Report:

in this work, I mainly implemented the KNN and MLP and the NB is what I placed for bonus

Implementation Details:

preprocess:

due to the data leakage, thus I follow the email to do the K-fold first. Later I put the train in the preprocess first, in the class preprocess I calculate the mean with the F1 to F17 column and the mode in the F18 to F77 column with each column and fill the mean and the mode into the train data. After doing this, I do the validation data preprocess. The function used is different with the train thus in the class preprocess I used the _preprocess_numerical and _preprocess_val_numerical to distinguish them. that's because, we have to use the mean and mode calculate in the train to fill the the validation dataset blank. In the normalization, I used the min max method to do the normalization to let the data fit into the range[0,1]

NB:

fit:

I have two dict, the dict to save the row which the outcome is 0 and which the outcome is 1. And another is to save the probability of the 0 in the outcome and so does 1. After doing this, I have to calulate the mean and std in all of the feature in the row and put into a stack to save them. with the 0 and 1 to distinguish them..

joint_log_likelihood:

is the calculation to the below, it will return the two value one for 0 and one for 1

$$\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Predict:

let each row in the test to do the joint_log_likelihood and take the bigger value in the return then it is the prediction for 1 or 0

Predict proba:

let each row in the test to do the joint_log_likelihood and the the return while it is negative so I change it to positive and let it to be the range in the[0,1] then return it.

Hyperparameters:

due to the zero probability, I just add a small value to the probability, so it won't be 0.

KNN:

fit:

just to save the train and outcome

_euclidean_distance

calculate the distance between the given row.

Predict:

just let every row to calculate the distance with the test row and pick up for the best k ,then count their occurrence with 0 and 1. Which is more then the result will be that.

Predict proba:

just let every row to calculate the distance with the test row and pick up for the best k ,then count their occurrence with 0 and 1. After counting, to dive the k to get probability and return.

Hyperparameters: to find the best k to let k row give the vote just give every k a chance and get the best.

MLP:

initialize_network:

this function just have to create the network. Thus, give each layer of node the initial weight and save them. and the range is the $(0, \sqrt{\frac{2}{n_{in}^k + n_{out}^k}})$,

fit:

create a thing named network. It will save the weights of each layer and each node in the each layer. And the delta will also be saved in it. Thus, we have to create it first then go to the initialize_network function. After doing this, then to go through every Epoch, in every row in train, I have to do the forward propogaiton to get the output to used in the backward propagation. And after backward propagation we need to update the weight then go next row. After all row done than do the next epoch

forward propogaiton:

Calculate neuron activation, and do the 1.0 / (1.0 + exp(-activation)), then get the output of a neuron

backeard propagation:

Error is calculated between the expected outputs and the outputs forward propagated from the network. These errors are then propagated backward through the network from the output layer to the hidden layer and in the calculation it will used output * (1.0 - output). The back-propagated error signal is accumulated and then used to determine the error for the neuron, Where error_j is the error signal from the jth neuron in the output layer, weight_k is the weight that connects the kth neuron to the current neuron and output is the output for the current neuron.

update weight:
to update the weight. The main calculation is that weight = weight - learning_rate * error * input, Where weight is a given weight, learning_rate is a parameter that you must specify, error is the error calculated by the backpropagation procedure for the neuron and input is the input value that caused the error.
Hyperparameters:
hidden_layer_size:the node in the hidden layer, it will effect the time and the performance
output_size:the node in the output layer can be 1 or 2
epoch:determine the number of times that updating the network
learning_rate: Learning rate controls how much to change the weight to correct for the error.

Discussion:

Challenge:in the mlp implementation no matter what I change the hidden_layer_size, epoch and the learning_rate the result is always all 0. I think that it maybe is the Weight Vanishing and the Weight Exploding problem then just have to handle the given weight.
Result:
NB:
I think maybe is my implement problem, all the index compare will be the worst. And the result for the the fold will be all zero.while the all zero will have 70% accuracy is what I shocked.
Key insight: Based on Bayes' theorem, it assumes that the features used to describe an observation are conditionally independent.
improvement:
Maybe to change the probability calculation method.
KNN:
all the index except the auc and the recall are the best. The prediction in the fold seems a little bit great.
key insight: Classifies data points based on the majority class of their k-nearest neighbors in the feature space, it can be mislead in higher features.
improvement:
In this work, I don’t give the weight of the each row distance, maybe give the weight and the best k won’t be 9 and the index can be greater. Also, maybe drop column I a matter in it. But I think drop column will have expect and danger so I don’t do that. Maybe that would cause some effect.

MLP:

the auc and the recall are the best of the three. And the prediction is the most likely to come up with a one. Maybe it is about the Hyperparameters I change or my computer can't afford more layers or the ore nodes in the layer. the best Hyperparameters I test will be hidden layer size: [3,2], epoch will be 500, learning rate will be 0.01

Key insight: A type of artificial neural network with multiple layers of nodes (neurons) and non-linear activation functions, in expect it must be the best of the three, due to its complexity can be very high.

Improvement: maybe have to go to the work station in school to run this code, to extend its limitation, or improve the weight vanishing problem or the implementation of the back_propogaion.