A Quick Primer on Machine Learning in Wireless Communications

Faris B. Mismar

Abstract

This is a first draft of a quick primer on the use of Python (and relevant libraries) to build a wireless communication prototype that supports multiple-input and multiple-output (MIMO) systems with orthogonal frequency division multiplexing (OFDM) in addition to some machine learning use cases. This primer is intended to empower researchers with a means to efficiently create simulations. This draft is aligned with the syllabus of a graduate course we created to be taught in Fall 2022 and we aspire to update this draft occasionally based on feedback from the larger research community.

Index Terms

Python, wireless channels, MIMO, OFDM, machine learning, deep learning.

I. Introduction

There is no shortage of publications in the research area of wireless communication and machine learning. These numerous publications bring about numerous assumptions, configurations, and source codes. These source codes—especially when proprietary or released under restrictive license agreements—can make the idea of reproducibility extremely challenging.

Because of this, and keeping simplicity in mind, we share this draft of a primer to machine learning in wireless communication in addition to the source code related to it. Because the wireless communication system of our choice supports multiple-input and multiple-output (MIMO) systems and orthogonal frequency division multiplexing (OFDM), this can source code can be suitable for prototyping of 4G LTE and 5G systems alike. The source code is available on GitHub [1] and runs on Python 3.

The author is an adjunct professor of electrical and computer engineering at The University of Texas at Dallas (email: fbm090020@utdallas.edu).

II. SYSTEM MODEL

We consider a MIMO system model with N_t transmit antennas at the base station (BS) with OFDM symbols each of energy E_s and a subcarrier spacing of Δf, and N_r receive antennas at the user equipment (UE). This MIMO system can be written as:

$$\mathbf{y} = \mathbf{H}\mathbf{F}\mathbf{x} + \mathbf{n} \tag{1}$$

where $\mathbf{y} \in \mathbb{C}^{N_r}$ is a column vector containing received OFDM symbols from a transmitted column vector $\mathbf{x} \in \mathbb{C}^{N_t}$ such that $\mathbb{E}[\|\mathbf{x}\|^2] = E_s\Delta f/N_t$, $\mathbf{H} \in \mathbb{C}^{N_r \times N_t}$ is a channel state information matrix. $\mathbf{F} \in \mathbb{C}^{N_t \times N_s}$ is the precoding matrix responsible for both power control and the conversion of the number of streams $N_s \leq N_t$ to the number of transmit antennas. The precoder is also normalized such that $\|\mathbf{F}\|_F^2 = N_t$ and is currently set to the identity matrix, and finally $\mathbf{n} \in \mathbb{C}^{N_r}$ is a column vector of additive noise the entries of which are independent and identically sampled from a zero-mean complex Normal distribution $\mathbf{n} \sim \mathcal{N}_{\mathbb{C}}(0, \sigma_n^2)$. The noise power σ_n^2 is further defined with respect to the noise power spectral density N_0 multiplied by the bandwidth of an OFDM resource element (RE) which is equal to the subcarrier spacing (i.e., $\sigma_n^2 = N_0\Delta f$). The OFDM symbols are Gray-coded and are either based on M-PSK or M-QAM constellations, with M being square (i.e., $M \in \{4, 16, 64, 256\}$). We define $k \coloneqq \log_2 M$ which represents the instantaneous number of bits per symbol. The total bandwidth B is computed through the number of subcarriers N_{SC} multiplied by the subcarrier spacing Δf (i.e., $B \coloneqq N_{\text{SC}}\Delta f$). The capacity C is defined as $C \coloneqq R_b/B$ for a given bit rate R_b. Currently, the source code only supports *one user* at a time. Thus, the model is for a single UE served by a single BS (or beam).

