Neural Networks Ex1- What Breast cancer will recur?

Report:

We have used Python(with several libraries such as NumPy pandas Seaborn Sklearn)

All of the codes are attached in py files.

(i) Time it took to train the get the final results:
Perceptron: 0.06 sec
Adaline: 0.02 sec
Backpropagation: 0.15 sec

We can see that our implementation of adaline is faster than the others.

The reason for this is the fact that adaline algorithm is more simple then backpropogation(one neuron with simple algorithm vs complicated algorithm with NN) , and in perceptron we stoped the fit at the middle because the data is non linear so the results may vary.

Parameters Adjustment:

Perceptron:

In order to find the best parameters we have run the model each time with different values for the learning rate and for the amount of iterations.

The best parameters:

- **eta** = 0.01
- **n_iter** =10

Adaline:

In order to find the best parameters we have run the model each time with different values for the learning rate and for the amount of iterations.

The same method was used also in the perceptron algorithm.

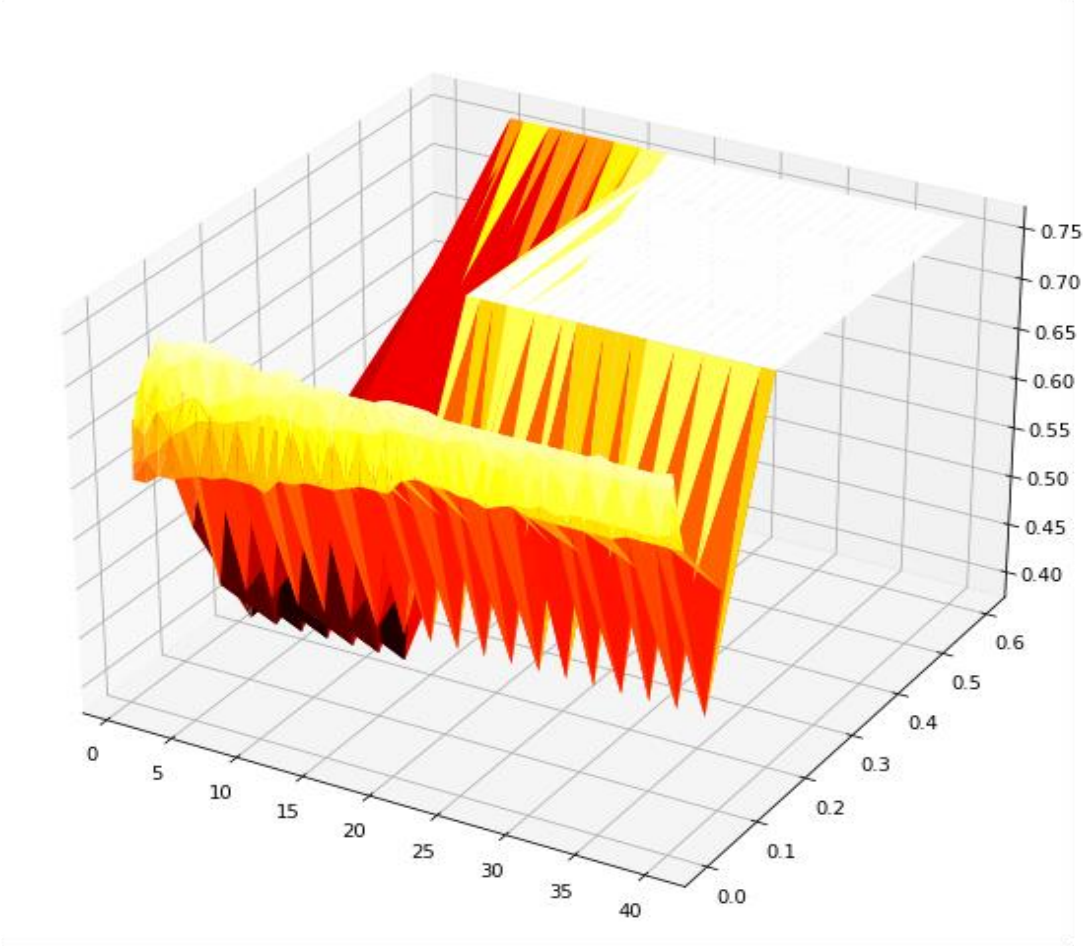
We made a graph shown the highest score and the best parameters.

