

```

1 from google.colab import drive
2 drive.mount('/content/drive/')
3
4

```

Drive already mounted at /content/drive/; to attempt to forcibly remount, call d

```

1 import numpy as np
2 x = np.array([-1.5, -2, -1, 0.7, 2.3, -1.9])
3 W = np.array([[-3.5, 0, 3.1, 2.4, 1.8, 0.9], [1.7, -3.8, 0, 1.6, 2.3, -3.0], [2.8
4 b = np.array([1.1, -2.1, -3.0])
5 W_t = np.transpose(W)
6 print(W_t)
7 y = np.dot(x,W_t) + b
8 print(y)

```

```

[[-3.5  1.7  2.8]
 [ 0.  -3.8  3.1]
 [ 3.1  0.  -2.9]
 [ 2.4  1.6  0. ]
 [ 1.8  2.3 -2.1]
 [ 0.9 -3.  -2.5]]
[  7.36 15.06 -10.58]

```

```
1 %autosave 60
```

Autosaving every 60 seconds

```

1 # from google.colab import files
2 # src = list(files.upload().values())[0]
3
4 import os
5 os.chdir("/content/drive/My Drive/CS444_assignments/CS444/assignment1")
6 import sys
7 # sys.path.append('/content/drive/My Drive/CS444_assignments/CS444/assignment1/')
8 sys.path.append(".")
9

```

```

1 !ls
2 pwd = !pwd
3 print("Current working directory is: ",pwd)
4 !ls "models"

```

Assign1_sandbox.ipynb	ksa5_mp1_report.gdoc
assignment1.zip	ks-projects-201801-utf8.csv
cifar_net.pth	models
colab_setup.ipynb	mushroom
'CS 444 Assignment-1.ipynb'	mylib.py

```

data 'Numpy_logistic_reg_CS 444 Assignment-1.ipynb'
data_process.py __pycache__
fashion-mnist pytorch_tutorial.ipynb
kaggle sandbox
kaggle_submission.py 'Sandbox_Assign1_CS 444.ipynb'
Current working directory is: ['/content/drive/My Drive/CS444_assignments/CS444
__init__.py logistic.py perceptron.py __pycache__ softmax.py svm.py

```

```

1 import random
2 import numpy as np
3 import pandas as pd
4 # helpful character encoding module
5 import chardet
6 import math
7
8 from data_process import get_FASHION_data, get_MUSHROOM_data
9 from scipy.spatial import distance
10 # from models import Perceptron, SVM, Softmax, Logistic
11
12 from models.logistic import *
13 from models.perceptron import *
14 from models.softmax import *
15 from models.svm import *
16
17
18 from kaggle_submission import output_submission_csv
19 %matplotlib inline
20
21 # For auto-reloading external modules
22 # See http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipythor
23 %load_ext autoreload
24 %autoreload 2

```

The autoreload extension is already loaded. To reload it, use:
 %reload_ext autoreload

```
1 !pip install kaggle
```

```

Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/d
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dis

```

▼ Loading Fashion-MNIST

In the following cells we determine the number of images for each split and load the images.

TRAIN_IMAGES + VAL_IMAGES = (0, 60000], TEST_IMAGES = 10000

```
1 # You can change these numbers for experimentation
2 # For submission we will use the default values
3 TRAIN_IMAGES = 50000
4 VAL_IMAGES = 10000
5 normalize = True
```

```
1 !ls
```

```
Assign1_sandbox.ipynb      ksa5_mp1_report.gdoc
assignment1.zip            ks-projects-201801-utf8.csv
cifar_net.pth             models
colab_setup.ipynb         mushroom
'CS 444 Assignment-1.ipynb' mylib.py
data                      'Numpy_logistic_reg_CS 444 Assignment-1.ipynb'
data_process.py           __pycache__
fashion-mnist             pytorch_tutorial.ipynb
kaggle                   sandbox
kaggle_submission.py      'Sandbox_Assign1_CS 444.ipynb'
```

```
1 data = get_FASHION_data(TRAIN_IMAGES, VAL_IMAGES, normalize=normalize)
2 X_train_fashion, y_train_fashion = data['X_train'], data['y_train']
3 X_val_fashion, y_val_fashion = data['X_val'], data['y_val']
4 X_test_fashion, y_test_fashion = data['X_test'], data['y_test']
5 n_class_fashion = len(np.unique(y_test_fashion))
```

▼ Loading Mushroom

In the following cells we determine the splitting of the mushroom dataset.

TRAINING + VALIDATION = 0.8, TESTING = 0.2

```
1 # TRAINING = 0.6 indicates 60% of the data is used as the training dataset.
2 VALIDATION = 0.2
```

```
1 # TRAINING = 0.6 indicates 60% of the data is used as the training dataset.
2 VALIDATION = 0.2
3 data = get_MUSHROOM_data(VALIDATION)
4 X_train_MR, y_train_MR = data['X_train'], data['y_train']
```

```

5 X_val_MR, y_val_MR = data['X_val'], data['y_val']
6 X_test_MR, y_test_MR = data['X_test'], data['y_test']
7 n_class_MR = len(np.unique(y_test_MR))
8
9 print("Number of train samples: ", X_train_MR.shape[0])
10 print("Number of val samples: ", X_val_MR.shape[0])
11 print("Number of test samples: ", X_test_MR.shape[0])

```

```

Number of train samples: 4874
Number of val samples: 1625
Number of test samples: 1625

```

▼ Get Accuracy

This function computes how well your model performs using accuracy as a metric.

```

1 def get_acc(pred, y_test):
2     return np.sum(y_test == pred) / len(y_test) * 100

```

▼ Perceptron

Perceptron has 2 hyperparameters that you can experiment with:

- **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, but you should experiment with different values. We recommend changing the learning rate by factors of 10 and observing how the performance of the classifier changes. You should also try adding a **decay** which slowly reduces the learning rate over each epoch.
- **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according to the perceptron update rule for each sample in the training set. You should try different values for the number of training epochs and report your results.

