Pattern Recognition
Department of Electrical and Information Engineering
University of Cassino and Southern Latium, Second Semester 2018
Homework Assignment 2
Assigned 23 May 2018; due 11:59pm, 01 June 2018

In preparing my solutions, I did not look at any old home work, copy anybody's answers or let them copy mine.

Name: Md. Kamrul Hasan **Signature:** [illegible]

Submitted By:

Md. Kamrul Hasan

E-mail: kamruleeekuet@gmail.com

MAIA (M2), UNICAS, Italy.

Submitted To:

Prof. Francesco Tortorella, Ph.D.

E-mail: francesco.tortorella@unicas.it

Universita degli Studi di Cassino e del Lazio Meridionale, Italy.

Problem 2.1 [60 %] Consider the dataset contained in the file hw2data.csv available on the course site. It contains 8,000 samples coming from a two {class problem, each made of 10 numerical features and a binary label (±1). Split the data into training and test sets by randomly selecting 25% of the examples from each class for the test set. For the classification of the test data, use TFLearn to train a Multi-Layer Perceptron with 10 input nodes, 2 hidden layers and 2 output nodes. Consider the following options:

(a) Weight initialization:
- (a.1) all the weights set to zero
- (a.2) random values from a uniform distribution
- (a.3) uniform Xavier distribution

(b) Batch normalization
- (b.1) yes
- (b.2) no

(c) Activation function for the hidden layers
- (c.1) ReLu
- (c.2) tanh

Evaluate the accuracy obtained on the test set with the various options and discuss the results obtained. Choose reasonable values for the remaining parameters (e.g. learning rate, batch size, number of epochs, ..) and keep them fixed during the experiments.

Solution of 2.1: Data given for this homework contains 8000 rows and 11 columns where 8000 rows indicate the measurements of the observations that also known as the instance of the class. There are 10 features (Column 1st to 10th) for each instances of the observations. Last column (11th) indicates the class label (+1 or -1) corresponding to each instance. In this data set, there are 2 class which are +1 and -1. The overall work flow used for the question is shown in Figure 2.1.1.

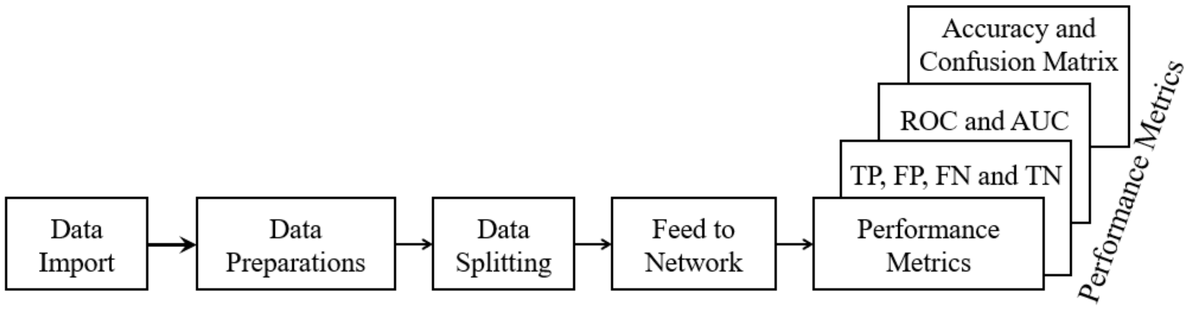

Figure 2.1.1: Overall pipeline of the home work for question 2.1

After importing the data, some preparations have been done to this data. Firstly, data are not normalized which means the variations of the values among each column are too much. To solve features variance problem, standardizations has been done by using the equation Eq. 2.1.1.

$$Data\ (:,1:10) = \frac{Data(:,1:10) - mean}{Standard\ Deviation} \qquad (2.1.1)$$

Where, both mean and Standard Deviation are the 1×10 $vectors$. And we don't need to normalize last column because it's the class level only. To normalize the data, I have used *"sklearn"* preprocessing that are available in Python Library. After normalizing, I need to split whole data into Train (75 %) and Test (25 %). I have used "*train_test_split*" to split the data into Train and Test with the random state 100.

After that, Train data was fed to the network as shown in Figure 2.1.2. for the training of the network. Once network is trained, test data was used to the trained model to measure some performance metrics e.g. *TP, TN, FP, FN, ROC, AUC, Accuracy and confusion Metrix*. The multilayer perceptron used in this problem was just like the Figure in 2.1.2. The different hyperparameters of the network are mentioned afterward.

