Problem-2

January 30, 2023

0.1 imports

[1]:
```
# Scientific and vector computation for python
import numpy as np

# Plotting library
import matplotlib.pyplot as plt

# Optimization module in scipy
from scipy import optimize

# Module to load MATLAB .mat datafile format (Input and output module of scipy)
# from scipy.io import loadmat
import pandas as pd

# figure size, dpi and font size
plt.rcParams['figure.figsize'] = [10, 5]
plt.rcParams['figure.dpi'] = 150
plt.rcParams['font.size'] = 14
```

- let's load the train and test data

[2]:
```
train_fashion = pd.read_csv('data/fashion-mnist_train.csv')
test_fashion = pd.read_csv('data/fashion-mnist_test.csv')

train_fashion.head()
```

[2]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	\
0	0	0	0	0	0	0	0	0	9	
1	1	0	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	14	53	
3	2	0	0	0	0	0	0	0	0	
4	3	0	0	0	0	0	0	0	0	

	pixel9	…	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	\
0	8	…	103	87	56	0	0	0	
1	0	…	34	0	0	0	0	0	
2	99	…	0	0	0	0	63	53	

```
3       0  …       137       126       140         0       133       224
4       0  …         0         0         0         0         0         0

   pixel781  pixel782  pixel783  pixel784
0         0         0         0         0
1         0         0         0         0
2        31         0         0         0
3       222        56         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

[3]:
```
# shape of the data
print('Shape of the training data: ', train_fashion.shape)
print('Shape of the test data: ', test_fashion.shape)
```

```
Shape of the training data:  (10000, 785)
Shape of the test data:  (30, 785)
```

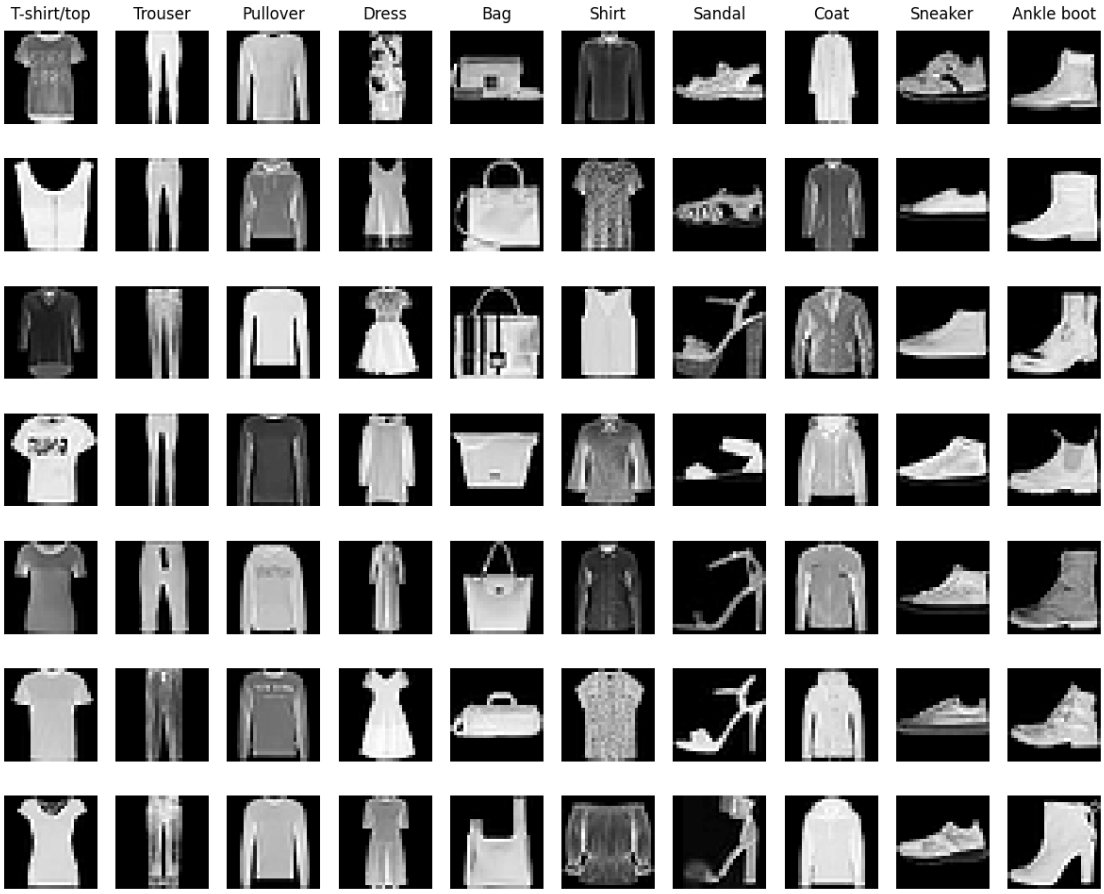
0.1.1 Visualize the data randomly

[4]:
```
title ={0: 'T-shirt/top', 1: 'Trouser', 2: 'Pullover', 3: 'Dress', 4: 'Coat', 5:
 ↪ 'Sandal',
        6: 'Shirt', 7: 'Sneaker', 8: 'Bag', 9: 'Ankle boot'}

