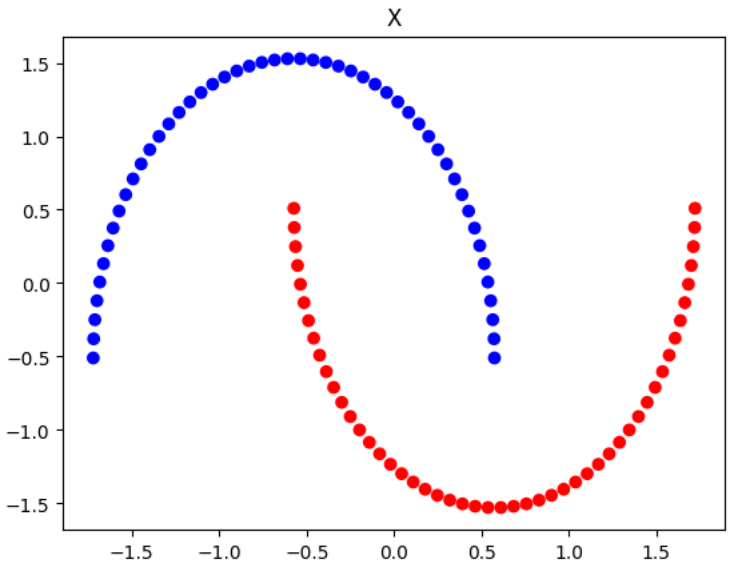
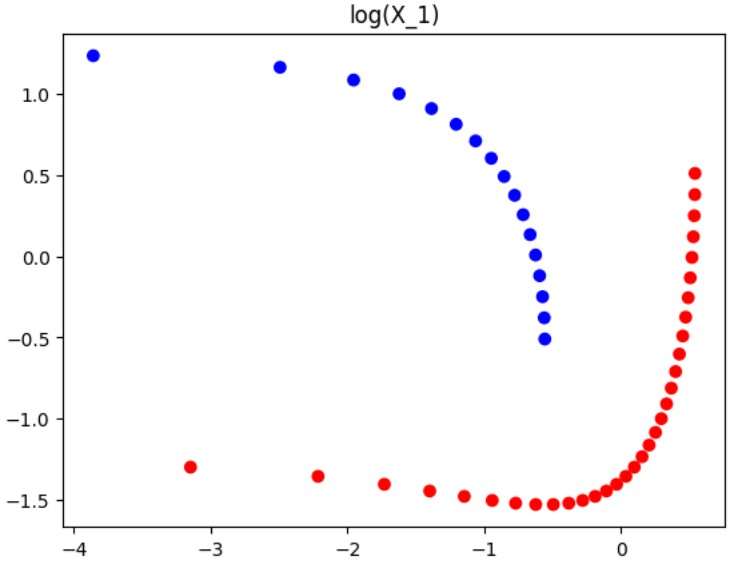
Week 4 Notebook - Linear Regression, Linear Discriminant, Logistic Regression

Basis Functions

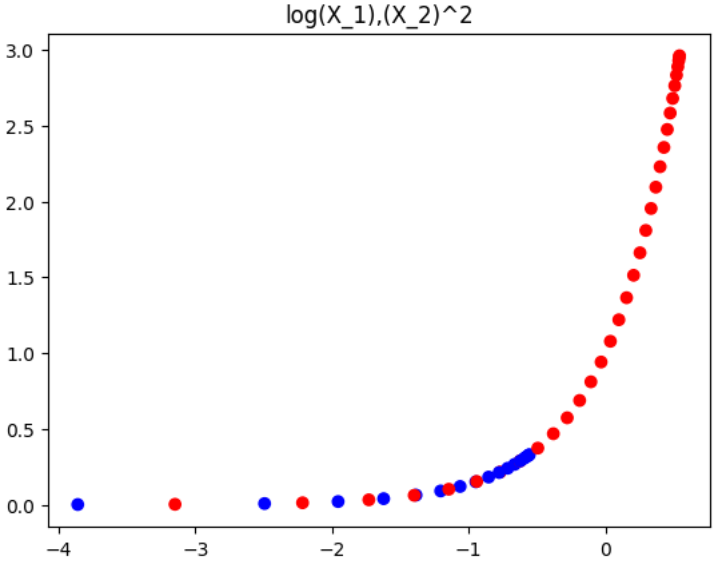
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons, make_blobs
from sklearn import preprocessing

