# Principles of Machine Learning

Lab 3 – Improving Machine Learning Models

## Overview

In this lab you will explore techniques for improving and evaluating the performance of machine learning models. You will continue to work with the diabetes patient classification dataset. Four methods for improving machine and evaluating learning models are investigated in this lab:

- Feature selection.
- Regularization.
- Finding optimal hyperparameters.
- Cross validation.

## What You'll Need

To complete this lab, you will need the following:

- An Azure ML account
- A web browser and Internet connection
- The lab files for this lab

**Note**: To set up the required environment for the lab, follow the instructions in the Setup Guide for this course.

### Important

In this lab you will build on the logistic regression classification experiment you created in Lab 1. If you did not complete lab 1, or if you have subsequently modified the experiment you created, you can copy a clean starting experiment to your Azure ML workspace from the Cortana Intelligence Gallery using the link for your preferred programming language below:

- **R**: http://gallery.cortanaintelligence.com/Experiment/DAT203-2x-Lab-3-Diabetes-Classification-R-1
- **Python**: http://gallery.cortanaintelligence.com/Experiment/DAT203-2x-Lab-3-Diabetes-Classification-Python-1

## Feature Selection

Selecting the best features is essential to the optimal performance of machine learning models. Only features that contribute to measurably improving model performance should be used.

Using extraneous features can contribute noise to training and predictions from machine learning models. This behavior can prevent machine learning models from generalizing from training data to data received in production.

To improve the chance of generalizing a machine learning model, features must be carefully selected. Feature selection is generally performed in one of two ways. In forward stepwise feature selection, the most promising features are added one at a time. In reverse stepwise feature selection, the least important features are pruned one, or a few, at a time. In this lab, reverse stepwise feature selection will be used.

Feature selection requires a method for measuring the importance of features. Ideally, feature importance should be measured by how its contribution to the reduction in model error. In principle, the reduction in error can be directly measured. Methods used in practice, rank features by their relative importance. A simple method is to measure the relationship between the features and the label. The Azure ML **Filter Based Feature Selection** module implements some of these methods. Another approach is to randomize the values of a feature and measure the increase in model error. This *Permutation Based* feature importance method is used in this lab.

## Determine Feature Importance

1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. Open your **Diabetes Classification** or **Diabetes Classification** experiment, and rename it to **Classification Model Improvement** (or copy the R or Python-based starting experiment from the Cortana Intelligence Gallery to a new experiment named **Classification Model Improvement** in your workspace)
3. Search for the **Permutation Feature Importance** module, and drag it onto the experiment canvas. Then connect the output of the **Train Model** module to its **Trained model** (left) input, and connect the **Result Dataset2** (right) output of the **Split Data** module to its Dataset (right) input.
4. Configure the **Permutation Feature Importance** module as follows:
    - **Random seed**: 123
    - **Metric for measuring performance**: Classification - Accuracy
5. Save and run your experiment.
6. When your experiment has finished running, visualize the output of the **Permutation Feature Importance** module. The visualization shows a list of the features, ordered by importance.
7. Scroll down until you can see features with zero importance. These features are candidates to be pruned.

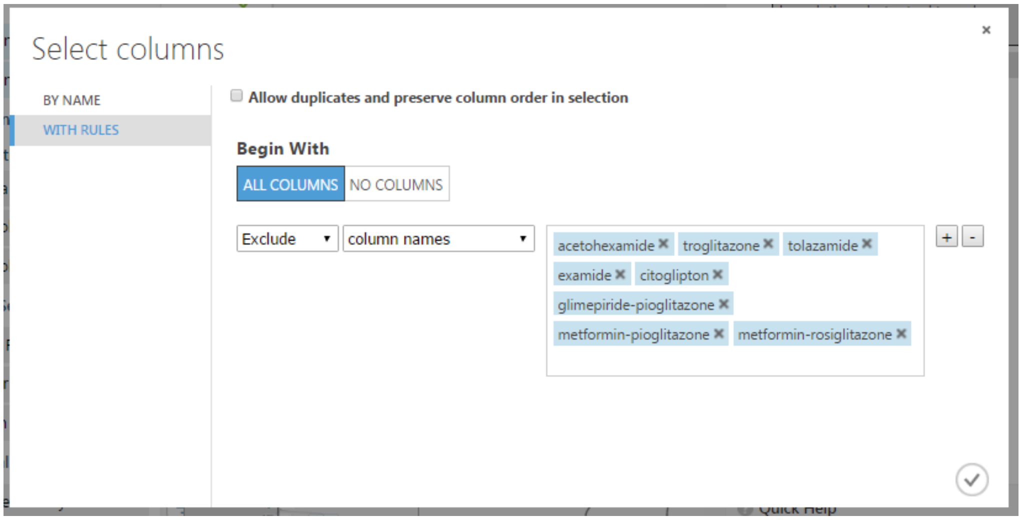
> **Note**: When examining the results produced by the **Permutation Feature Importance** module, the absolute values of the feature importance measure should be compared.

