Robert Heeter
ELEC 378
4/9/2023

Problem Set #9

1. a. See code and output below.

b. See code and output from gradient descent below. with $\lambda = 1$. The descent converges to the same final $\underline{w}^*$ and b^* (same line) regardless of the initializations of $\underline{w}$ and b, shown as the darkest colored line. This "solution"/decision hyperplane appears stable.

c. See code and output below with $\lambda = 0.5$ and $\lambda = 15$. For a smaller value of λ, the descent towards the ideal decision hyperplane is slightly faster/steeper as accuracy is weighted more than optimizing the margins. For a larger value of λ, the descent oscillates between two decision hyperplanes with similar accuracy and margins. λ trades off the size of the margin with the number of errors; a large λ allows for more errors to produce a larger margin while a small λ allows for fewer errors and thus a smaller margin — i.e. accuracy and margin size are inversely related.

(see code and output below)

d. Stochastic gradient steps produce a slower and less "controlled" descent as many more iterations are necessary than in G.D. to approach the same optimal solution/decision hyperplane from the same initial conditions.

e. See code and output below. The same implementation as for Train1.mat (which had linearly separable data) does not work well for Train2.mat with a misclassification error of about 24%. Looking at the data, it is clear that the Train2.mat data is not linearly separable (the "purple" data wraps around the "yellow" data), and the decision hyperplane cuts through both groups of data.

(see code and output below)

f. $\phi(\underline{x}_i) = x_i[0]^2 + x_i[1]^2$ works well as a nonlinear function to map the data points from $\underline{x}_i$ to $\phi(\underline{x}_i)$ where they are more easily linearly separable. This function ϕ represents the distance from a given point to the origin. Because the "yellow" data is grouped more closely to the origin overall than the "purple" data, the resulting mapping (see 3D scatter plot) appears successful in making the data linearly separable.

(1). g. See code and output below. The misclassification error of the transformed data is ~9% < 24% from the original data, so this SVM fit is much more accurate.

2. Next page.

2. $c(\underline{\hat{x}}) = \text{sign}\left(\langle \underline{w}^*, \underline{\hat{x}} \rangle + b^*\right)$

↑ weights ↑ unlabeled observation ↑ bias

Hard margin SVM: $\underline{w}^*, b^* = \underset{\underline{w}, b}{\text{argmin}} \|\underline{w}\|_2^2$ subject to

$$y_i\left(\langle \underline{w}, \underline{x}_i \rangle + b\right) \geq 1, \quad i = 1 \dots n$$

This only works when data is linearly separable in raw feature space

a. $$\underline{w}^*, b^* = \underset{\underline{w}, b}{\text{argmin}} \|\underline{w}\|_2^2 + \sum_{i=1}^{\hat{n}} \lambda_i \left(1 - y_i\left(\langle \underline{w}, \underline{x}_i \rangle + b\right)\right)$$

$$= \underset{\underline{w}, b}{\text{argmin}} \|\underline{w}\|_2^2 + \sum_{i=1}^{\hat{n}} \lambda_i - y_i \lambda_i \langle \underline{w}, \underline{x}_i \rangle - y_i \lambda_i b$$

Minimum where $\nabla = 0$:

$$0 = \frac{d}{d\underline{w}} \|\underline{w}\|_2^2 + \sum_{i=1}^{\hat{n}} \frac{d}{d\underline{w}} \lambda_i - \frac{d}{d\underline{w}} y_i \lambda_i \langle \underline{w}, \underline{x}_i \rangle - \frac{d}{d\underline{w}} y_i \lambda_i b$$

$$0 = 2\underline{w}^* - \sum_{i=1}^{\hat{n}} y_i \lambda_i \underline{x}_i$$

$$\therefore 2\underline{w}^* = \sum_{i=1}^{\hat{n}} y_i \lambda_i \underline{x}_i \longrightarrow \underline{w}^* = \frac{1}{2} \sum_{i=1}^{\hat{n}} y_i \lambda_i \underline{x}_i$$

$$\boxed{\underline{w}^* = \sum_{i=1}^{\hat{n}} \alpha_i \underline{x}_i \text{ where } \alpha_i = \frac{y_i \lambda_i}{2} \text{ and } \{\alpha\}_{i=1}^{\hat{n}}}$$

b. $$c(\underline{\hat{x}}) = \text{sign}\left(\langle \underline{w}^*, \underline{\hat{x}} \rangle + b^*\right)$$

$$= \text{sign}\left(\langle \sum_{i=1}^{\hat{n}} \alpha_i \underline{x}_i, \underline{\hat{x}} \rangle + b^*\right) \longrightarrow \boxed{c(\underline{\hat{x}}) = \left(\sum_{i=1}^{\hat{n}} \alpha_i \langle x_i, \underline{\hat{x}} \rangle + b^*\right)}$$

The class label $c(\underline{\hat{x}})$ is only a function of the inner product of $\underline{\hat{x}}$ and the training data points $\{\underline{x}_i\}_{i=1}^{\hat{n}}$ multiplied by a factor α_i and offset by intercept b^*.

c. $c(\hat{x}) = \left(\sum_{i=1}^{n} \alpha_i \langle x_i, \hat{x} \rangle + b^* \right)$

Only inner product computational complexity changes with p:

$x, \hat{x} \in \mathbb{R}^p \rightarrow \langle x, \hat{x} \rangle = \sum_{i=1}^{p} x[i]\hat{x}[i]$

↑ multiplied p times $\rightarrow O(p)$

summed p terms (i.e. added p times) $\rightarrow O(p)$

Computational cost is $O(p)$

d.

$$\underline{x} = \begin{bmatrix} x[1] \\ \vdots \\ x[p] \end{bmatrix} \rightarrow \begin{bmatrix} x[1]^2 \\ \vdots \\ x[p]^2 \\ \sqrt{2}x[1]x[2] \\ \sqrt{2}x[1]x[3] \\ \vdots \\ \sqrt{2}x[1]x[p-1] \\ \sqrt{2}x[1]x[p] \\ \sqrt{2}x[2]x[3] \\ \vdots \\ \sqrt{2}x[2]x[p] \\ \vdots \\ \sqrt{2}x[p-1]x[p] \end{bmatrix} \begin{matrix} \} 1 \rightarrow p = p \\ \} 2 \rightarrow p = p-1 \\ \\ \\ \\ \} 3 \rightarrow p = p-2 \\ \\ \vdots \\ \} p-1 \rightarrow p = 1 \end{matrix} = p + (p-1) + (p-2) + \ldots + 1 \text{ terms total}$$

$$= \begin{bmatrix} x\phi[1] \\ \vdots \\ x\phi[P] \end{bmatrix} \rightarrow P = p + (p-1) + (p-2) + \ldots + 1$$

$$\boxed{P = \frac{p(p+1)}{2}}$$

e. Next page

e. $\{(y_i, \phi(\underline{x}_i) \in \mathbb{R}^p\}_{i=1}^n \rightarrow$ linearly separable