Code:

```
##Finding best parameters
i_range = range(1, 41)
l_range = [0.01, 0.005, 0.001, 0.0005, 0.0001, 0.1, 0.025, 1.0,
0.003]
zdata = []
xdata = []
ydata = []
for i in i_range:
    #check every l_rangechecking it in adaline and add cv score ,
iter_n , eta  to list
    for l in l_range:
        ada = Adaline(n_iter=i, eta=l, random_state=random_state)
        ada.fit(X_train, y_train)
```

```
        y_predict = ada.predict(X_test)
        cv = cross_val_score(ada, X, y, cv=3,
scoring='accuracy').mean()
        zdata.append(cv)
        xdata.append(i)
        ydata.append(l)
matplotlib
##show it using 3D view
fig = plt.figure(figsize=(10, 10))
#https://matplotlib.org / mpl_toolkits / mplot3d / tutorial.html
ax = plt.axes(projection='3d')
#we choose the color cmap
#for more options see:
https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
ax.plot_trisurf(xdata, ydata, zdata, cmap='viridis');
fig.show()
#print best score
x = 0
for i in zdata:
    if (max(zdata) == i):
        break
    x += 1
print("best score: ", zdata[x])
print("n_iter: ", xdata[x])
print("eta: ", ydata[x])
```

```
C:\Users\user\PycharmProjects\NNEx1\venv\Scripts\python.exe C:/Users/user/PycharmProjects/NNEx1/adalinealgo.py
            Training dataset  Test dataset  Total
Recurrent                 34            13     47
Nonrecurrent              98            53    151
best score:  0.7626262626262627
n_iter:  5
eta:  0.6
```



As we can see in the graph, the number of iteration grows the score is lower(unless eta is high you can see the color get lighter), and the eta is around the average.

The best parameters:

- **eta** = 0.6
- **n_iter** =5

Backpropagation:

In order to find the best parameters we run Grid whose gets some optional parameters and returns the best parameters for the highest accuracy.
GridSearch is a function from sklearn library :" https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html"

The best parameters:

- **hidden_layer_sizes** = (32, 16), two hidden layers with 32 neurons in the first and 16 neurons in the second.
- **for one layer size:** (45)

- **activation** (Activation function for the hidden layer= ' relu'
- **solver** (The solver for weight optimization) = 'adam' (-refers to stochastic gradient descent

- **max_iter** (Maximum number of iterations. The solver iterates until convergence or this number of iterations) = 50, .

Performance on the training set and on the testing set:

Train-Test Split: we split the data into 2 classes : 'Train'-66%, 'verify'-33%.

Cross Validation:

We run it 3 times and calculate the average and std of the 3 results.

Perceptron:

```
Accuracy:  0.7557251908396947
The time for training the model is: 0.06 sec
accuracy of test:
0.8059701492537313
The time for Testing the model is: 0.00 sec
Total time for this model is: 0.06 sec
```

Adaline:

```
best score:  0.7575757575757575
n_iter:  5
eta:  0.005
The time for model fitting is: 0.02 sec
Accuracy of Adaline (train): 76.52 percents
Accuracy of Adaline (split): 74.24 percents
Accuracy of Adaline (cross-validation): 75.25 percents
Standart Deviation of Adaline (cross-validation) 3.98 precents
The time for getting all the model results: 0.09 sec
```

Backpropagation:

```
C:\Users\user\PycharmProjects\NNEx1\venv\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:582:
  warnings.warn(
Best parameters found:
 {'activation': 'tanh', 'hidden_layer_sizes': (32, 16), 'max_iter': 500, 'solver': 'sgd'}
The time for fitting model is: 0.11 sec
Accuracy of  Backpropagation train is:  86.36 precents
Accuracy of  Backpropagation split is:  77.27 precents
Accuracy of  Backpropagation cross-validation is: 78.79 precents
The time for getting all model results is: 0.46 sec

Process finished with exit code 0
```

- One layer size:

```
Best parameters found:
 {'activation': 'relu', 'hidden_layer_sizes': 45, 'max_iter': 50, 'solver': 'adam'}

Process finished with exit code 0
```

Score:

```
C:\Users\user\PycharmProjects\NNEx1\venv\Scripts\python.exe C:/Users/user/PycharmProjects/NNEx1/backpropagation.py
The time for fitting model is: 0.04 sec
Accuracy of  Backpropagation train is:  77.27 precents
Accuracy of  Backpropagation split is:  80.30 precents
Accuracy of  Backpropagation cross-validation is: 76.77 precents
The time for getting all model results is: 0.15 sec
True Positives (TP): 3
True Negatives (TN): 50
False Positives (FP): 3
False Negarives (FN): 10
```

```
#finding best parmeters going through all parameter space we made using gread search
#for more info https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selecti
clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=3)
clf.fit(X_train, y_train)
# Best paramete set
print('Best parameters found:\n', clf.best_params_)
```