You will implement the Perceptron classifier in the **models/perceptron.py**

The following code:

- Creates an instance of the Perceptron classifier class
- The train function of the Perceptron class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

▼ Train Perceptron on Fashion-MNIST

```

1 arr = np.array([1,2,3,7,12,768,2])
2 arr2 = np.arange(7)
3 #print(np.dot(np.transpose(arr),arr2))
4 weight = np.random.rand(2,2)
5 #print(arr.shape, weight.shape)
6 #print(weight)
7
8 #print(weight)

1 import numpy as np
2
3
4 class Perceptron:
5     def __init__(self, n_class: int, lr: float, epochs: int):
6         """Initialize a new classifier.
7
8         Parameters:
9             n_class: the number of classes
10            lr: the learning rate
11            epochs: the number of epochs to train for
12        """
13        self.w = None
14        self.lr = lr
15        self.epochs = epochs
16        self.n_class = n_class
17
18    def train(self, X_train: np.ndarray, y_train: np.ndarray):
19        """Train the classifier.
20
21        Use the perceptron update rule as introduced in the Lecture.
22
23        Parameters:
24            X_train: a number array of shape (N, D) containing training data;
25                    N examples with D dimensions
26            y_train: a numpy array of shape (N,) containing training labels
27        """
28        N, D = X_train.shape
29
30        #self.w = np.random.rand(self.n_class,D) # create a weight matrix of shape
31        self.w = np.zeros((self.n_class,D))
32        #print(self.w)
33        #print(self.w.shape)
34        #print(y_train[0:20])
35        for iter in range(self.epochs):
36            #if iter > 5:
37                # self.lr = 0.5

```

```

38     for example_num in range(N):
39         x = X_train[example_num]
40         y_label = y_train[example_num]
41         y_hat_list = np.dot(self.w, x) # get the dot product of weight and x
42         #print(y_label,y_hat_list)
43         y_hat_max = np.argmax(y_hat_list)
44
45         if y_label == y_hat_max:
46             pass
47         else: # update weight
48             y_yi = y_hat_list[y_label] # correct label w^T_yi*x_i
49             #y_c = np.argwhere(y_hat_list > y_yi).reshape(1,-1) # all labels l
50
51             coef_x = (self.lr)*x
52
53             for class_num in range(self.n_class):
54                 if iter == 0:
55                     #if class_num == y_label:
56                     self.w[y_label] = self.w[y_label] + coef_x
57                     #else:
58                     self.w[class_num] = self.w[class_num] - coef_x
59
60                 if y_hat_list[class_num] > y_yi:
61                     self.w[y_label] = self.w[y_label] + coef_x
62                     self.w[class_num] = self.w[class_num] - coef_x
63
64 def predict(self, X_test: np.ndarray) -> np.ndarray:
65     """Use the trained weights to predict labels for test data points.
66
67     Parameters:
68         X_test: a numpy array of shape (N, D) containing testing data;
69                 N examples with D dimensions
70
71     Returns:
72         predicted labels for the data in X_test; a 1-dimensional array of
73         length N, where each element is an integer giving the predicted
74         class.
75     """
76     N, D = X_test.shape
77     labels = np.zeros((N))
78     #print(self.w.shape)
79     for example_num in range(N):
80         x = X_test[example_num]
81         y_hat = np.dot(self.w,x)
82         labels[example_num] = np.argmax(y_hat)
83
84
85     return labels

```

```
1 lr = 0.55
```

```
2 n_epochs = 10
```

```

3
4 percept_fashion = Perceptron(n_class_fashion, lr, n_epochs)
5 percept_fashion.train(X_train_fashion, y_train_fashion)

1 pred_percept = percept_fashion.predict(X_train_fashion)
2 print('The training accuracy is given by: %f' % (get_acc(pred_percept, y_train_fa

The training accuracy is given by: 82.242000

```

▼ Validate Perceptron on Fashion-MNIST

```

1 pred_percept = percept_fashion.predict(X_val_fashion)
2 print('The validation accuracy is given by: %f' % (get_acc(pred_percept, y_val_fa

The validation accuracy is given by: 81.630000

```

▼ Test Perceptron on Fashion-MNIST

```

1 pred_percept = percept_fashion.predict(X_test_fashion)
2 print('The testing accuracy is given by: %f' % (get_acc(pred_percept, y_test_fas

The testing accuracy is given by: 80.790000

```

▼ Perceptron_Fashion-MNIST Kaggle Submission

Once you are satisfied with your solution and test accuracy, output a file to submit your test set predictions to the Kaggle for Assignment 1 Fashion-MNIST. Use the following code to do so:

```

1

1 #copy the kaggle.json token into kaggle folder
2 !mkdir -p ~/.kaggle
3 !cp kaggle/kaggle.json ~/.kaggle/

1 !pip install kaggle

Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-p

```

Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages
 Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/d
 Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-pac
 Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dis

```
1 !chmod 600 /root/.kaggle/kaggle.json
```

```
2 # !kaggle datasets list
```

```
1 #generate csv file for submission
```

```
2 output_submission_csv('kaggle/perceptron_submission_fashion.csv', percept_fashior  
3
```

```
4 import pandas as pd
```

```
5 intermediate_dataframe = pd.read_csv("kaggle/perceptron_submission_fashion.csv")
```

```
6 intermediate_dataframe.to_csv('kaggle/perceptron_submission_fashion_utf8_encoding  
7
```

```
8
```

```
9 # # from https://www.kaggle.com/alexisbcook/character-encodings
```

```
10 # # look at the first ten thousand bytes to guess the character encoding
```

```
11 with open("kaggle/perceptron_submission_fashion_utf8_encoding.csv", 'rb') as rawc
```

```
12     result = chardet.detect(rawdata.read(10000))
```

```
13
```

```
14 # check what the character encoding might be
```

```
15 print(result)
```

```
16
```

```
17
```

```
18
```

```
19 # # from https://www.kaggle.com/alexisbcook/character-encodings
```

```
20 # # look at the first ten thousand bytes to guess the character encoding
```

```
21 # with open("kaggle/perceptron_submission_fashion.csv", 'rb') as rawdata:
```

```
22 #     result = chardet.detect(rawdata.read(10000))
```

```
23 # # check what the character encoding might be
```

```
24 # print(result)
```

```
25
```

```
26 # #check top lines
```

```
27 # intermediate_dataframe.head()
```

```
28
```

```
29
```

```
30 # # intermediate_dataframe.to_csv("kaggle/perceptron_submission_fashion_utf8_enc
```

```
31
```

```
32
```

```
33
```

```
34
```

```
35
```

```
36
```

```
{'encoding': 'ascii', 'confidence': 1.0, 'language': ''}
```

```
1 # with open("kaggle/perceptron_submission_fashion.csv", 'rb') as source_file:
```



```

2 # with open("kaggle/perceptron_submission_fashion_utf8_encoding.csv", 'w+b') as
3 #     contents = source_file.read()
4 #     dest file write(contents.decode('utf-16-le').encode('utf-8'))

1 #measure the accuracy on the kaggle competition
2 # !kaggle competitions submit -c cs-444-assignment-1-perceptron -f kaggle/percept