Fading: We use G is the large scale fading coefficient which can be calculated based on the desired carrier frequency (f_c), antenna gains on both ends, path loss exponent (η), and distance of the UE from the BS (d). For a Rayleigh fading channel, the elements of $\mathbf{H} \coloneqq [h_{ij}]_{ij}$ are sampled from a zero-mean complex Normal distribution $h_{ij} \sim \mathcal{N}_{\mathbb{C}}(0, 1)$. Thus, the power gain of the channel becomes $G\|\mathbf{H}\|_F^2$, which means that to incorporate the large scale fading to $\mathbf{H}$, we simply multiply all the elements of $\mathbf{H}$ by $\sqrt{G}$. By employing the concept of the channel eigenmodes, which are the eigenvalues λ_i of $\mathbf{H}\mathbf{H}^*$, it is easy to prove that $\|\mathbf{H}\|_F^2 = \sum_{i=1}^{N_r} \lambda_i$. The proof is as follows:

$$\|\mathbf{H}\|_F^2 \coloneqq \text{tr}(\mathbf{H}\mathbf{H}^*) = \text{tr}(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}) = \text{tr}(\mathbf{U}^{-1}\mathbf{U}\mathbf{\Lambda}) = \text{tr}(\mathbf{\Lambda}) = \sum_i \lambda_i \tag{2}$$

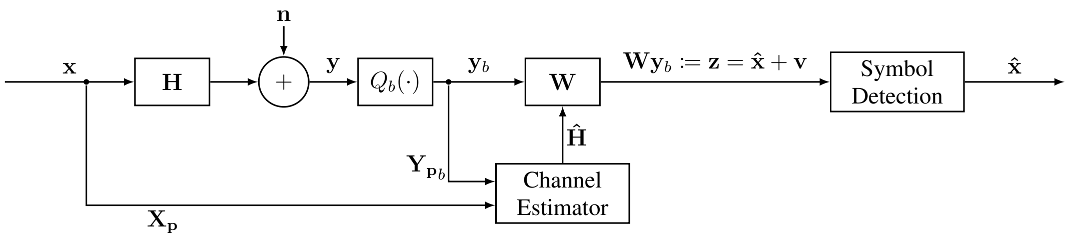


Fig. 1. System overall block diagram.

These eigenmodes are a representation of the number of parallel spatial data streams enabled by the channel and their value is the power each stream is expected to gain.

Beamforming: For the special case where $N_s = N_r = 1$, the precoder $\mathbf{F}$ becomes a column vector $\mathbf{f} \in \mathbb{C}^{N_t}$, $\|\mathbf{f}\|^2 = N_t$ and the channel becomes a row vector. Therefore, (1) becomes:

$$\mathbf{y} = \mathbf{h}^* \mathbf{f} \mathbf{x} + \mathbf{n}, \tag{3}$$

which is the reason a beamforming channel is often known as a "rank-1" channel. A well-known example of a beamforming codebook is the discrete Fourier transform (DFT) codebook. Regardless of the codebook, a power-optimal beamforming precoding vector $\mathbf{f}^\star$ has to be found:

$$\mathbf{f}^\star := \arg\max_{\mathbf{f} \in \mathcal{F}} |\mathbf{h}^* \mathbf{f}|^2, \tag{4}$$

which is how beamforming improves the received signal power.

III. Model-Driven Implementation

We start with the implementation of various functions of wireless network through models and statistics. Fig. 1 shows the functions supported in this version. We call this "model-driven" as opposed to "data-driven" which is discussed later in Sections V, IV, and VI. A b-bit quantizer is denoted as $Q_b(\cdot)$ and is optional. We use $\mathbf{W} \in \mathbb{C}^{N_r \times N_t}$ as a channel equalizer, which effectively removes the effect of the channel on the received data. However, equalizers require an estimate of the channel state information, which the receiver computes at its end. This estimate is denoted as $\hat{\mathbf{H}}$ and has the same dimensions as $\mathbf{H}$. Channel estimation is enabled through "pilots" which are a sequence known to both the transmitting and receiving ends of the wireless system. It should be clear that pilots symbols and cyclic prefix symbols (seen in both 4G LTE and 5G) are not the same.

Constellation: We follow the notation by [2] in generating the symbols for both QPSK and M-QAM as baseband symbols. For QPSK, the inphase (I) and quadrature (Q) branches take the values ± 1 and for M-QAM they take values in the integer interval $[-\sqrt{M}+1, \sqrt{M}]$. These constellation symbols are normalized so the power per symbol is equal to unity and are Gray-coded as stated earlier.

Noise: The noise signal is assumed to be a complex Normal random variable with a variance (or noise power) of $\sigma_n^2 := k_\mathrm{B} T \Delta f N_f$, where k_B is the Boltzmann constant, T is the temperature at 290 K and N_f is the noise figure at the receiving end.