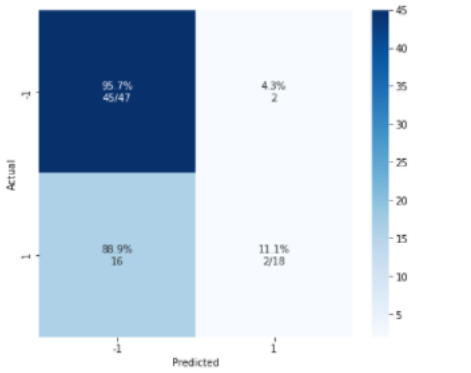
```
Best parameters found:
 {'activation': 'logistic', 'hidden_layer_sizes': 15, 'max_iter': 500, 'solver': 'adam'}
```

```
#training model
st = time.time()
clf = MLPClassifier(solver='adam',hidden_layer_sizes=(15),max_iter=500,activation='logistic',random_state=random_state)
clf.fit(X_train, y_train)
print("The time for fitting model is: %.2f sec" % (time.time() - st))

#training results
y_predict = clf.predict(X_train)
print("Accuracy of  Backpropagation train is:  %.2f precents" % (metrics.accuracy_score(y_train, y_predict) * 100))

#testing results
y_predict = clf.predict(X_test)
print("Accuracy of  Backpropagation split is:  %.2f precents" % (metrics.accuracy_score(y_test, y_predict) * 100))
cv = cross_val_score(clf, X, y, cv=3, scoring='accuracy').mean()
print("Accuracy of  Backpropagation cross-validation is: %.2f precents" % (cv * 100))

#time in toal
bpTime = time.time() - st
print("The time for getting all model results is: %.2f sec" % bpTime)
```

```
The time for fitting model is: 0.24 sec
Accuracy of  Backpropagation train is:  81.40 precents
Accuracy of  Backpropagation split is:  72.31 precents
Accuracy of  Backpropagation cross-validation is: 75.27 precents
The time for getting all model results is: 0.92 sec
```

- The accuracy of the training set implies that the model doesn't fit perfectly to the training data. We can refer, that the cause is that we have limited data.
- The cross-validation has higher results than the split to train-test. This is due to the fact that cross-validation is more reliable and also we train it few times then we get more accurate results.

```
#make confusion materix for this model
cm = confMat(y_test, y_predict, clf.classes_)
```



Code & Explanations:
We have used Python libraries:
MLPClassifier implementes Backpropagation algorithm.

Import the data from 'wpbc.data' using pandas

The description of the data:

1. Number of instances: 198
2. Number of attributes: 34 (ID, outcome, 32 real-valued input features)
3. Attribute information
   1) ID number
   2) Outcome (R = recur, N = nonrecur)
   3) Time (recurrence time if field 2 = R, disease-free time if field 2 = N)
   4-33) Ten real-valued features are computed for each cell nucleus:

      a) radius (mean of distances from center to points on the perimeter)

      b) texture (standard deviation of gray-scale values)

      c) perimeter

      d) area

      e) smoothness (local variation in radius lengths)

      f) compactness (perimeter^2 / area - 1.0)

      g) concavity (severity of concave portions of the contour)

      h) concave points (number of concave portions of the contour)

      i) symmetry

      j) fractal dimension ("coastline approximation" - 1)

```
data = pd.read_csv('wpbc.csv')
```

```
data.head()
```

	a	b	c	d	e	f	g	h	i	j	...	z	aa	ab	ac	ad	ae	af	ag	ah	ai
0	119513	N	31	18.02	27.60	117.50	1013.0	0.09489	0.1036	0.1086	...	139.70	1436.0	0.1195	0.1926	0.3140	0.1170	0.2677	0.08113	5.0	5
1	8423	N	61	17.99	10.38	122.80	1001.0	0.11840	0.2776	0.3001	...	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	3.0	2
2	842517	N	116	21.37	17.44	137.50	1373.0	0.08836	0.1189	0.1255	...	159.10	1949.0	0.1188	0.3449	0.3414	0.2032	0.4334	0.09067	2.5	0
3	843483	N	123	11.42	20.38	77.58	386.1	0.14250	0.2839	0.2414	...	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	2.0	0
4	843584	R	27	20.29	14.34	135.10	1297.0	0.10030	0.1328	0.1980	...	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	3.5	0

