Abdulshaheed Alqunber (asq@bu.edu)
Anuj Jain (anuj12@bu.edu)
Karan Mago (kmago@bu.edu)
Wayne Snyder
CS237

Using a Logistic Regression Model to Predict Benignancy/Malignancy

I. Abstract

The aim of our project is to create a predictive model that would predict whether or not a person has breast cancer based on an input of feature vectors which are available from a FNA. We use a variety of functions to help reach our goal which include comparing the accuracy with and without logistic regression. The last function summarizes the whole project as it give the accuracy, sensitivity, specificity and precision of the model. By utilizing the model with the data given, we concluded that our model is 92.8% accurate. The sensitivity (true positive rate) and specificity (false positive rate) of our model are 84.4% and 96.8% respectively. Lastly, the precision of our model is 92.7%, meaning our model will return the same results about 9 out of 10 times.

II. Introduction

In this project we used logistic regression and machine learning techniques in order to analyze data to determine whether women have benign (not likely to spread) or malignant (likely to spread and posing a significant threat to one's' life) breast cancer or not based on different variables and their values. We were given data for 683 women, with information about their cells such as uniformity of cell size and cell shape. The model we created takes this data and predicts whether someone has breast cancer or not in the form of a value of either 0.0 (benign) or malignant (1.0).

III. Background

The basic error faced while using linear regression is that the result is from 1-100 which can make it hard to estimate whether or not person has benign or malignant cancer. The data we have is in the form on different criteria each ranging from 0-10 with 0 being the weakest presence and 10 being the strongest. The goal of our model is to be able to predict whether or not a person has benign cancer or malignant cancer, a linear model needs to estimate a threshold value from 0-100 which if exceeded determines if a person has cancer or not as compared to a logistic regression model which gives the response in terms of success and failure, the algorithm it uses is more efficient in this case. We hope to achieve at least values of 90% in all of our statistics for our predictions (accuracy, sensitivity, specificity, and precision).

IV. Modeling and Analysis

To create our model, we first had to make a sigmoid squashing function as seen in Figure 1, which let us implement linear regression into our project and relate our independent variables to our dependent variable. We then had to find out how to compute cost

properly as that helped set the counter that was used for the functions after that, evidence can be seen in Figure 2. The code for this function models that of the cost function used in logistic regression. After that we created a logistic regression class, which uses the predict, predict_class, and score functions as shown in Figure 3 to get the optimal parameters to obtain the lowest cost function value. We use this function to determine whether someone has a benign or malignant tumor. We then tested our model by using it with the data we were given to determine the accuracy of it which can be seen in Figure 4. This outputted the number of the people in the dataset whose tumor the model predicted correctly (returned as true) and incorrectly (returned as false). We split our dataset into three individual datasets, we then used 60% of the original dataset for training, 20% for validating that we found the best lambda and 20% for testing to evaluate the model as whole which can been seen in Figure 5. We then tested the accuracy of our model using different values of lambda to find which value gives us the greatest accuracy which can be seen in Figure 6. In the end using the further exploration function and mathematical equations to find certain statistics of our model, we were able to find the accuracy, specificity, sensitivity and precision of our model (which helped us determine the correctness of our model) which is shown in Figure 7.

V. Prediction

Our model is 92.8% accurate which shows that it is pretty accurate in predicting if a woman has breast cancer or not. The sensitivity of our model is 84.4%, which means 84.4% of the time people with a malignant tumor are correctly identified to have the tumor. The specificity of our model is 96.8%, meaning 96.8% of the time people with a benign tumor are correctly identified as such. Lastly, our precision is 92.7%, which tells us that about 9 out of 10 times our model will give back the same results when identifying benign and malignant tumors.

VI. Discussion

We feel that we were able to reach our goal of creating a logistic regression model that allows us to predict whether a person has a benign or malignant tumor based on the independent variables provided in the data. The fact that our model is 92.8% accurate for a model that predicts whether or not the person has cancer or not is indicative of the extent to which we feel we were able to reach our goal given that our model is correct in its predictions about 92.8% of the time. The only value we felt we could have majorly improved is our value for sensitivity. While the percentage of our sensitivity is decently high, the value still isn't great as 15.6% of the time people who have a malignant tumor are incorrectly identified which could lead to them avoiding treatment. The implications of our specificity value (96.8%) aren't as bad as our sensitivity value, as a person with a benign tumor being told they have a malignant tumor, isn't as harmful as a person with a malignant tumor being told they have a benign tumor and only 3.2% of the time are people with benign tumors have their tumors predicted incorrectly whereas that value is 15.6% (almost five times) for people with malignant tumors having their tumors predicted incorrectly. This tells us that our model is much better at correctly determining benign tumors than malignant tumors. The initial study could be modified to utilize a much bigger sample size as that would allow us to state with more certainty the accuracy

of our model as even a 1% change in any of our results (accuracy, sensitivity, etc) when predicting if a tumor is benign or malignant is a big difference as less people would recieve false information about their health. Even 7.2% error isn't very good as the people who fall under this criteria could result in them avoiding treatment because they were told its benign or in them treating a benign tumor because they think its malignant. The implications of our work and the model created are huge. Physicians and doctors could utilize this model to determine with a lot more ease whether one of their patients has cancer or not. This would allow them to catch the cancer earlier, which would make it much easier to offer treatment to patients who have potentially life-threatening tumors, in turn saving many more lives. As stated above, since the model isn't 100% accurate, there is a chance a prediction is wrong, and if it is it could have dire consequences for a patient. While our results aren't perfect, we still were able to create a decently reliable model in predicting the state of a tumor based on the input given.

VII. Appendix

Figure 1.

```
def g(z):
    """Sigmoid function"""
    return 1 / (1+ (np.e**-z))
```

Figure 2.

```
def computeCost(theta,X,Y, Lambda = 0.):
    """
    theta is an n-dimensional vector of parameters
    X is data matrix
    Y is a matrix with m-rows and 1 column of target values
    This includes regularization if you set Lambda to not be zero (make sure it is non-negative)
    """
    m = Y.size
    X = X.values
    Y = np.matrix(Y)

    cost = np.sum([(-Y[i][0] * np.log(h(theta, X[i].T)) - (1 - Y[i][0]) * np.log(1 - h(theta,X[i].T))) for i in range(m)]) / (m)
    reg = (Lambda / (2*m)) * np.sum([pow(theta,2) for i in range(len(theta))])

    ret  = cost + reg
    ret = np.nan_to_num(ret)
    if ret == 0:
        return 99999999999
    return ret
```

Figure 3.

```
    def predict(self, X):
        X_val = X.values
        th = (np.matrix(self.theta))
        return [h(X_val[i],self.theta)[0] for i in range (len(X_val))]

    def score(self, X, Y):
        total = 0
        X = self.predict_class(X)
        c = []

        for i in range(len(Y)):
            if X[i] == Y[i][0]:
                total += 1
                c.append('True')
            else: c.append('False')

        self.accuracy = total/len(Y)
        print(Counter(c))
        print("Training Accuracy with Lambda = "+str(self.Lambda)+" "+str(self.accuracy))
        return self.accuracy

    def predict_class(self, X):
        X = self.predict(X)
        return [np.around(x, decimals=0) for x in X]
```

Figure 4.

```
clf_poly = LogisticRegression()
clf_poly.fit(X_tr,Y,0)
```

```
Counter({'True': 671, 'False': 12})
Training Accuracy with Lambda = 0 0.9824304538799414
```

Figure 5.

```
def train_validate_test_split(X,Y):
    # for reproducability
    np.random.seed(1234)
    X['Target'] = Y
    train, validate, test = None, None, None
    #train, validate, test= np.split(N.sample(frac=1), [int(.6*len(N)), int(.8*len(N))])

    perm = np.random.permutation(X.index)
    m = len(X)
    train_end = int(.6 * m)
    validate_end = int(.2 * m) + train_end
    train = X.iloc[perm[:train_end]]
    validate = X.iloc[perm[train_end:validate_end]]
    test = X.iloc[perm[validate_end:]]
    return train, validate, test


    #return train, validate, test

train, validate, test = train_validate_test_split(X,Y)
```

Figure 6.

```
def find_lambda(lambda_grid, X,Y):
    train, validate, test = train_validate_test_split(X,Y)
    trainX, trainY = data_to_XY(train)
    validateX, validateY = data_to_XY(validate)
    testX, testY = data_to_XY(test)
    best_lambda = 0
    best_lambda_score = 0
    best_clf = None
    s = []

    for i in range(len(lambda_grid)):
        clf = LogisticRegression()
        clf.fit(trainX,trainY,lambda_grid[i])
        s.append(clf.accuracy)
        print('after training')
        clf.fit(validateX,validateY,lambda_grid[i])
        print('after validating')
        if clf.accuracy > best_lambda_score:
            best_lambda_score = clf.accuracy
            best_lambda = clf.Lambda

    print('')
    print('DONE WITH FINDING LAMBDA')
    print('')

    print('best lambda score: '+ str(best_lambda_score))
    clf = LogisticRegression()
    clf.fit(trainX,trainY,best_lambda)
    #clf.fit(validateX,validateY,best_lambda)
    #clf.fit(testX,testY,best_lambda)
    best_clf = clf
    return best_lambda, best_lambda_score, best_clf  , s, testX,testY
```

Figure 7.

```
def classification_summary(cm):
    # 2 class confusion matrix
    tn, fp, fn, tp = cm.ravel()
    accuracy = (tp+tn) / (tn+fp+fn+tp)
    sensitivity = tp/(tp+fn)
    specificity = tn/(tn+fp)
    precision = tp/(tp+fp)
    print(tn,fp,fn,tp)
    print('accuracy: ',accuracy)
    print('sensitivity:' , sensitivity) #<-- this is a problem! Can you tihnk of how to fix it?
    print('specificity: ', specificity)
    print('precision: ', precision)
classification_summary(cm)
```

```
90 3 7 38
accuracy:  0.927536231884
sensitivity: 0.844444444444
specificity:  0.967741935484
precision:  0.926829268293
```

VIII. References

"How to Split Data into 3 Sets (Train, Validation and Test)?" Pandas - How to Split Data into 3 Sets (Train, Validation and Test)? - Stack Overflow, stackoverflow.com/questions/38250710/how-to-split-data-into-3-sets-train-validation-and-test.

The above reference was used in the train_validate_test_split function.