```

▼ Train Perceptron on Mushroom

```

1 lr = 0.15
2 n_epochs = 10
3
4 percept_MR = Perceptron(n_class_MR, lr, n_epochs)
5 percept_MR.train(X_train_MR, y_train_MR)

1 pred_percept = percept_MR.predict(X_train_MR)
2 print('The training accuracy is given by: %f' % (get_acc(pred_percept, y_train_MR)

The training accuracy is given by: 94.521953

```

▼ Validate Perceptron on Mushroom

```

1 pred_percept = percept_MR.predict(X_val_MR)
2 print('The validation accuracy is given by: %f' % (get_acc(pred_percept, y_val_MR)

The validation accuracy is given by: 94.030769

```

▼ Test Perceptron on Mushroom

```

1 pred_percept = percept_MR.predict(X_test_MR)
2 print('The testing accuracy is given by: %f' % (get_acc(pred_percept, y_test_MR))

The testing accuracy is given by: 94.215385

```

▼ Support Vector Machines (with SGD)

Next, you will implement a "soft margin" SVM. In this formulation you will maximize the margin between positive and negative training examples and penalize margin violations using a hinge loss.

We will optimize the SVM loss using SGD. This means you must compute the loss function with respect to model weights. You will use this gradient to update the model weights.

SVM optimized with SGD has 3 hyperparameters that you can experiment with:

- **Learning rate** - similar to as defined above in Perceptron, this parameter scales by how much the weights are changed according to the calculated gradient update.
- **Epochs** - similar to as defined above in Perceptron.
- **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case it is a coefficient on the term which maximizes the margin. You could try different values. The default value is set to 0.05.

You will implement the SVM using SGD in the **models/svm.py**

The following code:

- Creates an instance of the SVM classifier class
- The train function of the SVM class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

▼ Train SVM on Fashion-MNIST

```

1 X = X_train_fashion
2 Y = y_train_fashion
3
4 N, D = X.shape
5 #print(shuffle_in_unison(X[0:100], Y[0:100]))
6 batch_size = 100
7 limit = N/batch_size
8 rand_num = np.random.randint(0,10)
9 slice_x = X[rand_num*batch_size:rand_num*batch_size+batch_size]
10 slice_y = Y[rand_num*batch_size:rand_num*batch_size+batch_size]
11 print(slice_x.shape, slice_y.shape)
12

```

```

(100, 784) (100,)

```

```

1 class SVM:
2     def __init__(self, n_class: int, lr: float, epochs: int, reg_const: float, bat
3         """Initialize a new classifier.
4
5         Parameters:
6             n_class: the number of classes
7             lr: the learning rate
8             epochs: the number of epochs to train for
9             reg_const: the regularization constant

```

```

10     """
11     self.w = None # TODO: change this
12     self.lr = lr
13     self.epochs = epochs
14     self.reg_const = reg_const
15     self.n_class = n_class
16     self.batch_size = batch_size
17     self.learning_rate_exponent = learning_rate_exponent
18
19     def calc_gradient(self, X_train: np.ndarray, y_train: np.ndarray) -> np.ndarray:
20         """Calculate gradient of the svm hinge loss.
21
22         Inputs have dimension D, there are C classes
23
24         Parameters:
25             X_train: a numpy array of shape (N, D) containing a mini-batch
26                       of data
27             y_train: a numpy array of shape (N,) containing training labels;
28                     y[i] = c means that X[i] has label c, where 0 <= c < C
29
30         Returns:
31             the gradient with respect to weights w; an array of the same shape
32             as w
33         """
34         x = X_train
35
36         y_hat_list = self.reg_const + np.dot(self.reg_const + self.w, x) # get t
37
38         return y_hat_list
39
40
41     def train(self, X_train: np.ndarray, y_train: np.ndarray):
42         """Train the classifier.
43
44         Hint: operate on mini-batches of data for SGD.
45
46         Parameters:
47             X_train: a numpy array of shape (N, D) containing training data;
48                     N examples with D dimensions
49             y_train: a numpy array of shape (N,) containing training labels
50         """
51         N, D = X_train.shape
52         batch_size = self.batch_size
53         #self.w = np.random.uniform(low=0.1, high=0.8, size=(N,D))
54         #self.w = np.zeros((N,D))
55         self.w = np.random.rand(self.n_class,D)
56         #print(self.w.shape)
57
58         for iter in range(self.epochs):
59             #if iter > 4:
60                 # self.lr -= iter*self.lr/9

