**The Zhejiang University Artificial Intelligence Innovation Winter Training**

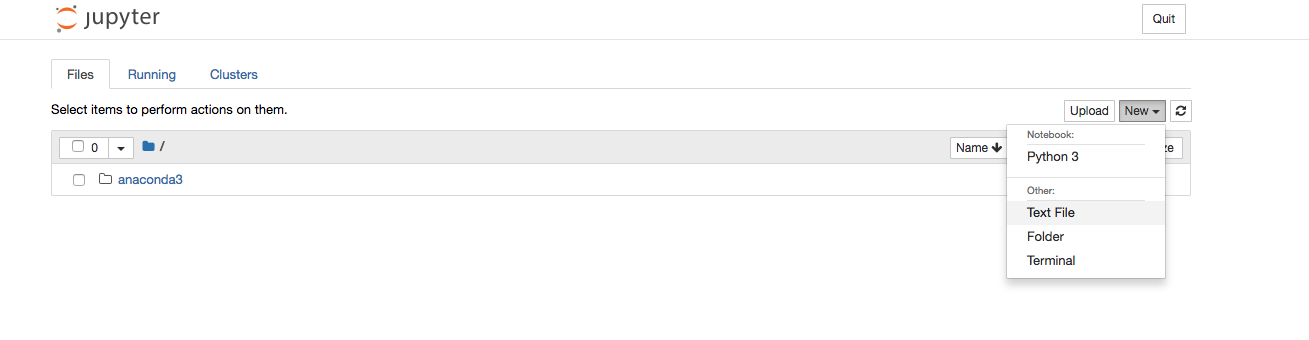
**AI Workshop**

**10/02/2019**

**1. A Brief Introduction to Python**

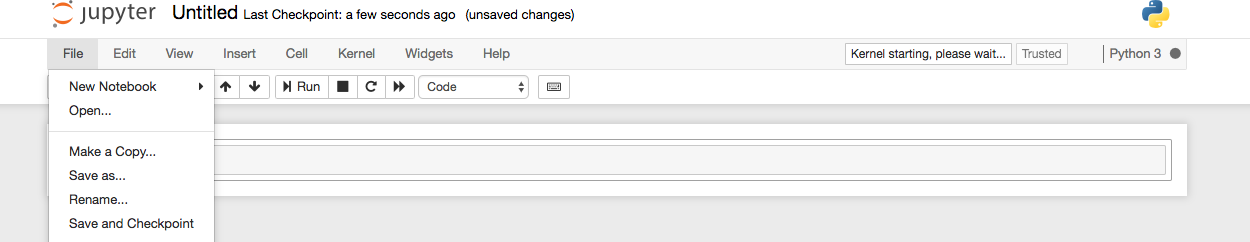
Open the link provided in a web browser. Create a new Python3 session.

(IMPORTANT: PLEASE DO NOT DELETE anaconda3 FOLDER.)



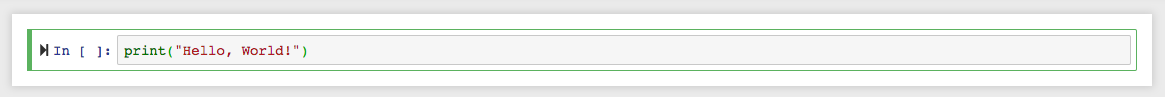


Save the session.





* 1. **Hello World**



Run the cell by pressing shift+entre.

* 1. **Variables**

Type and run the following code in the cell to understand how python handles variables.

***x = 1***

***y = “Hello”***

***z = “ world”***

***print(x,y)***

***print(y+z)***

Try running the following code,

***print(x+z)***

Can you explain the output and correct the error to get “5.1 world”?

* 1. **Numbers**

Type and run the following code in the cell to understand how python handles numbers.

***x = 1***

***y = 1.5***

***print(type(x))***

***print(type(y))***

***print(type(x+y))***

* 1. **Functions**

A function is a block of code which only runs when it is called.

Run the code,

***def my\_function():***

***print("Hello world")***

Then run,

**my\_function()**

* 1. **Modules**

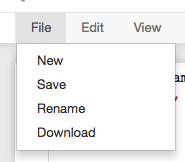
Consider a module to be the same as a code library.

