```
from google.colab import drive
drive.mount('/content/drive/')


```

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

```
import numpy as np
x = np.array([-1.5, -2, -1, 0.7, 2.3, -1.9])
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
b = np.array([1.1, -2.1, -3.0])
W_t = np.transpose(W)
print(W_t)
y = np.dot(x,W_t) + b
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]
```

```
%autosave 60
```

```
Autosaving every 60 seconds
```

```
# from google.colab import files
# src = list(files.upload().values())[0]

import os
os.chdir("/content/drive/My Drive/CS444_assignments/CS444/assignment1")
import sys
# sys.path.append('/content/drive/My Drive/CS444_assignments/CS444/assignment1/')
sys.path.append(".")

```

```
!ls
pwd = !pwd
print("Current working directory is: ",pwd)
!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
```

```
import random
import numpy as np
import pandas as pd
# helpful character encoding module
import chardet
import math

from data_process import get_FASHION_data, get_MUSHROOM_data
from scipy.spatial import distance
# from models import Perceptron, SVM, Softmax, Logistic

from models.logistic import *
from models.perceptron import *
from models.softmax import *
from models.svm import *


from kaggle_submission import output_submission_csv
%matplotlib inline

# For auto-reloading external modules
# See http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipythor
%load_ext autoreload
%autoreload 2
```

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

```
!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

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

```
!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'
```

```
data = get_FASHION_data(TRAIN_IMAGES, VAL_IMAGES, normalize=normalize)
X_train_fashion, y_train_fashion = data['X_train'], data['y_train']
X_val_fashion, y_val_fashion = data['X_val'], data['y_val']
X_test_fashion, y_test_fashion = data['X_test'], data['y_test']
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

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

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

```
X_val_MR, y_val_MR = data['X_val'], data['y_val']
X_test_MR, y_test_MR = data['X_test'], data['y_test']
n_class_MR = len(np.unique(y_test_MR))

print("Number of train samples: ", X_train_MR.shape[0])
print("Number of val samples: ", X_val_MR.shape[0])
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.

```
def get_acc(pred, y_test):
    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

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

#print(weight)
```

```
import numpy as np


class Perceptron:
    def __init__(self, n_class: int, lr: float, epochs: int):
        """Initialize a new classifier.

        Parameters:
            n_class: the number of classes
            lr: the learning rate
            epochs: the number of epochs to train for
        """
        self.w = None
        self.lr = lr
        self.epochs = epochs
        self.n_class = n_class

    def train(self, X_train: np.ndarray, y_train: np.ndarray):
        """Train the classifier.

        Use the perceptron update rule as introduced in the Lecture.

        Parameters:
            X_train: a number array of shape (N, D) containing training data;
                N examples with D dimensions
            y_train: a numpy array of shape (N,) containing training labels
        """
        N, D = X_train.shape

        #self.w = np.random.rand(self.n_class,D)  # create a weight matrix of sha
        self.w = np.zeros((self.n_class,D))
        #print(self.w)
        #print(self.w.shape)
        #print(y_train[0:20])
        for iter in range(self.epochs):
          #if iter > 5:
          #  self.lr = 0.5
```

```
          for example_num in range(N):
            x = X_train[example_num]
            y_label = y_train[example_num]
            y_hat_list = np.dot(self.w, x)  # get the dot product of weight and
            #print(y_label,y_hat_list)
            y_hat_max = np.argmax(y_hat_list)

            if y_label == y_hat_max:
              pass
            else:     # update weight
              y_yi = y_hat_list[y_label]      # correct label w^T_yi*xi
              #y_c = np.argwhere(y_hat_list > y_yi).reshape(1,-1)  # all labels

              coef_x = (self.lr)*x

              for class_num in range(self.n_class):
                if iter == 0:
                  #if class_num == y_label:
                  self.w[y_label] = self.w[y_label] + coef_x
                  #else:
                  self.w[class_num] = self.w[class_num] - coef_x

                if y_hat_list[class_num] > y_yi:
                  self.w[y_label] = self.w[y_label] + coef_x
                  self.w[class_num] = self.w[class_num] - coef_x

    def predict(self, X_test: np.ndarray) -> np.ndarray:
        """Use the trained weights to predict labels for test data points.

