Mini-Project Number 1. What Breast cancer will recur?

**Note: We have 2 pdf files: Documentation and the PDF file code with its examples (application file). When referring to the PDF file, our intention is the application file (the additional PDF file in the folder).

Authors

- **Eli Haimov (ID. 308019306)**
- **Warda Essa (ID. 208642793)**
- **Omar Essa (ID. 315535435)**

Documentation

The first thing we did was uploading the data as requested, we have 198 patient . there are approximately 151 patients that did not recur; and 47 that recurred (you can see the data in pages 1-3 in PDF file).

We know from the column descriptions provided with the dataset and from viewing the data that there are two non-numeric columns: The second and the last column are typed as “object” these two columns should have some non-numeric values within.

We should also talk about missing values such as “Lymph node status” which is also typed as object. The WPBC dataset contains a lot of missing values (specifically in the Lymph Nodes Removed column). Data could be missing due to a variety of reasons. Primarily it could be due to:

1. Missing at Random - Propensity for a data point to be missing is not related to the missing data, but it is related to some of the observed data.
2. Missing Completely at Random - The fact that a certain value is missing has nothing to do with its hypothetical value and with the values of other variables.
3. Missing not at Random - Two possible reasons are that the missing value depends on the hypothetical value (e.g. People with high salaries generally do not want to reveal their incomes in survey s) or missing value is dependent on some other variable’s value (e.g. Let’s assume that females generally don’t want to reveal their ages! Here the missing value in age variable is impacted by gen der variable) In the first two cases, it is safe to remove the data with missing values depending upon their occurrences, while in the third case removing observations with missing values can produce a bias in the model. So if you look at the data without there missing values : (1)-46 and (-1)-148

As for the functions we used (Adaline algorithm):

- `Get_train_n_test_len()` is a function for getting the train and test length
- `data_split()` is a method built to split data onto 2 options: some for the test and the rest to training.
- `Standardization()` function is a method to standardize values from one type to float type after updating or fixing. we also used the `mean()` function here to calculate mean/average of a given list of numbers.
- `Calc_actual()` is a function to calculate negative and positive results.
- `check_predictions()` is a function that calculates the prediction table (further explaining in the code)
- `check_score()` is a fuction for checking and printing the scores (true positive, false negative, …)
- `train_n_comp_loss()` function for training the data and weights and compute loss.
- `Predict()` function for getting predictions.

At page 10 (PDF file) we implement the Adaline algorithm, we implemented the algorithm with different numbers of rounds to see what will happen and each time we printed the time it took for it to execute the code and a list of true positive, false ….

From these results we concluded that:

- If the number of rounds is too large -> it will take the algorithm a long time to converge and it impairs the exact calculation of the final results.
- If the learning rate is too small -> the algorithm will not be able to converge on the ideal number of rounds we set (1000).
- If the learning rate is too large -> the algorithm can miss the convergence point of the number of ideal turns we have set (1000).

Explaining:

1. first time we implemented Adaline algorithm with 1000 epoch time of execution was 0.17 sec with accuracy of 0.83 that means that we succeed in predicting 83% of the data correctly , precision :0.8 , recall = 0.29 , true positive: 4 , false negative: 10 , false positive:1 , true negative:51
2. second time we did it with 5000 epoch and the time of execution was doubled and got to 0.31 sec with the same accuracy of 0.83 (83% correctly), precision: 0.71 ,recall: 0.36 , true positive: 5 , false negative: 9 , false positive:2 , true negative:50
3. third time we did it with 1000 epoch but changed the leaning rate from 0.0001 to 0.001 time didn’t change from the first time but accuracy decreased to 0.79 (79% correctly), precision: 0.5 ,recall: 0.36 , true positive: 5 , false negative: 9 , false positive:5 , true negative:47

From this data we can conclude:

That the more epochs we have the longer it will take to execute the code. The number of epoch doesn’t affect the accuracy rate, instead changing the learning rate does. The more epochs we have the more loss will decrease.

Adaline cross-validation

The validation process is the process of deciding whether the numerical results quantifying hypothesized relationships between variables, are acceptable as descriptions of the data.

Why use it? Cross-validation can be used to compare the performances of different predictive modeling procedures. For example, suppose we are interested in optical character recognition, and we are considering using either support vector machine (SVM) or k-nearest neighbors (KNN) to predict the true character from an image of a handwritten character. Using cross-validation, we could objectively compare these two methods in terms of their respective fractions of misclassified characters. If we simply compared the methods based on their in-sample error rates, the KNN method would likely appear to perform better, since it is more flexible and hence more prone to overfitting compared to the SVM method.

Cross-validation can also be used in variable selection Suppose we are using the expression levels of 20 proteins to predict whether a cancer patient will respond to a drug. A practical goal would be to determine which subset of the 20 features should be used to produce the best predictive model. For most modeling procedures, if

we compare feature subsets using the in-sample error rates, the best performance will occur when all 20 features are used. However under cross-validation, the model with the best fit will generally include only a subset of the features that are deemed truly informative. We divided the data to three subsets of data for training and testing , we did so we don't get overfitting . we do that 3 times because when we work with a model its profitable to do it three times and take the average of them . by doing so we get more accuracy answer and when we start dealing with new data we have a better chance to get it right.