def plot_grid(n, df):
    labels = df['label'].unique()
    fig, axes = plt.subplots(n, len(labels), figsize=(10, 8))
    for j, label in enumerate(labels):
        selected_rows = df[df['label'] == label]
        for i in range(n):
            # print(i, j)
            selected_row = selected_rows.sample(1)
            axes[0, j].set_title(title[selected_row['label'].values[0]],␣
 ↪fontsize=8)
            axes[i, j].imshow(selected_row.drop('label', axis=1).values.
 ↪reshape(28,28), cmap='gray')
            axes[i, j].axis('off')

    return fig
```

[5]:
```
fig = plot_grid(n=7, df=train_fashion)
fig.savefig('figures/0201.png')
```



0.1.2 Normalizing the data

Normalizing all the features by scaling them between 0 and 1

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

[6]:
```
def normalize(data):
    maximum = data.max()
    minimum = data.min()
    # avoid division by zero
    if maximum == minimum:
        return data
    return (data - minimum) / (maximum - minimum)
```

[7]:
```
for col in train_fashion.columns:
    if col != "label":
        train_fashion[col] = normalize(train_fashion[col])
        test_fashion[col] = normalize(test_fashion[col])
```

[8]:
```
train_fashion.head()
```

[8]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6   pixel7    pixel8  \
0      0     0.0     0.0     0.0     0.0     0.0     0.0  0.00000  0.041284
1      1     0.0     0.0     0.0     0.0     0.0     0.0  0.00000  0.000000
2      2     0.0     0.0     0.0     0.0     0.0     0.0  0.05668  0.243119
3      2     0.0     0.0     0.0     0.0     0.0     0.0  0.00000  0.000000
4      3     0.0     0.0     0.0     0.0     0.0     0.0  0.00000  0.000000

     pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0  0.032787  …  0.405512  0.345238  0.219608       0.0  0.000000  0.000000
1  0.000000  …  0.133858  0.000000  0.000000       0.0  0.000000  0.000000
2  0.405738  …  0.000000  0.000000  0.000000       0.0  0.247059  0.207843
3  0.000000  …  0.539370  0.500000  0.549020       0.0  0.521569  0.878431
4  0.000000  …  0.000000  0.000000  0.000000       0.0  0.000000  0.000000

   pixel781  pixel782  pixel783  pixel784
0  0.000000  0.000000       0.0       0.0
1  0.000000  0.000000       0.0       0.0
2  0.129167  0.000000       0.0       0.0
3  0.925000  0.248889       0.0       0.0
4  0.000000  0.000000       0.0       0.0

[5 rows x 785 columns]
```