Payload Construction: We generate a random sequence of bits as a payload. The size of this payload is a system parameter and so is the cyclic redundancy check (CRC) generator polynomial. Therefore, constructing the payload that will be represented by $\mathbf{x}$ is straightforward. To minimize the amount of zero-padding, it is a best practice to select the transmission block size and the CRC length to be an integer multiple of the number of symbols required and the transmission rank used. There is only one CRC sent per transmission, regardless of the transmission rank of the channel and it is positioned at the end of the transmission block. The receiver extracts the CRC and attempts to compare it with a fresh CRC computed on the received block for a match.

Quantization: Mapping continuous and infinite values that $\mathbf{y}$ takes to a smaller set of discrete finite values makes it easier to group symbols that "appear" similar and treat them like so—greatly reducing computational overheads (e.g., in machine learning). However quantization is irreversible, non-linear, and causes a degradation to the signal to noise ratio (SNR) of these symbols. Therefore, employing a quantizer comes with its drawbacks and many avoid it. The Lloyd-Max quantizer is the quantizer of choice.

Channel Estimation[†]**:** Let the pilot symbols have a length n_pilot OFDM symbols. We denote the fat matrix of known pilot symbols $\mathbf{X_P} \in \mathbb{C}^{N_t \times n_\text{pilot}}$. Note that this matrix cannot be tall because the number of pilots must at least be equal to the number of transmit antennas ($n_\text{pilot} \geq N_t$). Let the received pilot symbol values be stored in another matrix $\mathbf{Y_P} \in \mathbb{C}^{N_r \times n_\text{pilot}}$. Then we can

[†]For now the channel is the composite channel of $\mathbf{HF}$.

compute the least-squares estimate of the channel as follows:

$$\begin{aligned}
\mathbf{Y_P} &= \hat{\mathbf{H}}\mathbf{X_P} \\
\mathbf{Y_P}\mathbf{X_P}^* &= \hat{\mathbf{H}}\mathbf{X_P}\mathbf{X_P}^* \\
\mathbf{Y_P}\mathbf{X_P}^*(\mathbf{X_P}\mathbf{X_P}^*)^{-1} &= \hat{\mathbf{H}} \\
\hat{\mathbf{H}} &= \mathbf{Y_P}\mathbf{X_P}^*(\mathbf{X_P}\mathbf{X_P}^*)^{-1}
\end{aligned} \tag{5}$$

The linear minimum mean square error (L-MMSE) estimate of the channel minimizes an error term and can be written as:

$$\hat{\mathbf{H}}_{\text{L-MMSE}} = \mathbf{Y_P}\mathbf{X_P}^*(\mathbf{X_P}\mathbf{X_P}^* + \sigma_n^2\mathbf{I}_{N_t})^{-1} \tag{6}$$

Pilot Design: Since the matrix $\mathbf{X_P}$ is not square, it should be carefully designed so that $\mathbf{X_P}\mathbf{X_P}^*$ is invertible. An example of carefully designed codes is Zadoff-Chu codes in both 4G and 5G. However, for simplicity we exploit the idea of semi-unitary matrixes such that $\mathbf{X_P}\mathbf{X_P}^* = \mathbf{I}_{N_t}$ and therefore we do not need to worry about the invertibility of $\mathbf{X_P}\mathbf{X_P}^*$ any further. At a high level, the construction of a unitary matrix (i.e., $\mathbf{Q}^{-1} = \mathbf{Q}^*$) is possible through a combinatorial rearrangement of orthonormal bases. Since these lead to a square matrix, a fat matrix $\mathbf{A} \in \{0,1\}^{N_t \times n_{\text{pilot}}}$ with ones on the diagonal and zeros elsewhere is constructed and then right-multiplied by the unitary matrix: $\mathbf{X_P} := \mathbf{Q}\mathbf{A}$. Then, it is straightforward to verify that $\mathbf{X_P}\mathbf{X_P}^* = \mathbf{I}_{N_t}$. Consequently, the channel estimate $\hat{\mathbf{H}}$ from (5) and (6) become:

$$\hat{\mathbf{H}}_{\text{LS}} = \mathbf{Y_P}\mathbf{X_P}^*; \qquad \hat{\mathbf{H}}_{\text{L-MMSE}} = \frac{1}{1+\sigma_n^2}\mathbf{Y_P}\mathbf{X_P}^*. \tag{7}$$