X, y = make_moons(n_samples=100)
#X, y = make_blobs(n_samples=100)


X = preprocessing.StandardScaler().fit_transform(X)

plt.title('X')
plt.scatter(X[:,0],X[:,1], c=y, alpha=1, cmap=plt.cm.bwr)
plt.show()

X1 = np.log(X[:,0])
plt.title('log(X_1)')
plt.scatter(X1,X[:,1], c=y, alpha=1, cmap=plt.cm.bwr)
plt.show()

X1 = np.log(X[:,0])
X2 = np.square(X[:,0])
plt.title('log(X_1),(X_2)^2')
plt.scatter(X1,X2, c=y, alpha=1, cmap=plt.cm.bwr)
plt.show()

X1 = np.square(X[:,0])
plt.title('X_1^2')
plt.scatter(X1,X[:,1], c=y, alpha=1, cmap=plt.cm.bwr)
plt.show()

X2 = np.square(X[:,0])
plt.title('X^2')
plt.scatter(X1,X2, c=y, alpha=1, cmap=plt.cm.bwr)
plt.show()

X1_mm = np.exp(-((X[:,0] - X[:,0].mean()))**2)/2
X2_mm = np.exp(-((X[:,1] - X[:,1].mean()))**2)/2
X2 = np.square(X[:,0])
plt.scatter(X1_mm,X2_mm, c=y, alpha=1, cmap=plt.cm.bwr)
plt.show()
```

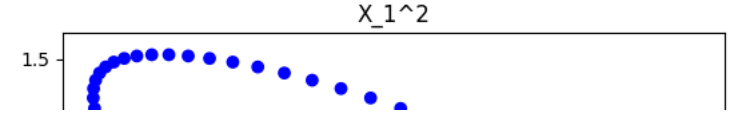


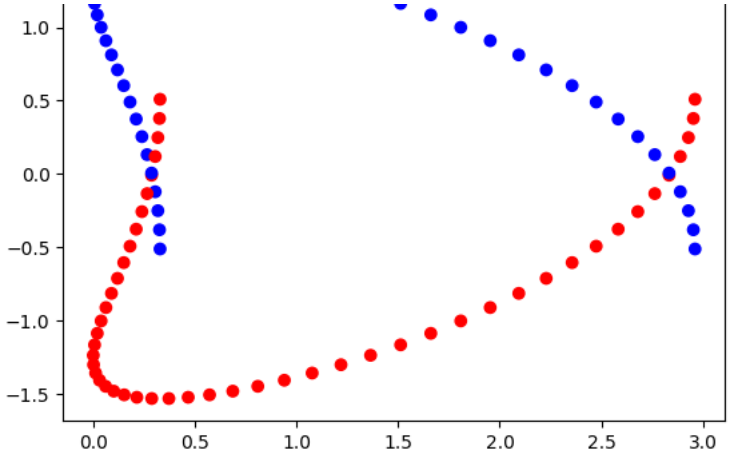

```
<ipython-input-8-ae4f10c7e1fa>:17: RuntimeWarning: invalid value encountered in log
  X1 = np.log(X[:,0])
```

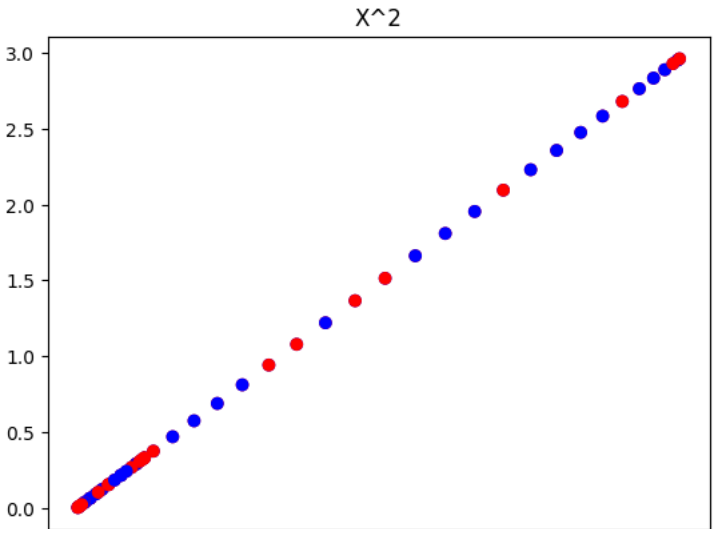



