Chem277B: Machine Learning Algorithms

Homework assignment #5: Regression

In [4]:
```python
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split, KFold
```

1. Baye’s Theorem.

(a) From the given data, the categories of testing results and their probabilities within proportion have been summarized in the table below:

Category	Probability within Proportion	Kidney Disease Positive	Marker	Proportion
P[+|M]	0.95	$+$	$+$	0.95 * 0.01
P[-|M]	(1-0.95)	$-$	$+$	(1-0.95) * 0.01
P[+|not M]	(1-0.95)	$+$	$-$	(1-0.95) * 0.99
P[-|not M]	0.95	$-$	$-$	0.95 * 0.99

Hence the quantities for the questions are:

(a1) P[-|M] = (1-0.95) = 0.05 within its proportion, the absolute probability is 0.05 * 0.01 = 0.0005

(a2) P[+|not M] = (1-0.95) = 0.05 within its proportion, the absolute probability is 0.05 * 0.99 = 0.0495

(a3) P[not M] = (1-0.95) *0.99 + 0.95* 0.99 = 0.99, or 1 - 0.01 = 0.99

(b) Using the Baye's Theorem, we try to differentiate between positive marker + positive test and positive marker + negative test. Hence the calculation is defined as:

$$P[M|+] = \frac{P[+|M] * P[M]}{P[+|M] * P[M] + P[+|notM] * P[notM]}$$

$$= \frac{0.95 \times 0.01}{(0.95 \times 0.01) + (0.05 \times 0.99)} = 0.161$$

Hence the chance of testing positive and actually have the marker is 16.1%. It warrants additional testing to confirm.

(c) When P[M] = 0.10, the categories and probabilities become the following:

Category	Probability within Proportion	Kidney Disease Positive	Marker	Proportion
P[+|M]	0.95	$+$	$+$	0.95 * 0.10
P[-|M]	(1-0.95)	$-$	$+$	(1-0.95) * 0.10
P[+|not M]	(1-0.95)	$+$	$-$	(1-0.95) * 0.90
P[-|not M]	0.95	$-$	$-$	0.95 * 0.90

Hence with the new frequency, the individual who test positive actually has the marker is:

$$P[M|+]' = \frac{P[+|M]' * P[M]'}{P[+|M]' * P[M]' + P[+|notM]' * P[notM]'}$$

$$= \frac{0.95 \times 0.1}{(0.95 \times 0.1) + (0.05 \times 0.9)} = 0.679$$

2. Gaussian Naive Bayes.

(a) The finished codes are shown below.

I chose Gaussian distribution because it's a widely used normal probability distributions in statistics, data analysis and visualization.

The Gaussian distribution has a bell curve with mean and standard deviation, hence suitable for modeling many real-world phenomena.

With the finished function, I calculated that a wine from cultivar 1 has a 23.24% probability of containinh 13% Alcohol.

In [5]:
```
class NaiveBayesClassifier():
    def __init__(self):
        self.type_indices={}    # store the indices of wines that belong to each cultivar as a boolean array of length
        self.type_stats={}      # store the mean and std of each cultivar
        self.ndata = 0
```

```python
        self.trained=False

    @staticmethod
    def gaussian(x,mean,std):
        exponent = -(x - mean)**2 / (2 * std**2)

        return (np.exp(exponent) / (np.sqrt(2 * np.pi) * std))

    @staticmethod
    def calculate_statistics(x_values):
        # Returns a list with length of input features. Each element is a tuple, with the input feature's average and s
        n_feats=x_values.shape[1]
        return [(np.average(x_values[:,n]),np.std(x_values[:,n])) for n in range(n_feats)]

    @staticmethod
    def calculate_prob(x_input,stats):
        """Calculate the probability that the input features belong to a specific class(P(X|C)), defined by the statist
        x_input: np.array shape(nfeatures)
        stats: list of tuple [(mean1,std1),(means2,std2),...]
        """
        init_prob = 1
        for i in range(len(x_input)):
            mean, std = stats[i]
            init_prob *= NaiveBayesClassifier.gaussian(x_input[i], mean, std)
        return init_prob

    def fit(self,xs,ys):
        # Train the classifier by calculating the statistics of different features in each class
        self.ndata = len(ys)
        for y in set(ys):
            type_filter= (ys==y)
            self.type_indices[y]=type_filter
            self.type_stats[y]=self.calculate_statistics(xs[type_filter])
        self.trained=True

    def predict(self,xs):
        # Do the prediction by outputing the class that has highest probability
        if len(xs.shape)>1:
            print("Only accepts one sample at a time!")
        if self.trained:
            guess=None
            max_prob=0
            # P(C|X) = P(X|C)*P(C) / sum_i(P(X|C_i)*P(C_i)) (deniminator for normalization only, can be ignored)
            for y_type in self.type_stats:
                pre = sum(self.type_indices[y_type]) / self.ndata
                prob= self.calculate_prob(xs, self.type_stats[y_type]) * pre
                if prob>max_prob:
                    max_prob=prob
                    guess=y_type
            return guess
```

```python
        else:
            print("Please train the classifier first!")

def calculate_accuracy(model,xs,ys):
    y_pred=np.zeros_like(ys)
    for idx,x in enumerate(xs):
        y_pred[idx]=model.predict(x)
    return np.sum(ys==y_pred)/len(ys)
```