Create a new txt file and entre the code,

***def greeting(name):***

***print("Hello, " + name)***

Rename the file as test.py





Type and run the code in the original notebook,

***import test***

***test.greeting(“Mike”)***

Let’s try a different way of importing the module,

***from test import greeting***

***greeting(“Mike”)***

**1.6 Plot a graph**

Run the code

***import matplotlib.pyplot as plt***

***import numpy as np***

***print(range(10))***

***plt.plot(range(10))***

***plt.scatter(np.random.rand(10),np.random.rand(10))***

**2. Machine Learning with Python**

**2.1 Linear Regression**

2.1.1 Introduction

Linear regression is used for finding linear relationship between target and one or more predictors. In this task, we will implement linear regression by scikit-learn, which is a widely-used Python module for machine learning. Scikit-learn provides many off-the-shelf unsupervised and supervised learning algorithms. These machine learning algorithms can be used in a Python program with just several lines of codes. What you need to do is just import the module like this:

**from** sklearn **import** linear\_model

2.1.2 Data Set

In this task, we use a diabetes dataset provided by scikit-learn. The characteristics of this dataset is shown as follows:

|  |  |
| --- | --- |
| Number of Instances | 442 |
| Number of Attributes | First 10 columns are numeric predictive values |
| Target | A quantitative measure of disease progression one year after baseline |
| Attribute Information | * Age * Sex * BMI * Average blood pressure * S1 * S2 * S3 * S4 * S5 * S6 |

In order to use this dataset in our program, we need to import datasets from sklearn and load the diabetes dataset.

**from** sklearn **import** datasets, linear\_model

diabetes = datasets.load\_diabetes()

2.1.3 Linear Regression with One Variable

This task we will choose one feature of the diabetes dataset, in order to illustrate a two-dimensional plot of this liner regression technique. The straight line can be seen in the plot, showing how linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

**Tips:**

1. The attributes data loaded from the dataset is a 442×10 matrix. We can choose certain feature like this:

diabetes\_X = diabetes.data[:, np.newaxis, 2]

The number 2 means which column you want to choose. You can change it to try other attributes. Print diabetes\_X to see what the data looks like. For the target data (Y) you can get is by diabetes.target.

1. In machine learning, we usually divide to dataset into a training set and a test set. Try to use the last 20 instances as the test set and others as the training set.
2. How to use the linear\_model:

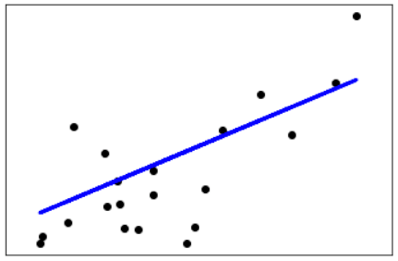
regr = linear\_model.LinearRegression()regr.fit(X\_train, y\_train)Y\_pred = regr.predict(X\_test)

**Task:**

Use the template code below to implement the linear regression with one variable plot the result graph.

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
( )  
*# Load the diabetes dataset*  
( )  
*# Use only one feature*( )  
*# Split the data into training/testing sets*( )  
*# Split the targets into training/testing sets*( )  
*# Create linear regression object*regr = linear\_model.LinearRegression()  
*# Train the model using the training sets*( )  
*# Make predictions using the testing set*( )  
*# Plot outputs*( )  
plt.xticks(())  
plt.yticks(())  
plt.show()

The result graph should look like this:



2.1.4 Linear Regression with Two Variable

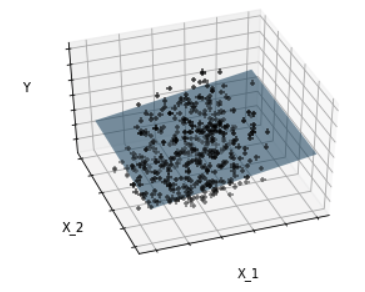
This task we will choose two features of the diabetes dataset, in order to illustrate a three-dimensional plot of the liner regression technique. Similar to the last task.

**Task:**

Use the template code below to implement the linear regression with two variable plot the result graph.