5 rows × 35 columns

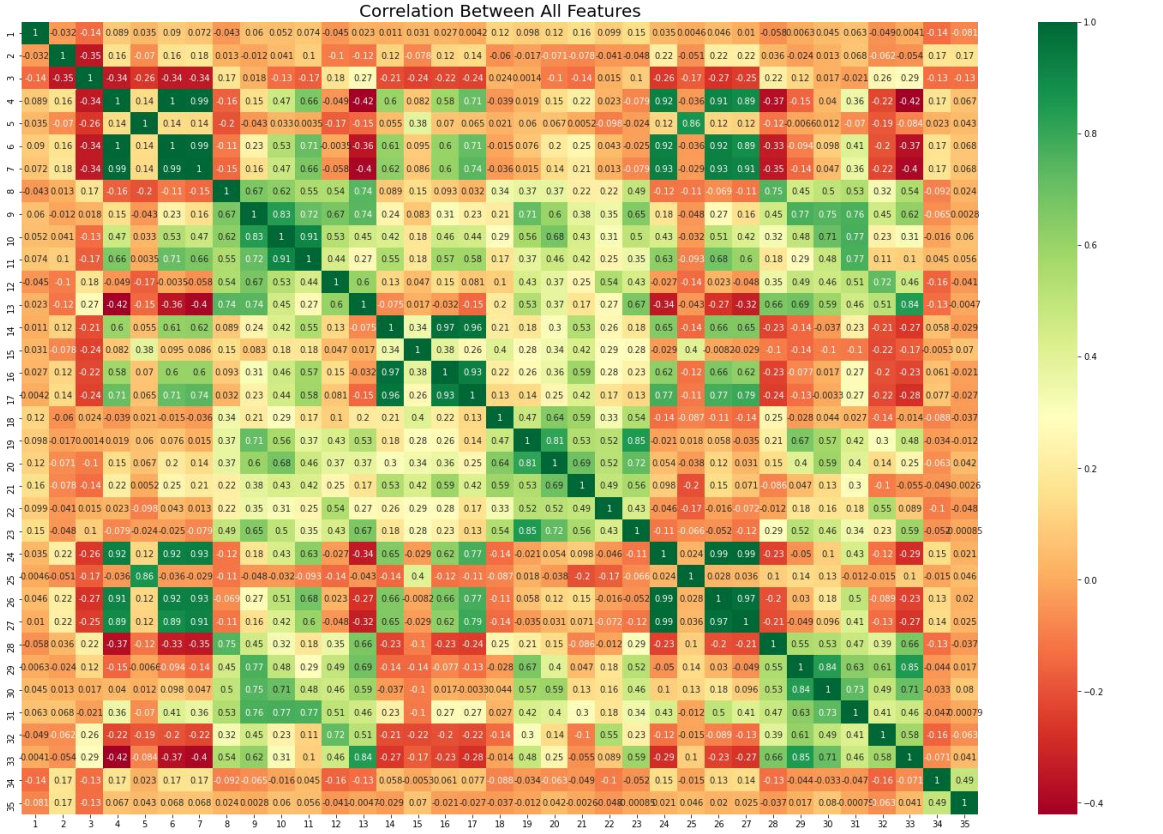
explain about filtering data:
we used panda to read the csv into dataframe.

After reading the data we filtered the data in a way that all of the features are presented with numbers and also we dropped few columns that are not needed for the training. After that we splatted the data as mentioned above.

Additionally to all above in Adaline and backpropagation we try to filter the data more. We made a correlation ,ap of all features given in'wpbc.csv' to try to find the best features for this algorithms.

```
Code: fig = plt.figure(figsize=(25,17))
sns.heatmap(data.corr(),annot=True,cmap='RdYlGn')
plt.title('Correlation Between All Features', size=20)
plt.show()
```

*we use plt to show the cor map and use pandas correlation function https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