```

```

61     self.lr *= (self.learning_rate_exponent ** iter)
62     # self.lr = self.lr * math.exp(-1*(self.learning_rate_exponent)*iter)
63     print("lr: ",self.lr)
64
65     #if self.lr > 6:
66     # self.reg_const /= 0.9
67     print("reg constant: ",self.reg_const)
68
69     for example_num in range(0,N,batch_size):
70         # print("example_num is: ",example_num)
71         x = X_train[example_num]
72         y_label = y_train[example_num]
73         #print(y_label)
74         #print(x.shape)
75         y_hat_list = self.calc_gradient(x,y_label)
76         y_correct = y_hat_list[y_label]
77         #print(y_correct)
78         #break
79
80         for class_num in range(self.n_class):
81             if y_correct != y_hat_list[class_num]:
82                 if y_correct - y_hat_list[class_num] < 1:
83                     self.w[y_label] = self.w[y_label] + self.lr*(x)
84                     self.w[class_num] = self.w[class_num] - self.lr*(x)
85
86                 self.w[class_num] = (1 - self.lr*(self.reg_const/self.n_class))*sel
87         print("weights are: ",self.w)
88         print("Epoch number finished: ",iter)
89     return
90
91 def predict(self, X_test: np.ndarray) -> np.ndarray:
92     """Use the trained weights to predict labels for test data points.
93
94     Parameters:
95         X_test: a numpy array of shape (N, D) containing testing data;
96                 N examples with D dimensions
97
98     Returns:
99         predicted labels for the data in X_test; a 1-dimensional array of
100         length N, where each element is an integer giving the predicted
101         class.
102     """
103     N, D = X_test.shape
104     labels = np.zeros(N)
105
106     for image_num in range(N):
107         x = X_test[image_num]
108         y_hat_list = np.dot(self.w, x)
109         labels[image_num] = np.argmax(y_hat_list)
110         if self.n_class == 2:
111             labels[image_num] = np.where(labels[image_num] == -1, 0, labels[image

```

112

```

1 lr = 0.005
2 n_epochs = 10
3 reg_const = 0.3
4 learning_rate_exponent = 0.2
5 batch_size = 1
6
7 svm_fashion = SVM(n_class_fashion, lr, n_epochs, reg_const, batch_size)
8 svm_fashion.train(X_train_fashion, y_train_fashion)

    1.61971904e-01 -3.35407491e-02]
    [-3.82721566e-03 -1.01911531e-03  6.14139403e-03 ...  1.50050996e+00
    5.41664896e-01  3.12687386e-01]
    ...
    [ 7.92817943e-05 -6.99862212e-03 -4.40244573e-02 ... -5.34792705e-01
    -1.26699856e-01 -7.13686019e-02]
    [-2.30833719e-03 -8.53695574e-03 -1.68489600e-01 ... -1.34582750e+00
    -1.27161145e+00 -2.65694086e-01]
    [ 7.86097126e-05 -1.32076961e-02 -4.94695638e-02 ... -1.27604630e-01
    2.36553648e-01  9.47255077e-02]]
Epoch number finished: 6
lr: 1.3421772800000025e-22
reg constant: 0.3
weights are: [[-2.03505170e-03  5.30068759e-02  1.81707331e-01 ... -1.2813268
-9.08759840e-01 -1.20393112e-01]
[ 7.63314513e-05 -6.22944984e-03 -1.05974062e-01 ... -2.83690732e-01
 1.61971904e-01 -3.35407491e-02]
[-3.82721566e-03 -1.01911531e-03  6.14139403e-03 ...  1.50050996e+00
 5.41664896e-01  3.12687386e-01]
...
[ 7.92817943e-05 -6.99862212e-03 -4.40244573e-02 ... -5.34792705e-01
-1.26699856e-01 -7.13686019e-02]
[-2.30833719e-03 -8.53695574e-03 -1.68489600e-01 ... -1.34582750e+00
-1.27161145e+00 -2.65694086e-01]
[ 7.86097126e-05 -1.32076961e-02 -4.94695638e-02 ... -1.27604630e-01
 2.36553648e-01  9.47255077e-02]]
Epoch number finished: 7
lr: 3.4359738368000008e-28
reg constant: 0.3
weights are: [[-2.03505170e-03  5.30068759e-02  1.81707331e-01 ... -1.2813268
-9.08759840e-01 -1.20393112e-01]
[ 7.63314513e-05 -6.22944984e-03 -1.05974062e-01 ... -2.83690732e-01
 1.61971904e-01 -3.35407491e-02]
[-3.82721566e-03 -1.01911531e-03  6.14139403e-03 ...  1.50050996e+00
 5.41664896e-01  3.12687386e-01]
...
[ 7.92817943e-05 -6.99862212e-03 -4.40244573e-02 ... -5.34792705e-01
-1.26699856e-01 -7.13686019e-02]
[-2.30833719e-03 -8.53695574e-03 -1.68489600e-01 ... -1.34582750e+00
-1.27161145e+00 -2.65694086e-01]
[ 7.86097126e-05 -1.32076961e-02 -4.94695638e-02 ... -1.27604630e-01
 2.36553648e-01  9.47255077e-02]]
Epoch number finished: 8
lr: 1.7592186044416049e-34
reg constant: 0.3

```

```
weights are: [[-2.03505170e-03  5.30068759e-02  1.81707331e-01 ... -1.2813268
-9.08759840e-01 -1.20393112e-01]
[ 7.63314513e-05 -6.22944984e-03 -1.05974062e-01 ... -2.83690732e-01
 1.61971904e-01 -3.35407491e-02]
[-3.82721566e-03 -1.01911531e-03  6.14139403e-03 ...  1.50050996e+00
 5.41664896e-01  3.12687386e-01]
...
[ 7.92817943e-05 -6.99862212e-03 -4.40244573e-02 ... -5.34792705e-01
-1.26699856e-01 -7.13686019e-02]
[-2.30833719e-03 -8.53695574e-03 -1.68489600e-01 ... -1.34582750e+00
-1.27161145e+00 -2.65694086e-01]
[ 7.86097126e-05 -1.32076961e-02 -4.94695638e-02 ... -1.27604630e-01
 2.00000000e-01  0.00000000e+00]
```

```
1 pred_svm = svm_fashion.predict(X_train_fashion)
2 print('The training accuracy is given by: %f' % (get_acc(pred_svm, y_train_fashion)))
```

The training accuracy is given by: 84.134000

▼ Validate SVM on Fashion-MNIST

```
1 pred_svm = svm_fashion.predict(X_val_fashion)
2 print('The validation accuracy is given by: %f' % (get_acc(pred_svm, y_val_fashion)))
```

The validation accuracy is given by: 82.730000

▼ Test SVM on Fashion-MNIST

```
1 pred_svm = svm_fashion.predict(X_test_fashion)
2 print('The testing accuracy is given by: %f' % (get_acc(pred_svm, y_test_fashion)))
```