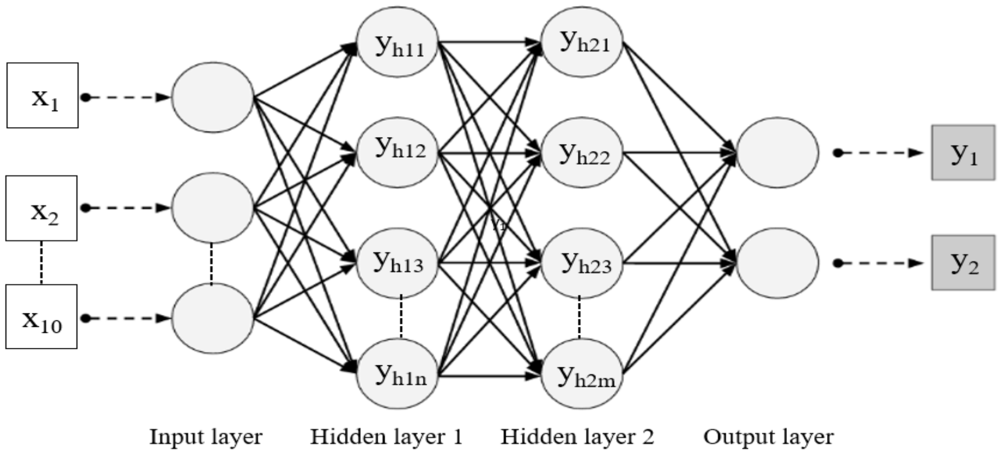

Figure 2.1.2: Multilayer perceptron model having input, output and hidden layers

Results and Discussions

To discuss the results for each condition as mention in 2.1 (a), 2.1 (b) and 2.1 (c), some hyperparameters have been fixed. So, over the experiment after lots of "*trial and error"* approaches, I have fixed the following hyperparameters-

i. Learning rate= **0.01**
ii. Batch size= **200**
iii. Number of epochs= **350**
iv. Optimizer= Adam optimization (Extension to stochastic gradient descent)
v. Bias initialization= Random values from a normal distribution (Mean=**0.0**, Std. dev.=**0.02**)
vi. Regularization= **L2-norm** (least squares error (LSE))
vii. Number of Class=**2**
viii. Number of Features=**10**
ix. Number of Neuron in Hidden layer 1=**25**
x. Number of Neuron in Hidden layer 2=**25**
xi. Loss Function=categorical_crossentropy

2.1 (a) ***Weight initialization:*** There are so many cases are possible for weight initialization where other parameters are kept constant. Among all those cases, only one cases are shown in this report as in Table 1 and Figures 2.1.3~ Figure 2.1.5.

Table 1: Weight initialization

No	**Types of Weight Initializations**	**Fixed Parameters**	**Performance Metrics**

a.1	All the weights set to zero	Batch normalization= '**No**'
		Activation for hidden layers is '**ReLu**'.
			TP= 0, FP= 0, FN= 984 and TN= 1016
			Accuracy= 50.8%, and AUC= 50 %

a.2	Random values from a uniform distribution	Batch normalization= 'No'
		Activation for hidden layers is '**ReLu**'.
			TP= 832, FP= 156, FN= 152 and TN= 860
			Accuracy= 84.6 %, and AUC=92.56 %

a.3	Uniform Xavier distribution	Batch normalization= 'No'
		Activation for hidden layers is '**ReLu**'.
			TP= 871, FP= 183, FN=113 and TN= 833
			Accuracy= 85.2 %, and AUC= 92.61%