```
<ipython-input-8-ae4f10c7e1fa>:22: RuntimeWarning: invalid value encountered in log
  X1 = np.log(X[:,0])
```









Manifold Learning

Sources for this code are Rashcka Kernel tricks and nonlinear dimensionality reduction via RBF kernel PCA and Jake Vanderplas sklearn: Comparison of Manifold Learning methods

```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from sklearn import manifold, datasets
from sklearn import datasets
from sklearn.datasets.samples_generator import make_s_curve

n_points = 1000
X, color = datasets.make_s_curve(n_points, random_state=0)
n_neighbors = 10
n_components = 2

fig = plt.figure(figsize=(20, 15))
plt.title("Manifold Learning")

ax = fig.gca(projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)
ax.view_init(4, -72)
plt.show()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-12-2194d3a59b03> in <cell line: 7>()
      5 from sklearn import manifold, datasets
      6 from sklearn import datasets
----> 7 from sklearn.datasets.samples_generator import make_s_curve
      8
      9 n_points = 1000

ModuleNotFoundError: No module named 'sklearn.datasets.samples_generator'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
```

```
from scipy.spatial.distance import pdist, squareform
from scipy import exp
from scipy.linalg import eigh
import numpy as np

def stepwise_kpca(X, gamma, n_components):
    """
    Implementation of a RBF kernel PCA.

    Arguments:
        X: A MxN dataset as NumPy array where the samples are stored as rows (M),
           and the attributes defined as columns (N).
        gamma: A free parameter (coefficient) for the RBF kernel.
        n_components: The number of components to be returned.

    Returns the k eigenvectors (alphas) that correspond to the k largest
        eigenvalues (lambdas).

    """
    # Calculating the squared Euclidean distances for every pair of points
    # in the MxN dimensional dataset.
    sq_dists = pdist(X, 'sqeuclidean')

    # Converting the pairwise distances into a symmetric MxM matrix.
    mat_sq_dists = squareform(sq_dists)

    # Computing the MxM kernel matrix.
    K = exp(-gamma * mat_sq_dists)

    # Centering the symmetric NxN kernel matrix.
    N = K.shape[0]
    one_n = np.ones((N,N)) / N
    K_norm = K - one_n.dot(K) - K.dot(one_n) + one_n.dot(K).dot(one_n)

    # Obtaining eigenvalues in descending order with corresponding
    # eigenvectors from the symmetric matrix.
    eigvals, eigvecs = eigh(K_norm)

    # Obtaining the i eigenvectors (alphas) that corresponds to the i highest eigenvalues (lambdas).
    alphas = np.column_stack((eigvecs[:,-i] for i in range(1,n_components+1)))
    lambdas = [eigvals[-i] for i in range(1,n_components+1)]

    return alphas, lambdas


Y = stepwise_kpca(X, gamma=0.2, n_components=2)[0]
plt.title('Kernelized PCA, gamma = 0.2')
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral);
plt.show()


Y = stepwise_kpca(X, gamma=1, n_components=2)[0]
plt.title('Kernelized PCA, gamma = 1')
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral);
plt.show()


Y = stepwise_kpca(X, gamma=10, n_components=2)[0]
plt.title('Kernelized PCA, gamma = 10')
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral);
plt.show()


Y = manifold.LocallyLinearEmbedding(n_neighbors, n_components,
                                    eigen_solver='auto',
                                    method='standard').fit_transform(X)
```

```
plt.title('Locally Linear Embedding')
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral);


Y = manifold.MDS(n_components, max_iter=100, n_init=1).fit_transform(X)
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("Isomap");


Y = manifold.SpectralEmbedding(n_components=n_components,
                               n_neighbors=n_neighbors).fit_transform(X)

plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("Spectral Embedding/Laplacian Eigenmaps");