The testing accuracy is given by: 81.460000

▼ SVM_Fashion-MNIST Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Fashion-MNIST. Use the following code to do so:

```
1 output_submission_csv('kaggle/svm_submission_fashion.csv', svm_fashion.predict(X_test_fashion))
```

▼ Train SVM on Mushroom

```

1 lr = 0.001
2 n_epochs = 10
3 reg_const = 0.6
4 batch_size = 1
5
6 svm_MR = SVM(n_class_MR, lr, n_epochs, reg_const, batch_size)
7 svm_MR.train(X_train_MR, y_train_MR)
-0.01065264  0.13460285  0.10964992  0.01682187  0.1643066  0.08857124
-0.05275008  0.047803  0.30035574  0.14252754]]
Epoch number finished: 4
lr: 3.2768000000000003e-14
reg constant: 0.6
weights are: [[ 0.07146993 -0.02145183  0.07778473  0.15622079  0.15996424  0
0.18087267 -0.09691478  0.20562084  0.22095555  0.2349655  0.20704153
0.17275992  0.14273808  0.12449457  0.06268333 -0.09482011  0.15787418
0.14939331  0.11528412 -0.13805703  0.04988181]
[ 0.11349462  0.08514871  0.11408066 -0.02530897  0.07635038  0.07212246
-0.03037723  0.32537601  0.09982113 -0.03034968 -0.08221091 -0.06225507
-0.01065264  0.13460285  0.10964992  0.01682187  0.1643066  0.08857124
-0.05275008  0.047803  0.30035574  0.14252754]]
Epoch number finished: 5
lr: 2.09715200000000025e-18
reg constant: 0.6
weights are: [[ 0.07146993 -0.02145183  0.07778473  0.15622079  0.15996424  0
0.18087267 -0.09691478  0.20562084  0.22095555  0.2349655  0.20704153
0.17275992  0.14273808  0.12449457  0.06268333 -0.09482011  0.15787418
0.14939331  0.11528412 -0.13805703  0.04988181]
[ 0.11349462  0.08514871  0.11408066 -0.02530897  0.07635038  0.07212246
-0.03037723  0.32537601  0.09982113 -0.03034968 -0.08221091 -0.06225507
-0.01065264  0.13460285  0.10964992  0.01682187  0.1643066  0.08857124
-0.05275008  0.047803  0.30035574  0.14252754]]
Epoch number finished: 6
lr: 2.68435456000000043e-23
reg constant: 0.6
weights are: [[ 0.07146993 -0.02145183  0.07778473  0.15622079  0.15996424  0
0.18087267 -0.09691478  0.20562084  0.22095555  0.2349655  0.20704153
0.17275992  0.14273808  0.12449457  0.06268333 -0.09482011  0.15787418
0.14939331  0.11528412 -0.13805703  0.04988181]
[ 0.11349462  0.08514871  0.11408066 -0.02530897  0.07635038  0.07212246
-0.03037723  0.32537601  0.09982113 -0.03034968 -0.08221091 -0.06225507
-0.01065264  0.13460285  0.10964992  0.01682187  0.1643066  0.08857124
-0.05275008  0.047803  0.30035574  0.14252754]]
Epoch number finished: 7
lr: 6.871947673600015e-29
reg constant: 0.6
weights are: [[ 0.07146993 -0.02145183  0.07778473  0.15622079  0.15996424  0
0.18087267 -0.09691478  0.20562084  0.22095555  0.2349655  0.20704153
0.17275992  0.14273808  0.12449457  0.06268333 -0.09482011  0.15787418
0.14939331  0.11528412 -0.13805703  0.04988181]
[ 0.11349462  0.08514871  0.11408066 -0.02530897  0.07635038  0.07212246
-0.03037723  0.32537601  0.09982113 -0.03034968 -0.08221091 -0.06225507
-0.01065264  0.13460285  0.10964992  0.01682187  0.1643066  0.08857124
-0.05275008  0.047803  0.30035574  0.14252754]]
Epoch number finished: 8
lr: 3.5184372088832095e-35
reg constant: 0.6

```

```

reg constant: 0.0
weights are: [[ 0.07146993 -0.02145183  0.07778473  0.15622079  0.15996424  0.
0.18087267 -0.09691478  0.20562084  0.22095555  0.2349655  0.20704153
0.17275992  0.14273808  0.12449457  0.06268333 -0.09482011  0.15787418
0.14939331  0.11528412 -0.13805703  0.04988181]
[ 0.11349462  0.08514871  0.11408066 -0.02530897  0.07635038  0.07212246
-0.03037723  0.32537601  0.09982113 -0.03034968 -0.08221091 -0.06225507
-0.01065264  0.13460285  0.10964992  0.01682187  0.1643066  0.08857124
-0.05275008  0.047803  0.30035574  0.14252754]]

```

```

1 pred_svm = svm_MR.predict(X_train_MR)
2 print('The training accuracy is given by: %f' % (get_acc(pred_svm, y_train_MR)))

```

The training accuracy is given by: 90.069758

▼ Validate SVM on Mushroom

```

1 pred_svm = svm_MR.predict(X_val_MR)
2 print('The validation accuracy is given by: %f' % (get_acc(pred_svm, y_val_MR)))

```

The validation accuracy is given by: 88.800000

▼ Test SVM on Mushroom

```

1 pred_svm = svm_MR.predict(X_test_MR)
2 print('The testing accuracy is given by: %f' % (get_acc(pred_svm, y_test_MR)))

```

The testing accuracy is given by: 88.800000

▼ Softmax Classifier (with SGD)

Next, you will train a Softmax classifier. This classifier consists of a linear function of the input data followed by a softmax function which outputs a vector of dimension C (number of classes) for each data point. Each entry of the softmax output vector corresponds to a confidence in one of the C classes, and like a probability distribution, the entries of the output vector sum to 1. We use a cross-entropy loss on this softmax output to train the model.