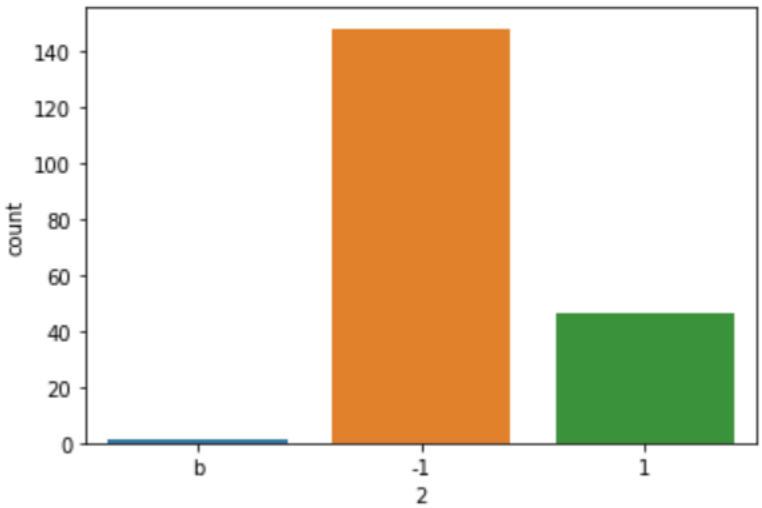


The first coloumn is the patients ID and the third coloumn(lables)
According to the map we can see that 1,9,19 has no cor with target coloumn, 2,6,7,2,26 has good correlation with coloumn 4 , 3 ,17 has good correlation with 14 so we choose to give up all this columns at this algorithms.

Pics of R/N amount:

```
sns.countplot(x=data[2],data=data)
data[2].value_counts()
```

```
-1    148
1      46
b       1
Name: 2, dtype: int64
```



From the pic above we see that we have 148 patients the are Non-recurrence and 46 the recurrence, we use this information in dividing to train test classes.

Code:

```
scaler = StandardScaler()
X = scaler.fit_transform(data.iloc[:, 1:].values)
y = data.iloc[:, 0].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=random_state)
freqs = pd.DataFrame({"Training dataset": [(y_train == 1).sum(),
(y_train == -1).sum()],
                      "Test dataset": [(y_test == 1).sum(), (y_test
== -1).sum()],
                      "Total": [(y_train == 1).sum() + (y_test ==
1).sum(),
                                (y_train == -1).sum() + (y_test == -
1).sum()]},
                      index=["Recurrent", "Nonrecurrent"])
freqs[["Training dataset", "Test dataset", "Total"]]
```

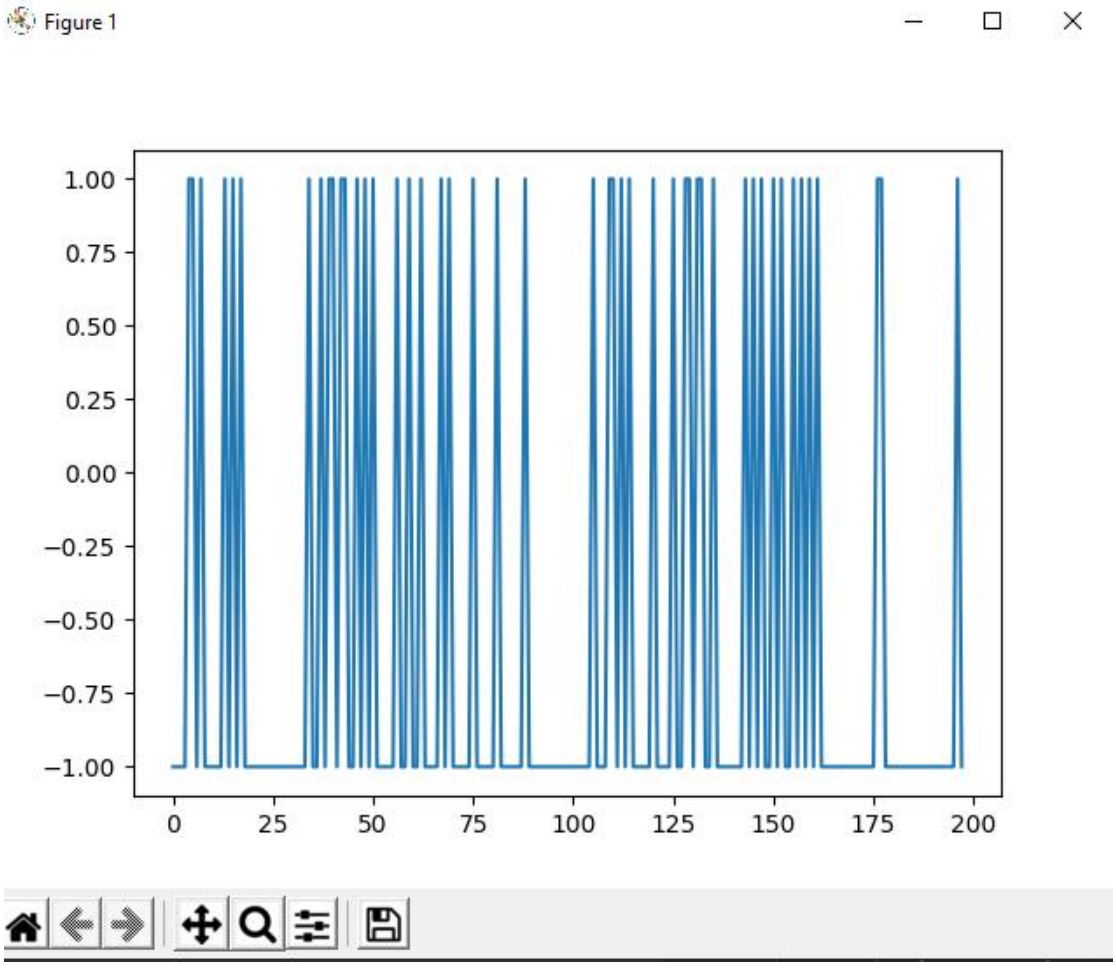
```
C:\Users\user\PycharmProjects\NNEx1\venv\Scripts\python.exe C:/Users/user/PycharmProjects/NNEx1/adalinealgo.py
             Training dataset  Test dataset  Total
Recurrent                  34            13     47
Nonrecurrent               98            53    151
best score:  0.7626262626262627
```