Channel Equalization: Channel equalization is intended to remove the impacts of the channel from the transmitted symbols. Two commonly used types of channel equalizations are 1) zero-forcing (ZF) and 2) minimum mean square error (MMSE) equalization. Continuing with the same notation, the zero-forcing equalizer of the estimated channel $\hat{\mathbf{H}}$ is given by:

$$\mathbf{W}_{\text{ZF}} = \hat{\mathbf{H}}^{\dagger} \tag{8}$$

Thus with a left-multiplication of $\mathbf{W}_{\text{ZF}}$ (or the Hermitian transpose for cases where $\hat{\mathbf{H}}$ is not square) into (1), we obtain:

$$\begin{aligned}\mathbf{z} &\coloneqq \mathbf{W}_{\text{ZF}}\mathbf{y} = \mathbf{W}_{\text{ZF}}\hat{\mathbf{H}}\mathbf{x} + \mathbf{W}_{\text{ZF}}\mathbf{n} \\ &= \hat{\mathbf{x}} + \mathbf{v}\end{aligned} \tag{9}$$

which is then fed into a detector to identify the OFDM symbols in the presence of the inseparable noise. Note that vector $\hat{\mathbf{x}}$ is equal to $\mathbf{x}$ only when a perfect channel state information is known under a perfect channel equalization. The vector $\mathbf{v}$ is known as the receiver enhanced noise. While it still has a zero mean, its autocorrelation matrix is $\mathbf{R}_{\mathbf{vv}} = \mathbf{R}_{\mathbf{nn}}\mathbb{E}[\text{tr}(\mathbf{W}^*\mathbf{W})] = \mathbf{R}_{\mathbf{nn}}\|\mathbf{W}\|_F^2$.

Also, for beamforming, the equalizer is $\mathbf{w} \in \mathbb{C}^{N_t}$ such that $\mathbf{w}^*\mathbf{f} = 1$.

MIMO Receivers: The average SNR per RE computed at the receiver *after* equalization is denoted γ. However, every antenna would have a different SNR based on the eigenmode it represents (a power gain in the form of $\lambda\|\mathbf{h}\|^2$). Therefore, the per-antenna SNR is expected to be different and the choice of the MIMO receiver enables this.

Symbol Detection: The last step is to convert the received symbols to the symbols belonging to the constellation. Since $\mathbf{v}$ is also Gaussian, we can use a maximum likelihood detector (or K-means clustering as detailed later). In this case, every symbol in $\mathbf{z}$, which we call $z \in \mathbb{C}$, can be found based on the Euclidean distance from the nearest symbol in the constellation $\{s_m\}_{m=0}^{M}$. Thus we obtain a column vector $\hat{\mathbf{m}}^\star$ from z, which is comprised of the entries $\hat{m}^\star$ that fulfill the following relationship:

$$\hat{m}^\star = \arg\min_m |z - s_m|, \qquad m \in \{0, 1, \ldots, M-1\}. \tag{10}$$

Furthermore, since every m corresponds to one I/Q symbol, we can also obtain the column vector $\hat{\mathbf{x}}$ the entries of which are the detected symbols. Because every symbol represents a string of bits, the received codeword (including the padding and the CRC) can be recovered from $\hat{\mathbf{x}}$ for further analysis.

So far we have covered model-driven applications of Python in wireless communications. Namely: construction of symbols, creation of a MIMO payload with CRC, creation of a channel, creation of additive noise, estimation of the channel using pilots, channel equalization, and symbol detection. Next, we focus on a data-driven approach where the value of machine learning is demonstrated. Three areas of interest are: unsupervised learning and supervised learning (including deep learning), and reinforcement learning.