Check the following link as an additional resource on softmax classification:

<http://cs231n.github.io/linear-classify/#softmax>

Once again we will train the classifier with SGD. This means you need to compute the gradients of the softmax cross-entropy loss function according to the weights and update the weights using this

gradient. Check the following link to help with implementing the gradient updates:

<https://deeptnotes.io/softmax-crossentropy>

The softmax classifier has 3 hyperparameters that you can experiment with:

- **Learning rate** - As above, this controls how much the model weights are updated with respect to their gradient.
- **Number of Epochs** - As described for perceptron.
- **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case, we minimize the L2 norm of the model weights as regularization, so the regularization constant is a coefficient on the L2 norm in the combined cross-entropy and regularization objective.

You will implement a softmax classifier using SGD in the **models/softmax.py**

The following code:

- Creates an instance of the Softmax classifier class
- The train function of the Softmax class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

▼ Train Softmax on Fashion-MNIST

```

1 y = np.array([-2.85], [0.86], [0.28]))
2 exp_y = np.exp(y)
3 log_k = -np.max(exp_y)
4 exp_y_logk = exp_y + log_k
5 #print(log_k)
6 sum_exp_y = np.sum(exp_y_logk)
7 #print(exp_y_logk)
8 #print(exp_y_logk/sum_exp_y)

1 z = np.random.uniform(low=0.01, high=0.1, size=(10,2))
2 #print(z)
3 #print(np.linalg.norm(z))

1 """Softmax model."""
2
3 import numpy as np
4
5
```

```

6 class Softmax:
7     def __init__(self, n_class: int, lr: float, epochs: int, reg_const: float):
8         """Initialize a new classifier.
9
10        Parameters:
11            n_class: the number of classes
12            lr: the learning rate
13            epochs: the number of epochs to train for
14            reg_const: the regularization constant
15        """
16        self.w = None # TODO: change this
17        self.lr = lr
18        self.epochs = epochs
19        self.reg_const = reg_const
20        self.n_class = n_class
21
22    def calc_gradient(self, X_train: np.ndarray, y_train: np.ndarray) -> np.ndarr
23        """Calculate gradient of the softmax loss.
24
25        Inputs have dimension D, there are C classes, and we operate on
26        mini-batches of N examples.
27
28        Parameters:
29            X_train: a numpy array of shape (N, D) containing a mini-batch
30                    of data
31            y_train: a numpy array of shape (N,) containing training labels;
32                    y[i] = c means that X[i] has label c, where 0 <= c < C
33
34        Returns:
35            gradient with respect to weights w; an array of same shape as w
36        """
37        #N, D = X_train.shape
38        #print(N,D)
39        #gradients = np.zeros((N,D))
40        x = X_train
41
42        y_hat_list = np.dot(self.reg_const + self.w, x) # get the dot product of
43        #print(y_hat_list)
44        #exp_y = np.exp(y_hat_list)
45        #print(exp_y)
46        log_k = -np.max(y_hat_list)
47        exp_y = np.exp(y_hat_list + log_k)
48        sum_exp_y = np.sum(exp_y)
49        gradients = exp_y / sum_exp_y
50
51        return gradients
52
53    def train(self, X_train: np.ndarray, y_train: np.ndarray):
54        """Train the classifier.
55
56        Hint: operate on mini-batches of data for SGD.

```

```

57
58     Parameters:
59         X_train: a numpy array of shape (N, D) containing training data;
60             N examples with D dimensions
61         y_train: a numpy array of shape (N,) containing training labels
62     """
63     N, D = X_train.shape
64     #self.w = np.random.uniform(low=0.1, high=0.8, size=(N,D))
65     #self.w = np.zeros((N,D))
66     self.w = np.random.rand(self.n_class,D)
67     #print(self.w.shape)
68
69     for iter in range(self.epochs):
70         #if iter > 4:
71             self.lr -= iter*self.lr/5
72
73         #if self.lr > 6:
74             self.reg_const /= 0.9
75
76         for example_num in range(N):
77             x = X_train[example_num]
78             y_label = y_train[example_num]
79             #print(y_label)
80             #print(x.shape)
81             gradients = self.calc_gradient(x,y_label)
82             #print(gradients)
83             #break
84
85             for class_num in range(self.n_class):
86                 if class_num == y_label:
87                     self.w[y_label] = self.w[y_label] + (self.lr*(1 - gradients[y_label]))
88                 else:
89                     self.w[class_num] = self.w[class_num] - (self.lr*(gradients[class_num]))
90
91     return
92
93
94     def predict(self, X_test: np.ndarray) -> np.ndarray:
95         """Use the trained weights to predict labels for test data points.
96
97         Parameters:
98             X_test: a numpy array of shape (N, D) containing testing data;
99                 N examples with D dimensions
100
101         Returns:
102             predicted labels for the data in X_test; a 1-dimensional array of
103                 length N, where each element is an integer giving the predicted
104                 class.
105         """
106         N, D = X_test.shape
107         labels = np.zeros(N)

```

```

108
109     for image_num in range(N):
110         x = X_test[image_num]
111         y_hat_list = np.dot(self.w, x)
112         labels[image_num] = np.argmax(y_hat_list)
113         if self.n_class == 2:
114             labels[image_num] = np.where(labels[image_num] == -1, 0, labels[image
115

```

```

1 lr = 0.01
2 n_epochs = 14
3 reg_const = 0.55
4
5 softmax_fashion = Softmax(n_class_fashion, lr, n_epochs, reg_const)
6 softmax_fashion.train(X_train_fashion, y_train_fashion)

1 pred_softmax = softmax_fashion.predict(X_train_fashion)
2 print('The training accuracy is given by: %f' % (get_acc(pred_softmax, y_train_fa

```

The training accuracy is given by: 84.976000

▼ Validate Softmax on Fashion-MNIST

```

1 pred_softmax = softmax_fashion.predict(X_val_fashion)
2 print('The validation accuracy is given by: %f' % (get_acc(pred_softmax, y_val_fa

```

The validation accuracy is given by: 81.580000

▼ Testing Softmax on Fashion-MNIST

```

1 pred_softmax = softmax_fashion.predict(X_test_fashion)
2 print('The testing accuracy is given by: %f' % (get_acc(pred_softmax, y_test_fas

