MSML 621

Assignment 3

For this assignment you will create a modulation classification algorithm that can distinguish between BPSK and QPSK. As part of the assignment, you will also create an RF dataset comprising of BSPK and QPSK signals with varying SNR and samples per symbol. The code provided during Lecture 7 (included on page 2) should give you a huge starting point for this assignment, you'll mainly just have to add some for loops around it, and think about what kind of features/metrics you can use to classify with, that give different values for BSPK vs QPSK. Your classifier does not have to be perfect, but it should at least work at the higher SNRs. Bonus points will be given to the best performing classifier in low SNR. This assignment does not involve your PlutoSDR.

1. Simulate signals based on each modulation scheme's constellation, like we did in Lecture 7. Have the samples per symbol vary between the values **2,4,6,8**. Do not add a random phase rotation for now, i.e., assume the receiver has already synchronized to the signal (this greatly simplifies the problem, to the point where I am confident you can figure out your own features to use for classification between BPSK and QPSK). Vary SNR between **-10 to 20 dB, in 2 dB increments**. Label each piece of data (i.e. each signal snippet) with its modulation scheme and SNR, because we will plot classification accuracy over SNR. You do not need to label the samples per symbol, just make sure it varies for both the training and testing data (we don't want to train a classifier that only works for one value). It is up to you whether you save each signal snippet as a file, or just keep everything in variables in Python, just make sure your dataset includes at least one example of each combination of modulation scheme, SNR, and samples per symbol. You can used pandas if you want, it is not required. The number of samples per signal snippet is up to you, I would recommend at least 100, so that there are many symbols per snippet.
2. Create your own modulation classifier, and test it by plotting accuracy over SNR. I'll leave it up to you to decide how complex you want to get with your classifier; which could range anywhere from you figuring out your own simple calculated metric and threshold, up to training a DNN to do it all for you.
3. Once the classifier above works well, add a random phase rotation (between 0 and 360 degrees) to the signal snippets, make sure the rotation is different for each signal snippet in the dataset. Now make a new classifier that works (assuming the one in part 2 fails for this new dataset). You can add a random phase rotation by multiplying your symbols by $e^{j\theta}$ where θ is the angle in radians to rotate by (make sure to convert degrees to radians!). Code hint: theta_degrees = np.random.uniform(0,360). For this classifier, you might need to use a more advanced feature compared to the first classifier, or multiple features, just try your best! Email me if you have trouble coming up with a working classifier at high SNR.

Submission instructions: Please submit the following items as individual files through Canvas:

1. Entire Python code as .py file, try to clean up the part that performs the classification and any features/metrics you calculate, so I can take a look at what you decided to do
2. Picture/screenshot of classification accuracy over SNR for Part 2
3. Picture/screenshot of classification accuracy over SNR for Part 3

Appendix: Code from Lecture 7

```
import numpy as np
import matplotlib.pyplot as plt

N = 10000 # number of symbols

# Note the samples per symbol has been hardcoded to 8 in this example

# QPSK
I = (np.random.randint(0, 2, N) - 0.5) * 2
Q = (np.random.randint(0, 2, N) - 0.5) * 2 # set this to 0 for BPSK
x = (I + 1j*Q)/np.sqrt(2) #  remove sqrt(2) for BSPK

x = np.repeat(x, 8)

# Create the noise
n = (np.random.randn(N*8) + 1j*np.random.randn(N*8))/np.sqrt(2) # unity power AWGN
n = n*0.1 # scaling the amount of noise (you'll want to change this line)

r = x + n # received signal is the transmitted signal x, plus noise n

print("signal power =", np.var(x))
print("noise power = ", np.var(n))
print("SNR [dB] =", 10*np.log10(np.var(x)/np.var(n)))

# IQ plot
plt.plot(np.real(r),np.imag(r),'.')
plt.axis([-1,1,-1,1])
plt.show()
```