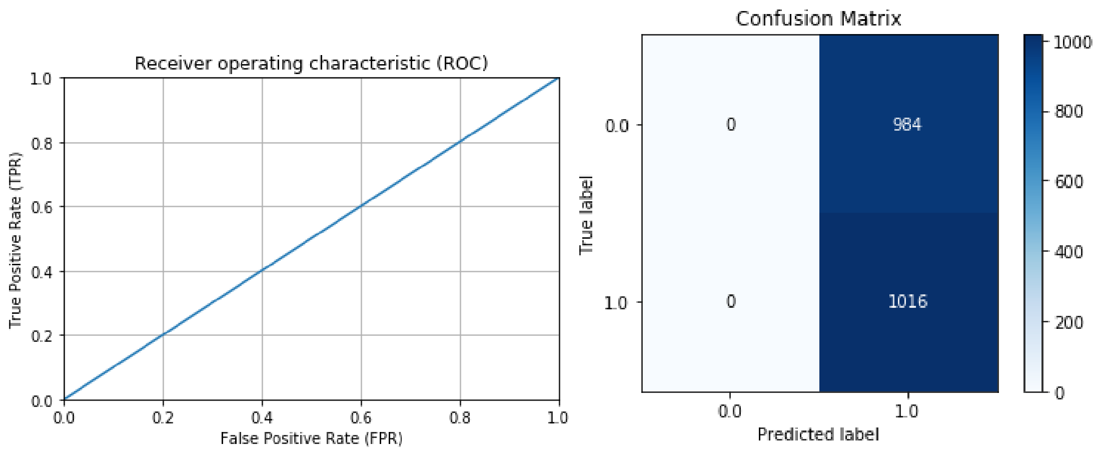

Figure 2.1.3: ROC (Left) and Confusion Matrix (Right) for weights set to zero

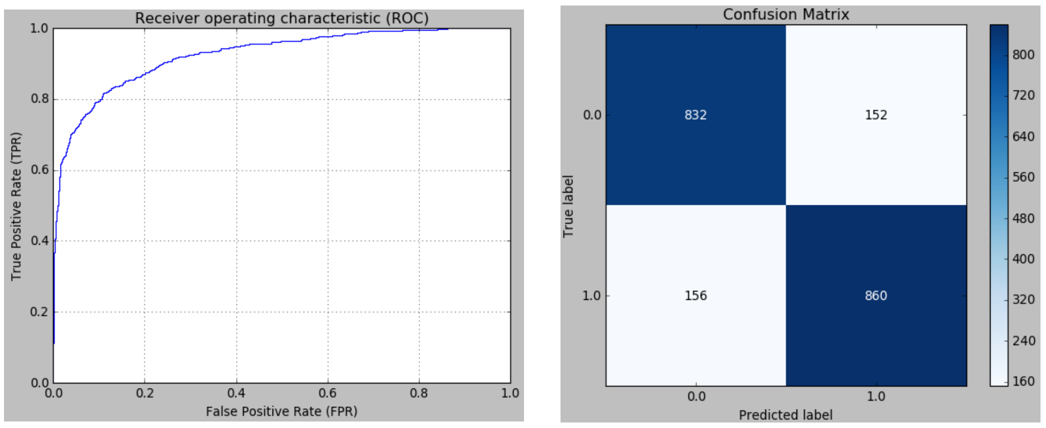

Figure 2.1.4: ROC (Left) and Confusion Matrix (Right) for weights uniform distribution

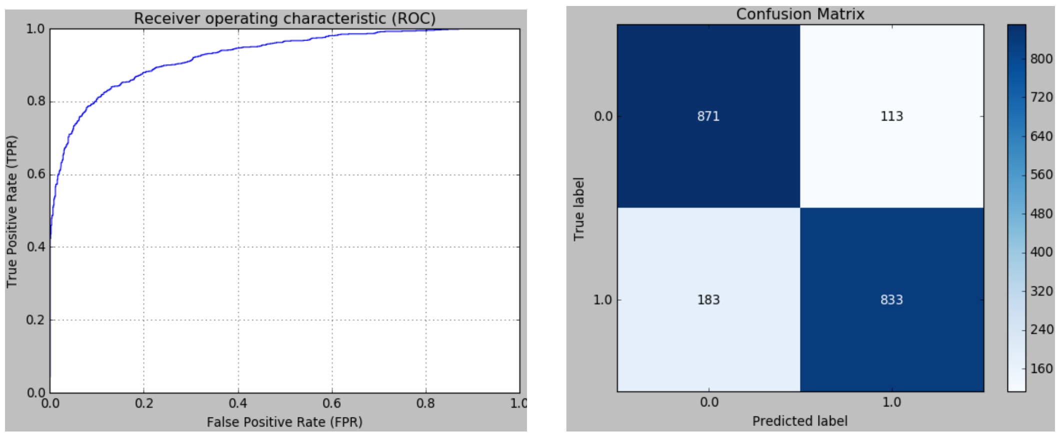

Figure 2.1.5: ROC (Left) and Confusion Matrix (Right) for weights Xavier distribution