tsne = manifold.TSNE(n_components=n_components, init='pca', random_state=0)
Y = tsne.fit_transform(X)
plt.scatter(Y[:, 0], Y[:, 1], c=color, cmap=plt.cm.Spectral)
plt.title("t-SNE/t-Stochastic Neighbor Estimation");
```

Simple 1D Regression (using Least-Squares)

In this section we use the Advertising data from Hastie/Tibshirani book An Introduction to Statistical Learning.

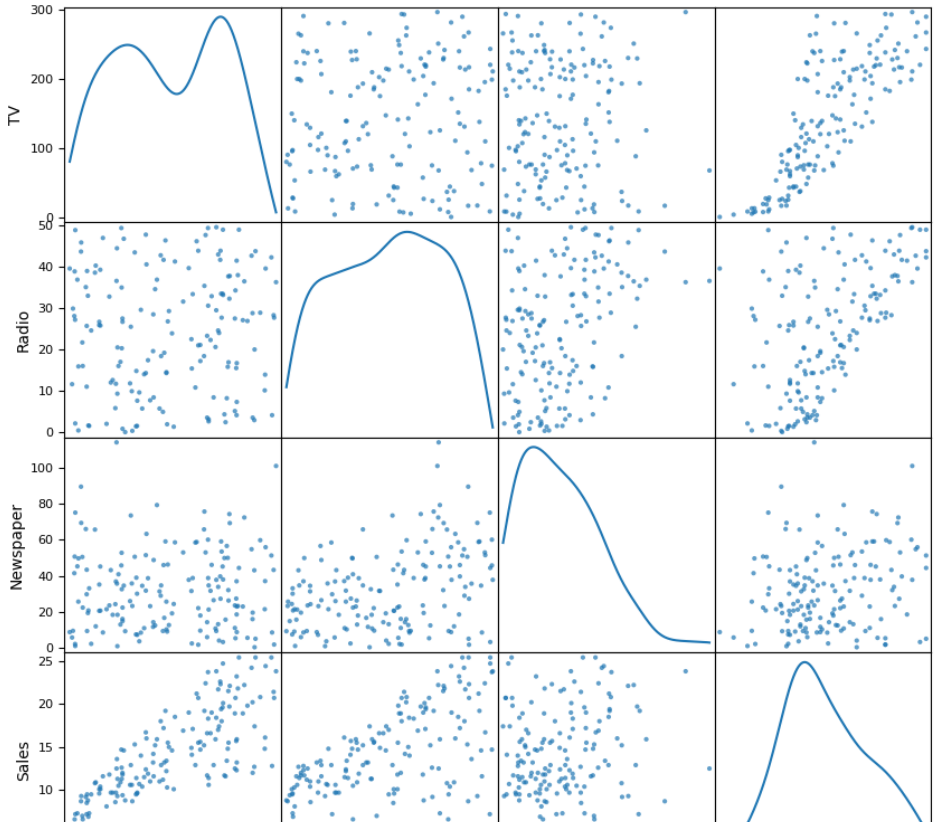
```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

np.random.seed(47)

# If you are NOT using google colab, you need to take this part out starting from here
from google.colab import files

uploaded=files.upload()
# till here

advertising = pd.read_csv('Advertising.csv',usecols=(1,2,3,4))

X = np.array(advertising['TV']).reshape(-1,1)
y = np.array(advertising['Sales'])

advertising.head()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Advertising.csv to Advertising (1).csv
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

```
from pandas.plotting import scatter_matrix
scatter_matrix(advertising.iloc[:160], alpha=0.7, figsize=(10, 10), diagonal='kde');
```



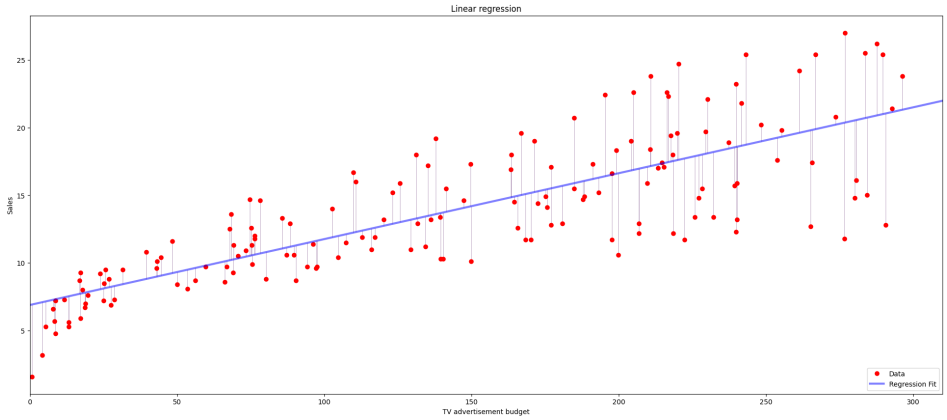

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=5)
```

TV Radio Newspaper Sales