## Prune Features and Measure Results.

Using the feature pruning candidates derived in the previous exercise, you will now prune the features for your machine learning model and evaluate the results. You will do this by removing features and using them in a duplicate of part of the work flow in your experiment. Follow these steps to identify additional features to prune:

1. Right-click the Train Model module and click **Edit Comment**. Then enter the comment *Original Model* and expand the **V** icon so that the comment is visible.

2. Add a **Select Columns in Dataset** module to the experiment and connect the output of the **Edit Metadata** module to its input. Then configure the **Select Columns in Dataset** module to *exclude* the columns with zero importance as shown in the figure:



Select columns

BY NAME

WITH RULES

☐ Allow duplicates and preserve column order in selection

Begin With

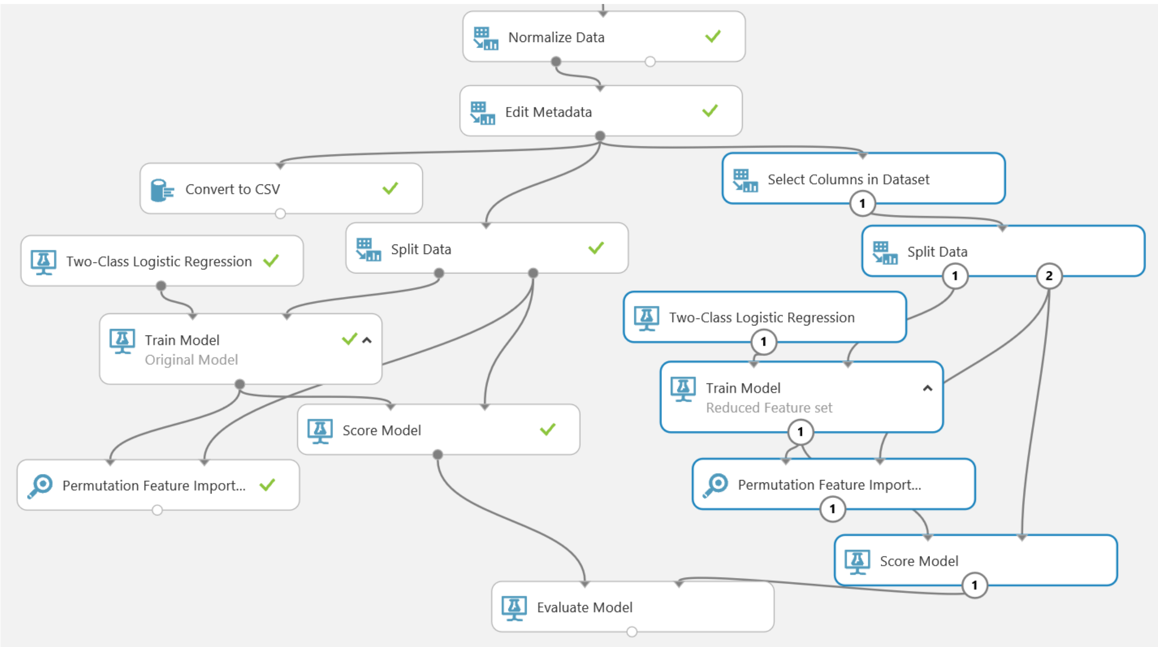
ALL COLUMNS | NO COLUMNS

Exclude ▾ | column names ▾

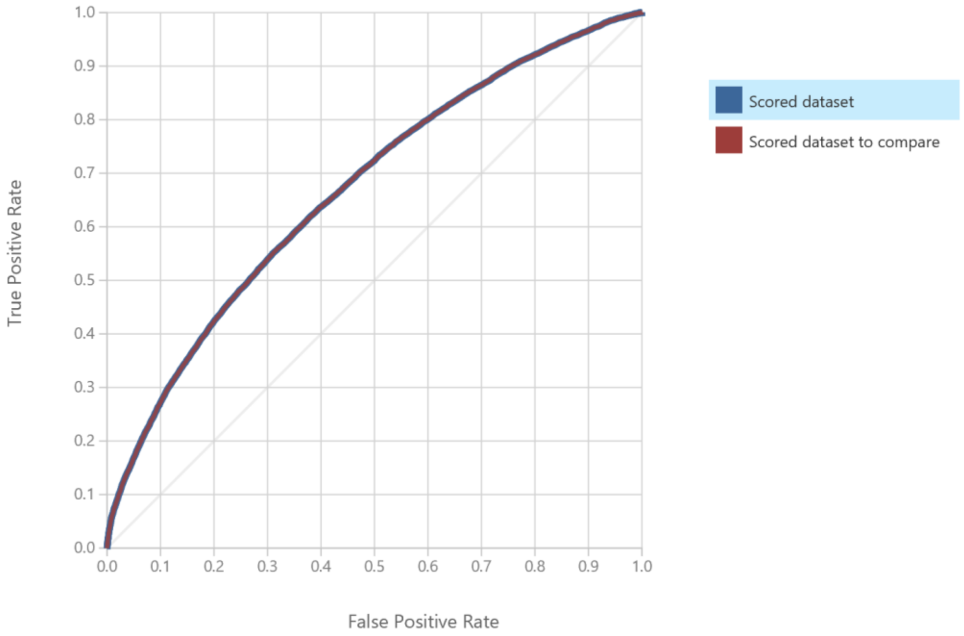
acetohexamide ✖ troglitazone ✖ tolazamide ✖ examide ✖ citoglipton ✖ glimepiride-pioglitazone ✖ metformin-pioglitazone ✖ metformin-rosiglitazone ✖