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** mpl\_toolkits.mplot3d **import** Axes3D  
( ) *# Define the Plot function***def** plot\_figs(fig\_num, elev, azim, X\_train, clf):  
 fig = plt.figure(fig\_num, figsize=(4, 3))  
 plt.clf()  
 ax = Axes3D(fig, elev=elev, azim=azim)  
 ax.scatter(X\_train[:, 0], X\_train[:, 1], y\_train, c=**'k'**, marker=**'+'**)  
 ax.plot\_surface(np.array([[-.1, -.1], [.15, .15]]),  
 np.array([[-.1, .15], [-.1, .15]]),  
 clf.predict(np.array([[-.1, -.1, .15, .15],  
 [-.1, .15, -.1, .15]]).T  
 ).reshape((2, 2)),  
 alpha=.5)  
 ax.set\_xlabel(**'X\_1'**)  
 ax.set\_ylabel(**'X\_2'**)  
 ax.set\_zlabel(**'Y'**)  
 ax.w\_xaxis.set\_ticklabels([])  
 ax.w\_yaxis.set\_ticklabels([])  
 ax.w\_zaxis.set\_ticklabels([])  
*#Generate the figure*elev = 43.5  
azim = -110  
plot\_figs(1, elev, azim, X\_train, ols)  
plt.show()

The result graph should look like this:



**2.2 Nearest Neighbours Classification**

**Dataset description**

Iris plants dataset

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information:

|  |
| --- |
| Sepal length in cm |
| Sepal width in cm |
| Petal length in cm |
| Petal width in cm |
| Class: Iris-Setosa, Iris-Versicolour, Iris-Virginica |

**Key class:** sklearn.neighbors.KNeighborsClassifier(n\_neighbors=5, weights=’uniform’)

**Parameters:**

n\_neighbors : int, optional (default = 5) Number of neighbors to use by default for kneighbors queries.

weights : str or callable, optional (default = ‘uniform’) weight function used in prediction. Possible values:

‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.

‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

**Task:**

**Reorder the following code and run the nearest neighbours classification with python.**

|  |
| --- |
| cmap\_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])  cmap\_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF']) |
| Z = Z.reshape(xx.shape)  plt.figure()  plt.pcolormesh(xx, yy, Z, cmap=cmap\_light) |
| plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap\_bold, edgecolor='k', s=20)  plt.xlim(xx.min(), xx.max())  plt.ylim(yy.min(), yy.max())  plt.title("3-Class classification (k = %i, weights = '%s')" % (n\_neighbors, weights)) |
| clf = neighbors.KNeighborsClassifier(n\_neighbors, weights=‘distance’) |
| X = iris.data[:, :2] |
| x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  xx, yy = np.meshgrid(np.arange(x\_min, x\_max, .02), np.arange(y\_min, y\_max, .02)) |
| clf.fit(X, y) |
| y = iris.target |
| iris = datasets.load\_iris() |
| n\_neighbors = 15 |
| Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()]) |
| import numpy as np  import matplotlib.pyplot as plt  from matplotlib.colors import ListedColormap  from sklearn import neighbors, datasets |

**Hint:**

#import some modules and classes

# import some data and assign a value to number of neighbour parameter

# take the first two features

# create colour maps

# create an instance of Neighbours Classifier and fit the data.

# plot the decision boundary. For that, we will assign a colour to each point in the mesh [x\_min, x\_max]x[y\_min, y\_max].

# put the result into a colour plot

# plot also the training points

**Questions:**

What would happen if we change the parameters in “KNeighborsClassifier”? Figure out the impact of changing the parameters on the classification result.

**2.3 Support Vector Machine (SVM)**

**Dataset description**

Same as 2.2

**Key class:** sklearn.svm.SVC(kernel=’rbf’, gamma=’ auto\_deprecated’)

**Parameters:**

kernel : string, optional (default=’rbf’)

Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used.

gamma : float, optional (default=’auto’)

Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.

