Problem 4: Gaussian Naive Bayes

1. Given a data set with m continuous features, what is the log-likelihood of the Gaussian NB model? Compute the MLE for each of the model parameters.

In [1]:
```
import pandas as pd
import numpy as np
import math
```

In [2]:
```
sonar_train = pd.read_csv('sonar_train.data', header=None)
sonar_test = pd.read_csv('sonar_test.data', header=None)
sonar_valid = pd.read_csv('sonar_valid.data', header=None)

sonar_train.loc[sonar_train[60] == 2, 60] = -1
sonar_test.loc[sonar_test[60] == 2, 60] = -1
sonar_valid.loc[sonar_valid[60] == 2, 60] = -1

def split_data(data):
    return data.iloc[:, :60].to_numpy(), data.iloc[:, 60:].to_numpy()

X_train, y_train = split_data(sonar_train)
X_validation, y_validation = split_data(sonar_valid)
X_test, y_test = split_data(sonar_test)
```

In [3]:
```
labels = {1, -1}
m, n = X_train.shape
data_stat = {}
for label in labels:
    mask = (y_train == label).ravel()
    y_prob = np.sum(mask.astype(np.float32)) / m
    X_label = X_train[mask]
    mean = X_label.mean(axis=0)
    var = X_label.var(axis=0)
    data_stat[label] = (y_prob, mean, var)
```

In [40]:
```
mask_1 = (y_train == 1).ravel()
print("y = 1")
print("Mean MLE\n", X_train[mask_1].mean(axis=0))
print("Variance MLE\n", (np.sum(X_train[mask_1], axis=0) - X_train[mask_1].mean(a
print("Unbiased Variance MLE\n", (np.sum(X_train[mask_1], axis=0) - X_train[mask_
```

```
y = 1
Mean MLE
 [0.0222902  0.03148824 0.03790196 0.04584314 0.0689     0.10116275
 0.11923529 0.12568235 0.13812941 0.16653529 0.17673137 0.18921569
 0.20863922 0.2563     0.30578039 0.3820451  0.42764706 0.44002157
 0.4440902  0.47582157 0.54373137 0.57872549 0.60416078 0.65514314
 0.68392745 0.70956667 0.69666275 0.67369216 0.63386471 0.57646078
 0.52397255 0.43650392 0.44531961 0.45542353 0.46760784 0.47771373
 0.44626667 0.36371765 0.31952745 0.33340392 0.29820196 0.26851569
 0.2234451  0.18886863 0.16330588 0.13898039 0.10754118 0.07797647
 0.04480588 0.01862353 0.01237647 0.01037255 0.00930784 0.00888431
 0.00874706 0.00722353 0.00836863 0.00688824 0.00730196 0.00610784]
Variance MLE
 [0.01857516 0.0262402  0.03158497 0.03820261 0.05741667 0.08430229
 0.09936275 0.10473529 0.11510784 0.13877941 0.14727614 0.15767974
 0.17386601 0.21358333 0.25481699 0.31837092 0.35637255 0.36668464
 0.37007516 0.39651797 0.45310948 0.48227124 0.50346732 0.54595261
 0.56993954 0.59130556 0.58055229 0.56141013 0.52822059 0.48038399
 0.43664379 0.36375327 0.37109967 0.37951961 0.3896732  0.39809477
 0.37188889 0.30309804 0.26627288 0.2778366  0.24850163 0.22376307
 0.18620425 0.15739052 0.13608824 0.11581699 0.08961765 0.06498039
 0.03733824 0.01551961 0.01031373 0.00864379 0.00775654 0.00740359
 0.00728922 0.00601961 0.00697386 0.0057402  0.00608497 0.00508987]
Unbiased Variance MLE
 [0.01889    0.02668495 0.03212031 0.03885012 0.05838983 0.08573114
 0.10104686 0.10651047 0.11705882 0.14113161 0.14977235 0.16035228
 0.17681289 0.21720339 0.25913593 0.32376703 0.36241276 0.37289963
 0.37634762 0.40323862 0.4607893  0.49044533 0.51200066 0.55520605
 0.57959953 0.60132768 0.59039216 0.57092556 0.53717348 0.48852609
 0.44404453 0.36991858 0.3773895  0.38595214 0.39627783 0.40484214
 0.37819209 0.30823529 0.27078598 0.2825457  0.25271353 0.22755567
 0.18936025 0.16005816 0.13839482 0.11777999 0.09113659 0.06608175
 0.03797109 0.01578265 0.01048853 0.0087903  0.007888   0.00752908
 0.00741276 0.00612164 0.00709206 0.00583749 0.0061881  0.00517614]
```

