Part II - Model Testing and Validation

Methods of testing

Consider a hypothetical dataset with information on 1000 patients.

1. **Hold out validation:** build a model from the first 500 patients and test on the next 500.
2. **Ten fold cross validation:** build a model from the first 900 patients and test on the next 100, then build a model using data from patients 101-1000 and test on the first 100, and so on until you cover the whole dataset.
3. **Re-substitution (weakest method)**: train on data from all 1000 patients and test on the same dataset.
4. **Independent validation (strongest):** use data from a different study to test the model.

Metrics for evaluating accuracy

Continuous response data (e.g. levels of blood glucose)

- **Pearson's Correlation:** measures the strength and direction of a linear relationship between two variables.
- **Rank Correlation:** measures the strength and direction of a monotonic relationship between two variables.
- **Root Mean Square Error (RMSE):** measures the standard deviation of the residuals. Residuals represent the difference between predicted values and observed values.

$$\mathrm{RMSE} = \sqrt{\frac{\left(\sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2\right)}{n}}$$

Discrete datasets (e.g. Normal vs Diabetes)

- **Accuracy:** the ratio of correct predictions to the total observations.
- **Precision:** the ratio of true positives to the total predicted positives.
- **Recall:** the ratio of true positives to the total actual positives. Also known as sensitivity.

There is no single metric that gives an overall picture of accuracy. It is necessary to use a range of metrics to get a holistic picture.

Testing regression models using hold out validation

Consider BMI to be a function of age, sex, glucose, cholesterol, and systolic blood pressure. How can we use these variables to determine BMI?

Begin by reading in the Framingham dataset, extracting the desired variables, and removing the rows containing NaN values.

```
fram = readtable("../frmgham.xls");
```

```
fram_revised =[fram.BMI,fram.AGE,fram.SEX,fram.GLUCOSE,fram.TOTCHOL,fram.SYSBP];
```

```
fram_noNaN = rmmissing(fram_revised);
```

Now store the BMI and the predictors into separate variables.

```
X = fram_noNaN(:,2:end);
Y = fram_noNaN(:,1);
```

Recall from last lecture that we can use the `fitlm` function to create a linear regression model.

```
model = fitlm(X,Y)
```

```
model =
Linear regression model:
    y ~ 1 + x1 + x2 + x3 + x4 + x5

Estimated Coefficients:
                   Estimate         SE         tStat       pValue
                   ________    __________    _______    ___________

    (Intercept)       19.817       0.33998      58.29             0
    x1             -0.027034      0.004449    -6.0765     1.2734e-09
    x2              -0.77047      0.078443    -9.8219     1.1467e-22
    x3             0.0077801     0.0015712     4.9518      7.474e-07
    x4             0.0052678    0.00087394     6.0276     1.7225e-09
    x5              0.049713     0.0018665     26.634    4.6066e-151


Number of observations: 10021, Error degrees of freedom: 10015
Root Mean Squared Error: 3.86
R-squared: 0.0865,  Adjusted R-Squared: 0.086
F-statistic vs. constant model: 190, p-value = 1.53e-193
```

The linear regression model is represented by $y = 19.817 - 0.027X_1 - 0.77X_2 + 0.008X_3 + 0.005X_4 + 0.05X_5$.
This model uses the entire predictor dataset, `X`, to create a best fit line for BMI.

When using the hold out validation method, a portion of the data is used as the training set while the rest is held out to be used as the testing test.

We have created a custom function for generating the training and test sets called `trainTestSplit`. Use the `open` function to view the script for `trainTestSplit`.

```
open trainTestSplit.mlx
```

The `trainTestSplit` function randomly assigns rows from our two datasets (*X, Y*) into the training and test sets, whose sizes are determined by our input for the `trainSize`.

Use `trainTestSplit` to perform hold out validation by training on 2/3 of the dataset and testing on the remainig 1/3.

```
[Xtrain, Ytrain, Xtest, Ytest] = trainTestSplit(X,Y,0.667);
```

Now that the datasets have been divided, we can use the `fitlm` function again to create a new linear regression model with the training dataset.

```
model = fitlm(Xtrain,Ytrain)
```

```
model =
Linear regression model:
    y ~ 1 + x1 + x2 + x3 + x4 + x5

Estimated Coefficients:
                   Estimate        SE         tStat       pValue
                   ________    _________    _______    ____________

    (Intercept)       19.906      0.41654     47.789             0
    x1             -0.029495    0.0054847    -5.3777     7.7985e-08
    x2              -0.64689     0.095695    -6.7599     1.4975e-11
    x3             0.0040749    0.0018812     2.1661       0.030337
    x4             0.0054153    0.0010715     5.0542     4.4375e-07
    x5              0.050481    0.0022979     21.968    2.4833e-103


Number of observations: 6684, Error degrees of freedom: 6678
Root Mean Squared Error: 3.85
R-squared: 0.0828,  Adjusted R-Squared: 0.0821
F-statistic vs. constant model: 121, p-value = 1.46e-122
```

What is the correlation between `Xtrain` and `Ytrain`? Recall that the `corr` function will output an array containing the correlation coefficients between BMI and each of the predictor variables.

```
corr(Xtrain,Ytrain)
```

```
ans = 5×1
    0.0513
   -0.0620
    0.0591
    0.0839
    0.2653
```

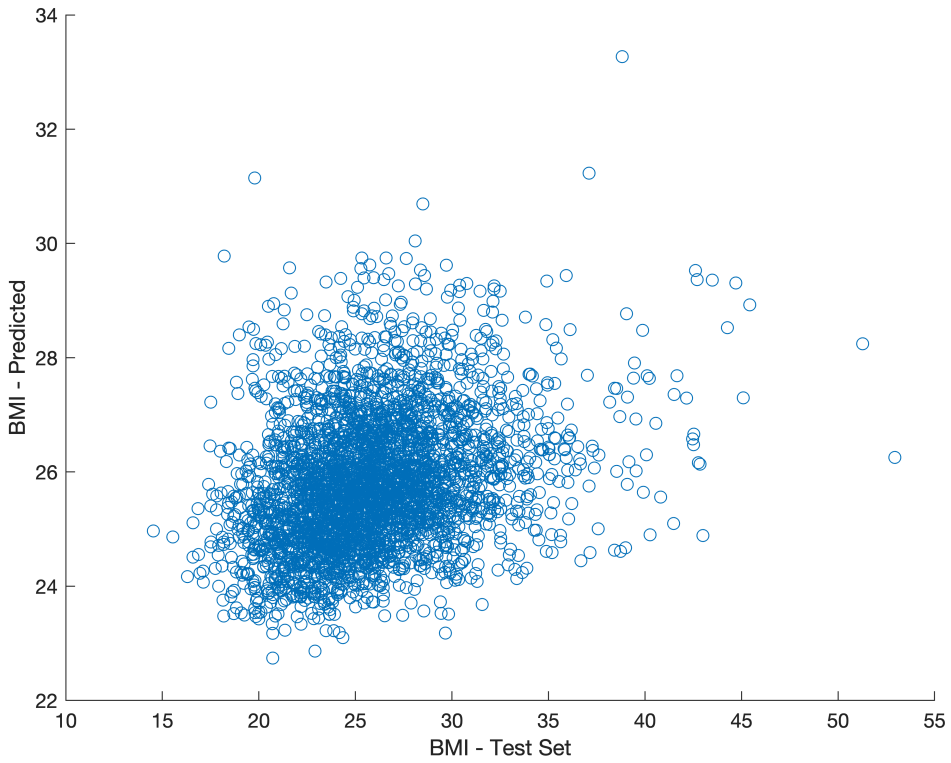
Evaluating accuracy in the test set

The model's predicted values for `Y` can be calculated using the `predict` function. Use this function to evaluate the model given the `Xtest` dataset.

```
Ypred = predict(model,Xtest);
```

How do the values in `Ypred` compare to the actual BMI values in `Ytest`? Use the `scatter` function to compare.

```
scatter(Ytest,Ypred)
xlabel('BMI - Test Set')
ylabel('BMI - Predicted')
```



How strong is the linear relationship that is displayed in the scatterplot?

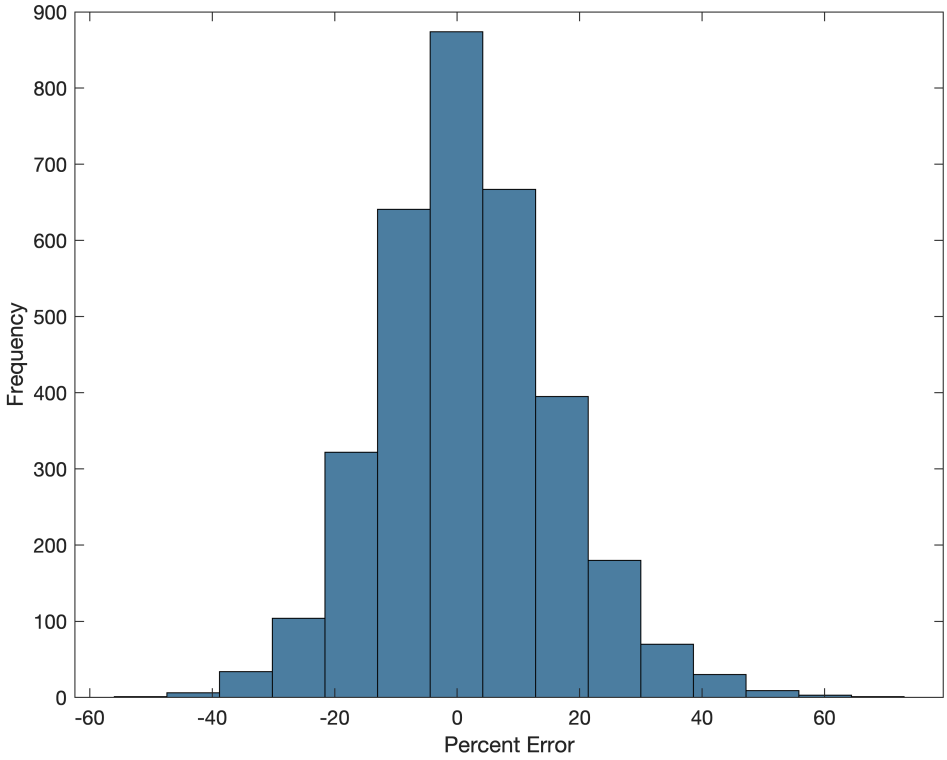
```
corr(Ytest,Ypred)
```

```
ans = 0.3022
```

There are several other parameters that we can calculate to examine the model.

Percent Error - measures the difference between estimed and actual values

```
percent_error = (Ypred-Ytest) ./ Ytest * 100;
histogram(percent_error,15)
xlabel('Percent Error')
ylabel('Frequency')
```



Percent Average Error

```
percent_avg_error = mean(percent_error)
```

```
percent_avg_error = 1.8346
```

Root Mean Square Error (RMSE) - measures the standard deviation of the residuals

```
RMSE = sqrt(mean((Ypred-Ytest).^2))
```

```
RMSE = 3.8807
```

RMSE is sensitive to large errors because the squaring process gives disproportionate weight to very large errors. Further, the RMSE can only be compared between models whose errors are measured in the same units. There is no absolute criterion for a "good" value of RMSE; it depends on the units in which the variable is measured and on the degree of predictive accuracy.

Testing Using Cross Validation

We will now repeat the previous analysis using five fold cross validation. For this analysis, the dataset will be broken into five groups of equal size, called folds. We will train the model wth 80% of the dataset and test the model with the remaining 20%. This process will be repeated five times so that a model is created for each test set.

Note that if there is any sort of ordering to the data, we cannot simply split the first 80% into the training set and the last 20% into the test set. It is better to populate the training and test sets by randomly ordering or shuffling the dataset first. There are a number of ways to randomize data in MATLAB. One method is shown below:

```
shuffled = fram_noNaN(randperm(size(fram_noNaN,1)),:);
```

There are several functions in MATLAB built for generating training and test sets, and they often shuffle the data internally. One of these functions is `crossvalind`. The function requires inputs for the cross validation method (`'kfold'` in our case), the number of observations, and the number of folds (`k`). The output contains a randomized array of indices (varying from 1 to `k`) which are assigned to each row in the dataset and create folds of approximately equal size.

Create an array which contains randomized indices for all of the rows in our dataset.

```
indices = crossvalind('kfold',size(fram_noNaN,1),5)
```

```
indices = 10021×1
     1
     1
     1
     5
     1
     1
     1
     3
     5
     2
     ⋮
```

We can see that each row in the `indices` array contains a value between 1 and 5, linking that row to a certain fold. These indices allow us to easily separate the data into the training and test sets for the five fold cross validation. Confirm that the new datasets are approximately the same size.

```
sum(indices==1)
```

```
ans = 2004
```

```
sum(indices==2)
```

```
ans = 2004
```

```
sum(indices==3)
```

```
ans = 2004
```

```
sum(indices==4)
```

```
ans = 2004
```

```
sum(indices==5)
```

```
ans = 2005
```

Now that we have a way to break the data into five folds, we can use similar steps as in the hold out validation test in order analyze the statistical relationship between `Ypred` and `Ytest`. Because five separate models must be created, a **for loop** will be used.

First, initialize the RMSE, correlation and average error variables. These parameters will be calculated for each of the five models.

```
RMSE_kfold = zeros(5,1);
corr_kfold = zeros(5,1);
avg_error = zeros(5,1);
```

Initializing variables prior to the for loop helps improve the speed of the script. Instead of having to reallocate memory each iteration for the growing matrix, the matrix zeros are simply replaced with the new values. However, it is important that the matrix is initialized to the correct size.

The for loop will use the indices to determine the training and test sets for each iteration. It will then calculate the correlation, RMSE and average percent error values as we did previously, and store them in the matrices that we initiliazed. For example, in the first iteration, every row with an assigned index of 1 is treated as the test set, while the remaining data is assigned to the training set. In the second iteration, rows with an index of 2 are used for the test set, and those with an index of 1, 3, 4 or 5 comprise the training set.

```
% Run through loop 5 times and assign iteration number to i. Identify rows
% in indices equal to i and assign to test; assign remaining rows to train.
for i = 1:5
    test = (indices == i);
    train = ~test;

    % Extract train rows from X and Y. Create linear model with train data.
    Xtrain = X(train,:);
    Ytrain = Y(train,:);
    model = fitlm(Xtrain,Ytrain);

    %Extract test rows from X and Y
    Xtest = X(test,:);
    Ytest = Y(test,:);

    % Predict Y values based on model. Calculate stat parameters and assign
    % to index i of variables
    Ypred = predict(model,Xtest);
    corr_kfold(i) = corr(Ytest,Ypred);
    RMSE_kfold(i) = sqrt(mean((Ypred-Ytest).^2));
    avg_error(i) = mean((Ypred-Ytest) ./ Ytest * 100);
end
```

Because there are five values in each of the output arrays, use the `mean` function to view the correlation, RMSE and average percent error values.

```
mean(corr_kfold)
```

```
ans = 0.2929
```

```
mean(RMSE_kfold)
```

```
ans = 3.8620
```

```
mean(avg_error)
```

```
ans = 2.1165
```

Evaluating discrete datasets using hold out validation

Similar methods can be used to evaluate discrete datasets, where the data can only take on certain values. An example would be classifying patients with diabetes versus those without diabetes. The values can only be one of two numbers: 0 (non-diabetic) or 1 (diabetic).

How well can we predict diabetes from the patients' glucose levels? Since we are using the hold out method, we can incorporate the `trainTestSplit` function again. This time, create a training set with 80% of the dataset.

```
fram_discrete = rmmissing([fram.GLUCOSE, fram.DIABETES]);
gluc = fram_discrete(:,1);
diab = fram_discrete(:,2);
[Xtrain, Ytrain, Xtest, Ytest] = trainTestSplit(gluc,diab,0.8);
```

Recall from last lecture that we used the `mnrfit` function for analyzing discrete categorical data. The response variable, diabetes, must first be converted into a categorical array with the `categorical` function.

```
Ytrain = categorical(Ytrain);
```

The `Ytrain` variable contains two categories: non-diabetic (0) and diabetic (1). The last category, in this case diabetic, is always used as the reference category for the `mnrfit` function. The logistic regression model will tell us the relative log odds of being non-diabetic versus diabetic. Create the model using `Xtrain` and `Ytrain` and then print the model coefficients.

```
[b,~,stats] = mnrfit(Xtrain,Ytrain);
b
```

```
b = 2×1
    8.3413
   -0.0545
```

The first term in `b` is the intercept term of the relative risk model and the second term is the coefficient for glucose levels. The resulting equation is: $\ln\left(\frac{\pi_{\text{normal}}}{\pi_{\text{diabetic}}}\right) = 8.5116 - 0.0563X_1$.

The relative log odds of being normal versus diabetic decreases 0.0563 times with a one-unit increase in X_1, assuming all else is equal. Here X_1 is our only predictor variable, glucose. While this does not seem like a strong relationship, the model makes sense. Diabetic patients typically have higher glucose levels, so for every unit increase in glucose, there is a lesser chance that the patient is non-diabetic.

The `mnrfit` function also outputs several other statistical parameters, such as the *p* value for each of the mdoel parameters.

```
stats.p
```

```
ans = 2×1
10^-117 ×
    0.0000
    0.3316
```

The *p* value for glucose is magnitudes below the accepted value of 0.05, meaning the measure of glucose levels is significant to the relative risk of being non-diabetic versus diabetic.

Estimating model accuracy in test data

The model accuracy can be evaluated using the test sets that we created, along with the `mnrval` function. Recall that this function returns the predicted probabilities for the logistic regression model given certain values for the predictor variables, which come from the predictor test set.

```
probability = mnrval(b,Xtest);
```

Let's look at the first ten rows of the predicted diabetes classifications.

```
probability(1:10,:)
```

```
ans = 10×2
    0.9843    0.0157
    0.9851    0.0149
    0.9887    0.0113
    0.9756    0.0244
    0.9881    0.0119
    0.9478    0.0522
    0.9756    0.0244
    0.9834    0.0166
    0.9804    0.0196
    0.9781    0.0219
```

The left column shows the probability, based on our model, that the patient is non-diabetic. The right colum shows the probabilty that they are diabetic.

In order to compare the predicted values to the test values, round the predicted values to the nearest integer (0 or 1). The model accuracy can then be determined by checking how many rows in the predicted and test sets match. We want to use the second column from the `probability` matrix because in that case, a 0 will indicate non-diabetic and 1 will indicate diabetic, which is the same system as the test dataset.

```
Ypred_round = round(probability);
accuracy = sum(Ypred_round(:,2) == Ytest)/length(Ypred_round)
```

```
accuracy = 0.9726
```

Hold out validation with multiple predictors

We will now repeat the hold out validation test, but this time we will use BMI, glucose and cholesterol to predict diabetes. What is the equation for this model and which variable has the greatest effect on predicting BMI? Also, which variables are statistically significant?

```
fram_discrete_2 = rmmissing([fram.GLUCOSE, fram.BMI, fram.TOTCHOL, fram.DIABETES]);
X = fram_discrete_2(:,1:3);
Y = fram_discrete_2(:,4);
[Xtrain, Ytrain, Xtest, Ytest] = trainTestSplit(X,Y,0.8);
Ytrain = categorical(Ytrain);
[b,~,stats] = mnrfit(Xtrain,Ytrain);
b
```

```
b = 4×1
   10.6350
   -0.0570
   -0.0670
   -0.0011
```

```
stats.p
```

```
ans = 4×1
    0.0000
    0.0000
    0.0000
    0.4391
```

Based on the results, the model's equation should look similar to this:

$\ln\left(\frac{\pi_{\text{normal}}}{\pi_{\text{diabetic}}}\right) = 10.635 - 0.057X_1 - 0.067X_2 - 0.0011X_3$. Our model suggests that BMI has the greatest impact towards being diabetic or non-diabetic. Based on the *p* values, we would reject the null hypothesis for glucose and BMI, and fail to reject for total cholesterol.

Next, find the accuracy of the model.

```
probability = mnrval(b,Xtest);
Ypred_round = round(probability);
accuracy = sum(Ypred_round(:,2) == Ytest)/length(Ypred_round)
```

```
accuracy = 0.9755
```

Example: 10 fold cross validation

We will now use 10 fold cross validation to evaluate our model for diabetes using glucose, BMI and cholesterol. A for loop similar to the one used in the previous example can be used.

```
% Assign random indices for each for row in data set
indices = crossvalind('kfold',size(fram_discrete_2,1),10);

% Initialze output variables
b = zeros(4,10);
p_values = zeros(4,10);
accuracy = zeros(1,10);

% Run through loop 10 times and assign iteration number to i. Identify rows
% in indices equal to i and assign to test; assign remaining rows to train.
for i = 1:10
    test = (indices == i);
    train = ~test;

    % Extract training rows from X and Y. Convert Ytrain to categorical.
    Xtrain = X(train,:);
    Ytrain = Y(train,:);
    Ytrain = categorical(Ytrain);

    % Create model w/ training sets. Store model coefficients and p values.
    [b(1:4,i),~,stats] = mnrfit(Xtrain,Ytrain);
    p_values(1:4,i)= stats.p;

    % Extract test rows from X and Y
    Xtest = X(test,:);
    Ytest = Y(test,:);

    % Calculate probability for each category, round up to nearest integer
    % and calculate accuracy using Ypred and Ytest
    probability = mnrval(b(:,i),Xtest);
    Ypred_round = round(probability(:,2));
    accuracy(1,i) = sum(Ypred_round == Ytest)/length(Ypred_round);
end
```

We can now see the model coefficients, *p* values and accuracy for each of the ten models. Calculate the mean for each of these values.

```
b_avg = mean(b,2)  % the 2 tells the function to take the mean of each row
```

```
b_avg = 4×1
   10.5998
   -0.0567
   -0.0670
   -0.0009
```

```
p_avg = mean(p_values,2)
```

```
p_avg = 4×1
    0.0000
    0.0000
    0.0000
    0.4984
```

```
accuracy_avg = mean(accuracy)
```

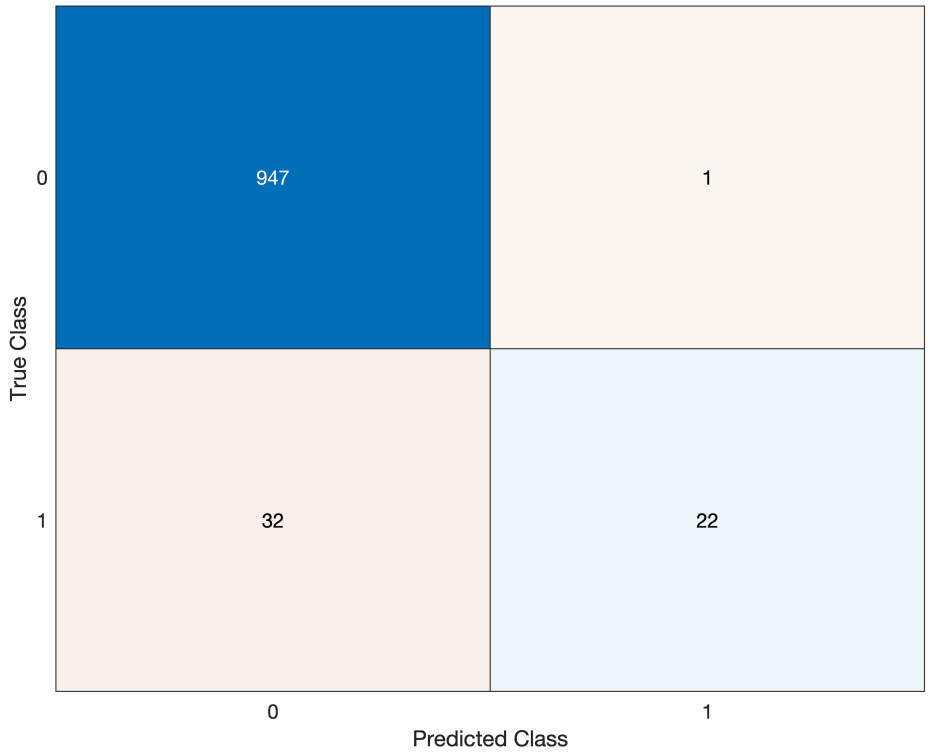
```
accuracy_avg = 0.9704
```

We now have a model with the equation of $\ln\left(\frac{\pi_{\text{non-diabetic}}}{\pi_{\text{diabetic}}}\right) = 10.5998 - 0.0567X_1 - 0.0670X_2 - 0.009X_3$ Once again, the *p* values indicate that glucose and BMI are significant to the model, but total cholesterol is not.

Additional metrics to evaluate accuracy

Confusion matrices are useful tools for evaluating how well a classification model performs. The `confusionchart` function allows us to visually compare the classifications of the predicted and test datasets.

```
confusionchart(Ytest,Ypred_round)
```




```
ans =
  ConfusionMatrixChart with properties:

    NormalizedValues: [2×2 double]
         ClassLabels: [2×1 double]

  Show all properties
```

There are four sections in the confusion matrix:

1. **True positive (TP)** - the model predicts TRUE and the actual outcome is TRUE
2. **True negative (TN)** - the model predicts FALSE and the actual outcome is FALSE
3. **False positive (FP)** - the model predicts TRUE and the actual outcome is FALSE

4. **False negative (FN)** - the model predicts FALSE and the actual outcome is TRUE

Using the values in the confusion matrix or the model variables (`Ypred` and `Ytest`), calculate the following parameters:

- **Precision** - the number of times that the model correctly predicted diabetes over the total number of times that the model predicted diabetes.

```
precision = sum(Ypred_round==1 & Ytest==1)/sum(Ypred_round==1)
```

```
precision = 0.9565
```

This is equivalent to the following calculation from the confusion matrix: $\text{precision} = \frac{22}{(22+1)} = 0.9565$

- **Recall** - the number of times the model correctly predicted diabetes over the total number of diabetes cases in the test set.

```
recall = sum(Ypred_round==1 & Ytest==1)/sum(Ytest==1)
```

```
recall = 0.4074
```

This is equivalent to the following calculation from the confusion matrix: $\text{recall} = \frac{22}{(32+22)} = 0.4074$

Outcome: model testing and validation

Errors arise due to both bad data and bad models

Need to have the right benchmark i.e. what accuracy is good enough?

Overfitting - model is too complex

- Solution: simplify the model, get more training data, reduce noise in the data

Underfitting - model is too simple

- Solution: add more features to the model, use more complex algorithms

Confounding variables - some features are irrelevant

- Solution: start with a strong hypothesis, use unsupervised learning methods to find relevant features, make sure the training data has appropriate features and sample size

A problem can arise when you measure the generalization error multiple times on the test set, and then adapt the model and hyperparameters to produce the best model *for that set.* This means that the model is unlikely to perform as well on new data. A common solution to this problem is to have a second holdout set

called the **validation set**, which is used to run a single final test against the test set to get an estimate of the generalization error.

Improving data quality by normalization

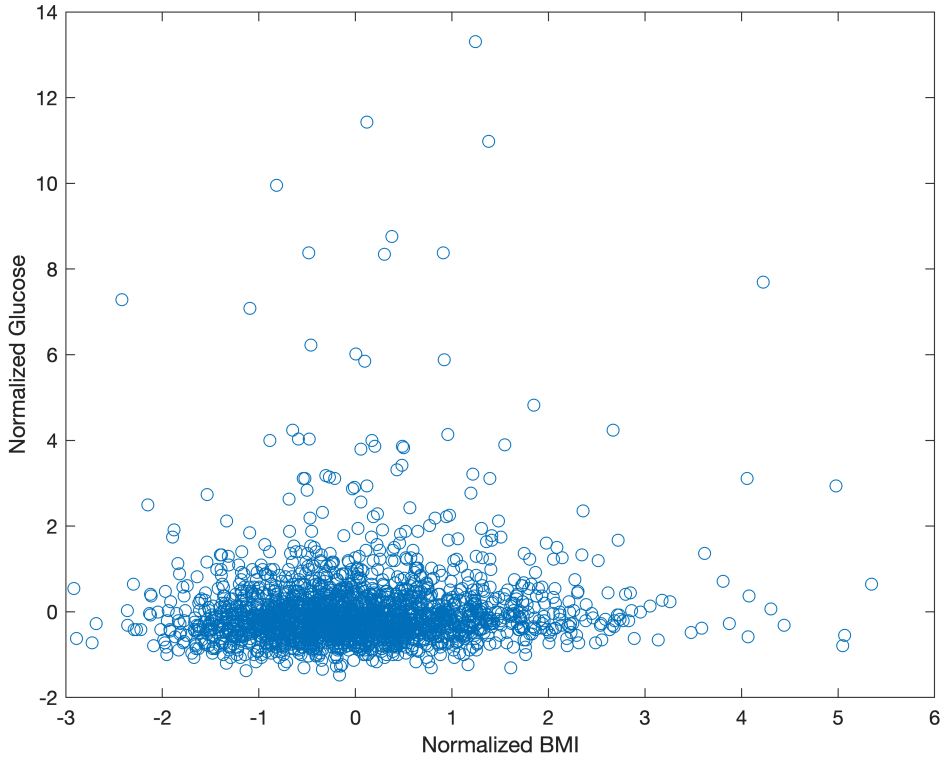
- Most machine learning algorithms calculate the distance between two data points.
- The range of numeric values cna be different between the various feature variables. For example, in the Framingham dataset, BMI has a larger value and range than Sex or Education.
- Euclidien distance will be scewed by magniture and range.
- The range of all features should be normalized so that each feature contributes equally to the distance.
- Scaling and normalization allows us to compare features at the same scale.

Re-scaling Framingham dataset

Getting the Framingham variables on the same scale requires subtracting the mean and dividing by the standard deviation. The MATLAB function `zscore` does these calculations for you.

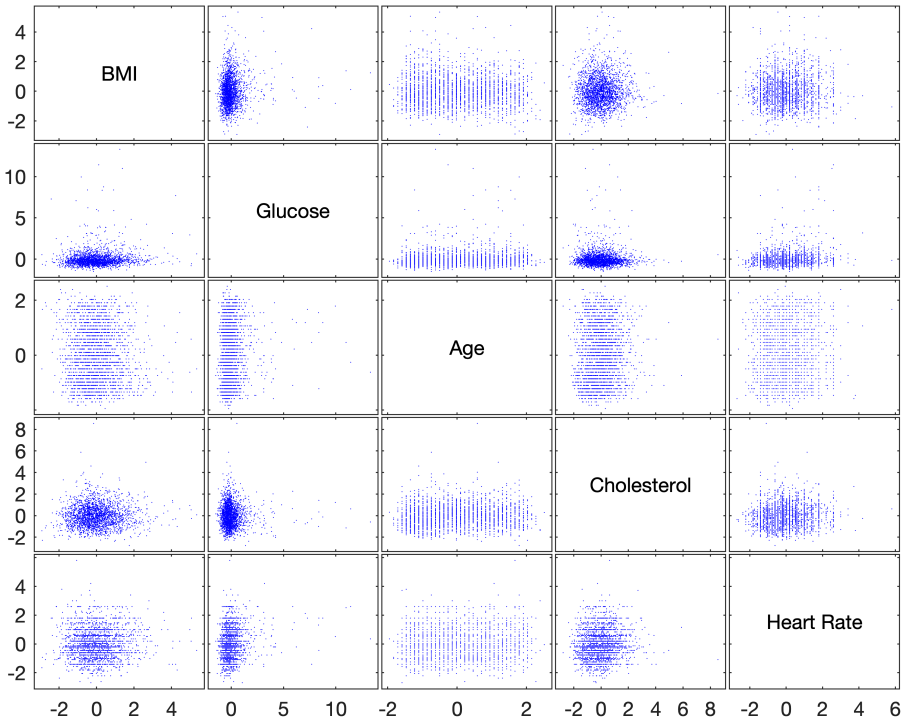
Example: plot normalized BMI vs Glucose

```
fram_noNaN = rmmissing(fram);
BMI_norm = zscore(fram_noNaN.BMI);
gluc_norm = zscore(fram_noNaN.GLUCOSE);
plot(BMI_norm,gluc_norm,'o')
xlabel('Normalized BMI')
ylabel('Normalized Glucose')
```



Now use `gplotmatrix` to create a scatterplot matrix which incudes normalized BMI, glucose, age, total cholesterol and heart rate.

```
age_norm = zscore(fram_noNaN.AGE);
chol_norm = zscore(fram_noNaN.TOTCHOL);
HR_norm = zscore(fram_noNaN.HEARTRTE);
X = [BMI_norm,gluc_norm,age_norm,chol_norm,HR_norm];
xnames = {'BMI', 'Glucose', 'Age', 'Cholesterol', 'Heart Rate'};
gplotmatrix(X,[],[],[],[],[],[],'variable',xnames);
```



The `pdist` function outputs the Euclidean distance between pairs of observations. We can use this function to see the effects of normalization.

```
BMI = fram_noNaN.BMI;
gluc = fram_noNaN.GLUCOSE;
Y1 = pdist([BMI,gluc]);
mean(Y1)
```

```
ans = 24.3267
```

```
Y2 = pdist([BMI_norm,gluc_norm]);
mean(Y2)
```

```
ans = 1.5298
```

Next, fit a linear regression line for BMI vs glucose before and after normalization. Compare the regression coefficient values.

```
fitlm(BMI,gluc)
```

```
ans =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE       tStat       pValue
                   ________    ________    ______    __________
```

```
    (Intercept)      72.193        4.13     17.48    3.1548e-64
    x1             0.65773     0.15858    4.1475    3.4866e-05


Number of observations: 2236, Error degrees of freedom: 2234
Root Mean Squared Error: 29.1
R-squared: 0.00764,  Adjusted R-Squared: 0.0072
F-statistic vs. constant model: 17.2, p-value = 3.49e-05
```

```
fitlm(BMI_norm,gluc_norm)
```

```
ans =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                    Estimate        SE         tStat        pValue
                   ___________    _________    _________    __________

    (Intercept)    2.8152e-16     0.021072    1.336e-14             1
    x1               0.087414     0.021076       4.1475    3.4866e-05


Number of observations: 2236, Error degrees of freedom: 2234
Root Mean Squared Error: 0.996
R-squared: 0.00764,  Adjusted R-Squared: 0.0072
F-statistic vs. constant model: 17.2, p-value = 3.49e-05
```

The two models have the following equations:

Before normalization: $\text{Glucose} = 72.192 + 0.6755 * \text{BMI}$

After normalization: $\text{Glucose} = 2.815 * 10^{-16} + 0.0874 * \text{BMI}$