In [6]:

```python
# Import wines.csv
wines = pd.read_csv('wines.csv')
wines.head()
```

Out[6]:

	Alcohol %	Malic Acid	Ash	Alkalinity	Mg	Phenols	Flavanoids	Phenols.1	Proantho-cyanins	Color intensity	Hue	OD280 315	Proline	Start assignment	ranking
0	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065	1	1
1	13.24	2.59	2.87	21.0	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735	1	1
2	14.83	1.64	2.17	14.0	97	2.8	2.98	0.29	1.98	5.20	1.08	2.85	1045	1	1
3	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	5.00	1.17	2.82	1280	1	1
4	13.75	1.73	2.41	16.0	89	2.6	2.76	0.29	1.81	5.60	1.15	2.90	1320	1	1

In [7]:

```python
# Define the instance from the Naive Bayes Classifier
nbc = NaiveBayesClassifier()

# Fit the Naive Bayes Classifier
nbc.fit(wines.loc[:, 'Alcohol %':'Proline'].values, wines.loc[:, 'ranking'].values)

# First get the stats for cultivar 1
type_stats = nbc.type_stats[1]

# Then calculate the probability of alcohol% = 13
probability = nbc.gaussian(13, type_stats[0][0], type_stats[0][1])

print(f"A wine from cultivar 1 has a {round(probability*100, 2)}% probability of containinh 13% Alcohol")
```

```
A wine from cultivar 1 has a 23.24% probability of containinh 13% Alcohol
```

(b) After 3-fold training, I can achieve close to 100% accuracy in very short term. The Naive Baye's method performs much better and much faster than the simulated annealing method.

In [8]:

```python
# First normalize the wines dataframe
wines_norm = wines.loc[:, 'Alcohol %':'Proline']
wines_norm = (wines_norm - np.mean(wines_norm, axis=0)) / np.std(wines_norm, axis=0)
```

```
wines_norm = wines_norm.merge(wines[['Start assignment','ranking']], left_index=True, right_index=True)
wines_norm.head()
```

Out[8]:

	Alcohol %	Malic Acid	Ash	Alkalinity	Mg	Phenols	Flavanoids	Phenols.1	Proantho-cyanins	Color intensity	Hue	OD280 315	Proline
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034819	-0.659563	1.224884	0.251717	0.362177	1.847920	1.01300
1	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.663351	0.226796	0.401404	-0.319276	0.362177	0.449601	-0.03787
2	2.259772	-0.625086	-0.718336	-1.650049	-0.192495	0.808997	0.954502	-0.578985	0.681738	0.061386	0.537671	0.336606	0.94931
3	1.382733	-0.768712	-0.170035	-0.809251	-0.332922	-0.152402	0.402320	-0.820719	-0.036617	-0.025128	0.932531	0.294232	1.69767
4	0.925685	-0.544297	0.158946	-1.049479	-0.754202	0.488531	0.733629	-0.578985	0.383884	0.234414	0.844785	0.407228	1.82505

In [9]:

```
# Divide the normalized wines data into 3-fold training and testing groups
# and use 2/3 training and 1/3 testing for the three divisions
kf = KFold(n_splits=3, shuffle=True)
xs = wines.loc[:, 'Alcohol %':'Proline'].values
ys = wines.loc[:, 'ranking'].values
nbc = NaiveBayesClassifier()
accuracy = []

for train_index, test_index in kf.split(xs):
    x_train, x_test = xs[train_index], xs[test_index]
    y_train, y_test = ys[train_index], ys[test_index]

    # train the classifier
    nbc.fit(x_train,y_train)
    accuracy.append(calculate_accuracy(nbc,x_test,y_test))
    print(f'Accuracy: {calculate_accuracy(nbc,x_test,y_test)}')
print(f'Average accuracy after 3-fold training is {np.array(accuracy).mean()}')
```

```
Accuracy: 0.95
Accuracy: 0.9661016949152542
Accuracy: 1.0
Average accuracy after 3-fold training is 0.9720338983050847
```

3. Softmax and Cross Entropy Loss.

(a) I did one PyTorch model without softmax and one PyTorch model with softmax. The output without softmax is a cluster of large positive or negative values. The output with softmax is more like probabilities that sum up to 1.