IV. Unsupervised Machine Learning Use Cases

A. K-means Clustering

The use of K-means clustering to perform symbol detection is the quintessential application of unsupervised learning in wireless communications. In the code implementation of this case, a two-dimensional plane representing the I/Q components of the complex-valued symbols is formed. The K-means centroids $\mathcal{C}$ are initialized (and fixed) to these M constellation symbols. Then a two-dimensional column vector $\mathbf{z}$ is constructed from the received symbols $z \coloneqq z_I + jz_Q$ as follows $\mathbf{z}^\top \coloneqq [z_I, z_Q]$. The centroids in $\mathcal{C}$ are constructed in a similar fashion. Next, the symbols in presence of additive noise are grouped in a way that minimizes the Euclidean distance to a centroid:

$$m^\star = \underset{\mathbf{c}\in\mathcal{C}\colon\, |\mathcal{C}|=M}{\arg\min}\; \|\mathbf{z} - \mathbf{c}\|. \tag{11}$$

which is identical to the result from (10).

V. Supervised Machine Learning Use Cases

We define the learning phase of supervised machine learning using a feature space $\mathbf{X}$, a supervisory signal of a column vector $\mathbf{y}$, and a set of hyperparameters $\boldsymbol{\Theta}$. To avoid confusion with the model-based development in Section III, any symbol used moving forward is expected to refer to this terminology unless explicitly referred to an equation from Section III.

Formally, a supervised learner minimizes a loss function $L(\cdot,\cdot)$ through a search space defined by the hyperparameters. The training dataset is used to find the optimal $\boldsymbol{\Theta}^\star$ and the validation set is used as a benchmark of the loss function. Formally:

$$\underset{\boldsymbol{\Theta}}{\text{minimize:}} \qquad L(\mathbf{y}, \hat{\mathbf{y}} \coloneqq f_{\boldsymbol{\Theta}}(\mathbf{X})), \tag{12}$$

where $f_{\boldsymbol{\Theta}}(\mathbf{X})$ is a supervised learning model specific function. Depending on the data type that $\mathbf{y}$ represents, we could have a "regressor" for continuous data $[\mathbf{y}]_i \in \mathbb{R}$ or a "classifier" for categorical data $[\mathbf{y}]_i \in \mathbb{Z}$, where the class identifiers have no quantitative significance.

A. Deep Learning

Fully connected deep neural networks: We build a deep neural network with depth D and width W in the source code to perform symbol detection where the supervisory signal (or label) is the one-hot encoded $\hat{\mathbf{m}}^\star$ and the learning feature space is $\mathbf{X} \coloneqq [\text{Re}[\mathbf{y}], \text{Im}[\mathbf{y}]]$, with $\mathbf{y}$ as in (1).

The activation function is the sigmoid function, which is a suitable choice since most variables are already scaled in the interval $[0, 1]$ due to the various normalizations in the development in Section II. Since the symbol detection problem is a multi-class classification problem, the loss function is the categorical cross-entropy function and the output layer has a dimension equal to the number of the one-hot encoded classes and a softmax activation function. More about deep learning is in [3].

Several approaches to training a supervised learning model have been discussed in literature. Here are a couple:

1) **Learn-exploit-invalidate:** In this approach, the deep neural network is turned off allowing the "default" algorithm to operate and collect data. Then after some time, the data trains a model and the model is then used (or exploited) for a longer duration. In this case, the performance of the trained model is observed during the exploitation state and if it falls below a threshold, it is invalidated and the cycle repeats.
2) **Transfer Learning:** This approach is suitable for deep learning specifically and for tasks that are controlled in reach (e.g., on a given road, a floor plan, or a Manhattan grid) and objective (e.g., optimal next beam and symbol detection). In this case, BSs in the area of interest collect data, send it to a higher tier for training (e.g., a cloud compute instance) and then the trained model is sent back to these BSs to perform inference (i.e., predictions) only. Similar to the previous approach, the performance of the inference is observed for degradation—the event for introducing more training.

VI. REINFORCEMENT LEARNING USE CASES

In the case where training data is not in a design matrix format (or simply unavailable), a policy (e.g., of a game) can be used to train an agent to "win" by learning a reward-maximizing policy. A policy $\pi(\cdot \mid \cdot)$ maps the state-action space to a probability. Formally:

$$\pi(a \mid s) \colon \mathcal{S} \times \mathcal{A} \to [0, 1] \tag{13}$$

The interaction of the algorithm that resides in the agent (and issues actions $a \in \mathcal{A}$) and the environment (our wireless network) for a reward R in return (and a new observation state $s \in \mathcal{S}$). This interaction is outlined in Fig. 2.