```
from sklearn.linear_model import LinearRegression
%time regr = LinearRegression().fit(X_train,y_train)
```

```
CPU times: user 3.55 ms, sys: 408 µs, total: 3.95 ms
Wall time: 18 ms
```

```
from matplotlib.collections import LineCollection

y_predicted = regr.predict(X_train)
N = len(y_train)

segments = [[[X_train[i], y_train[i]], [X_train[i], y_predicted[i]]] for i in range(N)]
lc = LineCollection(segments, zorder=0)
lc.set_array(np.ones(len(y_train)))
lc.set_alpha(0.5)
lc.set_linewidths(0.5 * np.ones(len(y_train)))

fig = plt.figure(figsize=[24,10])
# plot the training data
plt.plot(X_train, y_train, 'r.', markersize=12)
# plot the prediction line
x_lin = np.linspace(0,320,1000).reshape(-1,1)
plt.plot(x_lin, regr.predict(x_lin), color='blue',linewidth=3,alpha=0.5)
# plot the redisuals
plt.gca().add_collection(lc)

plt.xlim([0,310])
plt.xlabel('TV advertisement budget')
plt.ylabel('Sales')
plt.legend(('Data', 'Regression Fit'), loc='lower right')
plt.title('Linear regression')

# plot the regression line
plt.show()
```




```
# The mean square error
print("Residual sum of squares:", (np.mean((regr.predict(X_test) - y_test) ** 2)))
# Explained variance score: 1 is perfect prediction
print("Variance score:", regr.score(X_test, y_test))
```

```
Residual sum of squares: 12.579629630036752
Variance score: 0.4990784791512455
```

2D polynomial regression using least-squares

```
from mpl_toolkits.mplot3d.art3d import Line3DCollection
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

X = np.array(advertising[['TV','Radio']])
y = np.array(advertising['Sales'])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=9)

# pipeline with polynomial regression
regr = Pipeline([('poly', PolynomialFeatures(degree=1)),
                 ('linear', LinearRegression(fit_intercept=True))])

%time regr = regr.fit(X_train,y_train)
y_predicted = regr.predict(X_train)
```

```
CPU times: user 4.04 ms, sys: 0 ns, total: 4.04 ms
Wall time: 12 ms
```

```
# create 3d line segments for residuals
segments_a = np.concatenate((X_train,y_train.reshape(-1,1)),axis=1)
segments_b = np.concatenate((X_train,y_predicted.reshape(-1,1)),axis=1)

segments = np.array([])
for i in range(len(segments_a)):
    segments = np.append(segments,np.array([segments_a[i],segments_b[i]]))
segments = np.vsplit(segments.reshape(segments_a.shape[0]*2,segments_a.shape[1])
                     ,len(segments_a))

# create a 3D meshgrid
X1, X2 = np.mgrid[X_train[:,0].min():X_train[:,0].max():1,
                  X_train[:,0].min():X_train[:,1].max():1]
pos = np.empty(X1.shape + (2,))
pos[:, :, 0] = X1; pos[:, :, 1] = X2

Z = regr.predict(np.c_[X1.ravel(), X2.ravel()])
```

```
Z = Z.reshape(X1.shape)

fig = plt.figure(figsize=[24,10])
ax = fig.gca(projection='3d')

# plot the regression plane
ax.plot_wireframe(X1, X2, Z, rstride=15, cstride=3, alpha=0.4, linewidth=1)
# plot the points
ax.scatter(X_train[:,0],X_train[:,1],y_train,c='r',s=100,alpha=1,zorder=10000)

# Create the 3D-line collection object
lc = Line3DCollection(segments)
lc.set_color('k')
lc.set_linewidth(3)
ax.add_collection3d(lc)

ax.set_xlabel('TV budget'); ax.set_xlim(X_train[:,0].min(), X_train[:,0].max())
ax.set_ylabel('Radio budget'); ax.set_ylim(X_train[:,1].min(), X_train[:,1].max())
ax.set_zlabel('Sales');  ax.set_zlim(y_train.min(), y_train.max())

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-9fe8b7890a37> in <cell line: 2>()
      1 # create 3d line segments for residuals
----> 2 segments_a = np.concatenate((X_train,y_train.reshape(-1,1)),axis=1)
      3 segments_b = np.concatenate((X_train,y_predicted.reshape(-1,1)),axis=1)
      4
      5 segments = np.array([])