Cormap pic

Perceptron Alorithm:

```
Accuracy:  0.7557251908396947
The time for training the model is: 0.06 sec
accuracy of test:
0.8059701492537313
The time for Testing the model is: 0.00 sec
Total time for this model is: 0.06 sec
```

in the perceptron algorithm we noticed that because the data is non-linear the results may vary – some times the new weights improved the accuracy and in other cases it made it worse so we decided to stop the algorithm when it reaches 75% accuracy to avoid it from getting worse.
Also attached plot that shows that the data is non linear- the labels (1,-1) are scattered all over the dataset.



Adaline algorithm:

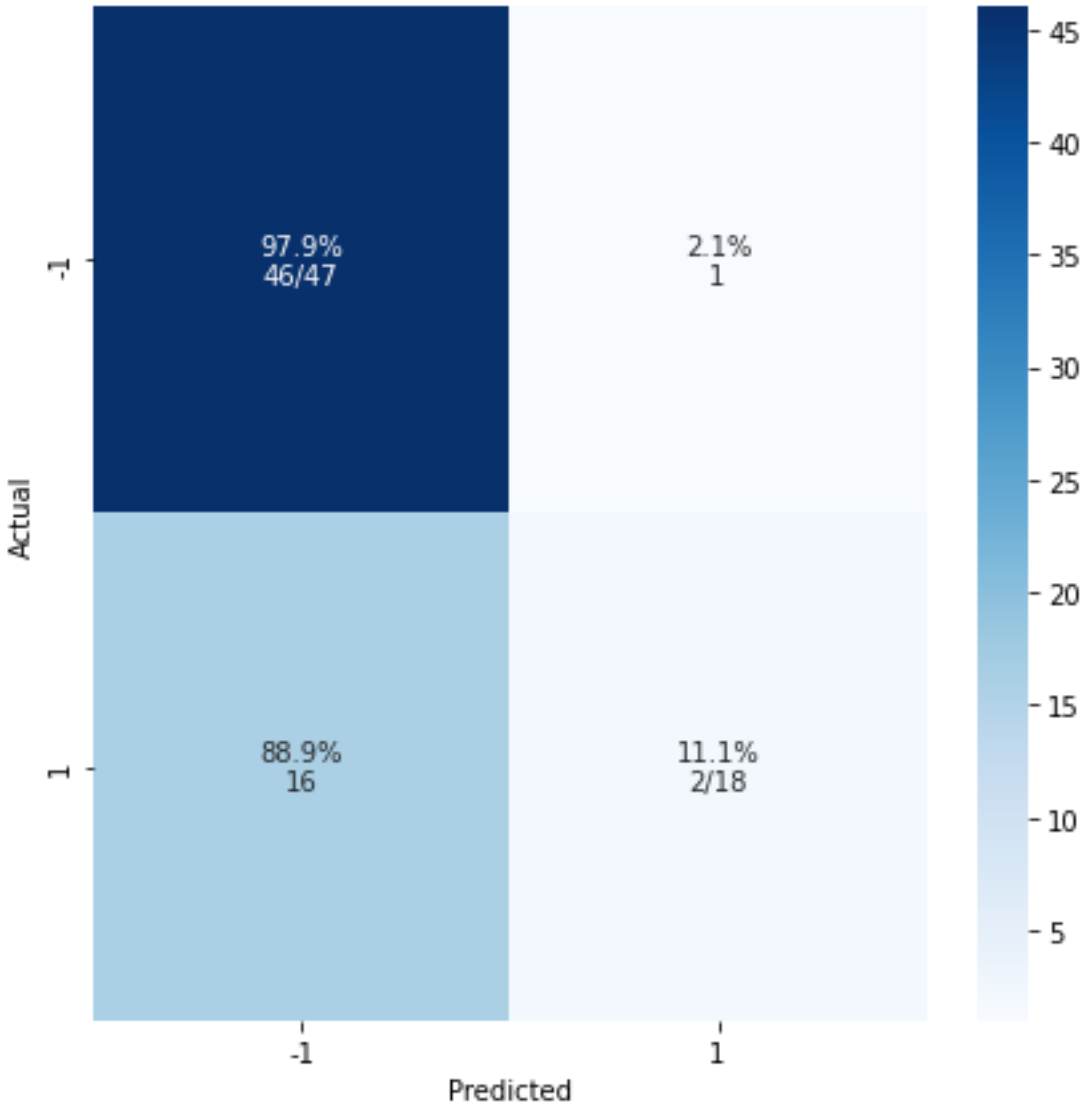
Score:

```
best score:  0.7575757575757575
n_iter:  5
eta:  0.005
The time for model fitting is: 0.02 sec
Accuracy of Adaline (train): 76.52 percents
Accuracy of Adaline (split): 74.24 percents
Accuracy of Adaline (cross-validation): 75.25 percents
Standart Deviation of Adaline (cross-validation) 3.98 precents
The time for getting all the model results: 0.09 sec
```

