Assignment 1 ANN

Dimitrios Passos, 20235090

GitHub link: https://github.com/DimitrisPassos/ANN1

Breast Cancer Dataset

Starting with the breast cancer dataset, we pass the data in a variable called cancer. Then variable x gets the input features while y gets the results. We define the test size to be equal to 30%, which we can do since the dataset is small. Then we split our data in four parameters, variables that contain train, are used to train the model, while variable containing test, are used for evaluation.

```
cancer = datasets.load_breast_cancer()

x = cancer.data
y = cancer.target

test_size = 0.3
x_train, x_test, y_train, y_test = x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=test_size)
```

Because the data have values that are not in the same scale, we will use the StandardScaler function, provided from sklearn, to normalize the features.

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler() # Normalize the range of the values

sc.fit(x_train)

x_train_std = sc.transform(x_train)
x_test_std = sc.transform(x_test)

print("Unique label: {0}".format(np.unique(y)))
```

The print helps us validate the unique target values of the dataset, zero and one. Zero is malignant while one is benign.

Next, we train our models. Since we are doing a comparison, we will train both logistic regression and perceptron model.

First let us analyze the variables.

```
n_iter = 40
eta0 = 0.5
random_state = 10
l1_ratio = 0.5
tol = 1e-10

ppn = Perceptron(max_iter=n_iter, eta0=eta0,
random_state=random_state, penalty='elasticnet', l1_ratio=l1_ratio,
tol=tol)
lrr = LogisticRegression(max_iter=n_iter, random_state=random_state,
solver='saga', penalty='elasticnet', l1_ratio=l1_ratio, tol=tol)

lrr.fit(x_train_std, y_train)
ppn.fit(x_train_std,y_train)