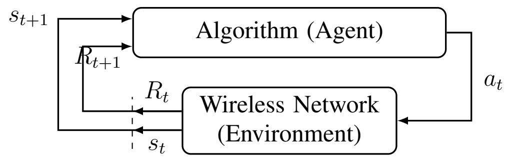


Fig. 2. Actions, states, and rewards: reinforcement learning.

VII. Performance Measures

Two groups of performance measures in the source code are observed and defined below. For the radio performance measures, the symbols refer to the model-based development in Section III.

A. Radio Performance Measures

1) Error vector: Defined as the difference of the vectorized channel estimate from the true vectorized channel. Therefore, $\mathbf{e} := \mathbf{vec}(\mathbf{H}) - \mathbf{vec}(\hat{\mathbf{H}})$.
2) Estimation mean squared error: Defined as the square of the Euclidean norm divided by the number of elements in the channel matrix. Thus MSE $:= \|\mathbf{e}\|^2/(N_r N_t)$.
3) Information bit rate (R_b): This is the codeword size (in bits) per stream divided by the transmit time interval (in seconds). The spectral efficiency or $C := R_b/B \le k$ in bps/Hz.
4) Signal to noise ratio per RE (SNR) at the transmitter is $10\log(E_s\Delta f/(N_t\sigma_n^2)) = 10\log(E_s/(N_t N_0))$ in dBm. The transmit power (P_{tx}) is equal to $E_s\Delta f/N_t$ in Watts.
5) Bit energy to noise power spectral density ratio (E_b/N_0) at the transmitter is $10\log(\text{SNR}/C) = \log(E_s/(CN_tN_0))$ in dBm. (Note that E_b/N_0 is the per-bit SNR).
6) The RE power measured at the receiver before equalization is $P_{\text{rx}} := G\|\hat{\mathbf{H}}\|_F^2 E_s\Delta f/N_t$ in Watts.
7) Pathloss: Arithmetically equal to $10\log P_{\text{tx}} - 10\log P_{\text{rx}}$ in dB and is a positive number.
8) Signal to noise ratio per RE at the receiver after equalization is $\gamma := 10\log(E_s\|\mathbf{W}\hat{\mathbf{H}}\|_F^2/N_t) - 10\log(N_0\|\mathbf{W}\|_F^2)$ in dBm.
9) Bit energy to noise power spectral density ratio (E_b/N_0) at the receiver after equalization is $10\log(\gamma/C)$ in dBm.
10) Block error rate (BLER): Defined at the receiver as the number of codewords the CRC of which is incorrect divided by the total number of received codewords.