/usr/local/lib/python3.10/dist-packages/numpy/core/overrides.py in concatenate(*args,
**kwargs)

ValueError: all the input array dimensions for the concatenation axis must match exactly,
but along dimension 0, the array at index 0 has size 14979 and the array at index 1 has
size 149790
```

SEARCH STACK OVERFLOW

```
# The mean square error
print("Residual sum of squares:",np.mean((regr.predict(X_test) - y_test) ** 2))
# Explained variance score: 1 is perfect prediction
print("Variance score:", regr.score(X_test, y_test))
```

```
Residual sum of squares: 1.8296065752684298
Variance score: 0.9258743223517101
```

Parametric Classification Revisited

[ ] ↳ *8 cells hidden*

Logistic Regression

[ ] ↳ *6 cells hidden*

Softmax Regression

[ ] ↳ *2 cells hidden*

Introduction to Keras

[ ] ↳ *6 cells hidden*

Programming Assignment 4: Logistic Regression on the notMIST dataset

For this assignment, we will perform logistic regression on the notMIST dataset using Keras. I have included the code to import the dataset and display the images. Since these images are black and white matrices, we flatten them to a vector so they can be used with logistic regression. I

also include the code to do the training and testing split.

Create a logistic regression model using Keras. In the first and only layer, set the only layer to:

```
model.add(Dense(output_dim=10, input_shape=[784], activation='sigmoid', W_regularizer=l2(0.01)))
```

Train the model for 100 iterations (`nb_epoch`). Change the loss function to `'categorical_crossentropy'` .

Display the loss after the final iteration and plot the training and testing loss, as above. Be sure to change the loss function label on the plot.

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy.io import loadmat

# If you are NOT using google colab, you need to take this part out starting from here
from google.colab import files

uploaded=files.upload()
# till here

data = loadmat('notMNIST_small.mat')
X_temp = data['images']/255

#for i in range(X_temp.shape[2]):

X = np.empty(shape=[X_temp.shape[2]] + [784], dtype='float32')
for i in range(X_temp.shape[2]):
    X[i,:] = X_temp[:,:,i].flatten()

y = pd.get_dummies(data['labels']).to_numpy()

print(X_temp.shape)
print(X.shape)
print(y.shape)
X[1,:]
```