```

The testing accuracy is given by: 80.640000

▼ Softmax_Fashion-MNIST Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Fashion-MNIST. Use the following code to do so:

```

1 output_submission_csv('kaggle/softmax_submission_fashion.csv', softmax_fashion.pr

```

▼ Train Softmax on Mushroom

```

1 lr = 0.5
2 n_epochs = 10
3 reg_const = 0.05
4
5 softmax_MR = Softmax(n_class_MR, lr, n_epochs, reg_const)
6 #rint(n_class_MR)
7 softmax_MR.train(X_train_MR, y_train_MR)
8 print(y_train_MR.shape)

(4874,)

1 pred_softmax = softmax_MR.predict(X_train_MR)
2 print('The training accuracy is given by: %f' % (get_acc(pred_softmax, y_train_MR)

The training accuracy is given by: 95.219532

```

▼ Validate Softmax on Mushroom

```

1 pred_softmax = softmax_MR.predict(X_val_MR)
2 print('The validation accuracy is given by: %f' % (get_acc(pred_softmax, y_val_MR)

The validation accuracy is given by: 94.523077

```

▼ Testing Softmax on Mushroom

```

1 pred_softmax = softmax_MR.predict(X_test_MR)
2 print('The testing accuracy is given by: %f' % (get_acc(pred_softmax, y_test_MR))

The testing accuracy is given by: 95.323077

```

▼ Logistic Classifier

The Logistic Classifier has 2 hyperparameters that you can experiment with:

- **Learning rate** - similar to as defined above in Perceptron, this parameter scales by how much the weights are changed according to the calculated gradient update.
- **Number of Epochs** - As described for perceptron.
- **Threshold** - The decision boundary of the classifier.

You will implement the Logistic Classifier in the **models/logistic.py**

The following code:

- Creates an instance of the Logistic classifier class
- The train function of the Logistic class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

▼ Training Logistic Classifier

1

▼ Load mushroom dataset

```
1 # TRAINING = 0.6 indicates 60% of the data is used as the training dataset.
2 VALIDATION = 0.2
3 data = get_MUSHROOM_data(VALIDATION)
4 X_train_MR, y_train_MR = data['X_train'], data['y_train']
5 X_val_MR, y_val_MR = data['X_val'], data['y_val']
6 X_test_MR, y_test_MR = data['X_test'], data['y_test']
7 n_class_MR = len(np.unique(y_test_MR))
8
9 print("Number of train samples: ", X_train_MR.shape[0])
10 print("Number of val samples: ", X_val_MR.shape[0])
11 print("Number of test samples: ", X_test_MR.shape[0])
```

```
Number of train samples: 4874
Number of val samples: 1625
Number of test samples: 1625
```

1

```
1 # TAKE a look at the x_sub_i mushroom example values
2 #TOTAL examples: 4874
3 print("X_train_MR is: ", X_train_MR)
4
5 # there are 22 features = dimensions of each mushroom example
6 print("X_train_MR shape is: ", X_train_MR.shape)
7
8 #see the training label values
9 print("\ny_train_MR is: ", y_train_MR)
10
11 # see the shape of ytrain
12 print("y_train_MR shape is: ", y_train_MR.shape)
```

```

13
14 #####
15 #see the validation label values
16 print("\ny_val_MR is: ", y_val_MR)
17
18 # see the shape of y for validation
19 print("y_val_MR is: ", y_val_MR.shape)
20 #####
21
22
23 #see the testing label values
24 print("\ny_test_MR is: ", y_test_MR)
25
26 # see the shape of ytest
27 print("y_test_MR is: ", y_test_MR.shape)
28
29 # y_train_MR is:  [1 0 0 ... 1 1 0]
30 # y_train_MR is:  (4874,)
31 #we notice that y_train has 1,0 as labels. We need to replace the zero labels wit
32 #convert the zero in y to -1.
33 y_train_MR = np.array([-1 if value==0 else 1 for value in y_train_MR])
34 print("\n\n\nconverted_y_train is: ",y_train_MR)
35
36
37 #convert for y_val_MR as well
38 y_val_MR = np.array([-1 if value==0 else 1 for value in y_val_MR])
39 print("\nconverted_y_val_MR is: ",y_val_MR)
40
41 #convert for y_test_MR as well
42 y_test_MR = np.array([-1 if value==0 else 1 for value in y_test_MR])
43 print("\nconverted_y_test_MR is: ",y_test_MR)
44
45
46
47
48
X_train_MR is:  [[5 0 8 ... 3 4 0]
 [5 0 4 ... 2 3 1]
 [5 2 8 ... 3 3 3]
 ...
 [3 3 4 ... 7 4 4]
 [2 0 3 ... 1 5 4]
 [5 3 2 ... 7 1 6]]
X_train_MR shape is:  (4874, 22)

y_train_MR is:  [1 0 0 ... 1 1 0]
y_train_MR shape is:  (4874,)

y_val_MR is:  [1 0 0 ... 0 0 0]
y_val_MR is:  (1625,)

y_test_MR is:  [0 0 0 ... 0 0 0]