Adaline cross validation, we used this method our data set. The number of epoch was 10000 with learning rate of 0.0001, We divided data to three sets then trained and tested them: 1. First set executed in 6.13 sec with accuracy of 0.79 (79% correctly), precision: 0.5 ,recall: 0.36 , true positive: 5 , false negative: 9 , false positive:5 , true negative:47 2. second set executed in 0.72 sec with accuracy of 0.8 (80% correctly) , precision: 0.56 ,recall: 0.6 , true positive: 9 , false negative: 6 , false positive:7 , true negative:44 3. third set executed in 0.52 sec with accuracy of 0.79 (79% correctly) (Each set got its discretion in the PDF , code and the result) precision: 0.5 ,recall: 0.36 , true positive: 5 , false negative: 9 , false positive:5 , true negative:47 4. In page 21 (PDF file) we have all three sets together in the same view, we also added the average of the three sets. We also calculated the total time to execute the last model which was: 24.36 sec.

Back-Propagation: One hidden layer:

Functions we used in this algorithm: * `Train_one_hidden_layer()` function: training features (data) with one hidden layer. * `Train_two_hidden_layer()` function: training features (data) with two hidden layers. * `Predict_back_prop()` function: for getting predictions (backpropagation), it also includes an `eval()` method which returns the result evaluated from expression.

We run the code for one hidden layer: Our results and conclusions (with one hidden layer): 1. First case: n_epochs = 10000, lr = 0.00001, stddev() = 0.1 => Min Loss = 0.9170, `(0,14,0,52)` 2. Second case: n_epochs = 10000, lr = 0.0001, stddev() = 0.1 => Min Loss = 0.7283, `(0,14,0,52)` 3. Third case: n_epochs = 10000, lr = 0.001, stddev() = 0.1 => Min Loss = 0.6170, `(0,14,0,52)` 4. Fourth case: n_epochs = 10000, lr = 0.01, stddev() = 0.1 => Min Loss = 0.3514, `(4,10,1,51)` , acc= 0.83 5. Fifth case: n_epochs = 10000, lr = 0.01, stddev() = 0.1 => Min Loss = 0.2980, `(4,10,0,52)` , acc= 0.85

- Our best result: `(4,10,0,52)` = true positive: 4 , false negative: 10 , false positive:0 , true negative:52

We can see that the standard deviation value, choosing number to hidden layer and choosing a learning rate plays a significant role in finding the convergence point and getting the maximum result accuracy.

`(4,10,0,52)` => It can be seen that this is the better solution because our algorithm matched its prediction accurately to the target results of cancer recurrence prediction and cancer recurrence prediction. In other results such as `(0,14,0,52)` and `(4,10,1,51)` , for example, we can see that our algorithm has made ## some predictions different from the target results. This is the reason we cited the factors (number of epochs, learning rate, standard deviation, etc.) that, if we do not precisely target them as accurately as possible, that may affect non-convergence to the minimum point and reach the most accurate result.

**Note: It is important to note that in the beginning, we trained our training data well (66 percent of the data) to train the algorithm and predict the test data as the target.

Back-Propagation: Two hidden layers:

We run backpropagation with two hidden layers now, 66% of dataset length to train and the other 34% to test (128 in training and 66 in test).

We created sets of train and test features by splitting the data according to the percent separated by function date_split().

We standardize set of train and test features from bits-type to float, then we did the training.

You will see the results in the PDF file (page 29), a graph describing the loss decrease according to the number of epoch. The result of training data with two hidden layers was:

- 0.88 accuracy (88% correctly) , precision: 0.88 ,recall: 0.5 , true positive: 7 , false negative: 7 , false positive:1 , true negative:51.

This tells us that training data and applying backpropagation algorithm with two hidden layers gave us better correct result than doing so with only one hidden layer.

One hidden layer have 85% correct prediction of the data while with two hidden layers we had 88% correct prediction of the data.

Back-propagation cross validation

- With one hidden layer: Same as before here 66% of the dataset length to train and the rest 34% to test.

Same as in Adaline algorithm we created 3 sets for train and test features (data) and just as normal back-propagation we ran the algorithm on the three sets.

We described the training error in a graph that contains the three sets and the total.

1. Result of the first set: 0.82 accuracy (82% correctly) , precision: 1.0 ,recall: 0.14 , true positive: 2 , false negative: 12 , false positive:0 , true negative:52
2. Result of the second set: 0.79 accuracy (79%correctly) , precision: 0.57 ,recall: 0.27 , true positive: 4 , false negative: 11 , false positive:3 , true negative:48
3. Result of the third set: 0.8 accuracy (80% correctly) , precision: 1.0 ,recall: 0.07 , true positive: 1 , false negative: 13 , false positive:0 , true negative:52

Each set got its discretion in the PDF, code and the result in page 31 we have all three sets together in the same view, we also added the total of the three sets.

- With two hidden layers: Same as training with one hidden layer we had 66% of the dataset length to train and 34% to test.

We described the training error in a graph that contains the three sets and the total.

1. Result of the first set: 0.83 accuracy (83% correctly), precision: 0.64 ,recall: 0.5 , true positive: 7 , false negative: 7 , false positive:4 , true negative:48
2. Result of the second set: 0.83 accuracy (83%correctly) , precision: 0.64 ,recall: 0.6 , true positive: 9 , false negative: 6 , false positive:5 , true negative:46
3. Result of the third set: 0.74 accuracy (74% correctly) , precision: 0.36 ,recall: 0.29 , true positive: 4 , false negative:10 , false positive:7 , true negative:45

**Note: we didn't add the tables and graphs in this paper, we didn't want to add many more unnecessary pages, but all the tables that includes the results and the performance of the training set and the testing set can be seen in the PDF file we are submitting. The PDF file includes in it the actual code with all the results we talked about in this paper and the results we didn't. It's an easy PDF file which can help anyone understand everything we did (codes and functions with descriptions and detailed results.