Form the above Figures and Table 1, we see that when all the weights are set to zero, model was not learning anything from the training data. As well as, from the Figure 2.1.3, ROC and confusion matrix indicates that for the zero weight, model become classification hell. But if bias has some weight initialization instead of zero initialization, then bias also learn something from training. For the uniform and Xavier weight initializations, there was not so much significant change in my model with the fixed hyperparameters. But, from the Table 1 and Figure 2.1.4 & Figure 2.1.5, we see that Xavier is little bit better in may model. Xavier provides less false negative (FN) that means sick people classified as a normal.

2.1 (b) ***Batch normalization:*** Batch Normalization is a technique that provides inputs to any layer in a Neural Network having zero mean and unit variance. Similarly, in this case only one experiment is showing in Table 2 and Figure 2.1.6 ~ Figure 2.1.7.

Table 2: Batch Normalization

No	**Batch Normalization**	**Fixed Parameters**	**Performance Metrics**
b.1	Batch normalization= **'No'**	Weight Initializations: **Xavier distribution** Activation for hidden layers is '**ReLu**'.	TP=901, FP= 227, FN= 83 and TN= 789 Accuracy= 84.5 %, and AUC= 92.91 %
b.2	Batch normalization= **'Yes'**	Weight Initializations: **Xavier distribution** Activation for hidden layers is '**ReLu**'.	TP= 862, FP= 173, FN= 122 and TN= 843 Accuracy= 86.5 %, and AUC= 93.22 %