3. Select and copy the following modules:
    - **Split Data**
    - **Two-Class Logistic Regression**
    - **Train Model**
    - **Score Model**
    - **Permutation Feature Importance**

4. Paste these copied modules, connect the output of the **Select Columns in Dataset** module to the input of the copied **Split Data** module, and connect the output of the copied **Score Model** to the left input of the existing **Evaluate Model** module. Then edit the comment on the copied **Train Model** module, changing it to *Reduced Feature Set* as shown below:



5. Save and run the experiment, and when it is finished, visualize the output of the **Evaluate Model** module to compare the relative performance of the two models. The *Scored dataset* is the original model, the *Scored dataset to compare* is the new model with the features removed). Note that the ROC curve shows that the performance of both models is almost identical – indicating that the features you removed have little impact on label prediction and simply add “noise” to the model.

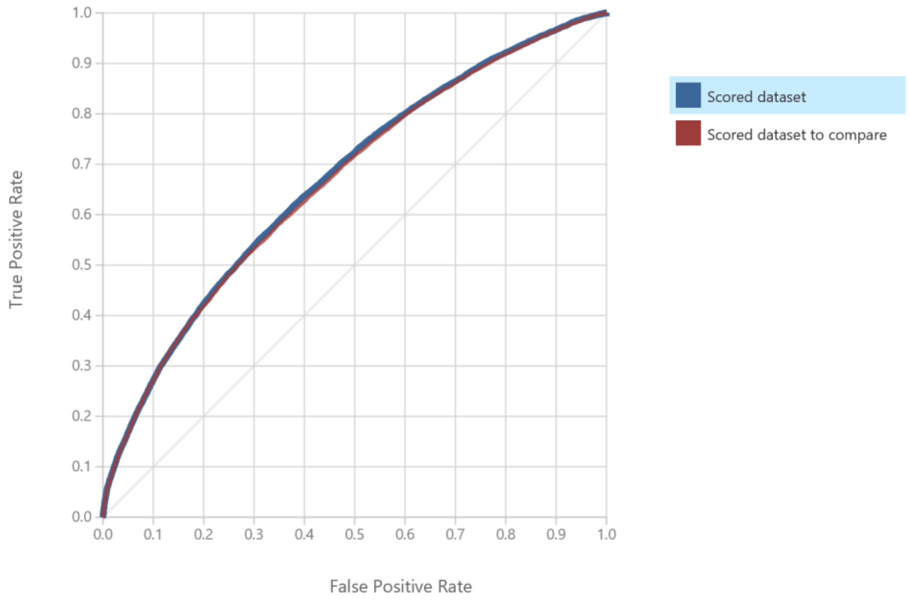


6. Visualize the new **Permutation Feature Importance** module. This indicates the relative importance of the remaining features in the new model.

   To continue pruning features, you typically go through several iterations, removing the least important features each time until the performance of the model starts to degrade. Then finalize the model using the minimum set of features that produces the same performance as the original full feature set.

7. Edit the properties of the **Select Columns in Dataset** module that excludes features from the second version of the model, and add the following columns to the columns being excluded:
   - nateglinide
   - glipizide-metformin
   - chlorpropamide
   - glipizide
   - glyburide
   - num_medications
   - glimepiride
   - miglitol
   - glyburide-metformin
   - tolbutamide
   - pioglitazone
   - num_procedures