Confusion matrix:

Code:

```
def confMat(y_true, y_pred, labels, figsize=(7,7)):
    #make conusion matrix usingg skl function
     #for more info https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.h
tml
    cm = confusion_matrix(y_true, y_pred, labels=labels)
    cm_sum = np.sum(cm, axis=1, keepdims=True)
    cm_perc = cm / cm_sum.astype(float) * 100
    annot = np.empty_like(cm).astype(str)
    nrows, ncols = cm.shape
    for i in range(nrows):
        for j in range(ncols):
            c = cm[i, j]
            p = cm_perc[i, j]
            if i == j:
                s = cm_sum[i]
                annot[i, j] = '%.1f%%\n%d/%d' % (p, c, s)
            elif c == 0:
                annot[i, j] = '0'
            else:
                annot[i, j] = '%.1f%%\n%d' % (p, c)
    cm = pd.DataFrame(cm, index=labels, columns=labels)
    cm.index.name = 'Actual'
    cm.columns.name = 'Predicted'
    fig, ax = plt.subplots(figsize=figsize)
    sns.heatmap(cm, annot=annot, fmt='', ax=ax, cmap='Blues')
    #plt.savefig(filename)
    plt.show()
    return cm
#make confusion matrix for adaline run
cm = confMat(y_test, y_predict, [-1,1])
```



Backpropogation Algorithm:

MLPClassifier implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.
The advantages of Multi-layer Perceptron are:
Capability to learn non-linear models.
Capability to learn models in real-time (on-line learning) using partial_fit.
For more about mlp: "https://scikit-learn.org/stable/modules/neural_networks_supervised.html"