```

```
y_test_MR is: (1625,)
```

```
converted_y_train is: [ 1 -1 -1 ...  1  1 -1]
```

```
converted_y_val_MR is: [ 1 -1 -1 ... -1 -1 -1]
```

```
converted_y_test_MR is: [-1 -1 -1 ... -1 -1 -1]
```

```
1 def scalar_value_of_sigmoid(sigmoid_input):
2     """Sigmoid function.
3
4     Parameters:
5         z: the input
6
7     Returns:
8         the sigmoid of the input
9     """
10
11     sigmoid_value = 1/(1+math.exp(-1*sigmoid_input))
12     # print("sigmoid function returns: ",sigmoid_value)
13     return sigmoid_value
```

```
1
```

```
1 # find the gradient of loss at a point
2 def sgd_gradient_of_loss_for_a_point(weight_vec,y_sub_i,x_sub_i,learning_rate,sig
3
4
5     # print("y_sub_i is: ",y_sub_i)
6
7     sigmoid_input_for_gradient = -1*y_sub_i*(np.dot(x_sub_i,weight_vec))
8     # print("sigmoid input of gradient is: ",sigmoid_input_for_gradient)
9     # print("shape of sigmoid input of gradient is: ",sigmoid_input_for_gradient.sh
10
11     # print("x_sub_i is: ",x_sub_i)
12     # print("x_sub_i shape is: ",x_sub_i.shape)
13
14     # print("weight_vec is: ",weight_vec)
15     # print("weight_vec shape is: ",weight_vec.shape)
16
17     output_of_sigmoid_function = scalar_value_of_sigmoid(sigmoid_input_for_gradient
18     # print("output_of_sigmoid_function is: ",output_of_sigmoid_function)
19
20
21     gradient_of_loss_multiplied_by_eta = (x_sub_i)*learning_rate*(output_of_sigmoid
22     # print("gradient_of_loss_multiplied_by_eta is: ",gradient_of_loss_multiplied_b
23     # print("shape of gradient_of_loss_multiplied_by_eta is: ",gradient_of_loss_mul
24     gradient_of_loss_multiplied_by_eta = gradient_of_loss_multiplied_by_eta.reshape
```



```
25
26 return gradient_of_loss_multiplied_by_eta
27

1 def get_acc(pred, y_test):
2     return np.sum(y_test == pred) / len(y_test) * 100
3

1 """Logistic regression model."""
2
3 import numpy as np
4
5
6 class Logistic:
7     def __init__(self, lr: float, epochs: int, threshold: float):
8         """Initialize a new classifier.
9
10         Parameters:
11             lr: the learning rate
12             epochs: the number of epochs to train for
13         """
14         self.weight_vec = None # TODO: change this
15         self.lr = lr
16         self.epoch_number = epochs
17         self.threshold = threshold
18         self.logistic_loss = []
19
20     # def sigmoid(self, z: np.ndarray) -> np.ndarray:
21     #     """Sigmoid function.
22
23     #     Parameters:
24     #         z: the input
25
26     #     Returns:
27     #         the sigmoid of the input
28     #     """
29     #     exp_z = np.exp(-z)
30     #     # print("exp_z is: ", exp_z)
31     #     # ones_array = np.ones(len(z))
32     #     sum = 1+exp_z
33     #     print("sum is: ", sum)
34     #     sigmoid_value = 1/(1+exp_z)
35     #     print("sigmoid function returns: ", sigmoid_value)
36     #     return sigmoid_value
37
38     def train(self, X_train: np.ndarray, y_train: np.ndarray):
39         """Train the classifier.
40
41         Use the *logistic regression update rule* as introduced in lecture.
42
```

```

43     Parameters:
44         X_train: a numpy array of shape (N, D) containing training data;
45             N examples with D dimensions
46         y_train: a numpy array of shape (N,) containing training labels
47     """
48     #in class notes, x_rows=n and x_cols=d
49     x_rows,x_cols = X_train.shape
50     self.weight_vec = np.zeros((x_cols,1))
51     # print("self.weight_vec is: ",self.weight_vec)
52     # print("self.weight_vec.shape is: ",self.weight_vec.shape)
53
54     #reshape y_train to a column vector that is n by 1,
55     y_train = y_train.reshape(x_rows,1)
56
57
58     #loop for each epoch
59     for epoch_number in range(self.epoch_number):
60
61         #we need to iterate over the weight matrix and take each row as input
62         for x_row_index in range(x_rows):
63             x_row_for_example = X_train[x_row_index]
64             # print("x_row_for_example:",x_row_for_example)
65             y_label = y_train[x_row_index]
66             # print("y_label:",y_label)
67
68             sigmoid_input = y_label*np.dot(x_row_for_example,self.weight_vec)
69             sigmoid_output = scalar_value_of_sigmoid(sigmoid_input)
70             delta_weight_vector = sgd_gradient_of_loss_for_a_point(self.weight_vec, x_row_for_example, y_label)
71             # print("delta_weight_vector is: ",delta_weight_vector)
72
73             # Updating the weight vector.
74             # print("weight vector before update is: ",self.weight_vec)
75             self.weight_vec = self.weight_vec + delta_weight_vector
76             # print("weight vector after update is: ",self.weight_vec)
77
78
79     return self.weight_vec
80
81
82
83
84     def predict(self, X_test: np.ndarray) -> np.ndarray:
85         """Use the trained weights to predict labels for test data points.
86
87         Parameters:
88             X_test: a numpy array of shape (N, D) containing testing data;
89                 N examples with D dimensions
90
91         Returns:
92             predicted labels for the data in X_test; a 1-dimensional array of
93                 length N, where each element is an integer giving the predicted

```

```

94         class.
95         """
96
97         N, D = X_test.shape
98         labels = np.zeros((N))
99         #print(self.w.shape)
100        for example_num in range(N):
101            x = X_test[example_num]
102            y_hat = np.dot(x,self.weight_vec)
103            if y_hat>=self.threshold:
104                labels[example_num] = 1
105            else:
106                labels[example_num] = -1
107
108
109        return labels
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

array([[ -2.01724437],
       [ 22.23359795],
       [  1.65088044],
       [-12.09568461],
       [-14.47913435],
       [ 42.75131934],
       [-178.24504802],
       [ 262.5225279 ],
       [  -5.72225053],
       [-22.68513477],
       [-68.93669088],
       [-147.59746428],
       [-24.0227172 ],
       [-2.93313362],
       [-3.43449207],
       [  0.          ],
       [ 191.40812511],
       [  1.70174918],
       [ 22.76219888],
       [-22.57656537],
       [-11.98547559],
       [  3.82978282]])

```

1

```

1  pred_lr = lr.predict(X_train_MR)
2  print('The training accuracy is given by: %f' % (get_acc(pred_lr, y_train_MR)))
3

```

```
1  
2  
3  
4 print("True y labels for training set of mushroom dataset are:",y_train_MR)  
5 print("Predicted y labels for training set of mushroom dataset are:",pred_lr)  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
2213  
2214  
2215  
2216  
2217  
2218  
2219  
2220  
2221  
2222  
2223  
2224  

```