        Parameters:
            X_test: a numpy array of shape (N, D) containing testing data;
                N examples with D dimensions

        Returns:
            predicted labels for the data in X_test; a 1-dimensional array of
                length N, where each element is an integer giving the predicted
                class.
        """
        N, D = X_test.shape
        labels = np.zeros((N))
        #print(self.w.shape)
        for example_num in range(N):
          x = X_test[example_num]
          y_hat = np.dot(self.w,x)
          labels[example_num] = np.argmax(y_hat)


        return labels
```

```
lr = 0.55
n_epochs = 10
```

```

percept_fashion = Perceptron(n_class_fashion, lr, n_epochs)
percept_fashion.train(X_train_fashion, y_train_fashion)
```

```
pred_percept = percept_fashion.predict(X_train_fashion)
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

```
pred_percept = percept_fashion.predict(X_val_fashion)
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

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

```
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:

```

```

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

```
!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-packag
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-
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-package
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
```

```
!chmod 600 /root/.kaggle/kaggle.json
# !kaggle datasets list
```

```
#generate csv file for submission
output_submission_csv('kaggle/perceptron_submission_fashion.csv', percept_fashion

import pandas as pd
intermediate_dataframe = pd.read_csv("kaggle/perceptron_submission_fashion.csv")
intermediate_dataframe.to_csv('kaggle/perceptron_submission_fashion_utf8_encoding


# # from https://www.kaggle.com/alexisbcook/character-encodings
# # look at the first ten thousand bytes to guess the character encoding
with open("kaggle/perceptron_submission_fashion_utf8_encoding.csv", 'rb') as rawd
    result = chardet.detect(rawdata.read(10000))

# check what the character encoding might be
print(result)



# # from https://www.kaggle.com/alexisbcook/character-encodings
# # look at the first ten thousand bytes to guess the character encoding
# with open("kaggle/perceptron_submission_fashion.csv", 'rb') as rawdata:
#     result = chardet.detect(rawdata.read(10000))
# # check what the character encoding might be
# print(result)

# #check top lines
# intermediate_dataframe.head()


# # intermediate_dataframe.to_csv("kaggle/perceptron_submission_fashion_utf8_enco
```

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

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

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

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

Train Perceptron on Mushroom

```
lr = 0.15
n_epochs = 10

percept_MR = Perceptron(n_class_MR, lr, n_epochs)
percept_MR.train(X_train_MR, y_train_MR)
```

```
pred_percept = percept_MR.predict(X_train_MR)
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

```
pred_percept = percept_MR.predict(X_val_MR)
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

```
pred_percept = percept_MR.predict(X_test_MR)
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

```
X = X_train_fashion
Y = y_train_fashion

N, D = X.shape
#print(shuffle_in_unison(X[0:100], Y[0:100]))
batch_size = 100
limit = N/batch_size
rand_num = np.random.randint(0,10)
slice_x = X[rand_num*batch_size:rand_num*batch_size+batch_size]
slice_y = Y[rand_num*batch_size:rand_num*batch_size+batch_size]
print(slice_x.shape, slice_y.shape)

```

```
(100, 784) (100,)
```

```
class SVM:
    def __init__(self, n_class: int, lr: float, epochs: int, reg_const: float,bat
        """Initialize a new classifier.

        Parameters:
            n_class: the number of classes
            lr: the learning rate
            epochs: the number of epochs to train for
            reg_const: the regularization constant
```

```
        """
        self.w = None  # TODO: change this
        self.lr = lr
        self.epochs = epochs
        self.reg_const = reg_const
        self.n_class = n_class
        self.batch_size = batch_size
        self.learning_rate_exponent = learning_rate_exponent

    def calc_gradient(self, X_train: np.ndarray, y_train: np.ndarray) -> np.ndarr
        """Calculate gradient of the svm hinge loss.

        Inputs have dimension D, there are C classes

        Parameters:
            X_train: a numpy array of shape (N, D) containing a mini-batch
                of data
            y_train: a numpy array of shape (N,) containing training labels;
                y[i] = c means that X[i] has label c, where 0 <= c < C

        Returns:
            the gradient with respect to weights w; an array of the same shape
                as w
        """
        x = X_train

        y_hat_list = self.reg_const + np.dot(self.reg_const + self.w, x)  # get t

        return y_hat_list


    def train(self, X_train: np.ndarray, y_train: np.ndarray):
        """Train the classifier.

        Hint: operate on mini-batches of data for SGD.

        Parameters:
            X_train: a numpy array of shape (N, D) containing training data;
                N examples with D dimensions
            y_train: a numpy array of shape (N,) containing training labels
        """
        N, D = X_train.shape
        batch_size = self.batch_size
        #self.w = np.random.uniform(low=0.1, high=0.8,size=(N,D))
        #self.w = np.zeros((N,D))
        self.w = np.random.rand(self.n_class,D)
        #print(self.w.shape)

        for iter in range(self.epochs):
          #if iter > 4:
          # self.lr -= iter*self.lr/9
```

```
        self.lr *= (self.learning_rate_exponent ** iter)
        # self.lr = self.lr * math.exp(-1*(self.learning_rate_exponent)*iter)
        print("lr: ",self.lr)

        #if self.lr > 6:
        # self.reg_const /= 0.9
        print("reg constant: ",self.reg_const)

        for example_num in range(0,N,batch_size):
          # print("example_num is: ",example_num)
          x = X_train[example_num]
          y_label = y_train[example_num]
          #print(y_label)
          #print(x.shape)
          y_hat_list = self.calc_gradient(x,y_label)
          y_correct = y_hat_list[y_label]
          #print(y_correct)
          #break

          for class_num in range(self.n_class):
            if y_correct != y_hat_list[class_num]:
              if y_correct - y_hat_list[class_num] < 1:
                self.w[y_label] = self.w[y_label] + self.lr*(x)
                self.w[class_num] = self.w[class_num] - self.lr*(x)

            self.w[class_num] = (1 - self.lr*(self.reg_const/self.n_class))*sel
        print("weights are: ",self.w)
        print("Epoch number finished: ",iter)
      return

  def predict(self, X_test: np.ndarray) -> np.ndarray:
      """Use the trained weights to predict labels for test data points.

      Parameters:
          X_test: a numpy array of shape (N, D) containing testing data;
              N examples with D dimensions

      Returns:
          predicted labels for the data in X_test; a 1-dimensional array of
              length N, where each element is an integer giving the predicted
              class.
      """
      N, D = X_test.shape
      labels = np.zeros(N)

      for image_num in range(N):
        x = X_test[image_num]
        y_hat_list = np.dot(self.w, x)
        labels[image_num] = np.argmax(y_hat_list)
        if self.n_class == 2:
          labels[image_num] = np.where(labels[image_num] == -1, 0, labels[image
```

112

```
lr = 0.005
n_epochs = 10
reg_const = 0.3
learning_rate_exponent = 0.2
batch_size = 1

svm_fashion = SVM(n_class_fashion, lr, n_epochs, reg_const,batch_size)
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:  3.435973836800008e-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
```

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

```
The training accuracy is given by: 84.134000
```

Validate SVM on Fashion-MNIST

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

```
The validation accuracy is given by: 82.730000
```

Test SVM on Fashion-MNIST

```
pred_svm = svm_fashion.predict(X_test_fashion)
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:

```
output_submission_csv('kaggle/svm_submission_fashion.csv', svm_fashion.predict(X_
```

Train SVM on Mushroom

```
lr = 0.001
n_epochs = 10
reg_const = 0.6
batch_size = 1

svm_MR = SVM(n_class_MR, lr, n_epochs, reg_const,batch_size)
svm_MR.train(X_train_MR, y_train_MR)
```

```
  -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:  4
lr:  3.276800000000003e-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:  2.0971520000000025e-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:  2.6843545600000043e-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:  0.6
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]]
```

```
pred_svm = svm_MR.predict(X_train_MR)
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

```
pred_svm = svm_MR.predict(X_val_MR)
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

```
pred_svm = svm_MR.predict(X_test_MR)
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 sotmax 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://deepnotes.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

```
y = np.array(([-2.85],[0.86],[0.28]))
exp_y = np.exp(y)
log_k = -np.max(exp_y)
exp_y_logk = exp_y + log_k
#print(log_k)
sum_exp_y = np.sum(exp_y_logk)
#print(exp_y_logk)
#print(exp_y_logk/sum_exp_y)
```

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

```
"""Softmax model."""

import numpy as np


```

```
class Softmax:
    def __init__(self, n_class: int, lr: float, epochs: int, reg_const: float):
        """Initialize a new classifier.

        Parameters:
            n_class: the number of classes
            lr: the learning rate
            epochs: the number of epochs to train for
            reg_const: the regularization constant
        """
        self.w = None  # TODO: change this
        self.lr = lr
        self.epochs = epochs
        self.reg_const = reg_const
        self.n_class = n_class

    def calc_gradient(self, X_train: np.ndarray, y_train: np.ndarray) -> np.ndarr
        """Calculate gradient of the softmax loss.

        Inputs have dimension D, there are C classes, and we operate on
        mini-batches of N examples.

        Parameters:
            X_train: a numpy array of shape (N, D) containing a mini-batch
                of data
            y_train: a numpy array of shape (N,) containing training labels;
                y[i] = c means that X[i] has label c, where 0 <= c < C

        Returns:
            gradient with respect to weights w; an array of same shape as w
        """
        #N, D = X_train.shape
        #print(N,D)
        #gradients = np.zeros((N,D))
        x = X_train

        y_hat_list = np.dot(self.reg_const + self.w, x)  # get the dot product o
        #print(y_hat_list)
        #exp_y = np.exp(y_hat_list)
        #print(exp_y)
        log_k = -np.max(y_hat_list)
        exp_y = np.exp(y_hat_list + log_k)
        sum_exp_y = np.sum(exp_y)
        gradients = exp_y / sum_exp_y

        return gradients

    def train(self, X_train: np.ndarray, y_train: np.ndarray):
        """Train the classifier.

        Hint: operate on mini-batches of data for SGD.
```

```

        Parameters:
            X_train: a numpy array of shape (N, D) containing training data;
                N examples with D dimensions
            y_train: a numpy array of shape (N,) containing training labels
        """
        N, D = X_train.shape
        #self.w = np.random.uniform(low=0.1, high=0.8,size=(N,D))
        #self.w = np.zeros((N,D))
        self.w = np.random.rand(self.n_class,D)
        #print(self.w.shape)

        for iter in range(self.epochs):
          #if iter > 4:
          self.lr -= iter*self.lr/5

          #if self.lr > 6:
          self.reg_const /= 0.9

          for example_num in range(N):
            x = X_train[example_num]
            y_label = y_train[example_num]
            #print(y_label)
            #print(x.shape)
            gradients = self.calc_gradient(x,y_label)
            #print(gradients)
            #break

            for class_num in range(self.n_class):
              if class_num == y_label:
                self.w[y_label] = self.w[y_label] + (self.lr*(1 - gradients[y_lab
              else:
                self.w[class_num] = self.w[class_num] - (self.lr*(gradients[class


        return

    def predict(self, X_test: np.ndarray) -> np.ndarray:
        """Use the trained weights to predict labels for test data points.

        Parameters:
            X_test: a numpy array of shape (N, D) containing testing data;
                N examples with D dimensions

        Returns:
            predicted labels for the data in X_test; a 1-dimensional array of
                length N, where each element is an integer giving the predicted
                class.
        """
        N, D = X_test.shape
        labels = np.zeros(N)
```

```
      for image_num in range(N):
        x = X_test[image_num]
        y_hat_list = np.dot(self.w, x)
        labels[image_num] = np.argmax(y_hat_list)
        if self.n_class == 2:
          labels[image_num] = np.where(labels[image_num] == -1, 0, labels[image

```

```
lr = 0.01
n_epochs = 14
reg_const = 0.55

softmax_fashion = Softmax(n_class_fashion, lr, n_epochs, reg_const)
softmax_fashion.train(X_train_fashion, y_train_fashion)
```

```
pred_softmax = softmax_fashion.predict(X_train_fashion)
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

```
pred_softmax = softmax_fashion.predict(X_val_fashion)
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

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

```
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:

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

Train Softmax on Mushroom

```
lr = 0.5
n_epochs = 10
reg_const = 0.05

softmax_MR = Softmax(n_class_MR, lr, n_epochs, reg_const)
#rint(n_class_MR)
softmax_MR.train(X_train_MR, y_train_MR)
print(y_train_MR.shape)
```

```
(4874,)
```

```
pred_softmax = softmax_MR.predict(X_train_MR)
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