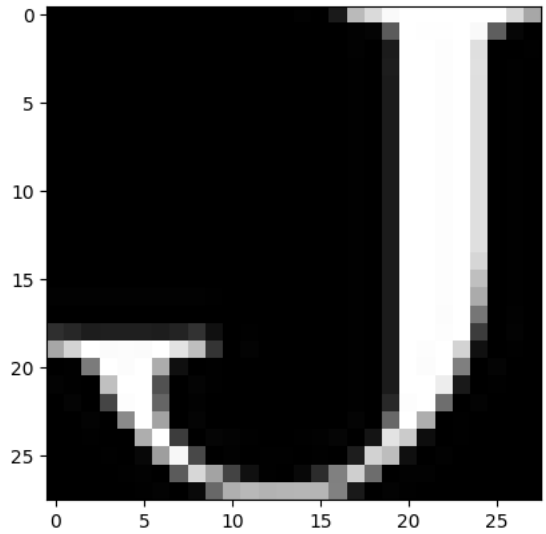
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving notMNIST_small.mat to notMNIST_small (1).mat
(28, 28, 18724)
(18724, 784)
(18724, 10)
array([0.        , 0.00392157, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.03137255, 0.16078432, 0.38039216, 0.6509804 , 0.87058824,
       0.9764706 , 0.90588236, 0.6627451 , 0.4       , 0.11764706,
       0.        , 0.00784314, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.07450981,
       0.3647059 , 0.44313726, 0.42745098, 0.4627451 , 0.5019608 ,
       0.54509807, 0.6392157 , 0.77254903, 0.8627451 , 0.9607843 ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.99607843, 1.        , 0.94509804, 0.42352942, 0.        ,
       0.00392157, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.01176471, 0.654902  , 1.        , 0.9843137 ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 0.99607843, 0.98039216, 0.972549  ,
       0.99215686, 1.        , 0.99607843, 0.9882353 , 0.99215686,
       0.99215686, 1.        , 0.21568628, 0.        , 0.00784314,
       0.        , 0.        , 0.        , 0.        , 0.05882353,
       0.7529412 , 1.        , 0.98039216, 0.9764706 , 0.96862745,
       0.96862745, 0.98039216, 0.99607843, 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.99607843, 1.        ,
       1.        , 1.        , 1.        , 0.9843137 , 1.        ,
       0.5372549 , 0.        , 0.00392157, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.18039216, 0.74509805,
       0.96862745, 1.        , 1.        , 1.        , 1.        ,
       0.98039216, 0.9372549 , 0.85882354, 0.76862746, 0.6666667 ,
       0.6627451 , 0.9372549 , 1.        , 0.99607843, 1.        ,
       1.        , 0.99607843, 1.        , 0.85882354, 0.00392157,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.1254902 , 0.2784314 ,
       0.3372549 , 0.3372549 , 0.24705882, 0.15294118, 0.07058824,
       0.01960784, 0.        , 0.01568628, 0.        , 0.3764706 ,
       1.        , 1.        , 0.99607843, 1.        , 1.        ,
       1.        , 0.9764706 , 0.24705882, 0.        , 0.00784314,
       0.        , 0.        , 0.        , 0.        , 0.00784314,
       0.00392157, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.00392157,
       0.01176471, 0.01568628, 0.        , 0.8117647 , 1.        ,
       0.99607843, 1.        , 1.        , 0.99215686, 1.        ,
       0.58431375, 0.        , 0.01568628, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.00392157, 0.00392157,
       0.01176471, 0.01568628, 0.01568628, 0.01176471, 0.00784314,
       0.00392157, 0.00392157, 0.        , 0.        , 0.00784314,
       0.        , 0.45882353, 1.        , 0.99607843, 1.        ,
       1.        , 0.9882353 , 1.        , 0.8       , 0.03137255,
       0.        , 0.00392157, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.00392157, 0.        , 0.13725491,
       0.9882353 , 1.        , 1.        , 1.        , 0.99607843,
       1.        , 0.9764706 , 0.2901961 , 0.        , 0.01568628,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00392157, 0.        , 0.01960784, 0.6901961 , 1.        ,
       0.9882353 , 1.        , 1.        , 0.98039216, 1.        ,
       0.52156866, 0.        , 0.01568628, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.01176471,
       0.        , 0.36862746, 1.        , 0.9843137 , 1.        ,
       1.        , 0.9882353 , 1.        , 0.8       , 0.07450981,
       0.        , 0.00392157, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.00784314, 0.        , 0.15294118,
       0.88235295, 1.        , 0.99215686, 1.        , 1.        ,
       0.99215686, 1.        , 0.21960784, 0.        , 0.00784314,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00392157, 0.01176471, 0.        , 0.6666667 , 1.        ,
       0.9843137 , 1.        , 1.        , 0.99215686, 1.        ,
       0.5176471 , 0.        , 0.00392157, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.01568628,
       0.        , 0.41568628, 1.        , 0.99607843, 0.99607843,
       1.        , 0.99607843, 1.        , 0.85490197, 0.01176471,
```

