
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)

mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
mtp.title("Salary vs Experience (Training Dataset)")
mtp.xlabel("Years of Experience")
mtp.ylabel("Salary(In Rupees)")
mtp.show()
```

In [ ]:

```
#Logistic Regression Exercise-4b
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.linear_model import LogisticRegression
l=LogisticRegression()
l.fit(X_train,Y_train)
y_pred=l.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,y_pred)*100)
```

In [ ]:

```
#Binary Classifier Exercise-4c
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive (1)\heart.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.ensemble import RandomForestClassifier
```

```
r=RandomForestClassifier()
r.fit(X_train,Y_train)
y_pred=r.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,y_pred)*100)
```

In [ ]:

```
# 5.estimate the bias and variance for a regression model
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
dataframe = read_csv(url, header=None)
# separate into inputs and outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_s
# define the model
model = LinearRegression()
# estimate bias and variance
mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test, los
# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# load the dataset
data = pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv", header=0)

# encode the target variable
label_encoder = LabelEncoder()
label_ids = label_encoder.fit_transform(data['Species'])
onehot_encoder = OneHotEncoder(sparse=False)
reshaped = label_ids.reshape(len(label_ids), 1)
targetvar = onehot_encoder.fit_transform(reshaped)

# get the independent variables
inde_vars = []
for col in data.columns:
    if col not in ['Id', 'Species']:
        inde_vars.append(col)

# split the data into training and testing sets
X = data[inde_vars]
y = targetvar
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, random_st

# normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)

# train the model using Naive Bayes algorithm
```

```
gnb = GaussianNB()
gnb.fit(X_train, Y_train)

# make predictions on test data
y_pred = gnb.predict(X_test)

# evaluate the model accuracy
accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", accuracy * 100)
```

In [ ]:

```
#KNN Exercise-8
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.neighbors import KNeighborsClassifier
k=KNeighborsClassifier(n_neighbors=4)
k.fit(X_train,Y_train)
y_pred=k.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,y_pred)*100)
```

In [ ]:

```
#NaiveBayes Exercise-10
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.naive_bayes import GaussianNB
g=GaussianNB()
g.fit(X_train,Y_train)
y_pred=g.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,y_pred)*100)
```

In [ ]:

```
#11.Python Program to Implement the K-Means and Estimation & MAximization Algorithm

from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']
```

```
dataset = pd.read_csv("8-dataset.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

In [ ]:

```
#NaiveBayes Exercise-13
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive (1)\heart.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.naive_bayes import GaussianNB
g=GaussianNB()
g.fit(X_train,Y_train)
y_pred=g.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,y_pred)*100)
```

In [ ]:

```
#PCA and SVM Exercise-14
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.decomposition import PCA
principal=PCA(n_components=3)
principal.fit(X_train)
principal.fit(X_test)
from sklearn.svm import SVC
s=SVC()
s.fit(X_train,Y_train)
y_pred=s.predict(X_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,y_pred)*100)
```

In [ ]:

```
#PCA Exercise-15
import numpy as mp
import pandas as pd
d=pd.read_csv(r"C:\Users\20B91A12J0\Downloads\archive\Iris.csv",header=0)

x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 1].values

from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
s=StandardScaler()
X_train=s.fit_transform(X_train)
X_test=s.fit_transform(X_test)
from sklearn.tree import DecisionTreeClassifier
k=DecisionTreeClassifier()
k.fit(X_train,Y_train)
y_pred=k.predict(X_test)
from sklearn.metrics import accuracy_score
print("Before PCA")
print(accuracy_score(Y_test,y_pred)*100)
from sklearn.decomposition import PCA
principal=PCA(n_components=3)
X_Train1=principal.fit_transform(X_train)
X_Test1=principal.fit_transform(X_test)
k1=DecisionTreeClassifier()
k1.fit(X_Train1,Y_train)
Y_pred1=k1.predict(X_Test1)
print("After PCA")
print(accuracy_score(Y_test,Y_pred1)*100)
```

In [ ]:

implementing-neural-networks

May 8, 2023

[3]:
```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\20b91a12d1\anaconda3\lib\site-
packages (22.2.2)
Collecting pip
  Downloading pip-23.1.1-py3-none-any.whl (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 2.1 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.2.2
    Uninstalling pip-22.2.2:
      Successfully uninstalled pip-22.2.2
Successfully installed pip-23.1.1
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
pip install tensorflow
```

```
Collecting tensorflowNote: you may need to restart the kernel to use updated
packages.

  Using cached tensorflow-2.12.0-cp39-cp39-win_amd64.whl (1.9 kB)
Collecting tensorflow-intel==2.12.0 (from tensorflow)
  Using cached tensorflow_intel-2.12.0-cp39-cp39-win_amd64.whl (272.8 MB)
Collecting absl-py>=1.0.0 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached absl_py-1.4.0-py3-none-any.whl (126 kB)
Collecting astunparse>=1.6.0 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: flatbuffers>=2.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (23.3.3)
Collecting gast<=0.4.0,>=0.2.1 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached gast-0.4.0-py3-none-any.whl (9.8 kB)
Collecting google-pasta>=0.1.1 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached google_pasta-0.2.0-py3-none-any.whl (57 kB)
Requirement already satisfied: h5py>=2.9.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (3.7.0)
Collecting jax>=0.3.15 (from tensorflow-intel==2.12.0->tensorflow)
```

```
  Using cached jax-0.4.8-py3-none-any.whl
Requirement already satisfied: libclang>=13.0.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (16.0.0)
Requirement already satisfied: numpy<1.24,>=1.22 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (1.23.5)
Collecting opt-einsum>=2.3.2 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached opt_einsum-3.3.0-py3-none-any.whl (65 kB)
Requirement already satisfied: packaging in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (21.3)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (4.22.3)
Requirement already satisfied: setuptools in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (63.4.1)
Requirement already satisfied: six>=1.12.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (4.3.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (1.14.1)
Collecting grpcio<2.0,>=1.24.3 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached grpcio-1.54.0-cp39-cp39-win_amd64.whl (4.1 MB)
Collecting tensorboard<2.13,>=2.12 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached tensorboard-2.12.2-py3-none-any.whl (5.6 MB)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (2.12.0)
Collecting keras<2.13,>=2.12.0 (from tensorflow-intel==2.12.0->tensorflow)
  Using cached keras-2.12.0-py2.py3-none-any.whl (1.7 MB)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from tensorflow-
intel==2.12.0->tensorflow) (0.31.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
astunparse>=1.6.0->tensorflow-intel==2.12.0->tensorflow) (0.37.1)
Collecting ml-dtypes>=0.0.3 (from jax>=0.3.15->tensorflow-
intel==2.12.0->tensorflow)
```

```
  Using cached ml_dtypes-0.1.0-cp39-cp39-win_amd64.whl (120 kB)
Requirement already satisfied: scipy>=1.7 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from jax>=0.3.15->tensorflow-
intel==2.12.0->tensorflow) (1.9.1)
Collecting google-auth<3,>=1.6.3 (from tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow)
  Using cached google_auth-2.17.3-py2.py3-none-any.whl (178 kB)
Collecting google-auth-oauthlib<1.1,>=0.5 (from
tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow)
  Using cached google_auth_oauthlib-1.0.0-py2.py3-none-any.whl (18 kB)
Requirement already satisfied: markdown>=2.6.8 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (3.3.4)
Requirement already satisfied: requests<3,>=2.21.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (2.28.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (0.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (1.8.1)
Requirement already satisfied: werkzeug>=1.0.1 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (2.0.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from packaging->tensorflow-
intel==2.12.0->tensorflow) (3.0.9)
Collecting cachetools<6.0,>=2.0.0 (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow)
  Using cached cachetools-5.3.0-py3-none-any.whl (9.3 kB)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow)
(0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow)
(4.9)
Collecting requests-oauthlib>=0.7.0 (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow)
  Using cached requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: charset-normalizer<3,>=2 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
```

```
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow) (2022.9.14)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from
pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow)
(0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in
c:\users\20b91a12d1\anaconda3\lib\site-packages (from requests-
oauthlib>=0.7.0->google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow-
intel==2.12.0->tensorflow) (3.2.2)
Installing collected packages: opt-einsum, ml-dtypes, keras, grpcio, google-
pasta, gast, cachetools, astunparse, absl-py, requests-oauthlib, jax, google-
auth, google-auth-oauthlib, tensorboard, tensorflow-intel, tensorflow
Successfully installed absl-py-1.4.0 astunparse-1.6.3 cachetools-5.3.0
gast-0.4.0 google-auth-2.17.3 google-auth-oauthlib-1.0.0 google-pasta-0.2.0
grpcio-1.54.0 jax-0.4.8 keras-2.12.0 ml-dtypes-0.1.0 opt-einsum-3.3.0 requests-
oauthlib-1.3.1 tensorboard-2.12.2 tensorflow-2.12.0 tensorflow-intel-2.12.0
```

[9]:

```python
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

[10]:

```python
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data
x_train = x_train.reshape((60000, 28*28)) / 255.0
x_test = x_test.reshape((10000, 28*28)) / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

[11]:

```python
# Define the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(28*28,)),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

[12]:

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=32, validation_data=(x_test,␣
 ↪y_test))
```

```
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2611 -
accuracy: 0.9246 - val_loss: 0.1301 - val_accuracy: 0.9619
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1132 -
accuracy: 0.9660 - val_loss: 0.1009 - val_accuracy: 0.9688
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0780 -
accuracy: 0.9760 - val_loss: 0.0858 - val_accuracy: 0.9728
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0585 -
accuracy: 0.9820 - val_loss: 0.0822 - val_accuracy: 0.9754
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0449 -
accuracy: 0.9863 - val_loss: 0.0713 - val_accuracy: 0.9789
```

[12]:

```
<keras.callbacks.History at 0x229bbb84a30>
```

[8]:

```
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)

# Print the test accuracy
print('Test accuracy:', test_accuracy)
```

```
313/313 - 0s - loss: 0.0895 - accuracy: 0.9730 - 359ms/epoch - 1ms/step
Test accuracy: 0.9729999899864197
```

[ ]:

```

```