In [10]:

```
# First convert the features and labels to PyTorch tensors
pytorch_features = torch.tensor(wines.loc[:, 'Alcohol %':'Proline'].values , dtype=torch.float32)
pytorch_labels = torch.tensor(wines.loc[:, 'ranking'].values, dtype=torch.int64)
```

```
# Then define a pytorch model without softmax
model_no_softmax = nn.Sequential(
    nn.Linear(pytorch_features.shape[1], len(np.unique(pytorch_labels)))
)

# Then pass the data through the model once without backpropagation
outputs_no_softmax = model_no_softmax(pytorch_features)

# Finally print out the outputs_no_softmax
print(outputs_no_softmax)
```

```
tensor([[ -95.7595,  -80.1433, -271.1181],
        [ -70.3680,  -55.5473, -186.5312],
        [ -89.5526,  -77.3873, -266.7516],
        [-105.8480,  -94.7881, -328.4174],
        [-107.6909,  -97.2809, -338.8018],
        [-109.6877,  -95.2311, -327.6151],
        [ -75.2029,  -59.1946, -197.1850],
        [ -86.9012,  -75.8067, -259.8968],
        [ -74.8449,  -63.2743, -215.8418],
        [-105.9978,  -95.4717, -329.8702],
        [ -88.4649,  -76.9907, -264.4617],
        [-108.5639,  -93.0019, -315.7316],
        [ -79.9209,  -66.1011, -224.1377],
        [ -93.8444,  -82.1973, -283.1660],
        [ -93.6766,  -81.3249, -279.8405],
        [ -79.4933,  -66.3302, -225.2316],
        [ -91.1583,  -78.3610, -270.7823],
        [-104.6139,  -93.3383, -324.4534],
        [-102.0187,  -88.3862, -303.7653],
        [-108.2847,  -95.1919, -328.8627],
        [ -64.1282,  -51.7691, -172.8694],
        [ -43.1073,  -31.5106, -105.4305],
        [ -47.6285,  -37.0139, -126.4444],
        [ -77.7044,  -65.4711, -222.4468],
        [ -52.6122,  -38.7805, -125.9749],
        [ -56.5051,  -47.1741, -161.2987],
        [ -51.6823,  -39.4758, -131.3857],
        [ -47.4287,  -34.8122, -113.2047],
        [ -60.9081,  -50.8997, -173.8416],
        [ -54.7720,  -46.7230, -160.1270],
        [ -92.5969,  -72.4996, -237.2635],
        [ -65.2154,  -48.7314, -158.0107],
        [ -42.2817,  -30.2529, -101.8373],
        [ -65.3198,  -53.5460, -180.6982],
        [ -42.4807,  -31.7434, -104.5706],
        [ -56.0758,  -42.3140, -141.3992],
        [ -44.3542,  -33.0702, -109.6971],
        [ -39.9226,  -29.4330,  -97.2258],
        [ -41.9280,  -30.9954, -102.9244],
        [ -58.9969,  -47.1536, -158.7496],
        [ -39.0347,  -27.9414,  -95.3846],
        [ -37.5140,  -26.1886,  -85.9839],
        [ -52.5705,  -42.9040, -147.9189],
        [ -53.9974,  -41.8098, -142.2261],
        [ -58.1301,  -44.9303, -152.1356],
        [ -53.6420,  -42.8173, -147.0570],
        [ -56.4847,  -44.4313, -149.7079],
        [ -50.9161,  -39.1205, -131.8629],
        [ -40.2738,  -30.0874, -104.7068],
        [ -55.1265,  -40.5552, -138.4230],
```

```
[ -47.2168,  -32.2037, -106.1813],
[ -46.2232,  -34.4733, -120.5233],
[ -64.0021,  -50.8870, -172.5027],
[ -64.9461,  -50.8716, -175.9956],
[ -60.6507,  -49.6607, -174.0285],
[ -59.6397,  -48.0405, -167.2719],
[ -65.4994,  -53.9129, -188.2240],
[ -67.7058,  -55.2292, -190.8036],
[ -90.5299,  -78.4755, -268.1708],
[-122.7202, -109.5233, -379.0020],
[-120.7422, -107.4394, -371.4403],
[ -89.3664,  -77.3397, -266.8197],
[ -96.4089,  -84.4749, -293.8468],
[ -98.1854,  -84.0856, -288.8749],
[ -69.9950,  -57.3209, -195.9571],
[ -78.1801,  -63.0056, -211.5188],
[ -81.6687,  -68.4600, -233.4466],
[-106.9510,  -94.9313, -329.7242],
[ -86.7286,  -73.9335, -252.8878],
[ -95.0400,  -81.9118, -280.2427],
[ -87.9826,  -76.1874, -260.9209],
[ -73.8215,  -57.3670, -191.7273],
[ -63.7275,  -50.7680, -172.3642],
[ -93.3901,  -80.3729, -275.9935],
[ -85.6678,  -73.0847, -251.1188],
[ -96.1325,  -84.3870, -293.8909],
[-115.7293, -102.2474, -352.2450],
[ -93.7798,  -79.4673, -270.3569],