y_pred = ppn.predict(x_test_std)
y_pred_lrr = lrr.predict(x_test_std)
print("accuracy perceptron:
{0:.2f}%".format(accuracy_score(y_test,y_pred) * 100))
print("accuracy logistic regression:
{0:.2f}%".format(accuracy_score(y_test,y_pred_lrr) * 100))
```

Variable n_iter is used for deciding how many iterations our models will do. I found that for anything over 1000, we do not get much value and we will combine it with a small learning rate eta0. This seems to make our perceptron more stable and return better accuracy score. By using the random state, we can make our model shuffle the training data, also by giving an integer value we can assure same results between our models. Since we are using a penalty of elastic net and we are setting the l1 ration to be 0.5, we practically also use l2 ratio of 0.5. If l1 ration is 0 then we only use l2 ratio, if l1 ration is one we only use l1 ration, but since we set the l1 ratio to be 0.5, we use l1 ratio and l2 ration of 0.5. We also use a tolerance of 1e-3 for faster convergence. For bigger or smaller values of tolerance, based on the assignment, I got worse results.

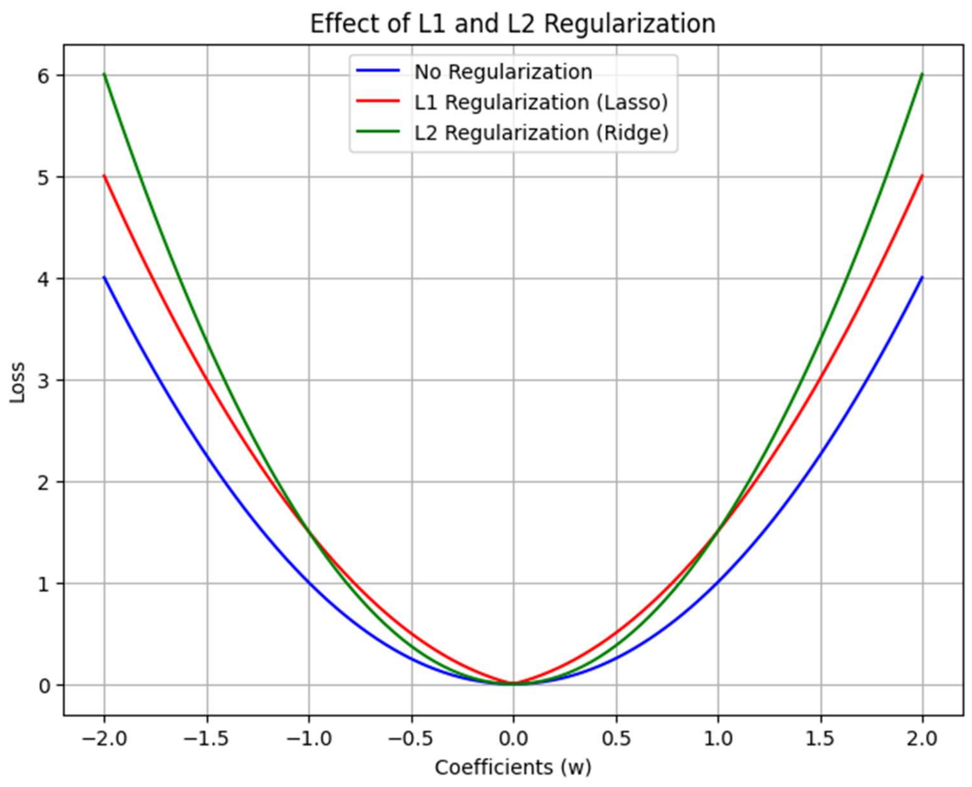


Figure 1 Effect of L1 and L2 Regularization

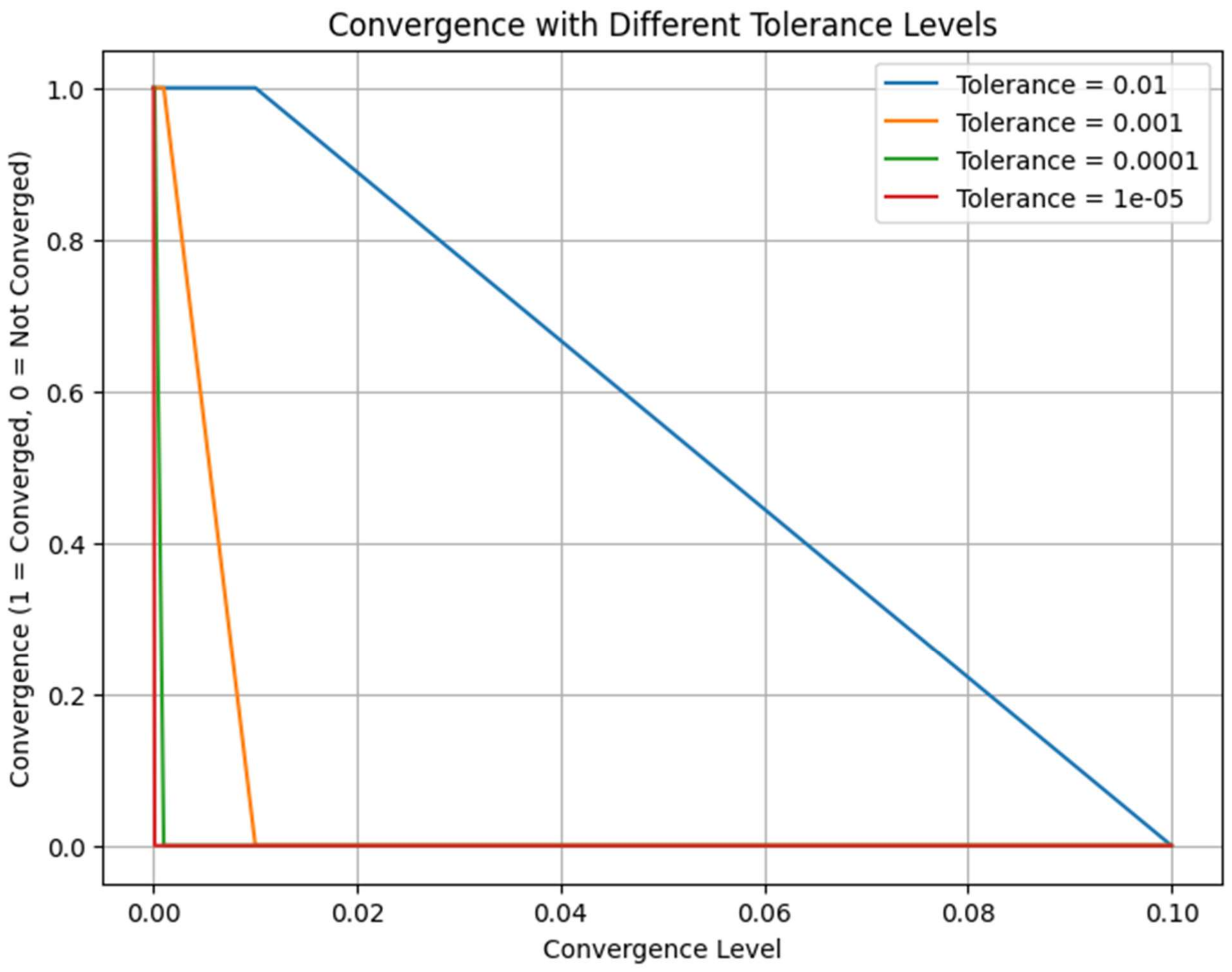
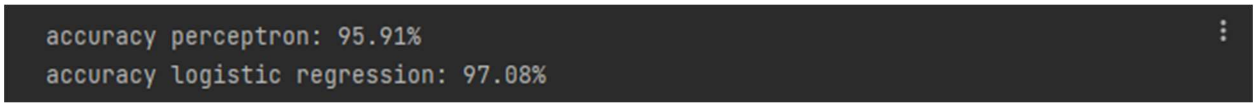


Figure 2 Convergence with Different Tolerance Levels

Finally, we train the model and print the accuracies for each classifier and print their confusion matrixes.



```
accuracy perceptron: 95.91%
accuracy logistic regression: 97.08%
```

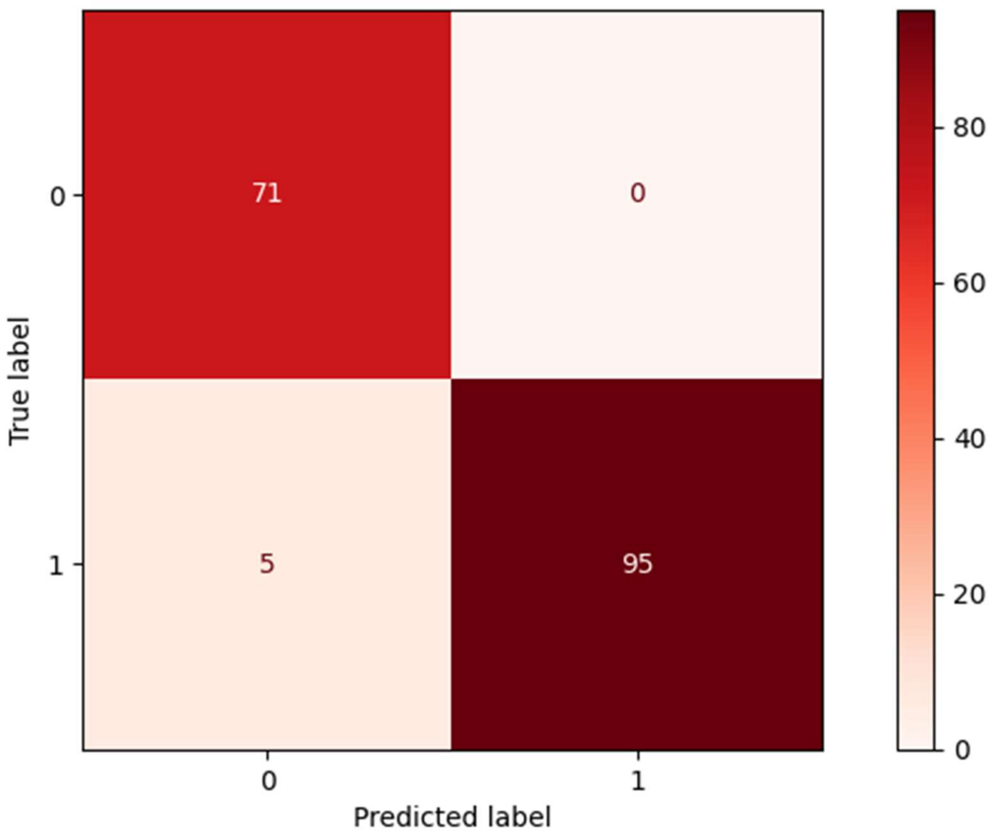


Figure 3 Perceptron confusion matrix

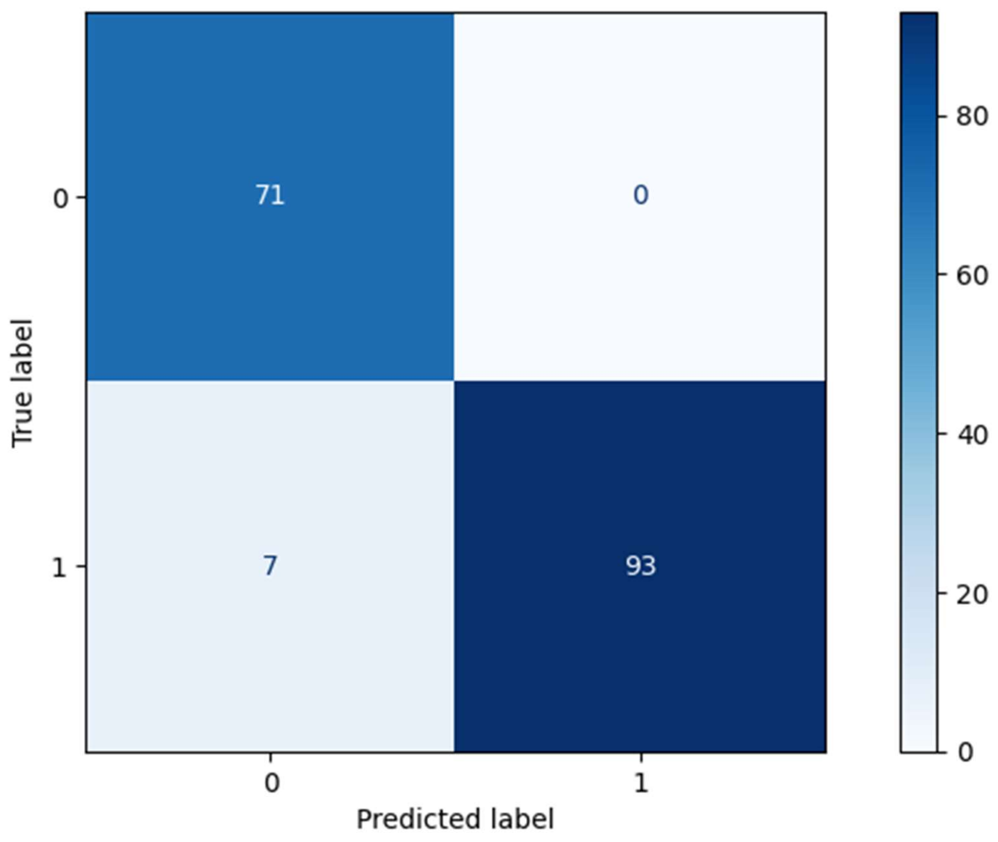


Figure 4 Logistic Regression confusion matrix

For the specific dataset, the accuracies after many tries and iterations seem to be very close. In this example, logistic regression seems to be more reliable than perceptron, but the overall performance seems to be close. Since this dataset requires a binary classification solution, we can see that both models give equally good results but, in the multiclass classification we will see that logistic regression is more stable.

Iris Dataset

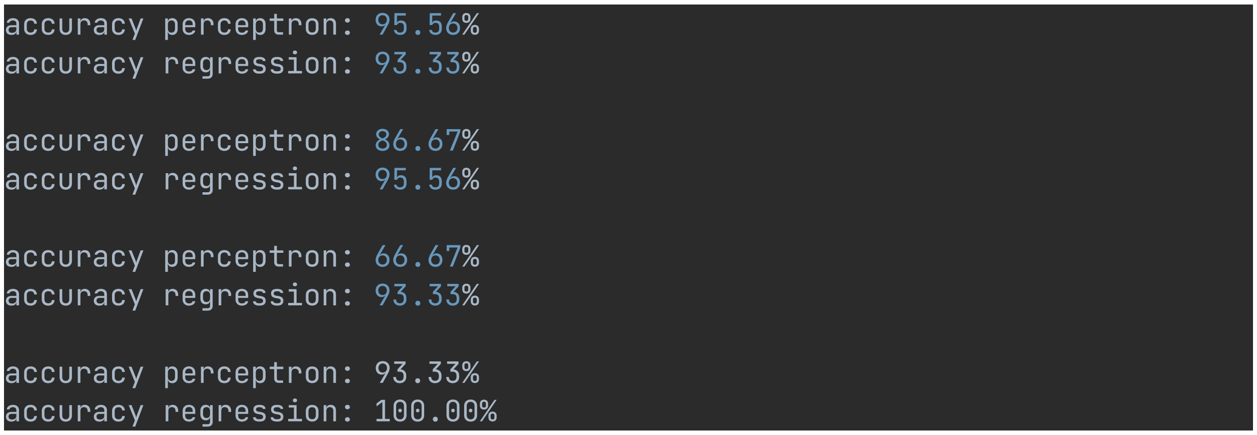
Since most of the code is the same, we will skip to the differences rather than explaining the whole code again.

In this example we will select petal length and petal width because it seems to give the most value for our model after experimenting with different features.

```
x_train_std = x_train_std[:, [2, 3]] # Feature selection
x_test_std = x_test_std[:, [2, 3]]
```

```
n_iter = 10000
eta0 = 0.3
random_state = 42
l1_ratio = 0.5
tol = 1e-6

ppn = Perceptron(alpha=0.01,max_iter=n_iter, eta0=eta0,
random_state=random_state, penalty='elasticnet', l1_ratio=l1_ratio,
fit_intercept=True, tol=tol)

lrr = LogisticRegression(penalty='elasticnet', l1_ratio=l1_ratio,
tol=tol, fit_intercept=True, random_state=random_state,
max_iter=n_iter, solver='saga')

lrr.fit(x_train_std, y_train)
ppn.fit(x_train_std,y_train)

y_pred = ppn.predict(x_test_std)
y_pred_lrr = lrr.predict(x_test_std)
print("accuracy perceptron:
{0:.2f}%".format(accuracy_score(y_test,y_pred) * 100))
print("accuracy regression:
{0:.2f}%".format(accuracy_score(y_test,y_pred_lrr) * 100))
```

Again, these values seem to give more stability to the perceptron. The problem is that since a single perceptron is a binary classifier, the accuracy of the model is unreliable. In this dataset I increased the number of iterations and learning rate, to get better results from the perceptron. I also reduced the error value, since bigger values were giving much lower accuracy score. Regression is more flexible and powerful for multiclass classification than a single perceptron, but in the future, we will build a network of perceptrons that will surely be more reliable and accurate.



```
accuracy perceptron: 95.56%
accuracy regression: 93.33%

accuracy perceptron: 86.67%
accuracy regression: 95.56%

accuracy perceptron: 66.67%
accuracy regression: 93.33%

accuracy perceptron: 93.33%
accuracy regression: 100.00%
```

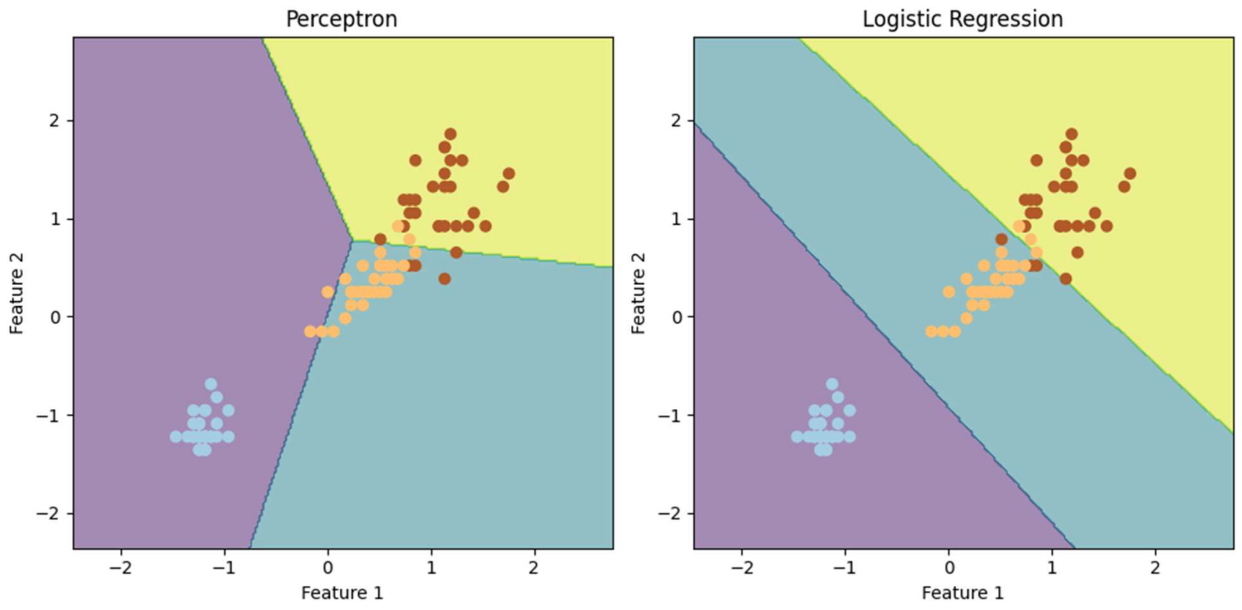


Figure 5 Perceptron and Logistic Regression hit and misses.

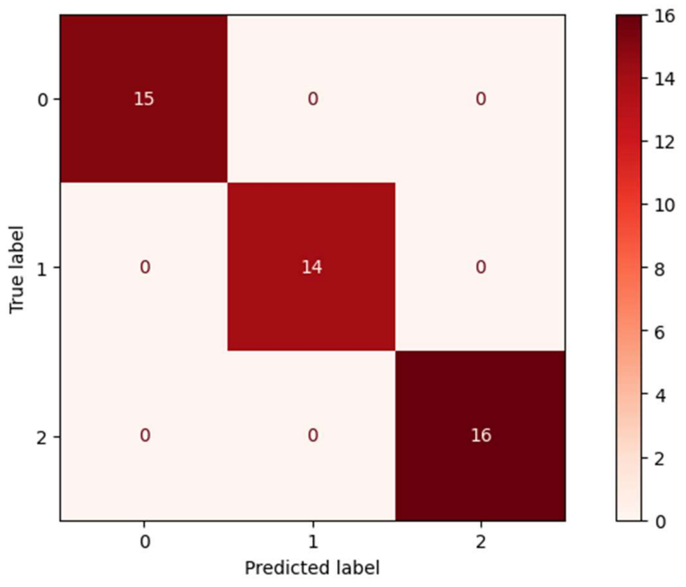


Figure 6 Logistic Regression confusion matrix

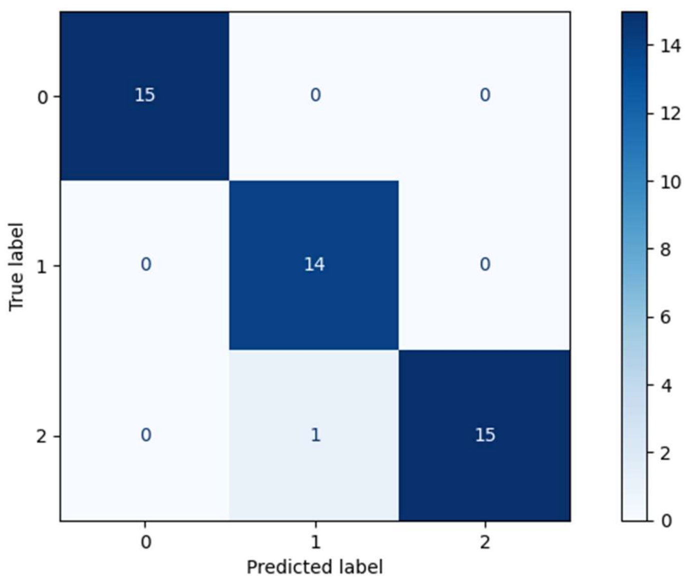


Figure 7 Perceptron confusion matrix