[9]:
```
test_fashion.head()
```

[9]:
```
   label  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0      2       0       0       0     0.0     0.0     0.0     0.0     0.0
1      9       0       0       0     0.0     0.0     0.0     0.0     0.0
2      6       0       0       0     0.0     0.0     0.0     0.0     1.0
3      0       0       0       0     0.2     0.5     0.0     0.0     0.0
4      3       0       0       0     0.0     0.0     0.0     0.0     0.0

   pixel9  …  pixel775  pixel776  pixel777  pixel778  pixel779  pixel780  \
0     0.0  …  0.000000       0.0       0.0  0.000000  0.000000  0.000000
1     0.0  …  0.000000       0.0       0.0  0.000000  0.000000  0.000000
2     0.0  …  0.000000       0.0       0.0  0.157895  0.259036  0.000000
3     0.0  …  0.014778       0.0       0.0  0.000000  0.000000  0.006061
4     0.0  …  0.000000       0.0       0.0  0.000000  0.000000  0.000000

   pixel781  pixel782  pixel783  pixel784
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
```

```
[5 rows x 785 columns]
```

- Split the data into features and labels

[10]:
```
# split the data into X and y
X_train = train_fashion.drop('label', axis=1)
y_train = train_fashion['label']

X_test = test_fashion.drop('label', axis=1)
y_test = test_fashion['label']
```

0.2 Softmax

Softmax: In Softmax, we calculate the probability of each class. The probability of each class is given by:

$$P(y_i = j) = \frac{e^{z_{ij}}}{\sum_{k=1}^{K} e^{z_{ik}}}$$

where z_{ij} is the score of the j^{th} class for the i^{th} data point.

Herer we have used the stable version of softmax to avoid overflow. which is given by:

$$P(y_i = j) = \frac{e^{z_{ij}-c}}{\sum_{k=1}^{K} e^{z_{ik}-c}}$$

`where $c$ is the maximum value of $z_{ij}$.`

z_{ij} is given by:

$$z_{ij} = w_{ij}x_i + b_j$$

where w_{ij} is the weight of the j^{th} class for the i^{th} data point and b_j is the bias of the j^{th} class.

0.2.1 Cross Entropy Loss

Cross Entropy Loss: Cross entropy loss is used to calculate the loss between the predicted probability distribution and the true distribution. The loss is given by:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} log(p_{ij})$$

where y_{ij} is the true label of the j^{th} class for the i^{th} data point and p_{ij} is the predicted probability of the j^{th} class for the i^{th} data point.

0.2.2 Gradient of Cross Entropy Loss

The gradient of the cross entropy loss is given by:

$$\frac{\partial L}{\partial w_{ij}} = -\frac{1}{N}\sum_{i=1}^{N}(y_{ij} - p_{ij})x_i$$

- This gradient is used to update the weights of the model.

0.2.3 Gradient Descent

Gradient descent isthe optimization algorithm used to update the weights of the model. The weights are updated by:

$$w_{ij} = w_{ij} - \alpha\frac{\partial L}{\partial w_{ij}}$$

where α is the learning rate.