In [41]:
```
mask_1 = (y_train == -1).ravel()
print("y = -1")
print("Mean MLE\n", X_train[mask_1].mean(axis=0))
print("Variance MLE\n", (np.sum(X_train[mask_1], axis=0) - X_train[mask_1].mean(a
print("Unbiased Variance MLE\n", (np.sum(X_train[mask_1], axis=0) - X_train[mask_
```

```
y = -1
Mean MLE
 [0.03253962 0.04104528 0.04472075 0.05795472 0.07641132 0.10650755
 0.12542075 0.14785283 0.21642453 0.26446415 0.29930566 0.30189434
 0.31160755 0.3128     0.33173774 0.3757434  0.40329623 0.44553396
 0.52754528 0.59813396 0.65068113 0.66556415 0.67275472 0.68820755
 0.67840566 0.69489245 0.70525472 0.70304906 0.65078113 0.57865094
 0.49120377 0.43141132 0.40898113 0.37285472 0.33809434 0.31356038
 0.30740377 0.30675283 0.30745283 0.28710755 0.25690566 0.27194528
 0.27778302 0.25384717 0.2275283  0.17933208 0.14156981 0.10864717
 0.06152264 0.01993396 0.01978679 0.01554717 0.01090755 0.0108
 0.00902075 0.00840377 0.00738302 0.00874151 0.00805472 0.00666792]
Variance MLE
 [0.02820101 0.03557258 0.03875799 0.05022742 0.06622314 0.09230654
 0.10869799 0.12813912 0.18756792 0.22920226 0.25939824 0.26164176
 0.27005987 0.27109333 0.28750604 0.32564428 0.3495234  0.38612943
 0.45720591 0.51838277 0.56392365 0.57682226 0.58305409 0.59644654
 0.58795157 0.60224013 0.61122075 0.60930918 0.56401031 0.50149748
 0.42570994 0.37388981 0.35445031 0.32314075 0.29301509 0.27175233
 0.2664166  0.26585245 0.26645912 0.24882654 0.22265157 0.23568591
 0.24074528 0.22000088 0.19719119 0.15542113 0.12269384 0.09416088
 0.05331962 0.0172761  0.01714855 0.01347421 0.00945321 0.00936
 0.00781799 0.00728327 0.00639862 0.00757597 0.00698075 0.00577887]
Unbiased Variance MLE
 [0.02867899 0.0361755  0.0394149  0.05107873 0.06734557 0.09387106
 0.11054033 0.13031097 0.19074704 0.23308705 0.26379482 0.26607637
 0.27463716 0.27568814 0.29237902 0.33116367 0.35544752 0.392674
 0.46495516 0.52716892 0.57348168 0.58659891 0.59293636 0.6065558
 0.59791685 0.61244759 0.62158043 0.61963646 0.57356981 0.50999744
 0.43292536 0.38022693 0.36045795 0.32861772 0.29798145 0.2763583
 0.27093214 0.27035843 0.27097538 0.25304394 0.22642533 0.23968059
 0.24482571 0.22372971 0.20053342 0.15805539 0.12477339 0.09575683
 0.05422335 0.01756892 0.01743921 0.01370259 0.00961343 0.00951864
 0.0079505  0.00740672 0.00650707 0.00770438 0.00709907 0.00587681]
```

In [4]:
```
def compute_gaussian(x, mean, var):
    return (1/np.sqrt(2*np.pi*var)) * np.exp(-np.power(x-mean,2)/(2*var))
```

2. Fit a Gaussian NB model to the training data. What is the accuracy of your trained model on the test set?

In [5]:
```python
def predict(X):
    m, n = X.shape
    preds = np.zeros((m, 1))
    for i in range(m):
        max_likely_y = 0
        best_prob = -1
        for y in {1, -1}:
            prob, mean, var = data_stat[y]
            for j in range(len(X[i])):
                gaus = compute_gaussian(X[i][j], mean[j], var[j])
                prob *= gaus
            if prob > best_prob:
                best_prob = prob
                max_likely_y = y
        preds[i] = max_likely_y
    return preds
```

In [6]:
```python
print("Test Accuracy: ", np.mean(predict(X_test) == y_test))
```

```
Test Accuracy:  0.6923076923076923
```

3. What kind of prior might make sense for this model? Explain.

Conjugate priors can be used as a prior

For fixed variance - Normal distribution

For fixed mean - Inverse Gamma distribution

4. Do you think the NB assumption is reasonable here?

Yes, since the attributes are represents the energy within a particular frequency band, integrated over a certain period of time. They might not be dependent on each other. Attribute independence can be assumed.