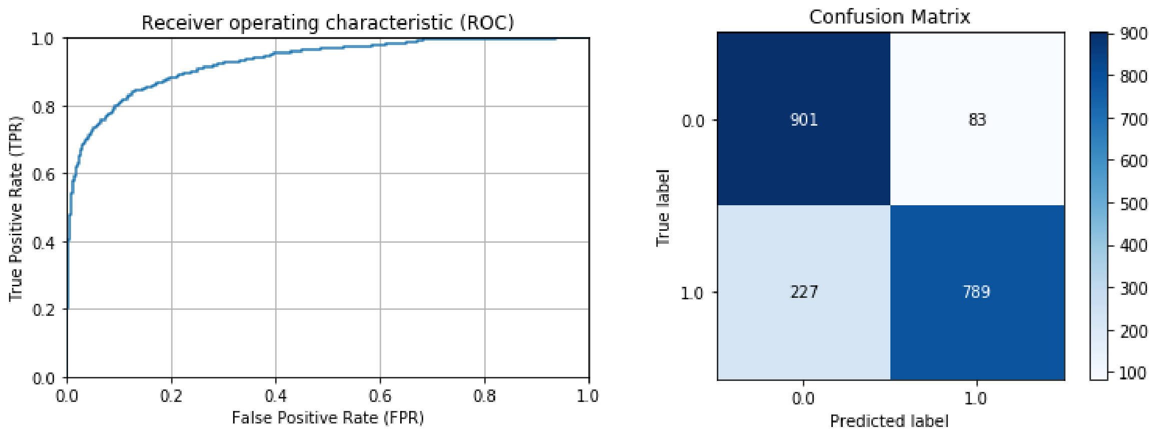

Figure 2.1.6: ROC (Left) and Confusion Matrix (Right) for No Batch Normalization

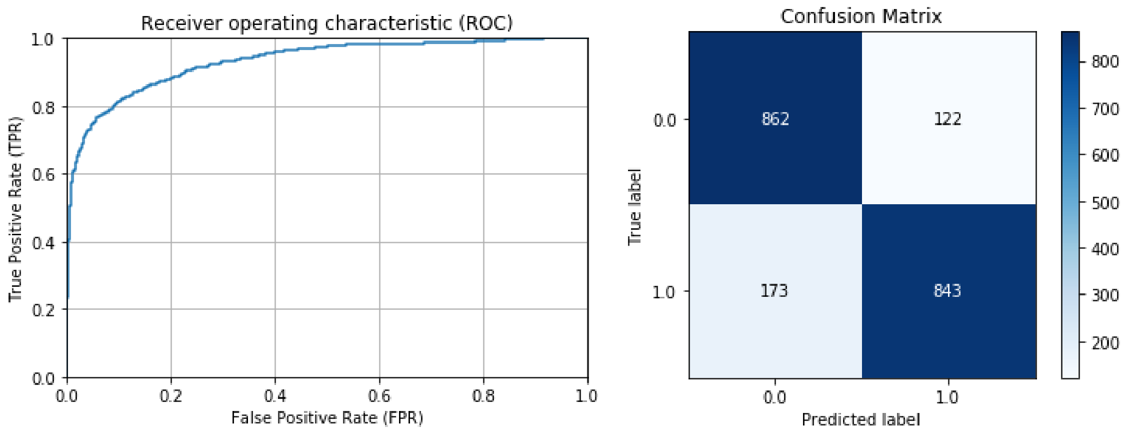

Figure 2.1.7: ROC (Left) and Confusion Matrix (Right) for Batch Normalization

From the Table 2 and Figure 2.1.6 & Figure 2.1.7, it is noticeable that after batch normalizations there is no significant change in the results. Although after normalizations, accuracy as well as AUC increase little bit. Actually, batch normalizations do not use for performance metrics controlling but it allows each layer of a network to learn by itself a little bit more independently of other layers. And this individual learning helps to reduce the overfitting effects as well makes the model to better generalized.

2.1 (c) ***Activation function for the hidden layers:*** The experiment that was done for this case is shown in Table 3 and Figure 2.1.8~ Figure 2.1.9

Table 3: Activation function used for hidden layer

No	Hidden layer Activation function	Fixed Parameters	Performance Metrics

c.1	Activation for hidden layers is '**tanh**.	Batch normalization= **'No'**
		Weight Initializations: **Xavier distribution**
			TP=818, FP=157, FN= 166 and TN=859
			Accuracy= 83.85 %, and AUC= 92.41 %

c.2	Activation for hidden layers is '**Relu**'.	Batch normalization= **'No'**
		Weight Initializations: **Xavier distribution**
			TP= 855, FP= 159, FN= 129 and TN= 857
			Accuracy= 85.6 %, and AUC= 92.75 %

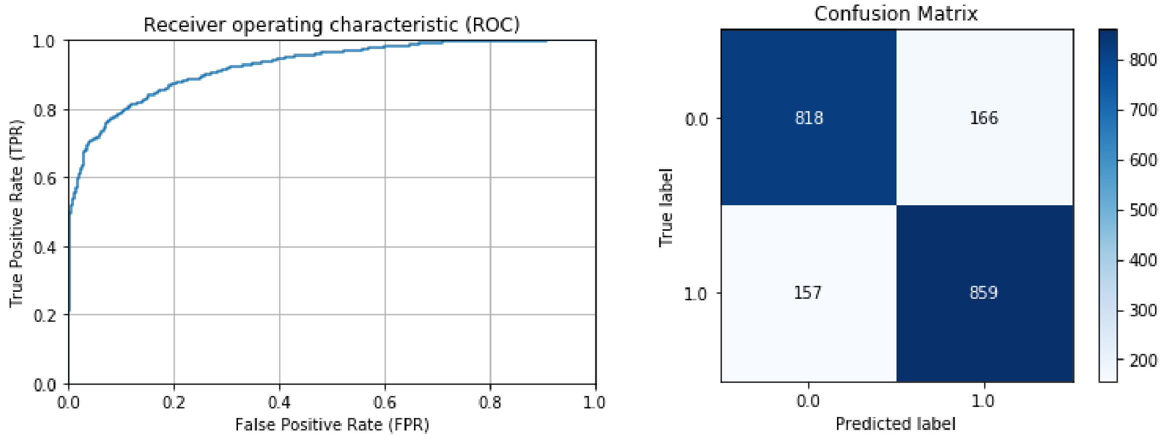

Figure 2.1.8: ROC (Left) and Confusion Matrix (Right) for tanh Activations in hidden Layer