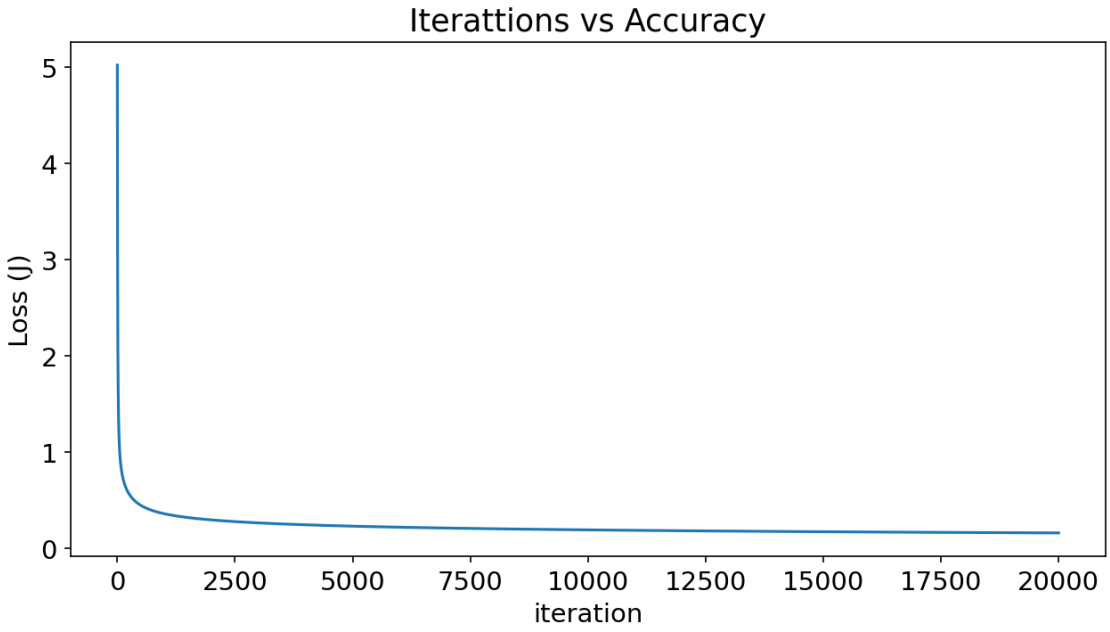
- Let's train a model by importing SoftmaxClassifier which I have implemented in the file Classifier.py

[11]:
```
from Classifier import SoftmaxClassifier
```

[16]:
```
softmax = SoftmaxClassifier(alpha=0.1, max_iter=20000, bias=True,
                            n_classes=10, tol=1e-5, penalty='l2', lambda_=0.05)

softmax.fit(X_train, y_train)
```

[17]:
```
plt.plot(softmax.loss_history);
plt.xlabel('iteration')
plt.ylabel("Loss (J)")
plt.title("Iterattions vs Accuracy")
plt.savefig("figures/0202.png")
```



- let's check the accuracy for training data

[18]:
```
acc_train = softmax.accuracy(softmax.predict(X_train), y_train)
print(f"Train accuracy: {acc_train:.4f}")
```

```
Train accuracy: 0.8917
```

- Accuracy for testing data

[19]:
```
acc_test = softmax.accuracy(softmax.predict(X_test), y_test)
print(f"Test accuracy: {acc_test:.4f}")
```

```
Test accuracy: 0.9000
```

- here we can see that training accuracy is less than test accuracy, hence we can say that our model is not overfitting.

0.3 OneVsAll implementation of the same

I have written a class for OneVsAll in the Classifier.py file. I have used Gradient Descent for the optimization purpose.

- Let's try that

[20]:
```
from Classifier import OneVsAll
```

[21]:
```
one_all = OneVsAll(alpha=0.1, max_iter=20000, bias=True, tol=1e-5,␣
↪penalty="l2", lambda_=0.05)

one_all.fit(X_train, y_train)
```

- let's check the training accyracy

[22]:
```
pred_train = one_all.predict(X_train)
train_acc = one_all.accuracy(pred_train, y_train)
print(f"Accuracy on Test data {train_acc:.2f}")
```

```
Accuracy on Test data 0.88
```

Test Accuracy

[23]:
```
pred_test = one_all.predict(X_test)
test_acc = one_all.accuracy(pred_test, y_test)
print(f"Accuracy on Test data {test_acc:.2f}")
```

```
Accuracy on Test data 0.87
```

- Here we can see that Softmax performed better than OneVsAll.
- The time taken by Softmax was also almost half of the time taken by OneVsAll.
- Hence Softmax is better than OneVsAll