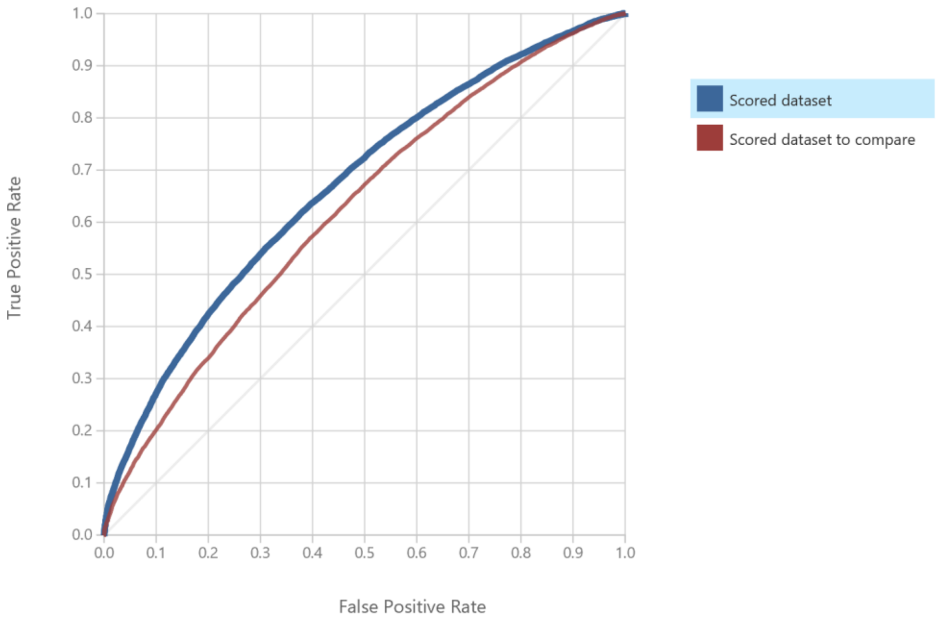
   - change
   - max_glu_serum
   - repaglinide
   - acarbose
   - rosiglitazone
   - num_lab_procedures
8. Add a second **Evaluate Model** module to the experiment and connect the output of the **Score Model** module for the model with the reduced feature set to its left input (so that this **Score Model** module is now connected to two **Evaluate Model** modules – one as the scored dataset to compare and one as the scored dataset)
9. Save and run the experiment, and when it is finished, visualize the output of the original **Evaluate Model** module to compare the relative performance of the two models. Note that the ROC curve shows that even with all of these features excluded, the performance of both models is almost identical. Then scroll down and note the **AUC** and **Accuracy** figures (which show area under the curve and accuracy for the original model).
10. Visualize the new **Evaluate Model** module and note the **AUC** and **Accuracy** figures for the model with a reduced feature set – this should be the same as the original model.
11. Visualize the new **Permutation Feature Importance** module. This indicates the relative importance of the remaining features in the new model.
12. Edit the properties of the **Select Columns in Dataset** module that excludes features from the second version of the model, and add the following columns to the columns being excluded:
    - gender
    - A1CResult
    - age
    - race
    - diag_1
13. Save and run the experiment, and then visualize the output of the original **Evaluate Model** module. Note that the ROC curve now shows a very slight degradation of performance in the *Scored dataset to compare* as shown here:



14. In the visualization of the original **Evaluate Model** module, scroll down and remind yourself the **AUC** and **Accuracy** figures for the original model.
15. Visualize the new **Evaluate Model** module and note the **AUC** and **Accuracy** figures for the model with a reduced feature set – these are now lower than the original model, indicating that the features you have removed had a measurable effect on predicting the label.

    The degradation in performance is slight, but could have an impact on the accuracy of the model when used with new data. However, to see the effect of removing too many features more clearly, you will make one final adjustment to the model by removing an important feature.

16. Re-edit the **Select Columns in Dataset** module and add the **number_inpatient** column to the list of columns to be excluded
17. Save and run the experiment, and then visualize the output of the **Evaluate Model** module connected to both models. Note that the ROC curve now shows a marked degradation of performance in the *Scored dataset to compare* as shown here:



18. In the visualization of the original **Evaluate Model** module, scroll down and remind yourself the **AUC** and **Accuracy** figures for the original model.
19. Visualize the new **Evaluate Model** module and note the **AUC** and **Accuracy** figures for the model with a reduced feature set – these are now lower than the original model, indicating that the features you have removed had a measurable effect on predicting the label.
20. Re-edit the **Select Columns in Dataset** module and remove the following columns from the list of columns to be excluded:
    - gender
    - A1CResult
    - age
    - race
    - diag_1
    - number_inpatient

21. Save and run the experiment and verify that the model with the reduced feature set now performs as well as the original model.

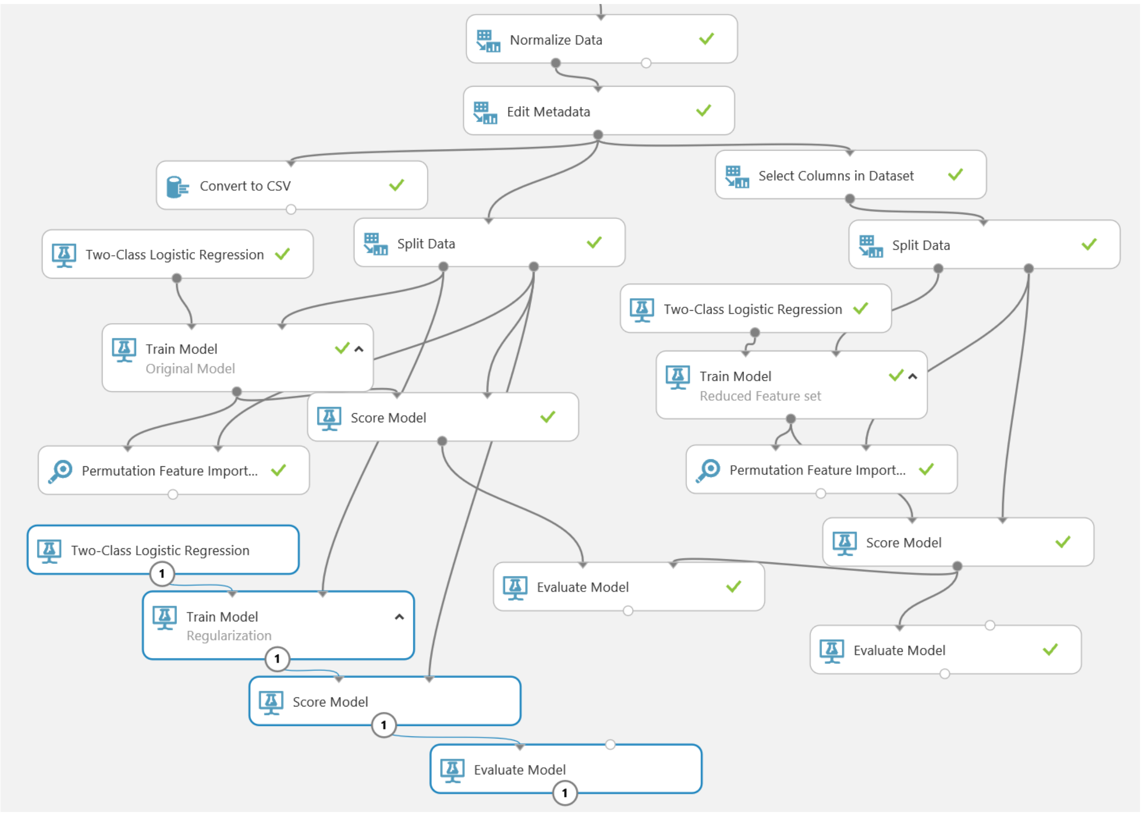
## Regularization

Regularization is an alternative to feature selection. In summary, regularization forces the weights of less important features toward zero. This process helps to ensure that models will generalize in production. Regularization can be performed on models with very large numbers of features, for which manual feature selection is infeasible.

Analogously with manual feature selection, the largest regularization weights which do not affect model performance are preferred. Larger regularization weights force the model coefficients for less important features toward zero. The lower influence of less important features helps to insure the model will generalize.

### Copy the Original Model

1. In the **Classification Model Improvement** experiment, copy the following modules from the original version of the model (which includes all features):
    - **Two-Class Logistic Regression**
    - **Train Model**
    - **Score Model**
    - **Evaluate Model**
2. Connect the left output of the original **Split Data** module to the input of the copied **Train Model** module, and the right output of the **Split Data** module to the right input of the copied **Score Model** module. Then change the comment of the copied **Train Model** module to **Regularization**, as shown below:



3. Save and run the experiment.
4. Visualize the output of the *Regularization* **Trained Model** module. Scroll down, noting that many of the features have a weight of zero, which means these features have no effect on the model. Note that these features include names for each category of categorical features. Regularization works on the indicator variables for each category of the categorical variables. Thus, regularization can apply weights to the coefficients of each category differently.

## Increase Regularization Weights

1. Change the configuration of the **Two-Class Logistic Regression** module for the *Regularization* model to increase the regularization weights as follows:
    - **Create trainer mode:** Single Parameter
    - **Optimization tolerance:** 1E-07
    - **L1 regularization weight:** 0.1
    - **L2 regularization weight:** 0.1
    - **Memory size for L-BFGS:** 20
    - **Random number seed:** 1234
    - **Allow unknown categorical levels:** checked
2. Save and run the experiment.
3. When the experiment finishes running, visualize the output of the *Regularization* **Trained Model** module. Note that some of the features that previously had a zero weight, now have a non-zero weight. Additionally, the order has changed.
4. Visualize the output of the **Evaluate Model** module for the original model, and note the performance metrics figures reported. Then visualize the output of the **Evaluate Model** module for the regularization model, and verify that the performance metrics are similar to the original model, despite the higher regularization weights.
5. Change the configuration of the **Two-Class Logistic Regression** module for the *Regularization* model to increase the regularization weights further using the following values:
    - **Create trainer mode:** Single Parameter
    - **Optimization tolerance:** 1E-07
    - **L1 regularization weight:** 10.0
    - **L2 regularization weight:** 10.0
    - **Memory size for L-BFGS:** 20
    - **Random number seed:** 1234
    - **Allow unknown categorical levels:** checked
6. Save and run the experiment.
7. When the experiment finishes running, visualize the output of the *Regularization* **Trained Model** module. Note that many more features now have zero coefficients. In addition, other features coefficients are smaller and the order has changed again. Note that all weight categories and most age categories now have zero coefficients. This demonstrates the concept of regularization forcing model coefficients toward zero.
8. Visualize the output of the **Evaluate Model** module for the original model, and note the performance metrics figures reported. Then visualize the output of the **Evaluate Model** module for the regularization model. All of the performance metrics are a bit worse with the higher regularization weight. Evidently, these weights are a bit too large.

**Note**: In this lab, you have tried three sets of regularization weights in this lab. You can further pursue finding the optimal weights. You can try weights intermediate between 10.0 and 0.1.

Additionally, the space of L1 and L2 weight combinations can be searched, either by random sampling or on a grid.

## Tuning Model Hyperparameters

You have investigated feature selection and regularization as manual processes. In this exercise you will use the **Tune Model Hyperparameters** module to automatically search for optimal combinations of machine learning model hyperparameters.

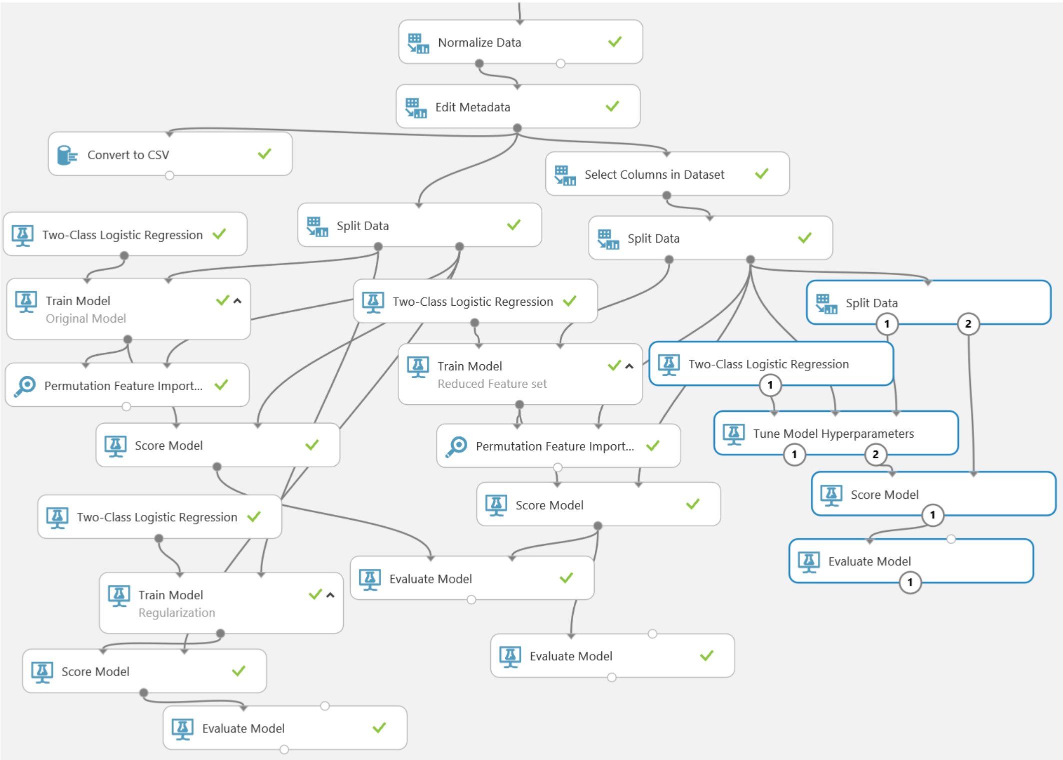
The hyperparameter space can be searched in several ways. The search can be on a regular or randomized grid of the hyperparameter space. Alternatively, the hyperparameter space can be randomly sampled. In either case, the search of hyperparameter space can be computationally intensive, since the model must re-compute and evaluate the model. Fortunately, most the performance commonly used machine learning models are not particularly sensitive to the choice of hyperparameters.

### Configure Hyperparameter Tuning

1. In the **Classification Model Improvement** experiment, add the following modules, placing them in free space to the right of the existing modules:
   - Split Data
   - Two Class Logistic Regression
   - Tune Model Hyperparameters
   - Score Model
   - Evaluate Model
2. Connect the **Results Dataset2** (right) output of the existing **Split Data** module for the *Reduced Feature Set* model to the input of the new **Split Data** module.
3. Configure the new **Split Data** module as shown:
   - **Splitting mode**: Split Rows
   - **Fraction of rows in the first output dataset**: 0.5
   - **Randomized split**: Checked
   - **Random seed**: 1234
   - **Stratified split**: False
4. Connect the output of the new **Two-Class Logistic Regression** module to the **Untrained model** (left-most) input of the **Tune Model Hyperparameters** module.
5. Connect the **Results dataset1** (left) output of the existing **Split Data** module for the *Reduced Feature Set* model to the **Training dataset** (middle) input of the **Tune Model Hyperparameters** module.
6. Connect the **Results dataset1** (left) output of the new **Split Data** module to the **Optional evaluation dataset** (right) input of the **Tune Model Hyperparameters** module.
7. Connect the **Trained model** (right) output of the **Tune Model Hyperparameters** module to the **Trained model** (left) input of the **Score Model** module.
8. Connect the **Results dataset2** (right) output of the new **Split Data** module to the **Dataset** (right) input of the **Score Model** module.
9. Connect the output of the new **Score Model** module to the scored dataset (left) input of the new **Evaluate Model** module.
10. Configure the new **Two Class Logistic Regression Module** as follows:
    - **Create trainer mode:** Parameter Range
    - **Use Range Builder (all four):** Unchecked
    - **Optimization tolerance:** 0.0001, 0.0000001
    - **L1 regularization weight:** 0.0, 0.01, 0.1, 1.0
    - **L2 regularization weight:** 0.01, 0.1, 1.0

- **Memory size for L-BFGS:** 5, 20, 50
- **Random number seed:** 1234
- **Allow unknown categorical levels:** checked

11. Configure the **Tune Model Hyperparameters** module as follows:
    - **Specific parameter sweeping mode**: Random sweep
    - **Maximum number of runs on random sweep**: 30
    - **Random seed**: 1234
    - **Label column, selected**: select the *readmitted* column
    - **Metric for measuring performance for classification**: Accuracy
    - **Metric for measuring performance for classification**: Mean absolute error
12. Ensure that the lower part of your experiment resembles this figure:



## View the Results

1. Save and run the experiment.
2. When you experiment has finished running visualize the output of the **Evaluate Model** module, and note that each of the model performance metrics is at least a bit better than the best obtained so far in this lab. Tuning the model hyperparameters has had the desired effect.
3. Visualize the **Trained model** (right) output of the **Tune Model Hyperparameters** module to examine the parameter values. Note that the L1 and L2 regularization weights are between 0.1 and 1.0. This was within the range that your manual regularization search identified. You could now apply these parameter values to the **Two-Class Logistic Regression** module used to train the *Reduced Feature Set* model to optimize its performance.

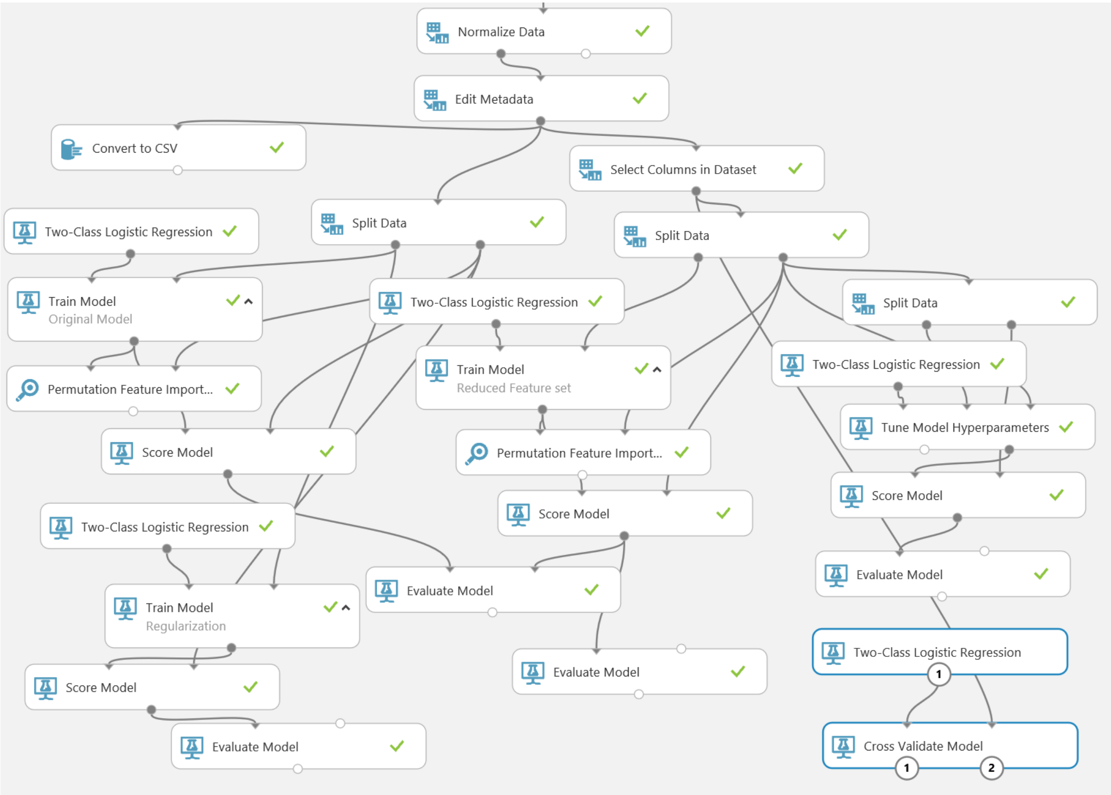
# Cross Validation

Up to now, you have evaluated the performance of machine learning models using a single randomized set of data splits. It is not unlikely that evaluation of models based on this single sample will be biased. In fact, it is often the case that this bias is toward the optimistic side, giving a false impression of expected model performance when it is put into production.

Cross validation uses a resampling scheme to train and evaluate machine learning model. The model is trained and evaluated with a series of folds, created by resampling with replacement. The model performance metrics are averaged over the results of the folds. This aggregated performance measure is more likely to represent the actual performance a model will achieve in production.

## Configure Cross Validation

1. Add another **Two-Class Logistic Regression** module and a **Cross Validate Model** module to the experiment, and place them in some free space.
2. Connect the output of the **Select Features in Dataset** module for the *Reduced Feature Set* model to the **Dataset** (right) input of the **Cross Validate Model** module, and connect the output of the new **Two-Class Logistic Regression** module to the **Untrained model** (left) input of the **Cross Validate Model** module.
3. Configure the new **Two-Class Logistic Regression** module using the hyperparameter values identified by the **Tune Model Hyperparameters** module in the previous exercise, which should be similar to:
    - **Create trainer mode:** Single Parameter
    - **Optimization tolerance:** 2.69260836E-05
    - **L1 regularization weight:** 0.0128837666
    - **L2 regularization weight:** 0.9810832
    - **Memory size for L-BFGS:** 45
    - **Random number seed:** 1234
    - **Allow unknown categorical levels:** checked
4. Configure the **Cross Validate Model** module as shown here:
    - **Label column, Selected**: select the *readmitted* column.
    - **Random seed**: 1234
5. Verify that the new portion of your experiment resembles this figure:

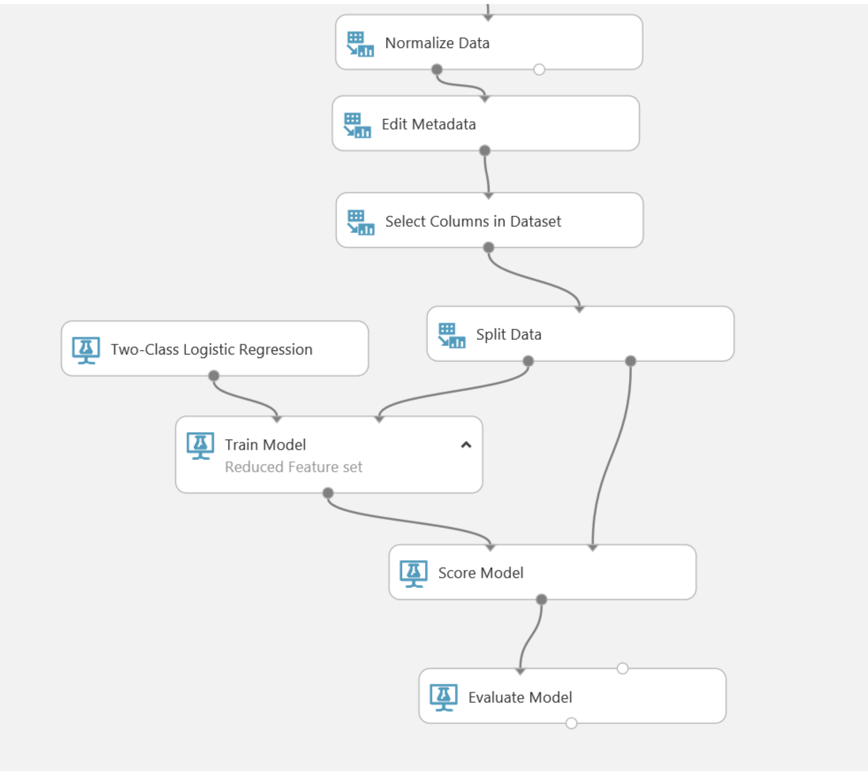


6. Save and run the experiment.

7. When your experiment has finished running, visualize the output of the **Cross Validate Model** module, and note the following:
    - The **Fold Number** is shown in the left most column.
    - The number of evaluation data in each fold is shown in the second column from the left.
    - The performance metrics you have been using, **Accuracy, Precision, Recall, F-Score,** and **AUC**, are shown for each of the folds. Notice that the values of these metrics do not change significantly from fold to fold, indicating the model is likely to generalize well in production.
    - The **Mean** of the performance metrics is shown in the second row from the bottom. Compare these metrics to the values obtained for the best model using single random splits of the dataset. Notice that each of the metrics is a bit worse in cross validation. This is expected, and is more representative of the performance the model is likely to achieve in production.
    - The **Standard Deviation** of the performance metrics is shown in the last row. In each case the **Standard Deviation** is several orders of magnitude less than the Mean. The low dispersion of the performance metrics between the folds is another indication that the model will generalize well in practice.

## Apply the Optimized Parameters

1. Save the experiment as **Optimized Diabetes Classification**. This makes a copy of the experiment while retaining the optimization trials you have created so far in this module.
2. In the **Optimized Diabetes Classification** experiment, modify the **Two Class Logistic Regression** module that is used with the *Reduced Feature Set* **Train Model** module so that it uses the parameters you identified, which should be similar to the following:
    - **Create trainer mode:** Single Parameter
    - **Optimization tolerance:** 2.69260836E-05
    - **L1 regularization weight:** 0.0128837666
    - **L2 regularization weight:** 0.9810832
    - **Memory size for L-BFGS:** 45
    - **Random number seed:** 1234
    - **Allow unknown categorical levels:** checked
3. Remove the extraneous modules so that you are left only with the modules required to train, score, and evaluate the *Reduced Feature Set* model. The bottom portion of your experiment, after the modules to perform data cleansing and initial feature selection, should look like this:



4. Save and run the experiment.
5. Visualize the output of the **Evaluate Model** module and verify that your optimized model is performing as expected.

## Summary

In this lab, you have used feature permutation and regularization to iteratively optimize a model. You have also tuned model hyperparameters to determine optimal parameter settings, and used cross-validation to check that your model will generalize well with new data.