```
pred_softmax = softmax_MR.predict(X_val_MR)
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

```
pred_softmax = softmax_MR.predict(X_test_MR)
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 Classifer

```

```

Load mushroom dataset

```
# TRAINING = 0.6 indicates 60% of the data is used as the training dataset.
VALIDATION = 0.2
data = get_MUSHROOM_data(VALIDATION)
X_train_MR, y_train_MR = data['X_train'], data['y_train']
X_val_MR, y_val_MR = data['X_val'], data['y_val']
X_test_MR, y_test_MR = data['X_test'], data['y_test']
n_class_MR = len(np.unique(y_test_MR))

print("Number of train samples: ", X_train_MR.shape[0])
print("Number of val samples: ", X_val_MR.shape[0])
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
```

```

```

```
# TAKE a look at the x_sub_i mushroom example values
#TOTAL examples: 4874
print("X_train_MR is: ", X_train_MR)

# there are 22 features = dimensions of each mushroom example
print("X_train_MR shape is: ", X_train_MR.shape)

#see the training label values
print("\ny_train_MR is: ", y_train_MR)

# see the shape of ytrain
print("y_train_MR shape is: ", y_train_MR.shape)
```

```

##################################################################
#see the validation label values
print("\ny_val_MR is: ", y_val_MR)

# see the shape of y for validation
print("y_val_MR is: ", y_val_MR.shape)
#####################################################################


#see the testing label values
print("\ny_test_MR is: ", y_test_MR)

# see the shape of ytest
print("y_test_MR is: ", y_test_MR.shape)

# y_train_MR is:  [1 0 0 ... 1 1 0]
# y_train_MR is:  (4874,)
#we notice that y_train has 1,0 as labels. We need to replace the zero labels wit
#convert the zero in y to -1.
y_train_MR = np.array([-1 if value==0 else 1 for value in y_train_MR])
print("\n\n\nconverted_y_train is: ",y_train_MR)


#convert for y_val_MR as well
y_val_MR = np.array([-1 if value==0 else 1 for value in y_val_MR])
print("\nconverted_y_val_MR is: ",y_val_MR)

#convert for y_test_MR as well
y_test_MR = np.array([-1 if value==0 else 1 for value in y_test_MR])
print("\nconverted_y_test_MR is: ",y_test_MR)





```

```
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]
```

```
def scalar_value_of_sigmoid(sigmoid_input):
        """Sigmoid function.

        Parameters:
            z: the input

        Returns:
            the sigmoid of the input
        """

        sigmoid_value = 1/(1+math.exp(-1*sigmoid_input))
        # print("sigmoid function returns: ",sigmoid_value)
        return sigmoid_value
```

```

```

```
# find the gradient of loss at a point
def sgd_gradient_of_loss_for_a_point(weight_vec,y_sub_i,x_sub_i,learning_rate,sig


  # print("y_sub_i is: ",y_sub_i)

  sigmoid_input_for_gradient = -1*y_sub_i*(np.dot(x_sub_i,weight_vec))
  # print("sigmoid input of gradient is: ",sigmoid_input_for_gradient)
  # print("shape of sigmoid input of gradient is: ",sigmoid_input_for_gradient.sh

  # print("x_sub_i is: ",x_sub_i)
  # print("x_sub_i shape is: ",x_sub_i.shape)

  # print("weight_vec is: ",weight_vec)
  # print("weight_vec shape is: ",weight_vec.shape)

  output_of_sigmoid_function = scalar_value_of_sigmoid(sigmoid_input_for_gradient
  # print("output_of_sigmoid_function is: ",output_of_sigmoid_function)


  gradient_of_loss_multiplied_by_eta = (x_sub_i)*learning_rate*(output_of_sigmoid
  # print("gradient_of_loss_multiplied_by_eta is: ",gradient_of_loss_multiplied_b
  # print("shape of gradient_of_loss_multiplied_by_eta is: ",gradient_of_loss_mul
  gradient_of_loss_multiplied_by_eta = gradient_of_loss_multiplied_by_eta.reshape
```

```

  return gradient_of_loss_multiplied_by_eta

```

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

```

```
"""Logistic regression model."""

import numpy as np


class Logistic:
    def __init__(self, lr: float, epochs: int, threshold: float):
        """Initialize a new classifier.

        Parameters:
            lr: the learning rate
            epochs: the number of epochs to train for
        """
        self.weight_vec = None  # TODO: change this
        self.lr = lr
        self.epoch_number = epochs
        self.threshold = threshold
        self.logistic_loss = []

    # def sigmoid(self, z: np.ndarray) -> np.ndarray:
    #     """Sigmoid function.

    #     Parameters:
    #         z: the input

    #     Returns:
    #         the sigmoid of the input
    #     """
    #     exp_z = np.exp(-z)
    #     # print("exp_z is: ",exp_z)
    #     # ones_array = np.ones(len(z))
    #     sum = 1+exp_z
    #     print("sum is: ",sum)
    #     sigmoid_value = 1/(1+exp_z)
    #     print("sigmoid function returns: ",sigmoid_value)
    #     return sigmoid_value

    def train(self, X_train: np.ndarray, y_train: np.ndarray):
        """Train the classifier.

        Use the *logistic regression update rule* as introduced in lecture.

```

```
        Parameters:
            X_train: a numpy array of shape (N, D) containing training data;
                N examples with D dimensions
            y_train: a numpy array of shape (N,) containing training labels
        """
        #in class notes, x_rows=n and x_cols=d
        x_rows,x_cols = X_train.shape
        self.weight_vec = np.zeros((x_cols,1))
        # print("self.weight_vec is: ",self.weight_vec)
        # print("self.weight_vec.shape is: ",self.weight_vec.shape)

        #reshape y_train to a column vector that is n by 1,
        y_train = y_train.reshape(x_rows,1)


        #loop for each epoch
        for epoch_number in range(self.epoch_number):

            #we need to iterate over the weight matrix and take each row as input
            for x_row_index in range(x_rows):
                  x_row_for_example = X_train[x_row_index]
                  # print("x_row_for_example:",x_row_for_example)
                  y_label = y_train[x_row_index]
                  # print("y_label:",y_label)

                  sigmoid_input = y_label*np.dot(x_row_for_example,self.weight_ve
                  sigmoid_output = scalar_value_of_sigmoid(sigmoid_input)
                  delta_weight_vector = sgd_gradient_of_loss_for_a_point(self.wei
                  # print("delta_weight_vector is: ",delta_weight_vector)

                  # Updating the weight vector.
                  # print("weight vector before update is: ",self.weight_vec)
                  self.weight_vec = self.weight_vec + delta_weight_vector
                  # print("weight vector after update is: ",self.weight_vec)


        return self.weight_vec




    def predict(self, X_test: np.ndarray) -> np.ndarray:
        """Use the trained weights to predict labels for test data points.

        Parameters:
            X_test: a numpy array of shape (N, D) containing testing data;
                N examples with D dimensions

        Returns:
            predicted labels for the data in X_test; a 1-dimensional array of
                length N, where each element is an integer giving the predicted
```

```
                class.
        """

        N, D = X_test.shape
        labels = np.zeros((N))
        #print(self.w.shape)
        for example_num in range(N):
          x = X_test[example_num]
          y_hat = np.dot(x,self.weight_vec)
          if y_hat>=self.threshold:
            labels[example_num] = 1
          else:
            labels[example_num] = -1


        return labels
```

```
learning_rate = 0.6
n_epochs = 20
threshold = 0.5

lr = Logistic(learning_rate, n_epochs, threshold)
lr.train(X_train_MR, y_train_MR)
```

```
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]])
```

```

```

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

```
print("True y labels for training set of mushroom dataset are:",y_train_MR)
print("Predicted y labels for training set of mushroom dataset are:",pred_lr)

```

```
The training accuracy is given by: 94.870743
True y labels for training set of mushroom dataset are: [ 1 -1 -1 ...  1  1 -1]
Predicted y labels for training set of mushroom dataset are: [ 1. -1. -1. ...  1
```

Validate Logistic Classifer

```
pred_lr = lr.predict(X_val_MR)
print('The validation accuracy is given by: %f' % (get_acc(pred_lr, y_val_MR)))
```

```
The validation accuracy is given by: 94.153846
```

Test Logistic Classifier

```
pred_lr = lr.predict(X_test_MR)
print('The testing accuracy is given by: %f' % (get_acc(pred_lr, y_test_MR)))
```

```
The testing accuracy is given by: 94.461538
```

us completed at 11:28 PM