[ -50.8155,  -39.9812, -131.2031],
[ -59.1775,  -47.5665, -160.0092],
[ -41.3422,  -27.7701,  -88.7323],
[ -47.7472,  -38.1787, -129.4783],
[ -41.9844,  -30.6733, -103.5136],
[ -46.3154,  -35.5393, -119.7110],
[ -41.0043,  -29.5392,  -98.1409],
[ -75.1354,  -57.8211, -189.2939],
[ -32.9722,  -21.7195,  -69.0966],
[ -63.8454,  -53.1075, -182.0839],
[ -52.9921,  -42.5161, -143.5636],
[ -46.4499,  -36.0199, -121.7188],
[ -47.2032,  -36.9527, -125.7045],
[ -33.8721,  -22.2086,  -71.9166],
[ -53.4146,  -42.6067, -142.7589],
[ -45.8384,  -33.6798, -110.2899],
[ -60.5284,  -50.5022, -171.3946],
[ -36.2448,  -24.2678,  -77.3887],
[ -63.0564,  -51.1417, -173.1173],
[ -58.8220,  -45.8668, -154.3202],
[ -48.1790,  -37.8065, -125.5902],
[ -38.0483,  -27.7569,  -93.3553],
```

```
[ -53.0801,  -42.4808, -143.2783],
[ -37.9239,  -26.5384,  -88.1518],
[ -46.6358,  -35.1526, -118.4888],
[ -52.9951,  -39.7576, -133.9356],
[ -63.7739,  -53.1868, -183.9596],
[ -56.6567,  -45.0805, -152.4304],
[ -68.2721,  -57.9437, -199.1737],
[ -52.0392,  -40.6550, -139.2529],
[ -73.9494,  -61.9691, -211.7534],
[ -58.9663,  -47.5144, -164.8894],
[ -76.0654,  -64.9467, -225.1627],
[ -49.0072,  -37.9552, -131.3719],
[ -55.8980,  -42.9190, -144.2426],
[ -63.5942,  -50.5809, -171.2624],
[ -48.9179,  -37.9178, -131.6362],
[ -50.1381,  -38.2544, -128.8955],
[ -45.1788,  -34.1721, -118.3403],
[ -99.8508,  -87.4786, -303.4079],
[-106.7419,  -95.4956, -330.5671],
[-110.9604,  -97.0445, -331.5586],
[-123.7389, -111.5977, -387.5805],
[-126.2076, -113.7205, -396.4001],
[-110.5025,  -97.1671, -335.5422],
[-136.3220, -123.7616, -431.3432],
[ -77.9980,  -63.3852, -214.6196],
[ -89.3988,  -77.4158, -264.6501],
[ -99.4892,  -88.5956, -306.3633],
[-124.1193, -112.0037, -388.9803],
[ -80.6667,  -68.5250, -234.9812],
[ -74.7416,  -59.7568, -201.2769],
[ -86.8939,  -76.3856, -265.1540],
[ -91.2729,  -78.7977, -271.8964],
[-106.1120,  -92.9291, -322.3038],
[ -97.4838,  -83.4479, -286.0666],
[ -87.3043,  -72.6440, -246.6375],
[-106.1832,  -94.0076, -325.4984],
[ -47.0953,  -34.2192, -112.8618],
[ -63.4865,  -50.6691, -172.2142],
[ -70.5792,  -57.3044, -190.6053],
[ -75.0449,  -56.4108, -180.7851],
[ -91.4002,  -75.2161, -251.5265],
[ -78.5728,  -66.8261, -226.2947],
[ -45.2534,  -32.7163, -107.5243],
[ -47.8508,  -35.0571, -116.7137],
[ -48.6012,  -37.7607, -130.6048],
[ -48.8070,  -37.4591, -125.7198],
[ -44.7871,  -33.9485, -113.9771],
[ -39.4460,  -26.7756,  -85.8938],
[ -43.6103,  -32.3360, -107.6418],
[ -60.5434,  -49.1194, -167.6660],
```

```
        [ -35.6747,  -24.3274,  -78.7909],
        [ -47.1800,  -36.5285, -123.9038],
        [ -36.2320,  -24.9942,  -81.3378],
        [ -41.0823,  -27.3728,  -86.0754],
        [ -51.0255,  -35.5192, -117.1997],
        [ -40.7060,  -27.9433,  -91.7698],
        [ -38.8315,  -27.9552,  -95.4119],
        [ -39.5275,  -28.7804,  -95.1505],
        [ -63.7631,  -48.4041, -159.2106],
        [ -58.8792,  -48.3266, -165.1284],
        [ -63.0256,  -51.4977, -176.6051],
        [ -49.7291,  -37.9999, -130.5413],
        [ -76.9455,  -63.4678, -217.5637],
        [ -55.9494,  -45.4670, -158.8678],
        [ -53.3975,  -37.4375, -125.4094],
        [ -50.0229,  -35.3705, -120.1836],
        [ -61.7129,  -49.0515, -170.6138],
        [ -60.6729,  -47.8728, -161.9878],
        [ -64.1084,  -52.6943, -184.3833],
        [ -60.4054,  -47.5032, -166.5674],
        [ -56.3295,  -44.6045, -156.6009],
        [ -56.0662,  -44.7320, -155.9557],
        [ -68.2930,  -55.3038, -190.7732],
        [ -60.3732,  -46.4871, -159.5021],
        [ -76.4289,  -61.6658, -211.7994],
        [ -77.3098,  -62.5640, -213.1622],
        [ -52.7916,  -40.5824, -141.4767],
        [ -48.2669,  -38.1544, -129.4740]], grad_fn=<AddmmBackward0>)
```