$\tilde{\underline{w}}^*, \tilde{b}^* = \underset{\underline{w}, b}{\text{argmin}} \|\underline{w}\|_2^2$ subject to $y_i(\langle \underline{w}, \phi(\underline{x}_i)\rangle + b) \geq 1, \quad i = 1 \ldots n$

$c(\hat{\underline{x}}) = c_\phi(\hat{\underline{x}}) = \text{sign}\left(\langle \tilde{\underline{w}}^*, \phi(\hat{\underline{x}})\rangle + \tilde{b}^*\right)$

$c_\phi(\hat{\underline{x}}) = \text{sign}\left(\langle \phi(\underline{x}), \phi(\hat{\underline{x}})\rangle + \tilde{b}^*\right)$ using the result from part B.

Computational complexity of inner product is $O(p)$ from part C.

$p \propto p^2$, so $O(p) = O(p^2)$ $\rightarrow$ Computational cost is $O(p^2)$, which is a factor of p worse than the $O(p)$ cost in part C.

f. Multinomial theorem shows:

$$\langle \phi(\underline{u}), \phi(\underline{v})\rangle = \sum_{i=1}^p \phi(u[i])\phi(v[i])$$

$$= \sum_{i=1}^p (u[i]^2)(v[i]^2) + \sum_{i=2}^p \sum_{j=1}^{i-1} (\sqrt{2}\, u[i]u[j])(\sqrt{2}\, v[i]v[j])$$

$$= \left(\sum_{i=1}^p u[i]v[i]\right)^2 = (\langle \underline{u}, \underline{v}\rangle)^2$$

Using this, to fit an SVM to nonlinearly separable data without computing any transformation of the data points:

$$c_\phi(\hat{\underline{x}}) = \text{sign}\left(\langle \phi(\underline{x}), \phi(\hat{\underline{x}})\rangle + \tilde{b}^*\right)$$

$$c_\phi(\hat{\underline{x}}) = \text{sign}\left((\langle \underline{x}, \hat{\underline{x}}\rangle)^2 + \tilde{b}^*\right)$$

$O(p)$ for inner product.

Computational cost is now $O(p)$ instead of $O(p^2)$ since $\phi(\underline{x})$ is no longer directly computed.

3. Next page.

3. See attached code which was used for group Kaggle submission, SVM uses sklearn's SVC function to train an SVM on the final project data.

ELEC378-HW9

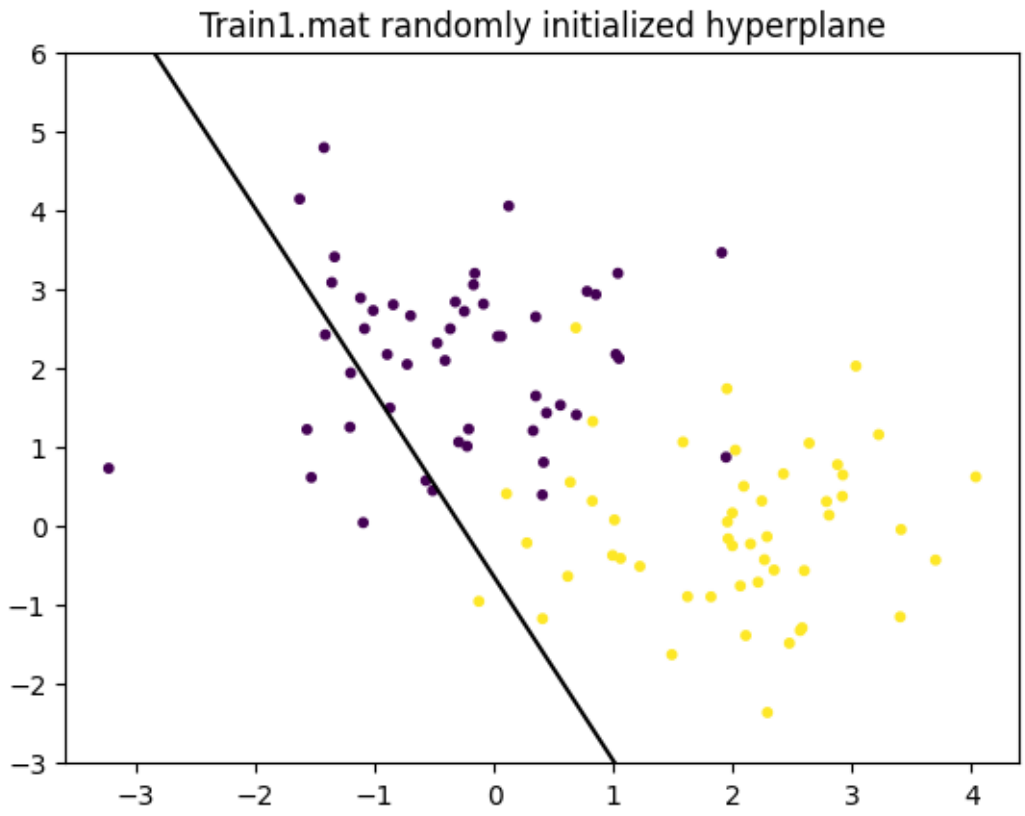
April 9, 2023

[1]:
```
# ROBERT HEETER
# ELEC 378 Machine Learning
# 7 April 2023

# PROBLEM SET 9
```

[2]:
```
# PROBLEM 1

import numpy as np
from matplotlib import pyplot as plt

import scipy.io
import random

# load files
train1 = scipy.io.loadmat('Train1.mat')
train2 = scipy.io.loadmat('Train2.mat')
```

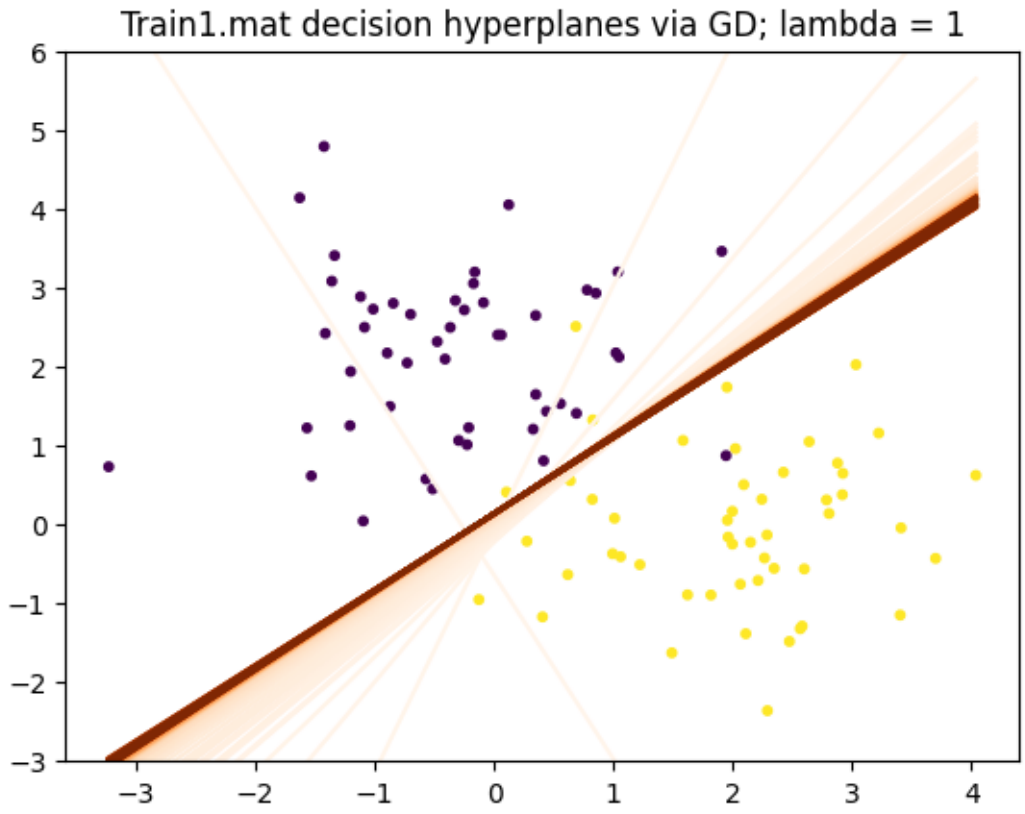
[3]:
```
# PART A

# load Train1.mat data
X = train1['X'] # X = data matrix w/ shape (n, p)
y = train1['y'] # Y = labels (±1)

# dimensions of X
n = train1['n'][0][0]
p = train1['p'][0][0]

T = 200 # T = number of iterations for subgradient descent

# initalize w_0 and b_0 to random values
w_0 = np.array([random.uniform(-2,2) for _ in range(p)]).reshape(p,)
b_0 = random.random()

w_s = np.zeros((T+1,p)) # w_s[i] = weights after ith iteration of subgradient␣
 ↪descent
```

```
b_s = np.zeros((T+1,)) # b_s[i] = intercept after ith iteration of subgradient␣
↪descent
w_s[0] = w_0
b_s[0] = b_0

w_s_stochastic = w_s # save for part D later
b_s_stochastic = b_s

# plot hyperplane function
def plot_hyperplane_2d(x_axis, w, b, c='k'):
    slope = -1*w[0]/w[1]
    intercept = -1*b/w[1]
    plt.plot(x_axis, (x_axis*slope)+intercept, color=c)

# calculate misclassification error function
def misclassification_error_2d(X, y, w, b):
    slope = -1*w[0]/w[1]
    intercept = -1*b/w[1]

    count = 0 # error count
    for i in range(len(y)):
        y_value = (X[i,0]*slope)+intercept

        if X[i,1] > y_value:
            y_pred = -1
        else:
            y_pred = 1

        y_label = y[i]
        if y_pred != y_label:
            count += 1
    return (count/len(y))*100

# create x-axis values to plot hyperplane
x_axis = np.linspace(np.min(X[:,0]), np.max(X[:,0]), n)

# plot randomly-initialized hyperplane
plt.figure()
plt.scatter(X[:,0], X[:,1], c=y, s=10)
plot_hyperplane_2d(x_axis, w_s[0], b_s[0])
plt.ylim(-3,6)
plt.title(f'Train1.mat randomly initialized hyperplane')
plt.show()
```



[4]:
```
# PART B

# plot sequence of hyperplanes
plt.figure()
plt.scatter(X[:,0], X[:,1], c=y, s=10)

colors = plt.cm.Oranges(np.linspace(0,1,T+1))

la = 1 # la = lambda/constraint
mu = 0.01 # mu = learning rate
y = y.reshape(n,)

# gradient descent
for t in range(T):
    plot_hyperplane_2d(x_axis, w_s[t], b_s[t], colors[t])

    y_pred = np.dot(X, w_s[t]) + b_s[t]
    s_b = np.where(y * y_pred < 1, -y, 0)

    s_w_sum = np.dot(X.T, s_b)
    w_subgrad = s_w_sum + (2*la*w_s[t])
```

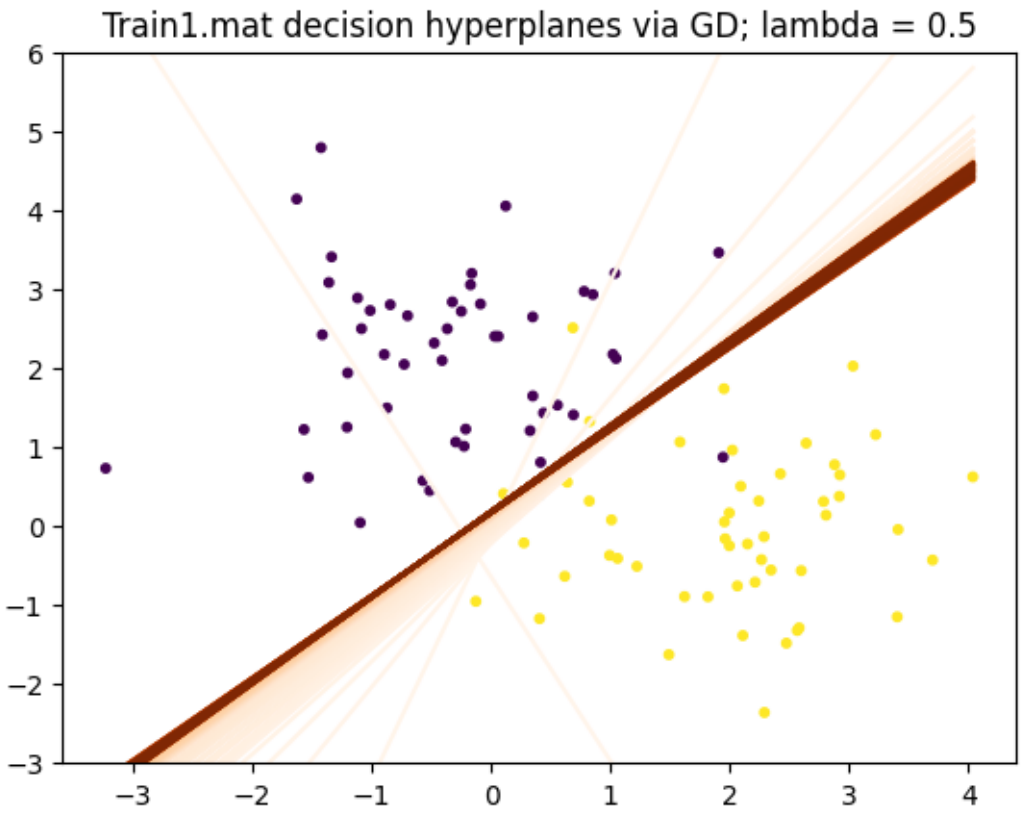
```
    b_subgrad = np.sum(s_b)

    w_s[t+1] = w_s[t] - (mu * w_subgrad)
    b_s[t+1] = b_s[t] - (mu * b_subgrad)

# plot final hyperplane
plot_hyperplane_2d(x_axis, w_s[T], b_s[T], colors[T])

plt.ylim(-3,6)
plt.title(f'Train1.mat decision hyperplanes via GD; lambda = {la}')
plt.show()

# calculate misclassification error
error = misclassification_error_2d(X, y, w_s[T], b_s[T])
print(f'MISCLASSIFICATION ERROR RATE: {error}%')
```

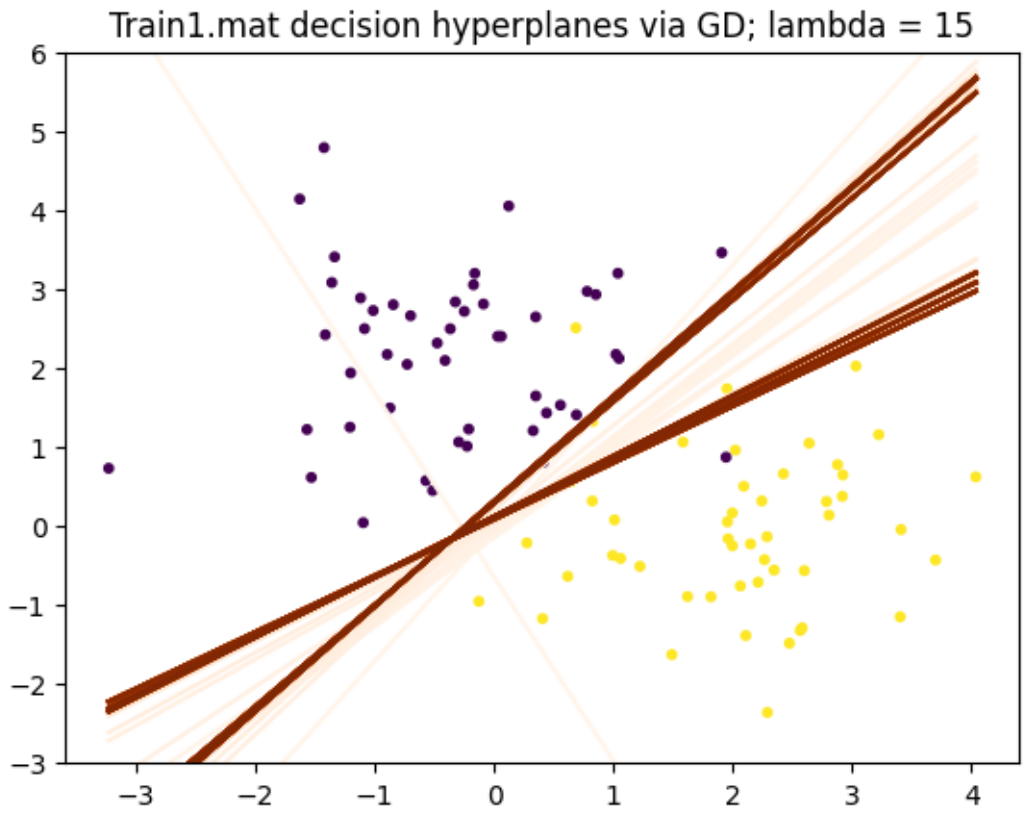



```
MISCLASSIFICATION ERROR RATE: 5.0%
```

[5]:
```
# PART C
```

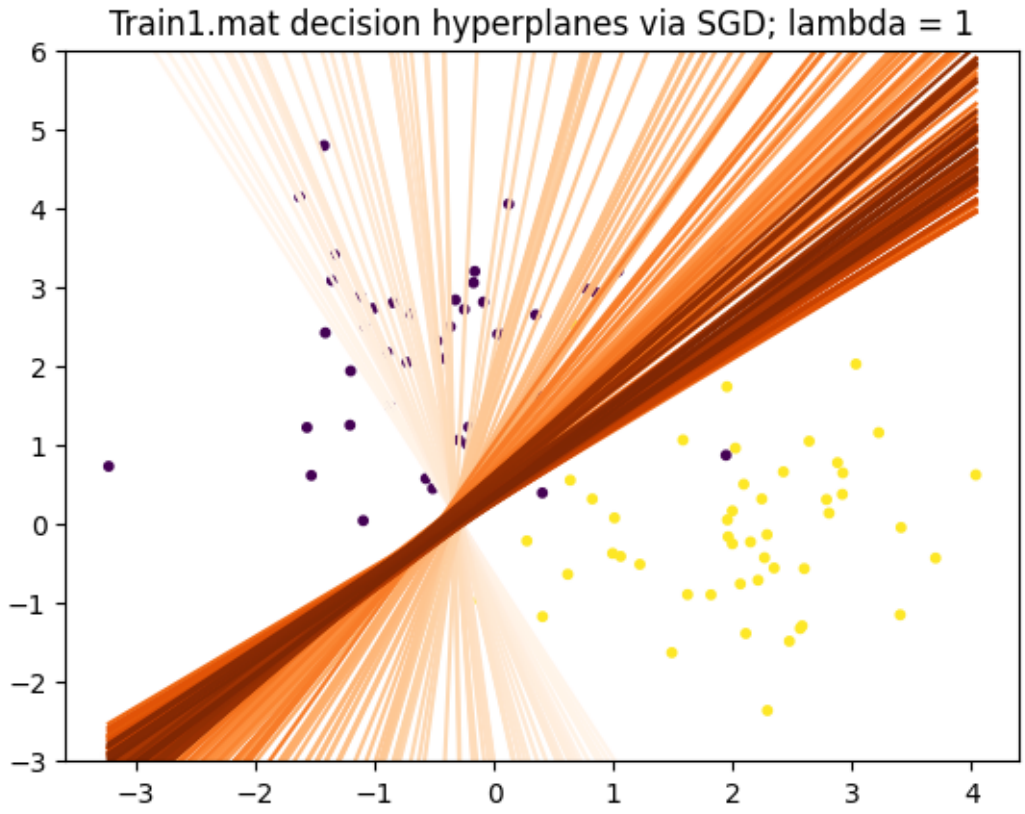
```
for la in [0.5,15]: # la = lambda/constraint

    # plot sequence of hyperplanes
    plt.figure()
    plt.scatter(X[:,0], X[:,1], c=y, s=10)

    colors = plt.cm.Oranges(np.linspace(0,1,T+1))

    mu = 0.01 # mu = learning rate
    y = y.reshape(n,)

    # gradient descent
    for t in range(T):
        plot_hyperplane_2d(x_axis, w_s[t], b_s[t], colors[t])

        y_pred = np.dot(X, w_s[t]) + b_s[t]
        s_b = np.where(y * y_pred < 1, -y, 0)

        s_w_sum = np.dot(X.T, s_b)
        w_subgrad = s_w_sum + (2*la*w_s[t])

        b_subgrad = np.sum(s_b)

        w_s[t+1] = w_s[t] - (mu * w_subgrad)
        b_s[t+1] = b_s[t] - (mu * b_subgrad)

    # plot final hyperplane
    plot_hyperplane_2d(x_axis, w_s[T], b_s[T], colors[T])

    plt.ylim(-3,6)
    plt.title(f'Train1.mat decision hyperplanes via GD; lambda = {la}')
    plt.show()
```


Train1.mat decision hyperplanes via GD; lambda = 0.5
6
5
4
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
4



[6]:
```
# PART D

w_s = w_s_stochastic
b_s = b_s_stochastic

# plot sequence of hyperplanes
plt.figure()
plt.scatter(X[:,0], X[:,1], c=y, s=10)

colors = plt.cm.Oranges(np.linspace(0,1,T+1))

la = 1 # la = lambda/constraint
mu = 0.01 # mu = learning rate
y = y.reshape(n,)

# stochastic gradient descent
for t in range(T):
    subset = np.random.choice(X.shape[0], size=1, replace=False)
    X_subset = X[subset]
    y_subset = y[subset]
```

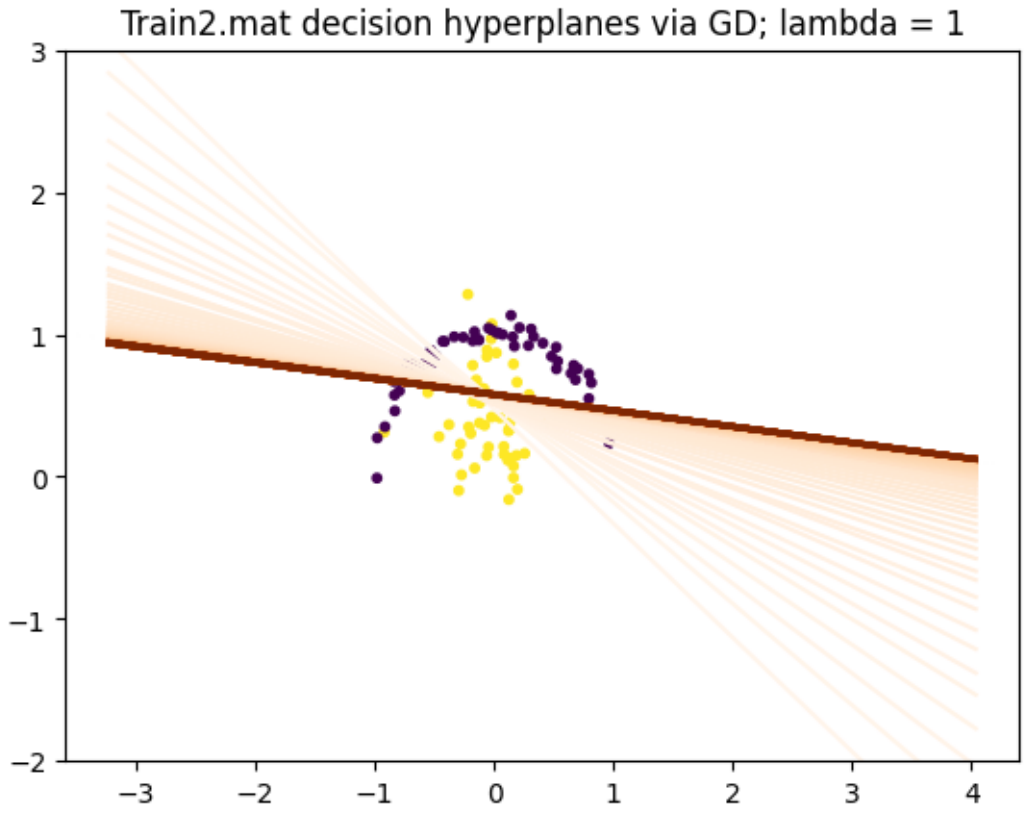
```
    plot_hyperplane_2d(x_axis, w_s[t], b_s[t], colors[t])

    y_pred = np.dot(X_subset, w_s[t]) + b_s[t]
    s_b = np.where(y_subset * y_pred < 1, -y_subset, 0)

    s_w_sum = np.dot(X_subset.T, s_b)
    w_subgrad = s_w_sum + (2*la*w_s[t])

    b_subgrad = np.sum(s_b)

    w_s[t+1] = w_s[t] - (mu * w_subgrad)
    b_s[t+1] = b_s[t] - (mu * b_subgrad)

# plot final hyperplane
plot_hyperplane_2d(x_axis, w_s[T], b_s[T], colors[T])

plt.ylim(-3,6)
plt.title(f'Train1.mat decision hyperplanes via SGD; lambda = {la}')
plt.show()

# calculate misclassification error
error = misclassification_error_2d(X, y, w_s[T], b_s[T])
print(f'MISCLASSIFICATION ERROR RATE: {error}%')
```



MISCLASSIFICATION ERROR RATE: 4.0%

[7]:
```
# PART E

# load Train2.mat data
X = train2['X'] # X = data matrix w/ shape (n, p)
y = train2['y'] # Y = labels (±1)

# dimensions of X
n = train1['n'][0][0]
p = train1['p'][0][0]

T = 200 # T = number of iterations for subgradient descent

# initalize w_0 and b_0 to random values
w_0 = np.array([random.uniform(-2,2) for _ in range(p)]).reshape(p,)
b_0 = random.random()

w_s = np.zeros((T+1,p)) # w_s[i] = weights after ith iteration of subgradient␣
 ↪descent
b_s = np.zeros((T+1,)) # b_s[i] = intercept after ith iteration of subgradient␣
 ↪descent
w_s[0] = w_0
b_s[0] = b_0

# plot sequence of hyperplanes
plt.figure()
plt.scatter(X[:,0], X[:,1], c=y, s=10)

colors = plt.cm.Oranges(np.linspace(0,1,T+1))

la = 1 # la = lambda/constraint
mu = 0.01 # mu = learning rate
y = y.reshape(n,)

# gradient descent
for t in range(T):
    plot_hyperplane_2d(x_axis, w_s[t], b_s[t], colors[t])

    y_pred = np.dot(X, w_s[t]) + b_s[t]
    s_b = np.where(y * y_pred < 1, -y, 0)

    s_w_sum = np.dot(X.T, s_b)
    w_subgrad = s_w_sum + (2*la*w_s[t])
```

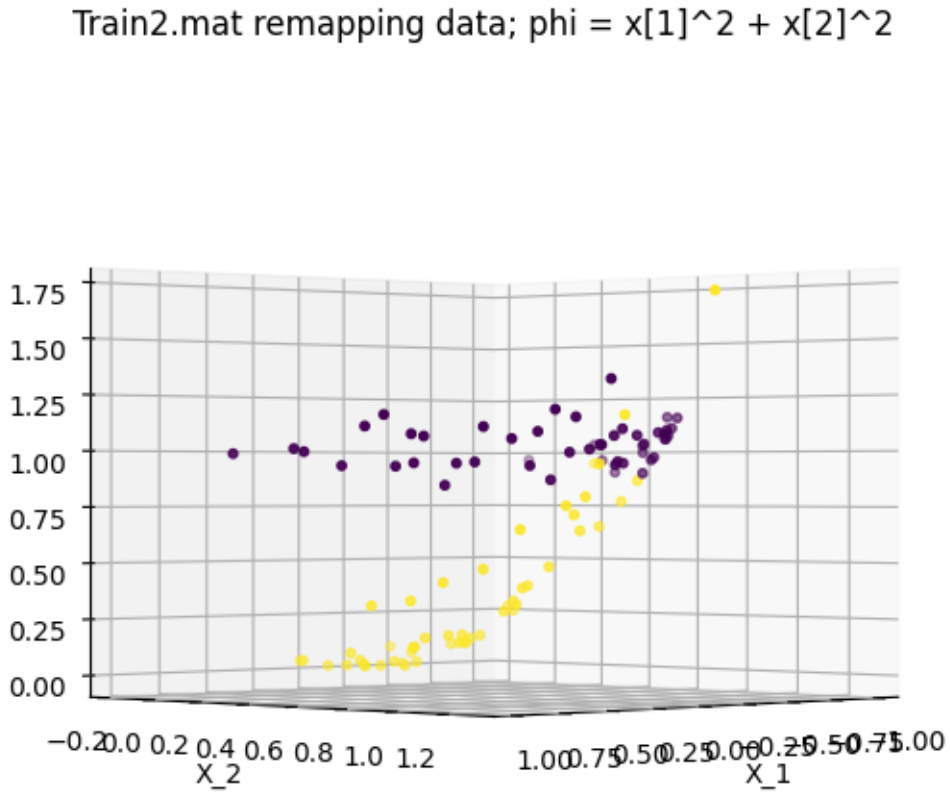
```
    b_subgrad = np.sum(s_b)

    w_s[t+1] = w_s[t] - (mu * w_subgrad)
    b_s[t+1] = b_s[t] - (mu * b_subgrad)

# plot final hyperplane
plot_hyperplane_2d(x_axis, w_s[T], b_s[T], colors[T])

plt.ylim(-2,3)
plt.title(f'Train2.mat decision hyperplanes via GD; lambda = {la}')
plt.show()

# calculate misclassification error
error = misclassification_error_2d(X, y, w_s[T], b_s[T])
print(f'MISCLASSIFICATION ERROR RATE: {error}%')
```




```
MISCLASSIFICATION ERROR RATE: 24.0%
```

[8]:
```
# PART F

# load Train2.mat data
```

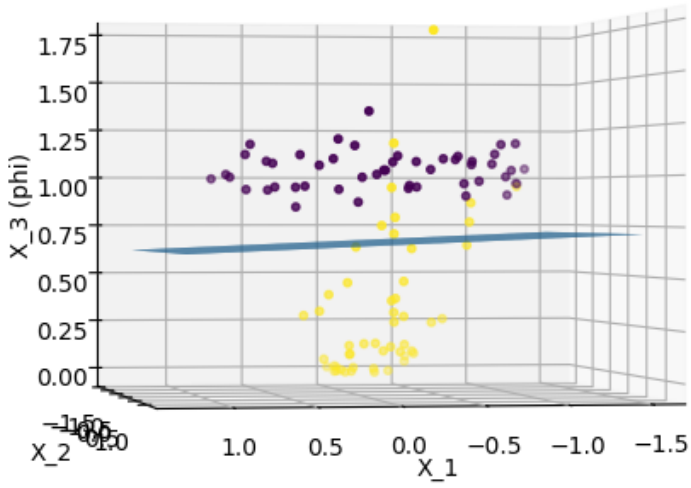
```
X = train2['X'] # X = data matrix w/ shape (n, p)
y = train2['y'] # Y = labels (±1)

# dimensions of X
n = train1['n'][0][0]
p = train1['p'][0][0]

# add third column with nonlinear function phi = x[1]^2 + x[2]^2
p = 3
X_mod = np.zeros((n,p))
X_mod[:,0:2] = X

phi = X[:,0]**2 + X[:,1]**2
X_mod[:,2] = phi

X = X_mod

# plot new data
fig = plt.figure(figsize = (7, 6))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X_1'); ax.set_ylabel('X_2'); ax.set_zlabel('X_3 (phi)')
ax.scatter(X[:,0], X[:,1], X[:,2], c=y, s=10)

ax.view_init(elev=0, azim=45)
plt.title(f'Train2.mat remapping data; phi = x[1]^2 + x[2]^2')
plt.show()
```



[9]:
```
# PART G

# calculate misclassification error function
def misclassification_error_3d(X, y, w, b):
    count = 0 # error count
    for i in range(len(y)):
        y_value = 1/w[2]*((-1*X[i,0]*w[0]) + (-1*X[i,1]*w[1]) - b)

        if X[i,2] > y_value:
            y_pred = -1
        else:
            y_pred = 1

        y_label = y[i]
        if y_pred != y_label:
            count += 1
    return (count/len(y))*100
```

```
T = 400 # T = number of iterations for subgradient descent

# initalize w_0 and b_0 to random values
w_0 = np.array([random.uniform(-2,2) for _ in range(p)]).reshape(p,)
b_0 = random.random()

w_s = np.zeros((T+1,p)) # w_s[i] = weights after ith iteration of subgradient descent
b_s = np.zeros((T+1,)) # b_s[i] = intercept after ith iteration of subgradient descent
w_s[0] = w_0
b_s[0] = b_0

la = 1 # la = lambda/constraint
mu = 0.01 # mu = learning rate
y = y.reshape(n,)

# gradient descent
for t in range(T):
    y_pred = np.dot(X, w_s[t]) + b_s[t]
    s_b = np.where(y * y_pred < 1, -y, 0)

    s_w_sum = np.dot(X.T, s_b)
    w_subgrad = s_w_sum + (2*la*w_s[t])

    b_subgrad = np.sum(s_b)

    w_s[t+1] = w_s[t] - (mu * w_subgrad)
    b_s[t+1] = b_s[t] - (mu * b_subgrad)

# plot final hyperplane
fig = plt.figure(figsize = (7, 6))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X_1'); ax.set_ylabel('X_2'); ax.set_zlabel('X_3 (phi)')
ax.scatter(X[:,0], X[:,1], X[:,2], c=y, s=10)

ax_range = np.arange(-1.5, 1.5, 0.25)
XX1, XX2 = np.meshgrid(ax_range, ax_range)
XX3 = 1/w_s[T,2]*(-XX1*w_s[T,0]-XX2*w_s[T,1]-b_s[T])
ax.plot_surface(XX1,XX2,XX3)

ax.view_init(elev=0, azim=80)
plt.title(f'Train2.mat decision hyperplanes via GD; lambda = {la}; phi = x[1]^2 + x[2]^2')
plt.show()
```

```
# calculate misclassification error
error = misclassification_error_3d(X, y, w_s[T], b_s[T])
print(f'MISCLASSIFICATION ERROR RATE: {error}%')
```




```
MISCLASSIFICATION ERROR RATE: 9.0%
```

[ ]:

SpeechEmotionClassification (1)

April 9, 2023

PCA for feature extraction, then logistic regression

[ ]:

Convolutional Neural Networks

[ ]:

Spectrograms → image processing (i.e. k-nearest neighbors)

[ ]:
```
from torch.utils.data import random_split
from torch.utils.data import DataLoader, Dataset, random_split
import torchaudio

import math
import random
import torch
import torchaudio
from torchaudio import transforms
from IPython.display import Audio
import pandas as pd
import numpy as np
import os
```

[ ]:
```
class AudioUtil():
  # ----------------------------
  # Load an audio file. Return the signal as a tensor and the sample rate
  # ----------------------------
    def open(audio_file):
        sig, sr = torchaudio.load(audio_file)
        return (sig, sr)

    def rechannel(aud, new_channel):
        sig, sr = aud
        if (sig.shape[0] == new_channel):
            # Nothing to do
            return aud
```

```
        if (new_channel == 1):
            # Convert from stereo to mono by selecting only the first channel
            resig = sig[:1, :]
        else:
            # Convert from mono to stereo by duplicating the first channel
            resig = torch.cat([sig, sig])

        return ((resig, sr))

    def resample(aud, newsr):
        sig, sr = aud

        if (sr == newsr):
            # Nothing to do
            return aud

        num_channels = sig.shape[0]
        # Resample first channel
        resig = torchaudio.transforms.Resample(sr, newsr)(sig[:1, :])
        if (num_channels > 1):
            # Resample the second channel and merge both channels
            retwo = torchaudio.transforms.Resample(sr, newsr)(sig[1:, :])
            resig = torch.cat([resig, retwo])

        return ((resig, newsr))

    def pad_trunc(aud, max_ms):
        sig, sr = aud
        num_rows, sig_len = sig.shape
        max_len = sr//1000 * max_ms

        if (sig_len > max_len):
            # Truncate the signal to the given length
            sig = sig[:, :max_len]

        elif (sig_len < max_len):
            # Length of padding to add at the beginning and end of the signal
            pad_begin_len = random.randint(0, max_len - sig_len)
            pad_end_len = max_len - sig_len - pad_begin_len

            # Pad with 0s
            pad_begin = torch.zeros((num_rows, pad_begin_len))
            pad_end = torch.zeros((num_rows, pad_end_len))

            sig = torch.cat((pad_begin, sig, pad_end), 1)

        return (sig, sr)
```

```
    def time_shift(aud, shift_limit):
        sig, sr = aud
        _, sig_len = sig.shape
        shift_amt = int(random.random() * shift_limit * sig_len)
        return (sig.roll(shift_amt), sr)

    def spectro_gram(aud, n_mels=64, n_fft=1024, hop_len=None):
        sig, sr = aud
        top_db = 80

        # spec has shape [channel, n_mels, time], where channel is mono, stereo␣
↪etc
        spec = transforms.MelSpectrogram(
            sr, n_fft=n_fft, hop_length=hop_len, n_mels=n_mels)(sig)

        # Convert to decibels
        spec = transforms.AmplitudeToDB(top_db=top_db)(spec)
        return (spec)

    def spectro_augment(spec, max_mask_pct=0.1, n_freq_masks=1, n_time_masks=1):
        _, n_mels, n_steps = spec.shape
        mask_value = spec.mean()
        aug_spec = spec

        freq_mask_param = max_mask_pct * n_mels
        for _ in range(n_freq_masks):
            aug_spec = transforms.FrequencyMasking(
                freq_mask_param)(aug_spec, mask_value)

        time_mask_param = max_mask_pct * n_steps
        for _ in range(n_time_masks):
            aug_spec = transforms.TimeMasking(
                time_mask_param)(aug_spec, mask_value)

        return aug_spec
```

[ ]:
```
class SoundDS(Dataset):
    def __init__(self, df, data_path):
        self.df = df
        self.data_path = str(data_path)
        self.duration = 4000
        self.sr = 44100
        self.channel = 2
        self.shift_pct = 0.4

    # ----------------------------
```

```
    # Number of items in dataset
    # ----------------------------
    def __len__(self):
        return len(self.df)

    # ----------------------------
    # Get i'th item in dataset
    # ----------------------------
    def __getitem__(self, idx):
        # Absolute file path of the audio file - concatenate the audio␣
↪directory with
        # the relative path
        audio_file = self.data_path + self.df.loc[idx, 'relative_path']
        # Get the Class ID
        class_id = self.df.loc[idx, 'classID']

        aud = AudioUtil.open(audio_file)
        # Some sounds have a higher sample rate, or fewer channels compared to␣
↪the
        # majority. So make all sounds have the same number of channels and same
        # sample rate. Unless the sample rate is the same, the pad_trunc will␣
↪still
        # result in arrays of different lengths, even though the sound duration␣
↪is
        # the same.
        reaud = AudioUtil.resample(aud, self.sr)
        rechan = AudioUtil.rechannel(reaud, self.channel)

        dur_aud = AudioUtil.pad_trunc(rechan, self.duration)
        shift_aud = AudioUtil.time_shift(dur_aud, self.shift_pct)
        sgram = AudioUtil.spectro_gram(
            shift_aud, n_mels=64, n_fft=1024, hop_len=None)
        aug_sgram = AudioUtil.spectro_augment(
            sgram, max_mask_pct=0.1, n_freq_masks=2, n_time_masks=2)

        return aug_sgram, class_id
```

[ ]:
```
data = np.empty((1125, 2), dtype=object)
count = 0
#directory = "/Users/ariellesanford/Desktop/ELEC378/data/data"
directory = "/Users/rch/Documents/RICE UNIVERSITY/JUNIOR SPRING/ELEC 378␣
↪Machine Learning/ELEC 378 Project/
↪elec-378-sp2023-speech-emotion-classification/data/data"
for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    # checking if it is a file
```

```
    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        data[count][0] = "/"+filename
        data[count][1] = name
        count += 1

df = pd.DataFrame(data)
df.rename(columns={0: "relative_path", 1: "classID"}, inplace=True)
myds = SoundDS(df, directory)

# Random split of 80:20 between training and validation
num_items = len(myds)
num_train = round(num_items * 1)
num_val = num_items - num_train
train_ds,_ = random_split(myds, [num_train, num_val])

directory = "/Users/rch/Documents/RICE UNIVERSITY/JUNIOR SPRING/ELEC 378␣
 ↪Machine Learning/ELEC 378 Project/
 ↪elec-378-sp2023-speech-emotion-classification/test/test"

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    # checking if it is a file

    if os.path.isfile(f):
        name = filename[:len(filename)-7]
        data[count][0] = "/"+filename
        data[count][1] = name
        count += 1

df = pd.DataFrame(data)
df.rename(columns={0: "relative_path", 1: "classID"}, inplace=True)
valds = SoundDS(df, directory)


##https://towardsdatascience.com/
 ↪audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5
##all above code is heavily influenced from the resource above
```

[ ]:
```
##Training data
X = []
y = []

for i in range(len(train_ds)):
    sample = train_ds[i]
    X.append(sample[0])
    y.append(sample[1])
```

```
X_train = np.concatenate([spectrogram.flatten().reshape(1, -1) for spectrogram␣
 ↪in X])

label_to_number = {}
for label in set(y):
    label_to_number[label] = len(label_to_number)

# Map each label to its corresponding number
y_train = [label_to_number[label] for label in y]
```

[ ]:
```
##Testing data
Xval = []
yval = []
for i in range(len(val_ds)):
    sample = val_ds[i]
    Xval.append(sample[0])
    yval.append(sample[1])

X_test = np.concatenate([spectrogram.flatten().reshape(1, -1) for spectrogram␣
 ↪in Xval])

label_to_number = {}
for label in set(yval):
    label_to_number[label] = len(label_to_number)

# Map each label to its corresponding number
y_test = [label_to_number[label] for label in yval]
```

[ ]:
```
import numpy as np
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

#training
clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
clf.fit(X_train, y_train)
```

[ ]:
```
from sklearn.metrics import accuracy_score
# Predict on validation data
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

[ ]:
```
#Convert to kaggle upload
number_to_label = {v: k for k, v in label_to_number.items()}

# Map each number back to its corresponding label
y_kaggle = [number_to_label[number]b for number in y_test]
```

[ ]:
```

```

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