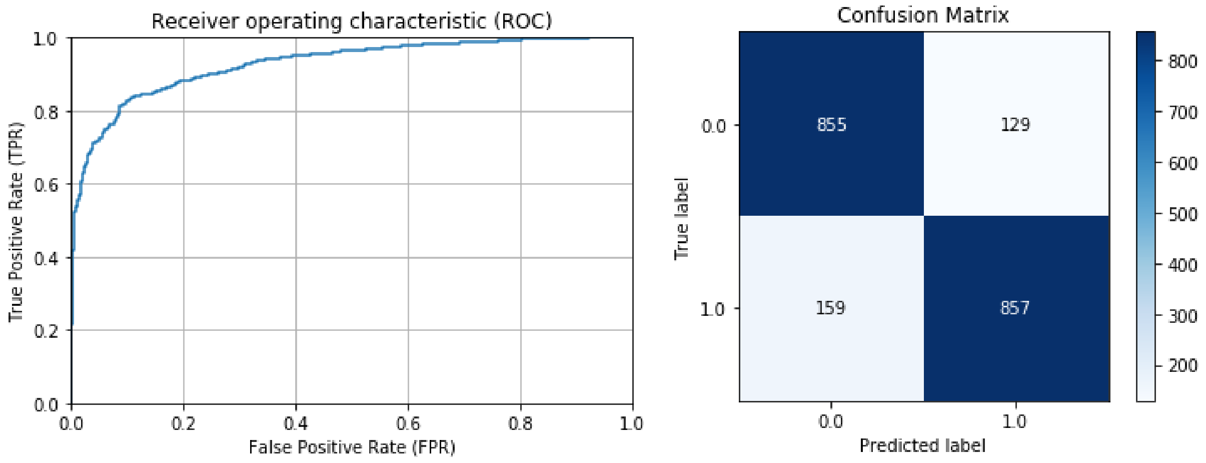

Figure 2.1.9: ROC (Left) and Confusion Matrix (Right) for Relu Activations in hidden Layer

Table 2 and Figure 2.1.8 & Figure 2.1.9 are the results for two different activation functions. And their performances are almost same in the implemented classifier. But, ***Relu*** has little bit more accuracy and AUC than ***tanh***. This improvement of ***Relu*** is due having zero sparsity that means for activation less than 0 it has zero response. And another reason might be considered that ***Relu*** is indeed non-saturation of its gradient, which greatly accelerates the convergence of gradient descent compared to the ***tanh*** functions.

Problem 2.2 [40%] Use the dataset of Problem 2.1 and perform several splits into a training set and a test set (also with different sizes) to determine which combination of options among the ones you considered in Problem 2.1 ensures the best accuracy. Once you picked out the best model, save it by using the method **tflearn.models.dnn.save.** The saved model must be submitted together with your report and will be run on a separate matrix containing new test data. Your grade will be based on the performance of your classifier on the new test data, which will contain a very large number of examples generated from the same distribution.

***Solution of 2.2*:** After analyzing the performance of all the conditions in Problem 2.1, some hyperparameters have been selected to save the model for future classifications.

To access this code, anyone need to follow the block diagram as shown in Figure 2.2.1.

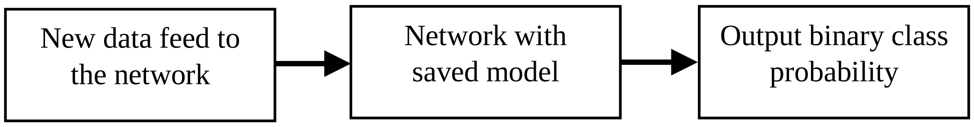

Figure 2.2.1: Block presentation of Problem 2.2

In the 1st block, you need to load new data for testing. Here, it is mentionable that the new data that will be classified with this model should be as like as Eq. 2.2.1. And must have the same distribution of the training data that means that same standard deviations and mean values. Suppose your new data matrix is A.

$$A = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{row \times 10} \qquad (2.2.1)$$

Which means that matrix A may have any numbers of rows but 10 columns only. After hitting the RUN button, this model will return the predicted binary class as like as Eq. 2.2.2. If rows in A are in class label 0 then 1st column of y will have more probability and vice versa.

$$y = \begin{bmatrix} \quad & \quad \end{bmatrix}_{row \times 2} \qquad (2.2.2)$$

N.B. The python code named "*mytest.py*" and all the saved items listed below should be in the same working directory.

Attachments with this report:

1. Source data named as *"hw2data.csv"*.
2. *Python* Source code for 2.1 named as "*Problem_2_1.py*".
3. *Python* Source code for 2.2 named as *"mytest.py"*.
4. Saved Item for the model
   i. checkpoint
   ii. mymodel.tfl.data-00000-of-00001
   iii. mymodel.tfl.index
   iv. mymodel.tfl.meta