In [11]:

```
# Second is to define a pytorch model with softmax
model_softmax = nn.Sequential(
    nn.Linear(pytorch_features.shape[1], len(np.unique(pytorch_labels))),
    nn.Softmax(dim=1)
)

# Then pass the data through the model once without backpropagation
outputs_softmax = model_softmax(pytorch_features)

# Finally print out the outputs_softmax
print(outputs_softmax)
```

```
tensor([[1.8472e-04, 0.0000e+00, 9.9982e-01],
        [8.6887e-05, 0.0000e+00, 9.9991e-01],
        [3.2321e-02, 0.0000e+00, 9.6768e-01],
        [5.2833e-01, 0.0000e+00, 4.7167e-01],
        [8.3785e-01, 0.0000e+00, 1.6215e-01],
        [1.6370e-02, 0.0000e+00, 9.8363e-01],
        [8.5346e-06, 0.0000e+00, 9.9999e-01],
        [7.0744e-02, 0.0000e+00, 9.2926e-01],
        [1.5047e-02, 0.0000e+00, 9.8495e-01],
        [6.7305e-01, 0.0000e+00, 3.2695e-01],
        [5.4550e-02, 0.0000e+00, 9.4545e-01],
        [8.9474e-04, 0.0000e+00, 9.9911e-01],
        [4.5460e-04, 0.0000e+00, 9.9955e-01],
        [1.0055e-01, 0.0000e+00, 8.9945e-01],
        [3.4522e-02, 0.0000e+00, 9.6548e-01],
        [1.5719e-03, 0.0000e+00, 9.9843e-01],
        [2.9096e-02, 0.0000e+00, 9.7090e-01],
        [5.5085e-01, 0.0000e+00, 4.4915e-01],
        [1.0845e-02, 0.0000e+00, 9.8915e-01],
        [7.0314e-02, 0.0000e+00, 9.2969e-01],
        [2.5039e-04, 0.0000e+00, 9.9975e-01],
        [3.3493e-04, 8.6922e-32, 9.9967e-01],
        [2.2942e-03, 4.1237e-35, 9.9771e-01],
        [4.4389e-03, 0.0000e+00, 9.9556e-01],
        [1.5888e-05, 8.3079e-39, 9.9998e-01],
        [5.6468e-02, 6.0180e-41, 9.4353e-01],
        [1.9038e-04, 8.1258e-38, 9.9981e-01],
        [4.3109e-05, 1.4603e-34, 9.9996e-01],
        [3.2355e-02, 1.6816e-44, 9.6764e-01],
        [2.4256e-01, 1.1164e-39, 7.5744e-01],
        [1.2097e-07, 0.0000e+00, 1.0000e+00],
        [9.9605e-07, 0.0000e+00, 1.0000e+00],
        [7.7131e-04, 8.2828e-31, 9.9923e-01],
        [2.4262e-03, 0.0000e+00, 9.9757e-01],
        [7.4628e-04, 5.7368e-31, 9.9925e-01],
        [2.1208e-04, 1.6608e-41, 9.9979e-01],
        [3.6215e-04, 1.9108e-32, 9.9964e-01],
        [9.2764e-04, 4.0758e-29, 9.9907e-01],
        [1.1742e-03, 1.7986e-30, 9.9883e-01],
        [1.6996e-03, 2.2841e-43, 9.9830e-01],
        [5.7814e-03, 8.8424e-29, 9.9422e-01],
        [4.8042e-04, 4.2283e-27, 9.9952e-01],
        [7.3373e-02, 1.3141e-38, 9.2663e-01],
        [5.1348e-04, 3.1990e-40, 9.9949e-01],
        [2.4543e-04, 2.6204e-43, 9.9975e-01],
        [4.3102e-03, 3.9934e-40, 9.9569e-01],
        [6.9607e-04, 4.0834e-42, 9.9930e-01],
        [6.6453e-04, 9.5720e-38, 9.9934e-01],
        [6.4963e-03, 3.2815e-30, 9.9350e-01],
        [2.3431e-05, 2.5714e-42, 9.9998e-01],
```

```
[2.6486e-06, 8.5243e-36, 1.0000e+00],
[8.1758e-04, 1.2263e-35, 9.9918e-01],
[2.9942e-04, 0.0000e+00, 9.9970e-01],
[1.7692e-04, 0.0000e+00, 9.9982e-01],
[6.4611e-03, 0.0000e+00, 9.9354e-01],
[1.8833e-03, 2.8026e-45, 9.9812e-01],
[1.7553e-02, 0.0000e+00, 9.8245e-01],
[3.4296e-03, 0.0000e+00, 9.9657e-01],
[1.5172e-02, 0.0000e+00, 9.8483e-01],
[2.6207e-01, 0.0000e+00, 7.3793e-01],
[1.8598e-01, 0.0000e+00, 8.1402e-01],
[2.6553e-02, 0.0000e+00, 9.7345e-01],
[1.8363e-01, 0.0000e+00, 8.1637e-01],
[6.2811e-03, 0.0000e+00, 9.9372e-01],
[3.5683e-03, 0.0000e+00, 9.9643e-01],
[1.3672e-04, 0.0000e+00, 9.9986e-01],
[3.2982e-03, 0.0000e+00, 9.9670e-01],
[4.8374e-01, 0.0000e+00, 5.1626e-01],
[1.0531e-02, 0.0000e+00, 9.8947e-01],
[1.3018e-02, 0.0000e+00, 9.8698e-01],
[3.0631e-02, 0.0000e+00, 9.6937e-01],
[8.9407e-06, 0.0000e+00, 9.9999e-01],
[1.0294e-03, 0.0000e+00, 9.9897e-01],
[2.8042e-02, 0.0000e+00, 9.7196e-01],
[1.0847e-02, 0.0000e+00, 9.8915e-01],
[1.5644e-01, 0.0000e+00, 8.4356e-01],
[7.3083e-02, 0.0000e+00, 9.2692e-01],
[1.0360e-03, 0.0000e+00, 9.9896e-01],
[2.3152e-04, 5.1354e-37, 9.9977e-01],
[9.4091e-04, 3.1950e-43, 9.9906e-01],
[6.8082e-06, 2.6367e-30, 9.9999e-01],
[6.1928e-03, 2.9662e-35, 9.9381e-01],
[1.3000e-03, 1.9358e-30, 9.9870e-01],
[1.8951e-03, 7.5669e-34, 9.9810e-01],
[1.4442e-04, 3.1113e-30, 9.9986e-01],
[5.9983e-07, 0.0000e+00, 1.0000e+00],
[9.3230e-05, 5.6534e-24, 9.9991e-01],
[1.7623e-02, 0.0000e+00, 9.8238e-01],
[7.4233e-03, 1.0843e-38, 9.9258e-01],
[2.8934e-03, 8.7052e-34, 9.9711e-01],
[4.8021e-03, 2.4598e-34, 9.9520e-01],
[1.5753e-04, 8.9675e-25, 9.9984e-01],
[1.7759e-03, 7.4345e-39, 9.9822e-01],
[1.5810e-04, 1.1561e-33, 9.9984e-01],
[2.1254e-02, 3.3631e-44, 9.7875e-01],
[4.8974e-05, 1.6064e-26, 9.9995e-01],
[2.6350e-03, 0.0000e+00, 9.9736e-01],
[2.9868e-04, 2.4242e-43, 9.9970e-01],
[2.2138e-03, 4.6780e-35, 9.9779e-01],
[1.3132e-03, 4.0459e-28, 9.9869e-01],
```

```
[8.8049e-03, 1.2341e-38, 9.9120e-01],
[3.5247e-04, 8.3555e-28, 9.9965e-01],
[1.9457e-03, 4.4292e-34, 9.9805e-01],
[8.3375e-05, 1.4907e-39, 9.9992e-01],
[4.5547e-02, 0.0000e+00, 9.5445e-01],
[1.0561e-03, 4.6341e-42, 9.9894e-01],
[2.2846e-02, 0.0000e+00, 9.7715e-01],
[3.7930e-03, 9.0769e-39, 9.9621e-01],
[8.8324e-03, 0.0000e+00, 9.9117e-01],
[3.3619e-03, 9.8091e-45, 9.9664e-01],
[5.0112e-02, 0.0000e+00, 9.4989e-01],
[2.2728e-03, 1.8445e-37, 9.9773e-01],
[1.5570e-04, 6.7767e-42, 9.9984e-01],
[3.1643e-04, 0.0000e+00, 9.9968e-01],
[4.6194e-03, 5.8100e-37, 9.9538e-01],
[2.6419e-04, 1.0546e-37, 9.9974e-01],
[4.6899e-04, 4.6372e-35, 9.9953e-01],
[1.8609e-01, 0.0000e+00, 8.1391e-01],
[4.5709e-01, 0.0000e+00, 5.4291e-01],
[1.5110e-02, 0.0000e+00, 9.8489e-01],
[8.1872e-01, 0.0000e+00, 1.8128e-01],
[7.2561e-01, 0.0000e+00, 2.7439e-01],
[5.2149e-02, 0.0000e+00, 9.4785e-01],
[8.7183e-01, 0.0000e+00, 1.2817e-01],
[2.1140e-04, 0.0000e+00, 9.9979e-01],
[3.1592e-02, 0.0000e+00, 9.6841e-01],
[3.8618e-01, 0.0000e+00, 6.1382e-01],
[7.4794e-01, 0.0000e+00, 2.5206e-01],
[1.5709e-02, 0.0000e+00, 9.8429e-01],
[5.5684e-05, 0.0000e+00, 9.9994e-01],
[5.0470e-01, 0.0000e+00, 4.9530e-01],
[6.7571e-02, 0.0000e+00, 9.3243e-01],
[5.9864e-02, 0.0000e+00, 9.4014e-01],
[5.7726e-03, 0.0000e+00, 9.9423e-01],
[3.3818e-04, 0.0000e+00, 9.9966e-01],
[2.2123e-01, 0.0000e+00, 7.7876e-01],
[2.0263e-05, 3.3096e-35, 9.9998e-01],
[5.7057e-04, 0.0000e+00, 9.9943e-01],
[1.1257e-04, 0.0000e+00, 9.9989e-01],
[4.5559e-08, 0.0000e+00, 1.0000e+00],
[1.3600e-04, 0.0000e+00, 9.9986e-01],
[8.1701e-03, 0.0000e+00, 9.9183e-01],
[4.5270e-05, 1.6220e-33, 9.9995e-01],
[4.1139e-04, 4.2205e-35, 9.9959e-01],
[8.6062e-03, 4.4454e-36, 9.9139e-01],
[1.2693e-03, 1.7133e-35, 9.9873e-01],
[1.0492e-03, 5.8773e-33, 9.9895e-01],
[2.3743e-05, 3.1789e-29, 9.9998e-01],
[4.4239e-04, 8.6426e-32, 9.9956e-01],
[4.2599e-03, 2.5223e-44, 9.9574e-01],
```

```
        [2.3700e-04, 4.0029e-26, 9.9976e-01],
        [3.2289e-03, 2.0931e-34, 9.9677e-01],
        [3.2947e-04, 2.6414e-26, 9.9967e-01],
        [7.8195e-06, 6.6988e-30, 9.9999e-01],
        [8.5738e-06, 5.8590e-38, 9.9999e-01],
        [2.4556e-04, 6.8928e-30, 9.9975e-01],
        [4.8841e-03, 1.1604e-28, 9.9512e-01],
        [9.3713e-04, 7.1952e-29, 9.9906e-01],
        [3.0651e-06, 0.0000e+00, 1.0000e+00],
        [3.4366e-03, 1.0229e-43, 9.9656e-01],
        [1.9740e-03, 0.0000e+00, 9.9803e-01],
        [3.2131e-03, 1.8661e-37, 9.9679e-01],
        [7.4541e-04, 0.0000e+00, 9.9925e-01],
        [1.7708e-02, 1.6339e-42, 9.8229e-01],
        [2.0427e-06, 1.0171e-40, 1.0000e+00],
        [5.5406e-06, 6.4215e-39, 9.9999e-01],
        [6.5805e-04, 0.0000e+00, 9.9934e-01],
        [1.2093e-04, 1.4013e-45, 9.9988e-01],
        [2.2252e-02, 0.0000e+00, 9.7775e-01],
        [5.1209e-04, 0.0000e+00, 9.9949e-01],
        [1.3205e-03, 1.6816e-43, 9.9868e-01],
        [2.4159e-03, 9.2626e-43, 9.9758e-01],
        [9.8771e-04, 0.0000e+00, 9.9901e-01],
        [1.9941e-04, 0.0000e+00, 9.9980e-01],
        [1.4798e-04, 0.0000e+00, 9.9985e-01],
        [7.1400e-05, 0.0000e+00, 9.9993e-01],
        [1.2024e-03, 1.9425e-40, 9.9880e-01],
        [5.3701e-03, 2.5709e-35, 9.9463e-01]], grad_fn=<SoftmaxBackward0>)
```

(b) I unfortunately always encounter an error of dead kernel when trying to evaluate the train_and_val function. I don't really understand the root cause of the issue.

In [14]:

```
def train_and_val(model,train_X,train_y,epochs,draw_curve=True):
    """
    Parameters
    --------------
    model: a PyTorch model
    train_X: np.array shape(ndata,nfeatures)
    train_y: np.array shape(ndata)
    epochs: int
    draw_curve: bool
    """

    ### Define your loss function, optimizer. Convert data to torch tensor ###
    loss = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr = 0.001)

    train_y -= 1
    Xs = torch.tensor(train_X).float()
```

```
    ys = torch.tensor(train_y).long()

    # Define Kfolds
    kf = KFold(n_splits = 3,shuffle = True)
    for train_index, test_index in kf.split(Xs):
        train_X, test_X = Xs[train_index], Xs[test_index]
        train_y, test_y = ys[train_index], ys[test_index]

    ### Split training examples further into training and validation ###
    train_X,val_X,train_y,val_y = train_test_split(train_X,train_y, test_size = 0.20)
    val_array=[]
    lowest_val_loss = np.inf

    for i in range(epochs):
        ### Compute the loss and do backpropagation ###
        optimizer.zero_grad()
        train_out = model(train_X)
        train_loss = loss(train_out, train_y)
        train_loss.backward()
        optimizer.step()

        ### compute validation loss and keep track of the lowest val loss ###
        # compute validation loss
        val_out = model(val_X)
        val_loss = loss(val_out, val_y)

        # append val loss to val_array
        val_array.append(val_loss.item())

        # keep track of the lowest val loss
        if val_loss < lowest_val_loss:
            lowest_val_loss = val_loss
            torch.save(model.state_dict(), 'model.pt')

    # The final number of epochs is when the minimum error in validation set occurs

    final_epochs = np.argmin(val_array) + 1
    print("Number of epochs with lowest validation:",final_epochs)
    ### Recover the model weight ###
    model.load_state_dict(torch.load('model.pt'))
    model.eval()

    ### Plot the validation loss curve ###

    if draw_curve:
        plt.figure()
        plt.plot(np.arange(len(val_array))+1,val_array,label='Validation loss')
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
```

In [ ]:
```
train_and_val(model_softmax,pytorch_features,pytorch_labels,1000,draw_curve=True)
```

In [ ]: