

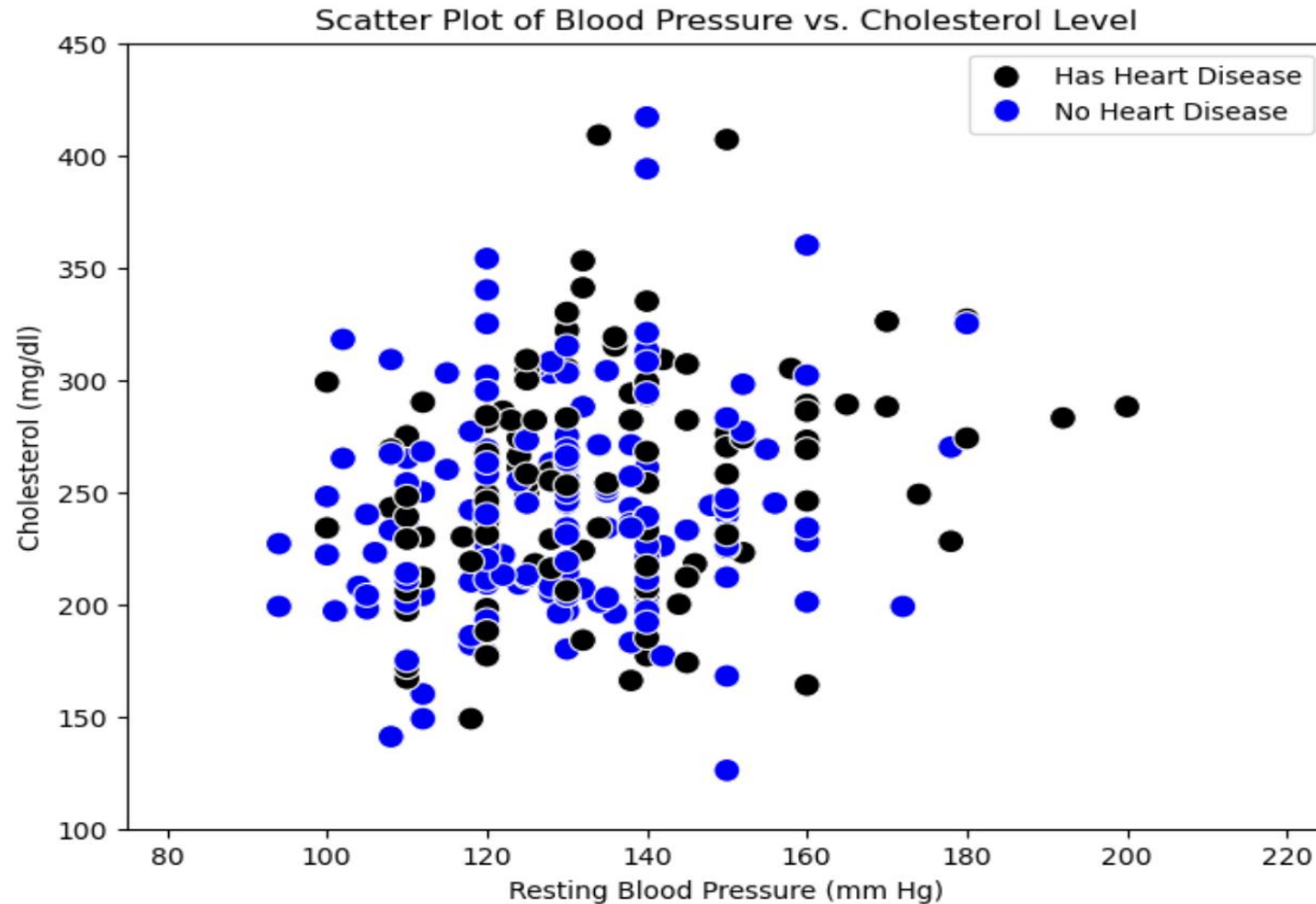
Logistic Regression Project Report

Heart Disease

By

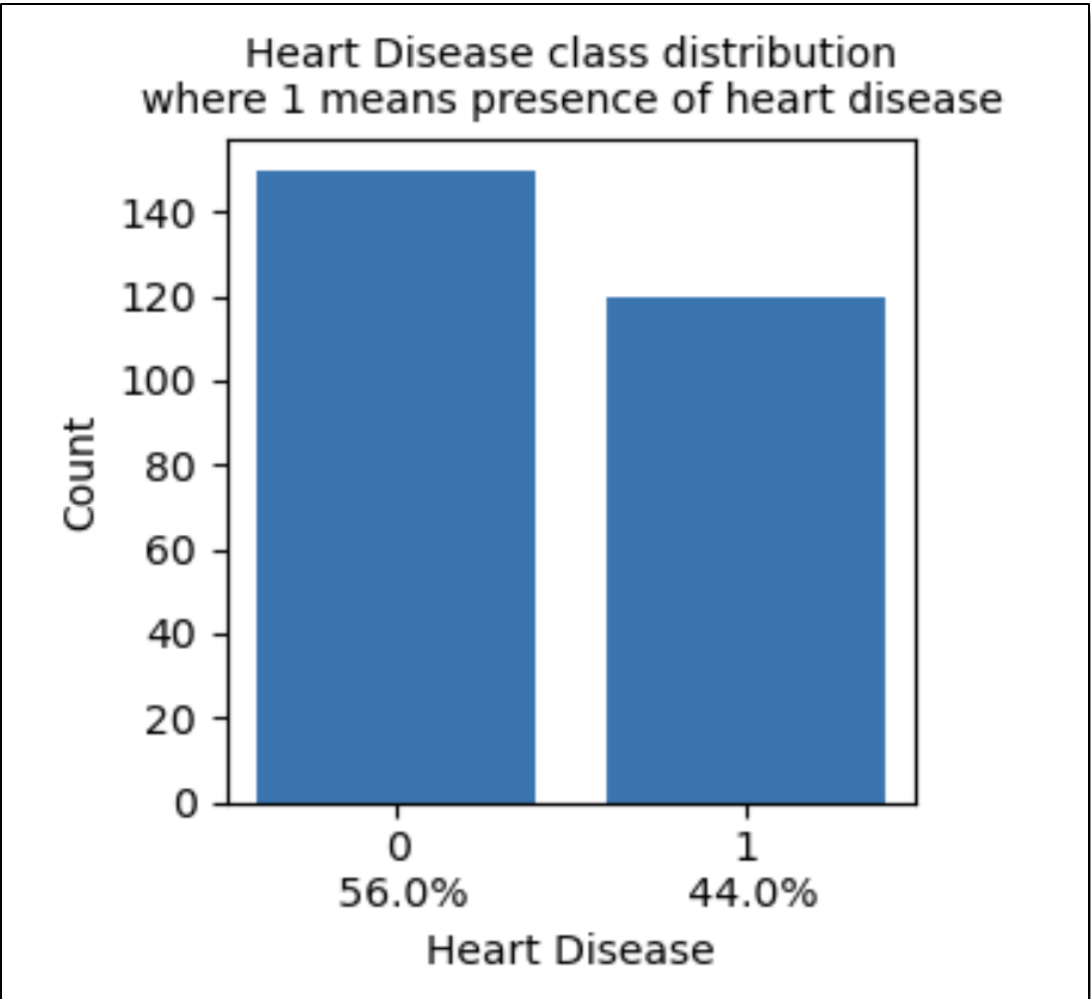
Roger Swartz

Logistic Regression: Speaks to the need to assign the likelihood of an event especially when there is no clear individual causal factor, but multiple independent variables associated with the likelihood of an event.



Gathered Extensive Variables on Patient Data to Run a Logistic Regression for Heart Disease:

The data have 270 rows and 14 columns	
column names:	
age_yr	Further Detail
sex_M_F	
chest_pain_value	
resting_BP_mm_Hg	
cholesterol_mg_dl	
fasting_blood_sugar_high	
ECG_value	
max_HR	
exercise_angina	
ST_depresssion_exercise	
ST_slope_peak	
number_vessels_involved	
defect_diag	
heart_disease	



Purpose

- Develop a Logistic Regression to Predict the Likelihood of Heart Disease
- Accomplish this with an extensive number of health indicators.
- Use both quantitative variables and categorical variables
- Use all features to build the Logistic Regression Model.
- Use Machine Learning Methods to Maximize the performance of the Logistic Regression
- Evaluate with a Confusion Matrix.
- Develop a Probability Classifier for Resting Blood Pressure (mm Hg) vs. Cholesterol Level (mg/dl)

The Model Building Cycle Using Linear Classifiers

- (a) train/test split
- (b) create an object of the class associated with the algorithm to be used--in this case LogisticRegression
- (c) build an actual model using the "fit" method from the class (applied to the training set)
- (d) predict with the built model using the "predict" method from the class (training set and test set)
- (e) compute performance metrics (in this case, accuracy) for the training and test predictions

```
Xtrain, Xtest, ytrain, ytest = train_test_split(dflog[['cholesterol_mg_dl', 'resting_BP_mm_Hg']],  
                                              dflog['heart_disease'],  
                                              train_size = 0.80,  
                                              random_state = 42)
```

▼ LogisticRegression ⓘ ?
LogisticRegression(C=1000, max_iter=500, solver='liblinear')

▼ LogisticRegression ⓘ ?
LogisticRegression(max_iter=500, solver='newton-cg')

The Model Building Cycle Using Linear Classifiers

```
Xtrain, Xtest, ytrain, ytest = train_test_split(dflog[['cholesterol_mg_dl', 'resting_BP_mm_Hg']],
                                              dflog['heart_disease'],
                                              train_size = 0.80,
                                              random_state = 42)
```

LogisticRegression

```
LogisticRegression(C=1000, max_iter=500, solver='liblinear')
```

```
print("Classification Report for Training Data")
print(classification_report(ytrain, classifier.predict(Xtrain)))
```

Classification Report for Training Data

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.61	0.78	0.68	117
1	0.61	0.40	0.48	99

accuracy			0.61	216
macro avg	0.61	0.59	0.58	216
weighted avg	0.61	0.61	0.59	216

```
print("Classification Report for Test Data")
print(classification_report(ytest, classifier.predict(Xtest)))
```

Classification Report for Test Data

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.63	0.82	0.71	33
1	0.45	0.24	0.31	21

accuracy			0.59	54
macro avg	0.54	0.53	0.51	54
weighted avg	0.56	0.59	0.56	54

LogisticRegression

```
LogisticRegression(max_iter=500, solver='newton-cg')
```

```
print("[Training] Accuracy score: (y_lstrat, y_predict_training)",
      f'{accuracy_score(y_lstrat, y_predict_training):.2f}')
```

[Training] Accuracy score: (y_lstrat, y_predict_training) 0.89

```
print("[Test] Accuracy score: (y_testlstrat, y_predict_test) [**note reversed order]",
      f'{accuracy_score(y_testlstrat, y_predict_test):.2f}')
```

[Test] Accuracy score: (y_testlstrat, y_predict_test) [**note reversed order] 0.81

Discovering an Optimal C Value (Regularization Parameter) for newton-cg linear Classifier

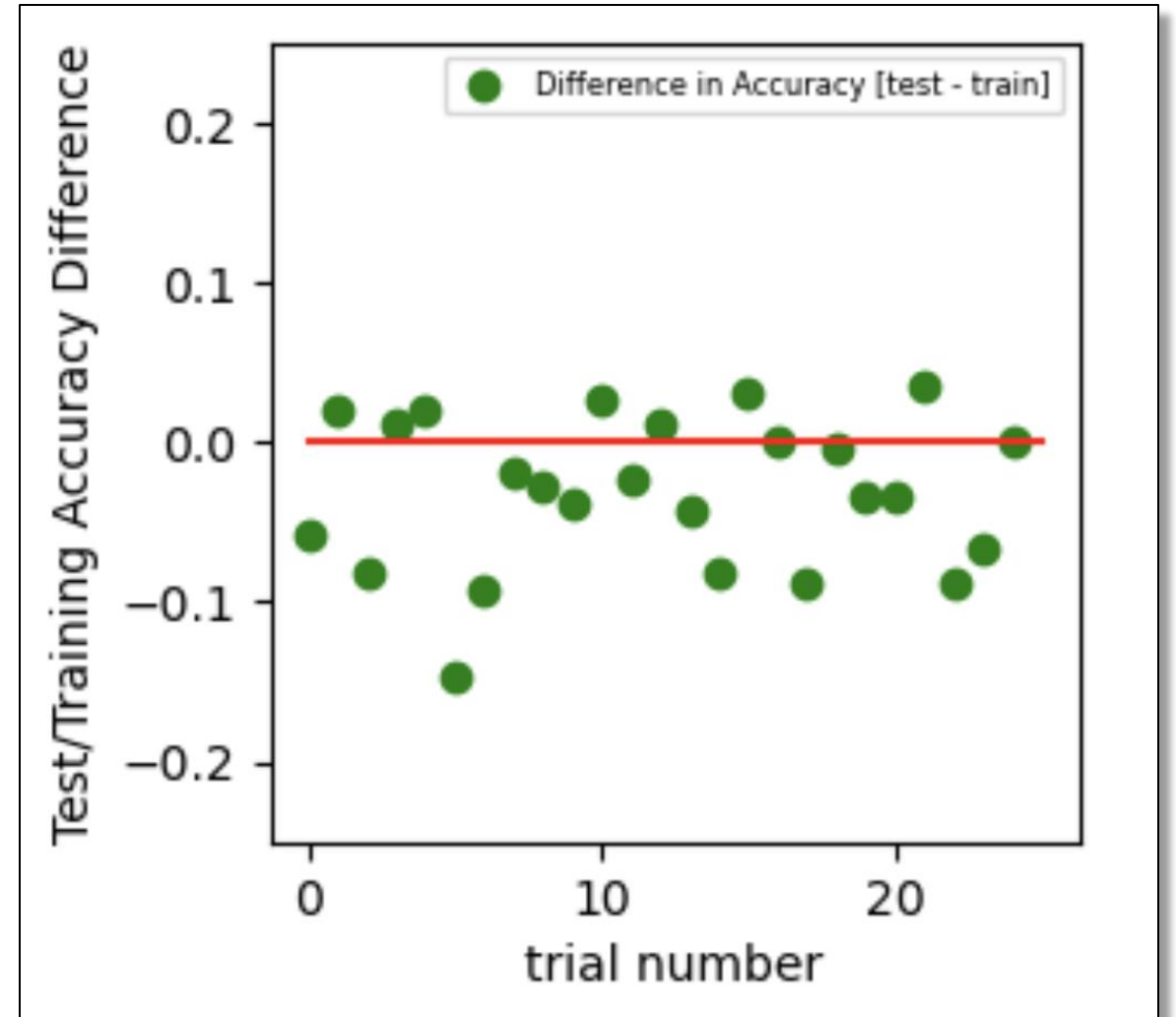
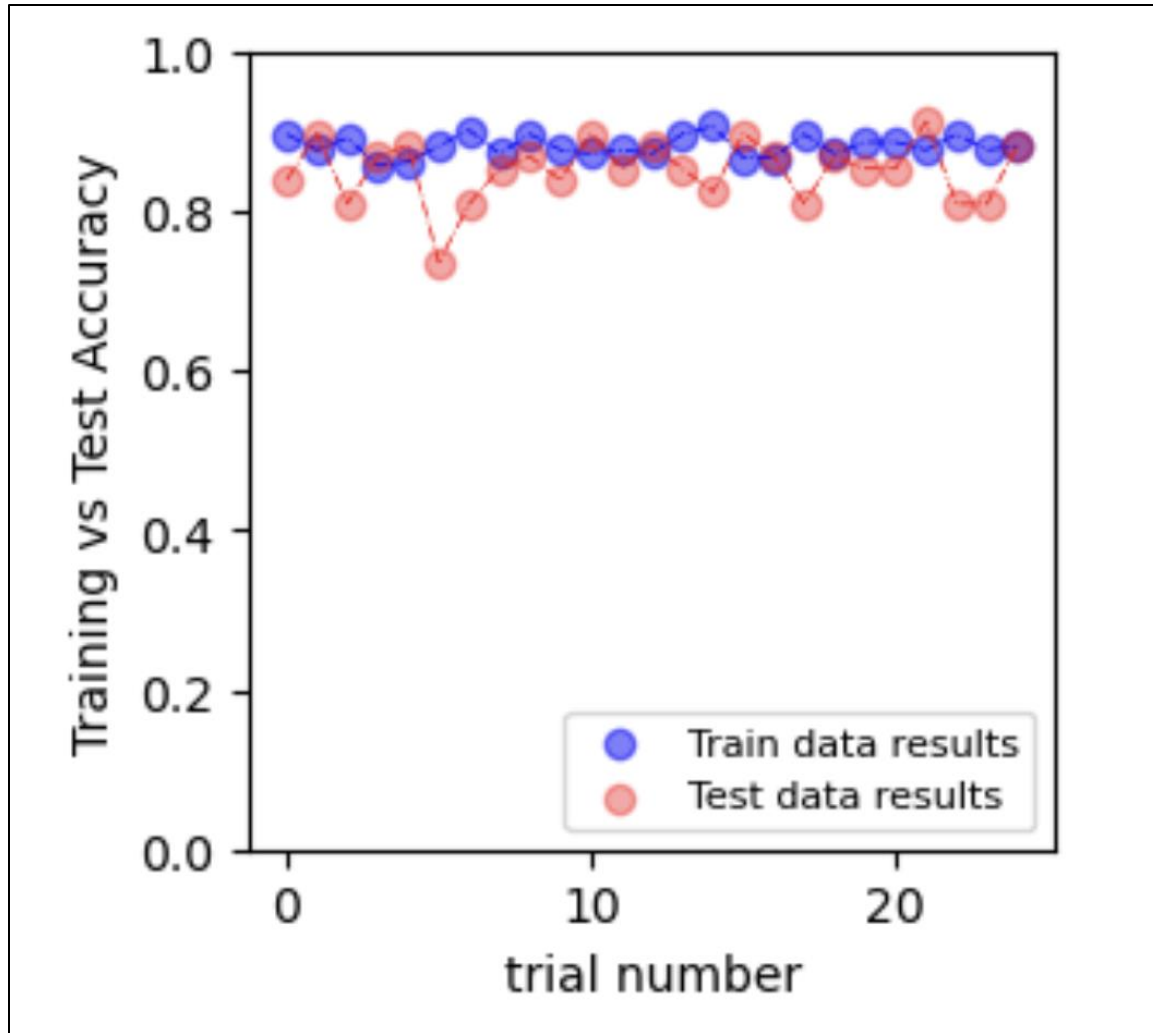
```
LogisticRegression
LogisticRegression(max_iter=500, solver='newton-cg')
```

[Training Accuracy for C=1]					
	precision	recall	f1-score	support	
0	0.88	0.91	0.89	112	
1	0.88	0.84	0.86	90	
accuracy			0.88	202	
macro avg	0.88	0.88	0.88	202	
weighted avg	0.88	0.88	0.88	202	

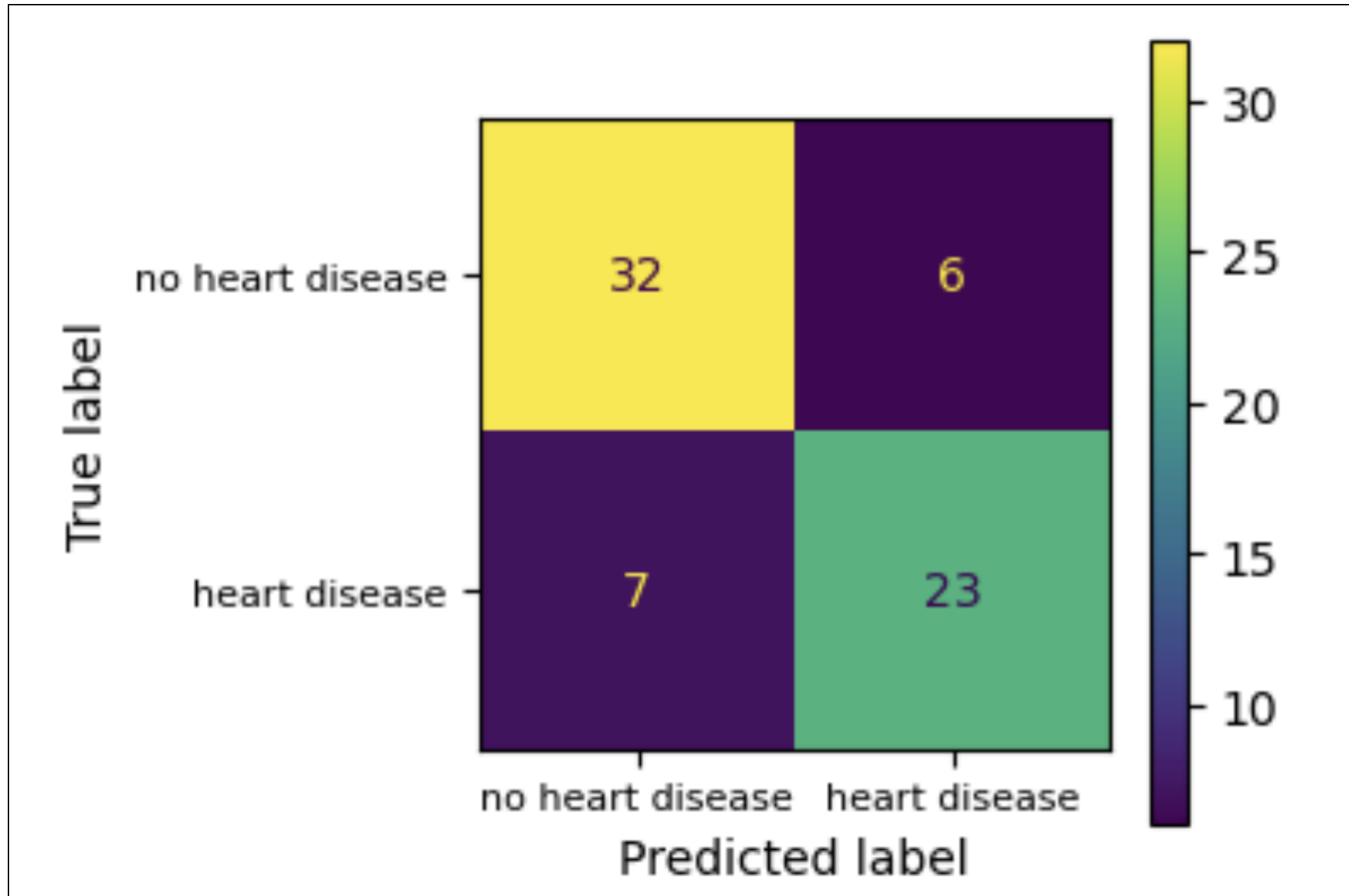
[Training Accuracy for C=10]					
	precision	recall	f1-score	support	
0	0.89	0.93	0.91	112	
1	0.91	0.86	0.88	90	
accuracy			0.90	202	
macro avg	0.90	0.89	0.89	202	
weighted avg	0.90	0.90	0.90	202	

```
print("Logistic Regression Weighted coefficients for linear:", best_estimator.coef_)
Logistic Regression Weighted coefficients for linear: [[-0.0067498  0.45052203  0.41301556  0.01076425  0.002496 -0.2150339
 0.14821705 -0.01590518  0.31524247  0.22654272  0.14303985  0.61583979
 0.30112849]]
```

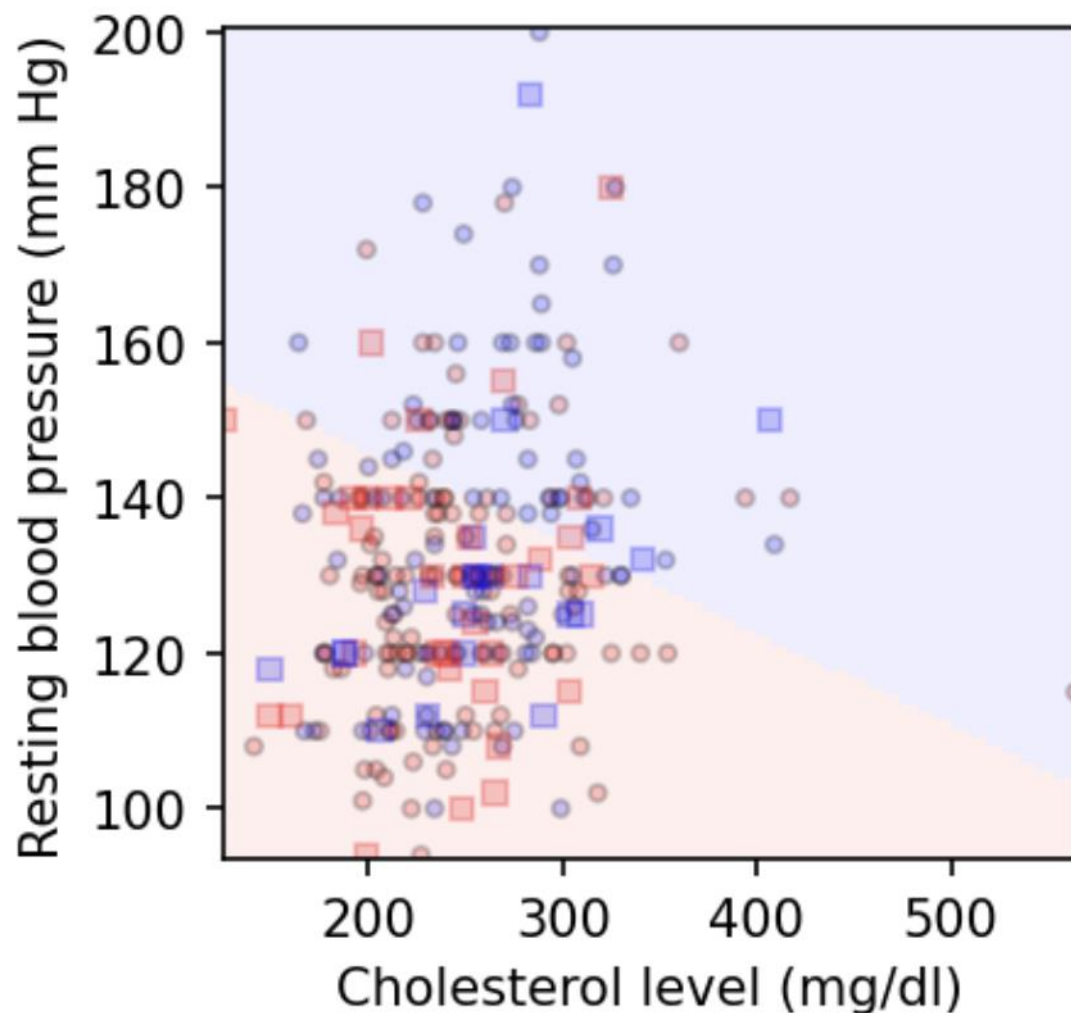
Reasonably Consistent Model Performance for newton-cg linear classifier



Confusion Matrix



Computed Decision Boundary:
Cholesterol Level (mg/dl) VS Resting Blood Pressure (mm Hg)
Red: Heart Disease | Blue: No Heart Disease
Circles: Training Set | Squares: Testing Set



```
plt.figure()
ax = plt.gca()
ax.set_ylabel('Resting blood pressure (mm Hg)')
ax.set_xlabel('Cholesterol level (mg/dl)')
ax.set_title('Computed Decision Boundary:\n ' +
             'Cholesterol Level (mg/dl) VS Resting Blood Pressure (mm Hg)' +
             '\n Red: Heart Disease | Blue: No Heart Disease' +
             '\n Circles: Training Set | Squares: Testing Set\n',
             fontsize = 10)
_ = points_plot(ax, Xtrain, Xtest, ytrain, ytest, classifier, alpha = 0.2)
```

Probability Classifier for Resting Blood Pressure (mm Hg) vs. Cholesterol Level (mg/dl)