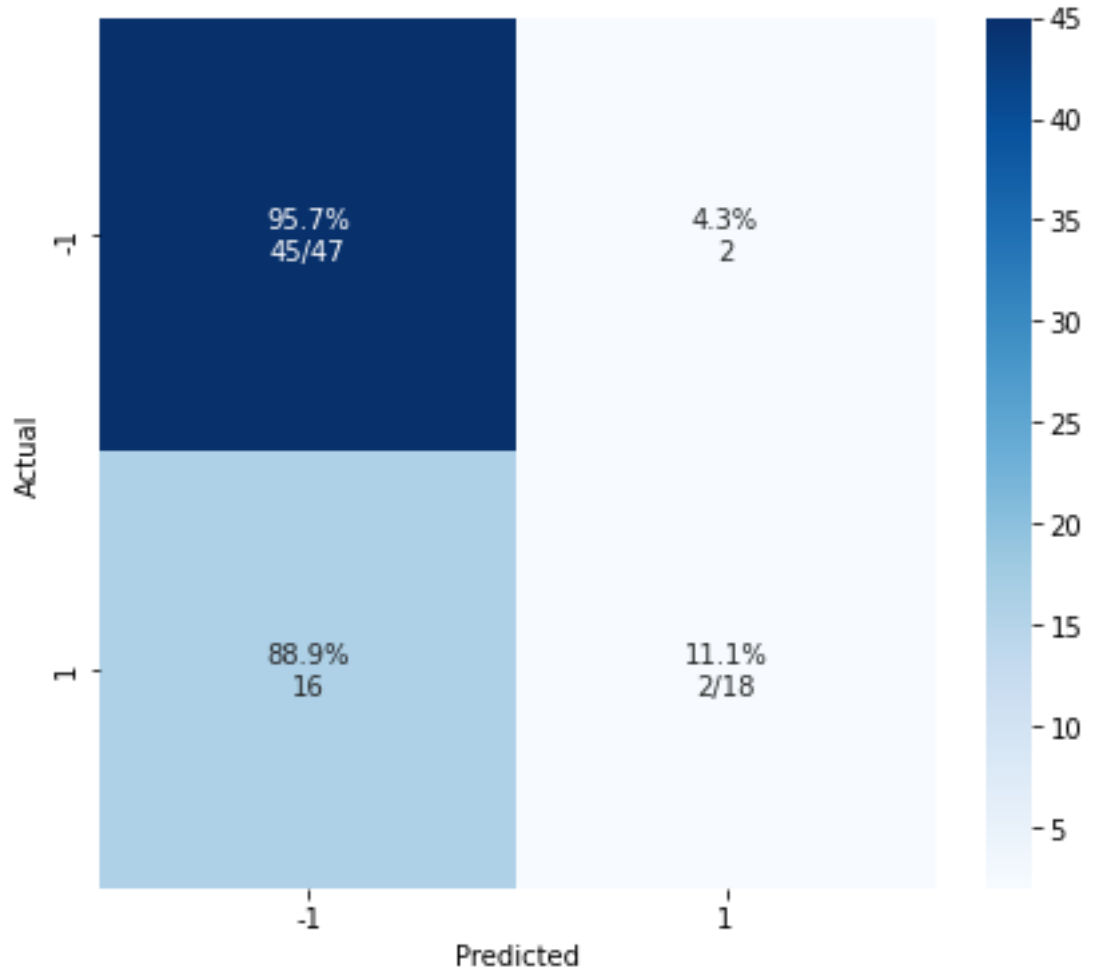
We use mlp classifier on our 'train test' data we explained above.

For more info about MLPClassifier**:** **https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html**

BackPropogation results:

```
C:\Users\user\PycharmProjects\NNEx1\venv\Scripts\python.exe C:/Users/user/PycharmProjects/NNEx1/backpropagation.py
The time for fitting model is: 0.04 sec
Accuracy of  Backpropagation train is:  77.27 precents
Accuracy of  Backpropagation split is:  80.30 precents
Accuracy of  Backpropagation cross-validation is: 76.77 precents
The time for getting all model results is: 0.15 sec
True Positives (TP): 3
True Negatives (TN): 50
False Positives (FP): 3
False Negarives (FN): 10
```

Pic corelation map



Summary and discussion of process/ problems:

The goal of our model is to predict if a patient is recurrent or not.

The target column is "b" which hold the information that represents if the patient was recurrent or not. *R* =recurrent ,*N* = nonrecurrent.

We switched the R to 1 and N to -1.

All of the models are not accurate enough – about 75% and that is after we tried to choose the best data and features as possible in order to get the best results.

Whenever we had missing data in the dataset we added data to the patient by using the average value for the column in order that the missing data wont impact the results.

We have a lot of features in the dataset, so we built a correlation map that shows the correlation between the different features and between them to the labels in order to see if there are features that are most likely not going to change the results so we can remove them to minimize the dataset.

We chose to remove features that have no correlation with the labels and also features with no correlations at all.

We observed that if we had more data and in particular, more data of recurrent patients- the chances of getting higher accuracy of the model grows up because the process will reflect more data about them that is missing in the current dataset.

We found that the best result for this problem with this dataset was achieved using the

Cross-validation and BackPropogation model and was 76.77%