**Task:**

**Reorder the following code and run the nearest neighbours classification with python.**

|  |
| --- |
| X\_train = X[:int(.9 \* n\_sample)]  y\_train = y[:int(.9 \* n\_sample)]  X\_test = X[int(.9 \* n\_sample):]  y\_test = y[int(.9 \* n\_sample):] |
| import numpy as np  import matplotlib.pyplot as plt  from sklearn import datasets, svm |
| clf = svm.SVC(kernel=’linear’, gamma=10) |
| iris = datasets.load\_iris()  X = iris.data  y = iris.target |
| XX, YY = np.mgrid[x\_min:x\_max:200j, y\_min:y\_max:200j]  Z = clf.decision\_function(np.c\_[XX.ravel(), YY.ravel()]) |
| X = X[y != 0, :2]  y = y[y != 0] |
| Z = Z.reshape(XX.shape)  plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)  plt.contour(XX, YY, Z, colors=['k', 'k', 'k'], linestyles=['--', '-', '--'], levels=[-.5, 0, .5])  plt.title(kernel) |
| n\_sample = len(X) |
| plt.scatter(X\_test[:, 0], X\_test[:, 1], s=80, facecolors='none', zorder=10, edgecolor='k')  plt.axis('tight')  x\_min = X[:, 0].min()  x\_max = X[:, 0].max()  y\_min = X[:, 1].min()  y\_max = X[:, 1].max() |
| clf.fit(X\_train, y\_train) |
| order = np.random.permutation(n\_sample)  X = X[order]  y = y[order].astype(np.float) |
| plt.figure(’linear’)  plt.clf()  plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired, edgecolor='k', s=20) |

Hint:

#import some modules and classes

# import some data

# take the first two features

# shuffle the dataset

# fit the model

# plot the dataset

# circle out the test data

# put the result into a colour plot

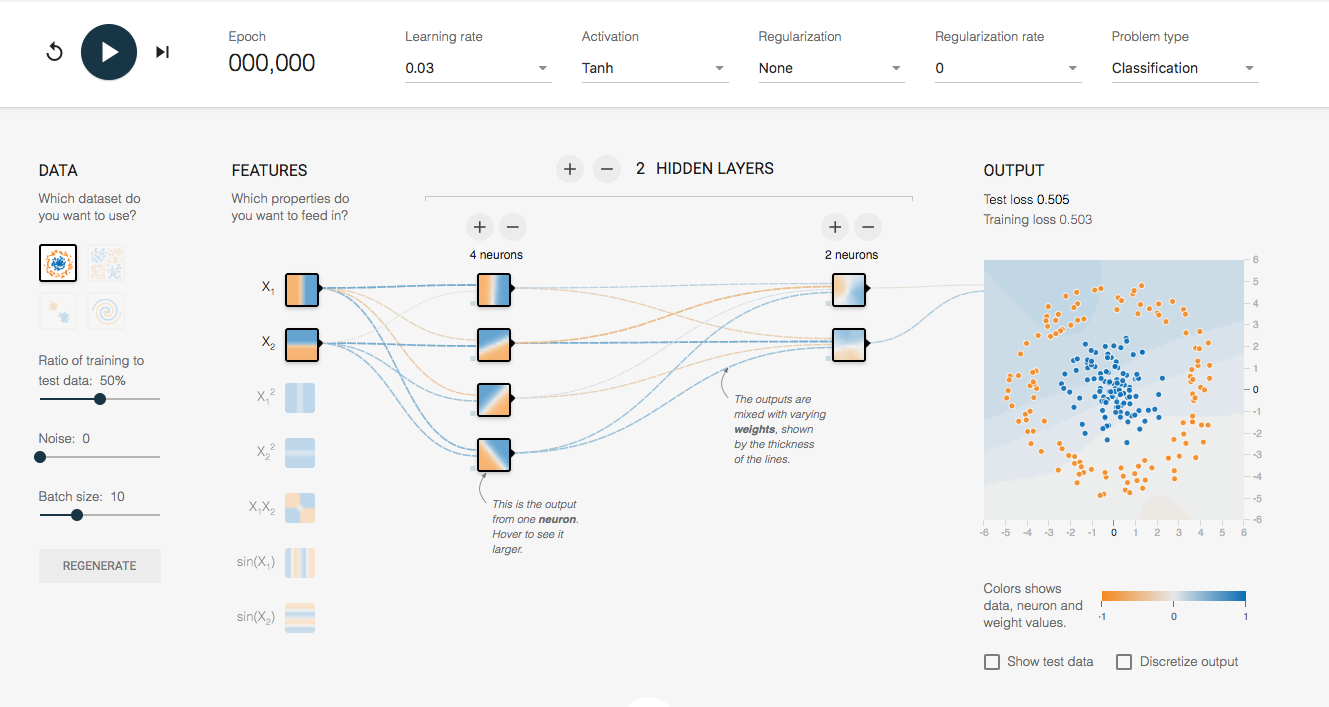
**Question**:

Given the modest result of linear SVC, can you think of a method to improve the performance of your SVM?

Hint: try gaussian kernel (RBF)

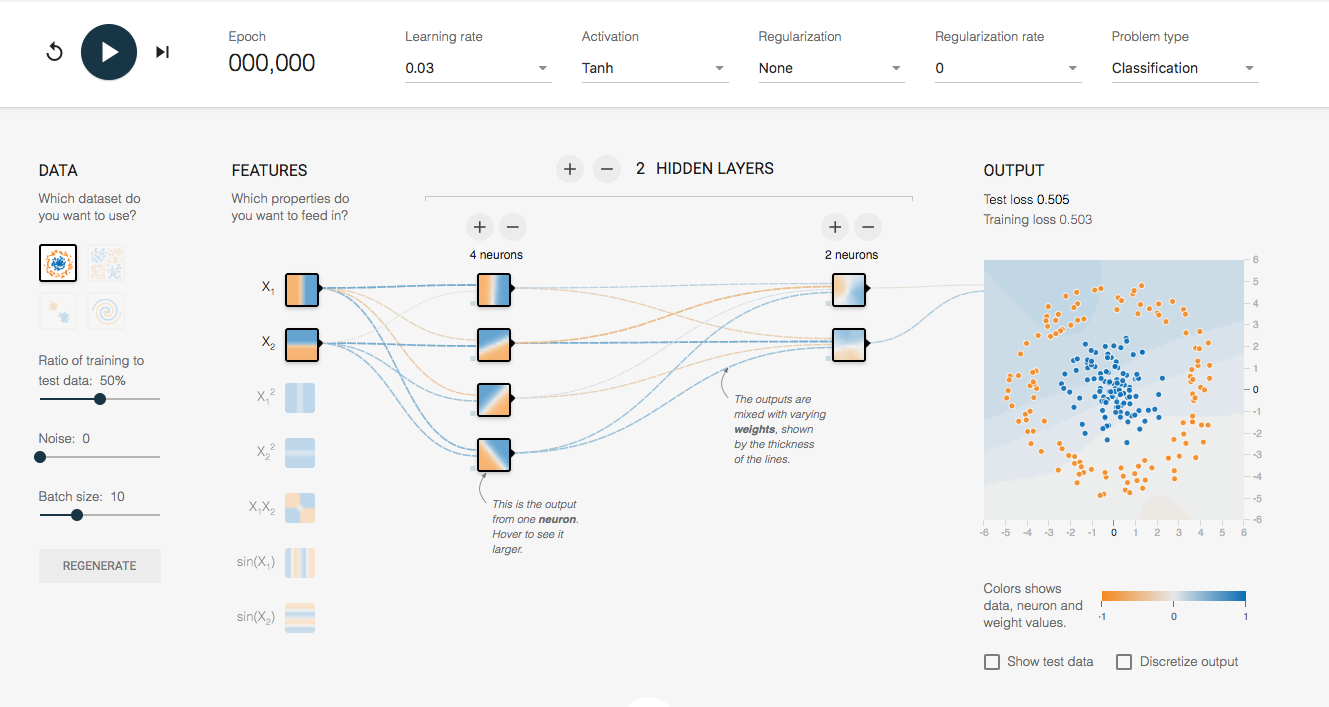
**2.4 A Neural Network Playground**

Start training your own neural network by clicking the ‘play’ button.





Tweak the parameters and initial settings to see if you can reduce the test loss.



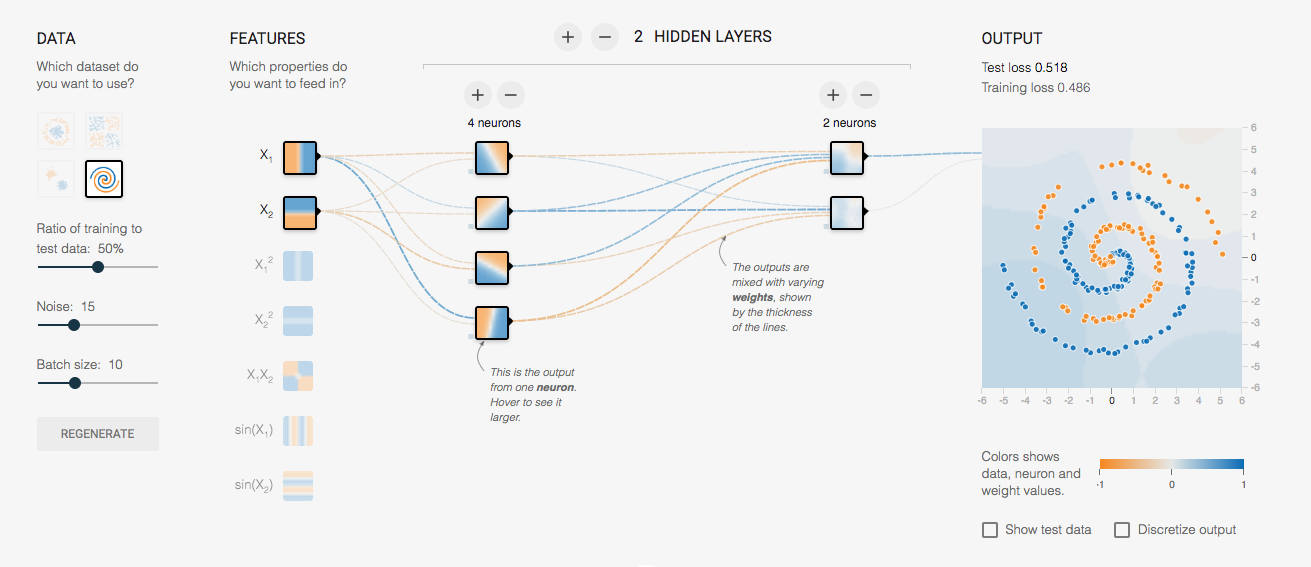


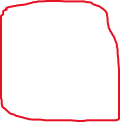
**Question**

For the first classification task, the default neural network structure should very well serve the purpose. What if we use this setup to solve more sophisticated task? Can you find out a method to improve the test loss? What would happen if we add some noise?

Hint:

You need a more complex neural network to deal with a complex problem.

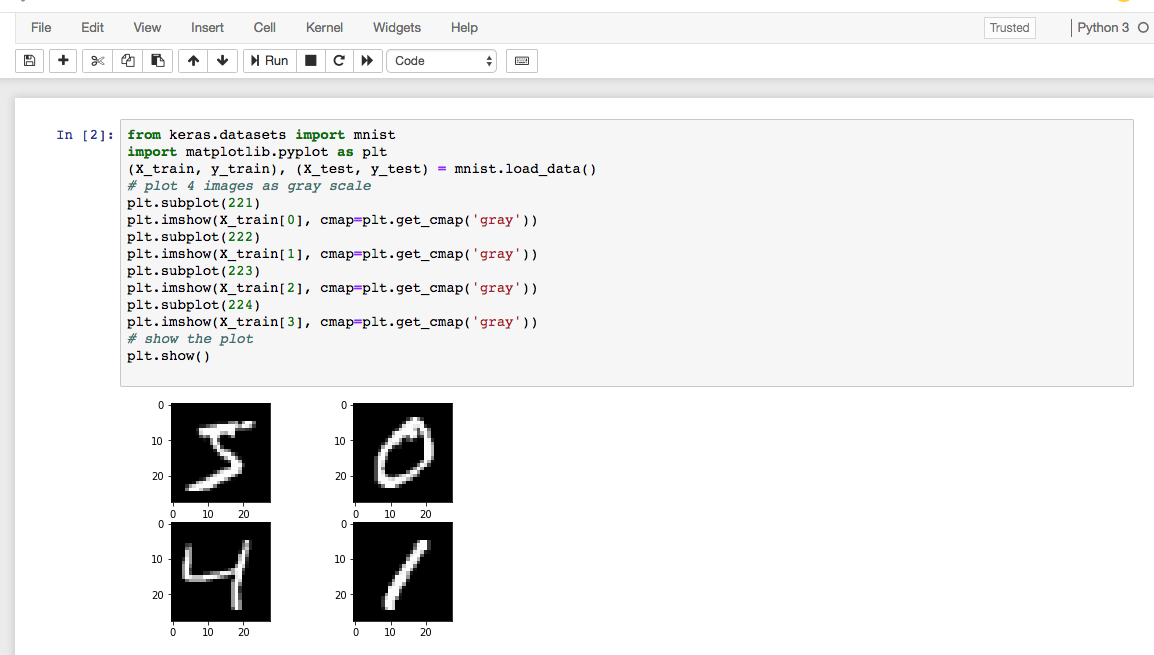




**2.4 Build your neural network with Python**

Finally it is your time to build your own neural network to solve a real problem.

Restart the kernel of your notebook and run the following code.



It might take a while to download and show the dataset we need for this part. The dataset is called MNIST, a commonly used one for learning neural network. Our task is to build a neural network that can classify these images.

**Task:**

As usual, a list of code is provided with you and please reorder the code to solve this problem.

|  |
| --- |
| X\_train = X\_train.reshape(X\_train.shape[0], num\_pixels).astype('float32')  X\_test = X\_test.reshape(X\_test.shape[0], num\_pixels).astype('float32') |
| model = baseline\_model() |
| def baseline\_model():  model = Sequential()  model.add(Dense(num\_pixels, input\_dim=num\_pixels, kernel\_initializer='normal', activation='relu'))  model.add(Dense(num\_classes, kernel\_initializer='normal', activation='softmax')) |
| import numpy  from keras.models import Sequential  from keras.layers import Dense  from keras.layers import Dropout  from keras.utils import np\_utils  import numpy as np |
| model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=10, batch\_size=200, verbose=2) |
| y\_train = np\_utils.to\_categorical(y\_train)  y\_test = np\_utils.to\_categorical(y\_test)  num\_classes = y\_test.shape[1]  y\_train.shape |
| model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  return model |
| scores = model.evaluate(X\_test, y\_test, verbose=0)  print("Baseline Error: %.2f%%" % (100-scores[1]\*100)) |
| num\_pixels = X\_train.shape[1] \* X\_train.shape[2] |
| X\_train = X\_train.reshape(X\_train.shape[0], num\_pixels).astype('float32')  X\_test = X\_test.reshape(X\_test.shape[0], num\_pixels).astype('float32') |

Hint:

#import some modules and classes

#flatten 28\*28 images to a 784 vector for each image

#flattening numpy array 28\*28 to 784

# normalize inputs from 0-255 to 0-1

# one hot encode outputs

# define baseline model

# compile model

# build the model

# fit the model

# final evaluation of the model

Question:

We have not mentioned the most famous Convolutional Neural Network (CNN) and its derivatives so far. CNN is a powerful network widely used in computer vision. The fundamental idea of CNN for classification is to automatic learn the ‘features’ of images in order to classify them.

The following code constructs a typical CNN for classifying the MNIST dataset. Try to understand and run it if you are interested. The test accuracy should be over 99% after 10 epochs’ training.

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

batch\_size = 128

num\_classes = 10

epochs = 10

# input image dimensions

img\_rows, img\_cols = 28, 28

# the data, split between train and test sets

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

if K.image\_data\_format() == 'channels\_first':

x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols)

x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols)

input\_shape = (1, img\_rows, img\_cols)

else:

x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)

x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)

input\_shape = (img\_rows, img\_cols, 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

# convert class vectors to binary class matrices

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

activation='relu',

input\_shape=input\_shape))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical\_crossentropy,

optimizer=keras.optimizers.Adadelta(),

metrics=['accuracy'])

model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

verbose=1,

validation\_data=(x\_test, y\_test))

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])