```
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.00392157, 0.01176471, 0.01960784, 0.        , 0.20392157,
        0.93333334, 1.        , 0.99607843, 1.        , 1.        ,
        1.        , 0.98039216, 0.23137255, 0.        , 0.00392157,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.00392157, 0.00784314,
        0.01568628, 0.01176471, 0.00392157, 0.        , 0.        ,
        0.        , 0.00784314, 0.        , 0.8352941 , 1.        ,
        0.99215686, 1.        , 1.        , 0.99215686, 1.        ,
        0.5803922 , 0.        , 0.01568628, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.00392157, 0.01176471,
        0.01176471, 0.        , 0.        , 0.        , 0.        ,
        0.00392157, 0.08235294, 0.2       , 0.31764707, 0.41960785,
        0.43529412, 0.8666667 , 1.        , 0.99607843, 1.        ,
        1.        , 0.9882353 , 1.        , 0.80784315, 0.01960784,
        0.        , 0.        , 0.        , 0.        , 0.00392157,
        0.01176471, 0.        , 0.        , 0.        , 0.01568628,
        0.16862746, 0.3764706 , 0.5803922 , 0.7764706 , 0.92941177,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 0.99607843, 1.        , 1.        , 0.99607843,
        1.        , 0.96862745, 0.28235295, 0.        , 0.        ,
        0.00392157, 0.01568628, 0.00392157, 0.        , 0.03529412,
        0.24705882, 0.5568628 , 0.8235294 , 0.98039216, 1.        ,
        1.        , 1.        , 1.        , 1.        , 0.99607843,
        1.        , 0.9411765 , 0.8235294 , 0.87058824, 1.        ,
        0.99607843, 1.        , 1.        , 0.98039216, 1.        ,
        0.52156866, 0.        , 0.00784314, 0.00784314, 0.        ,
        0.05882353, 0.48235294, 0.85882354, 0.99607843, 1.        ,
        1.        , 0.99607843, 0.9764706 , 0.98039216, 0.99607843,
        1.        , 0.95686275, 0.70980394, 0.38431373, 0.12156863,
        0.        , 0.07450981, 0.7921569 , 1.        , 0.9882353 ,
        1.        , 0.9882353 , 1.        , 0.76862746, 0.07058824,
        0.        , 0.03921569, 0.41960785, 0.88235295, 1.        ,
        0.99607843, 0.99607843, 0.9882353 , 0.9843137 , 0.9882353 ,
        1.        , 1.        , 0.84313726, 0.4862745 , 0.12941177,
        0.        , 0.        , 0.        , 0.01960784, 0.        ,
        0.4862745 , 1.        , 0.9882353 , 1.        , 0.99607843,
        1.        , 0.9647059 , 0.18039216, 0.2901961 , 0.8392157 ,
        1.        , 0.99607843, 0.9882353 , 0.99215686, 1.        ,
        1.        , 1.        , 1.        , 0.7921569 , 0.32156864,
        0.02745098, 0.        , 0.        , 0.00784314, 0.01568628,
        0.00784314, 0.01568628, 0.        , 0.30588236, 0.972549  ,
```

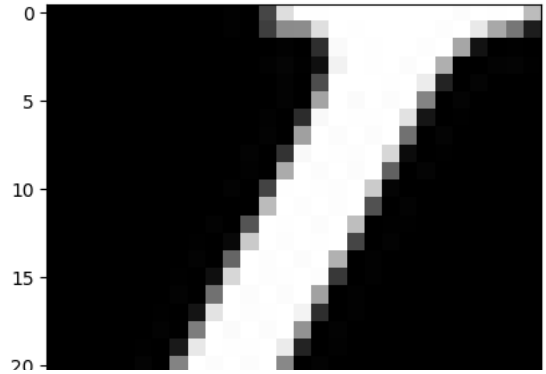
```
print(X_temp[:,:,3].shape)
plt.imshow(X_temp[:,:,3],cmap="gray");
```

```
(28, 28)
```




```
        0.3882353 . 0.        . 0.        . 0.01176471. 0.01176471.
```

```
plt.imshow(X_temp[:,:,4504],cmap="gray");
```




```
X_train, X_test, y_train, y_test =train_test_split(
    X, y, test_size=0.2, random_state=50)
print(X_train.shape)
print(y_train.shape)
```

```
(14979, 784)
(14979, 10)
```

```
print(y_test[7])
```

```
[0 1 0 0 0 0 0 0 0 0]
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2, l1
from tensorflow.keras.optimizers import SGD

model = Sequential()

model.add(Dense(units = 10,  input_shape=[784], activation='sigmoid', kernel_regularizer=l2(0.01)))

sgd = SGD(learning_rate=0.1)

model.compile(optimizer = sgd, loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 10)                7850

=================================================================
Total params: 7850 (30.66 KB)
Trainable params: 7850 (30.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(X_train, y_train, batch_size = 256, epochs = 100, verbose = 0, validation_data = (X_test, y_test))
```

```
%matplotlib inline
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(6,4))

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'], 'g--')

plt.title('Logistic Regression Model Loss')

plt.ylabel('categorical_crossentropy')

plt.xlabel('Epoch')

plt.legend(['Training Loss', 'Testing Loss'], loc='upper right')
```

```
plt.legend(['Training Loss', 'Testing Loss'], loc='upper right')

print("Loss after final iteration: ", history.history['val_loss'][-1])

print("Training Loss after final iteration: ", history.history['loss'][-1])

plt.show()
```

```
Loss after final iteration:  0.5614241361618042
Training Loss after final iteration:  0.5444937944412231
```