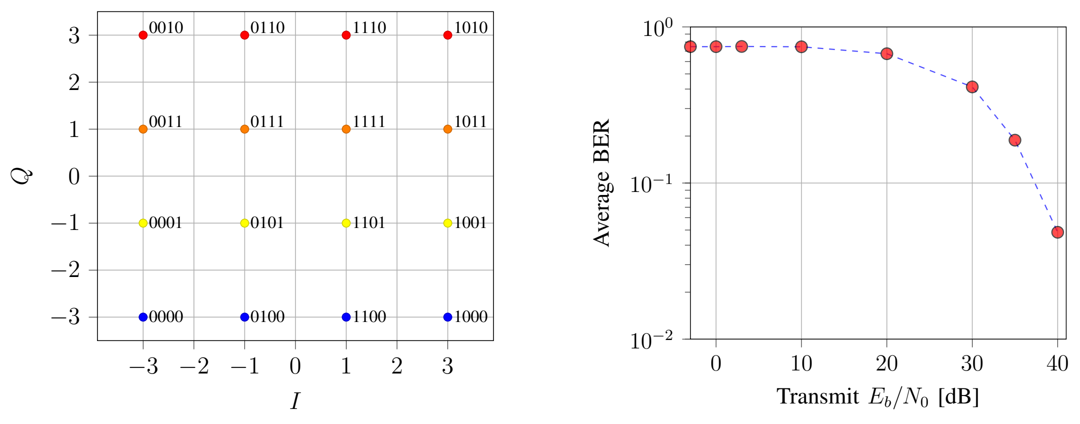
TABLE I
SYSTEM PARAMETERS

Parameter	Description	Python
Payload size	How many bits to be transmitted over the channel. Payload is divided into codewords.	`payload_size = 30000`
Random seed	This is the seed used by the pseudo-random number generator. Helps with reproducibility of results.	`seed = 7`
Number of transmit antennas (N_t)	The number of rows in the channel $\mathbf{H}$.	`N_t = 2`
Number of receive antennas (N_r)	The number of columns in the channel $\mathbf{H}$.	`N_r = 2`
Constellation	This is the modulation scheme.	`constellation = 'QAM'`
Constellation size (M)	Number of symbols in the constellation. Must be a power of two.	`M_constellation = 16`
Codeword size [bits]	The size of the transport block before CRC.	`codeword_size = 1024`
CRC generator	A string of bits that represent the CRC generator polynomial.	`crc_polynomial = 0b1001_0011`
CRC length [bits]	The length of the CRC in bits.	`crc_length = 4`
Number of pilot symbols n_{pilot}	Length of pilots required for channel estimation.	`n_pilot = 2`
Transmit SNR [dB]	The RE symbol power to noise ratio.	`Tx_SNRs = [-3, 0, 3, 10, 20, 30, 35, 40, 50]`
Center frequency [Hz] (f_c)	Avoid anything beyond sub-6 GHz at present.	`f_c = 1.8e6`
Independent variable	A quantity from the radio performance measures in Section VII.	`x_label = 'Tx_EbN0'`
Dependent variable	A quantity from the radio performance measures in Section VII.	`y_label = 'Avg_BER'`

B. Machine Learning Performance Measures

These measures are meant for supervised learning only since the ground truth is available by definition. Let us call the ground truth $\mathbf{y}$ and the predicted value based on the machine learning prediction $\hat{\mathbf{y}}$. These are column vectors the entries of which are integers. These integers represent categories (i.e., do not have any quantitative significance). Currently one performance measure is used in the source code:

- **Accuracy:** Defined as the complement of the mean classification error: $\frac{1}{M}\sum_{i=1}^{M} \mathbb{1}\left[[\mathbf{y}]_i \neq [\hat{\mathbf{y}}]_i\right]$. Note that accuracy can become meaningless if one category has rare occurrence and the classifier points to the normal class [4].



(a) 16-QAM constellation with Gray code (b) BER vs E_b/N_0 for a 2T2R MIMO channel

Fig. 3. 16-QAM constellation (*left*) and bit error rate performance curve (*right*). In Gray coding, from any symbol to any adjacent symbol only a change of one bit is permissible. This curve is for a rank-2 MIMO channel using 16-QAM and parameters as shown in Table I.

VIII. Using the Code

Code Download: The source code can be downloaded from [1]. Both Python 3 and Git need to be installed on your local machine. The main file is `main.py` and the library requirements is `requirements.txt`. While the latest versions of `tensorflow`, `keras`, `numpy`, `pandas`, and `matplotlib` seem to work fine, the first step is to ensure that the requirements are fulfilled. To this extent `pip3 install -r requirements.txt` can be issued from a terminal window (or Command Prompt) prior to downloading the code. The next step is to fork or download the code from Git:

```
git clone https://github.com/farismismar/eesc7v86-fall22
```

Code Editing: Editing parts of the code is possible and encouraged. However, while we invite the more advanced user to dig deep in the code and make changes, a beginner user can build a table as in Table I and adjust the parameters accordingly. The entry point to the code is `main.py`. There are several plotting functions available in the auxiliary file `PlottingUtils.py` that the reader is urged to take a look at. Examples of available plots are: probability density functions, cumultative distribution functions (both joint and marginal), and box plots.

Code Execution and Output: The code can be run from Python: `python3 main.py`. The output of the run is a comma separated value file named `output.csv`. This file has the radio performance measures defined in Section VII. We show some output plots in Fig. 3.

IX. Conclusion

In this draft we summarized the various concepts required to build a wireless communication prototype that supports both MIMO and OFDM. We showed how the prototype can be implemented in Python and showed a few use cases for machine learning applications. As this is an early draft, we do expect mistakes and errors and encourage the readers to share feedback with the author for further improvements.

References

[1] F. B. Mismar, “Source Code,” Oct. 2022. [Online]. Available: https://github.com/farismismar/eesc7v86-fall22

[2] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. McGraw-Hill, 2008.

[3] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[4] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf