Course: Machine Learning

Lab 2: Working with Classification Models

Overview

In this lab, you will train and evaluate a classification model. Classification is one of the fundamental machine learning methods used in data science. Classification models enable you to predict classes or categories of a label value. Classification algorithms can be two-class methods, where there are two possible categories, or multi-class methods. Like regression, classification is a supervised machine learning technique, wherein models are trained from labeled cases.

In this lab, you will use the dataset provided to categorize diabetes patients. The steps in this process include:

- Investigate relationships in the dataset with visualization using custom R or Python code.
- Create a machine learning classification model.
- Improve the model by pruning features, and sweeping parameter settings.
- Cross validate the model.
- Investigate model errors in detail with custom R or Python code.

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

Preparing and Exploring the Data

In this lab, you will work with a dataset that contains records of diabetes patients admitted to US hospitals. You will also train and evaluate a classification model to predict which hospitalized diabetes patients will be readmitted for their condition at a later date. Readmission of patients is both a metric of potential poor care as well as a financial burden to patients, insurers, governments, and health care providers.

Upload the Dataset

The full diabetes dataset requires considerable data cleaning and transformation. Additionally, the size of the dataset leads to fairly long model training time. To save this effort, you will upload a cleaned and reduced size version from the lab files following these steps:

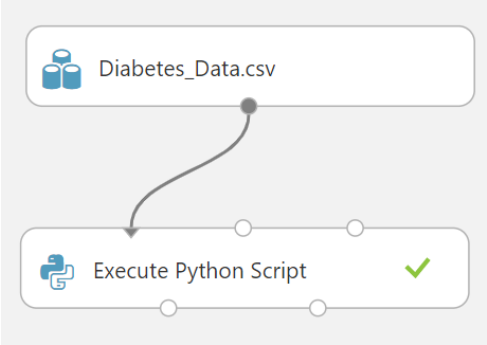
1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. Create a new blank experiment, and give it the title **Diabetes Classification**.

3. With the **Diabetes Classification** experiment open, at the bottom left, click **NEW**. Then in the **NEW** dialog box, click the **DATASET** tab.
4. Click **FROM LOCAL FILE**. Then in the **Upload a new dataset** dialog box, browse to select the **Diabetes_Data.csv** file from the folder where you extracted the lab files on your local computer and enter the following details, and then click the **OK** icon.
   - **This is a new version of an existing dataset:** Unselected
   - **Enter a name for the new dataset:** Diabetes_Data (Clean)
   - **Select a type for the new dataset:** Generic CSV file with a header (.csv)
   - **Provide an optional description:** Diabetes patient appointments
5. Wait for the upload of the dataset to be completed, and then on the experiment items pane, expand **Saved Datasets** and **My Datasets** to verify that the **Diabetes_Data (Clean)** dataset is listed.

Visualize the Data with Python

In this exercise, you will create custom Python code to visualize the dataset and examine the relationships. This dataset has 44 feature columns, with both numeric and categorical (string) features. The label column is titled **readmi_class**. The label column can have two values: "YES" and "NO". Having two values in the label makes this a two-class or binary classification problem.

1. Drag the **Diabetes_Data (Clean)** module onto the canvas.
2. Search for the **Execute Python Script** module and drag it onto the canvas. Connect the output of the dataset to the left input (**Dataset1**) port of the **Execute Python Script** module. At this point, your experiment should resemble the following.

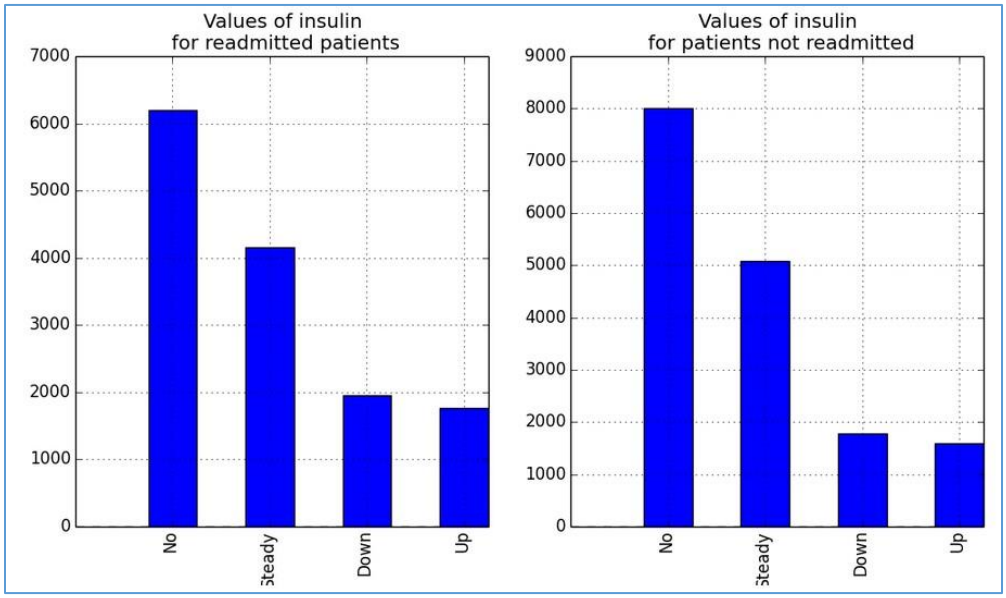


3. Click the **Execute Python Script** module, and in the **Properties** pane, replace the existing code with the following code, which you can copy from the **DiabetesVis.py.**

   This code creates bar plots for the categorical variables and box plots for numeric variables in the dataset with the following steps:

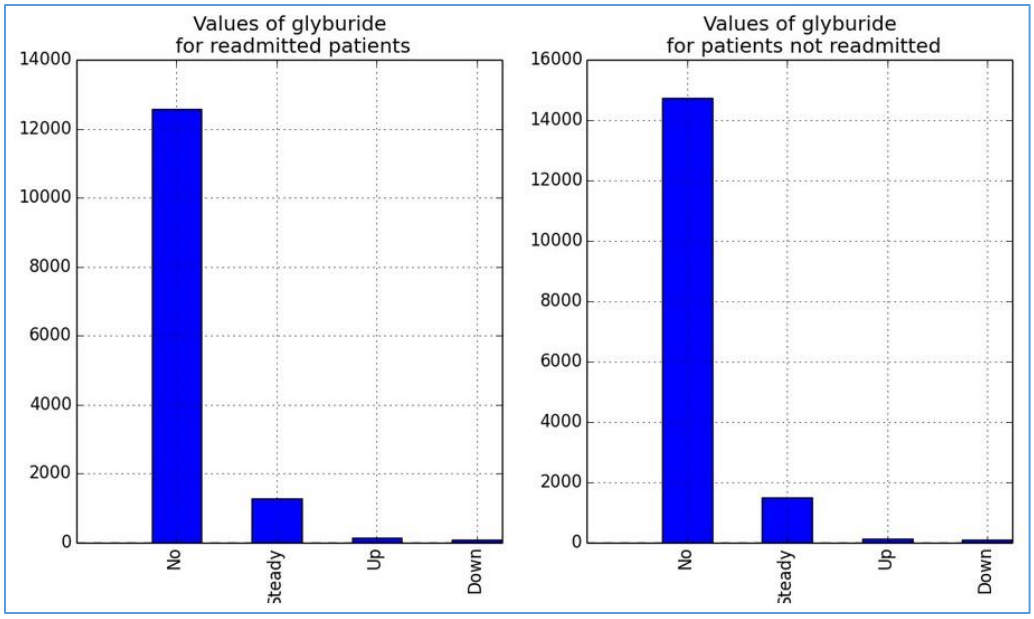
   - Create bar plots for each categorical column in the data frame. A subplot is created for each level of the label, **readmi_class**.
   - Create box plots for each categorical column in the data frame. A subplot is created for each level of the label, **readmi_class**.

4. Save and run the experiment. When the experiment has finished running, visualize the output from the **Python Device** port.
5. You will see a conditioned bar plot for each of the categorical features. Examine these bar plots to develop an understanding of the relationship between these features and the label values.
6. Examine the bar plots of the **insulin** feature, as shown in the following image.



Note, the vertical (frequency) scale is different for each value of the label readmitted or not readmitted. There are more total cases not readmitted than readmitted. The frequency of Down and Up are proportionately more likely for patients who are latter readmitted. This feature exhibits some differences between the two label values, indicating there is likely to have some predictive value. However, there is considerable overlap between the two values of the label, so separation will be prone to error based on this one feature.

7. Locate the bar plot of the **glyburide** feature, as shown in the following image.

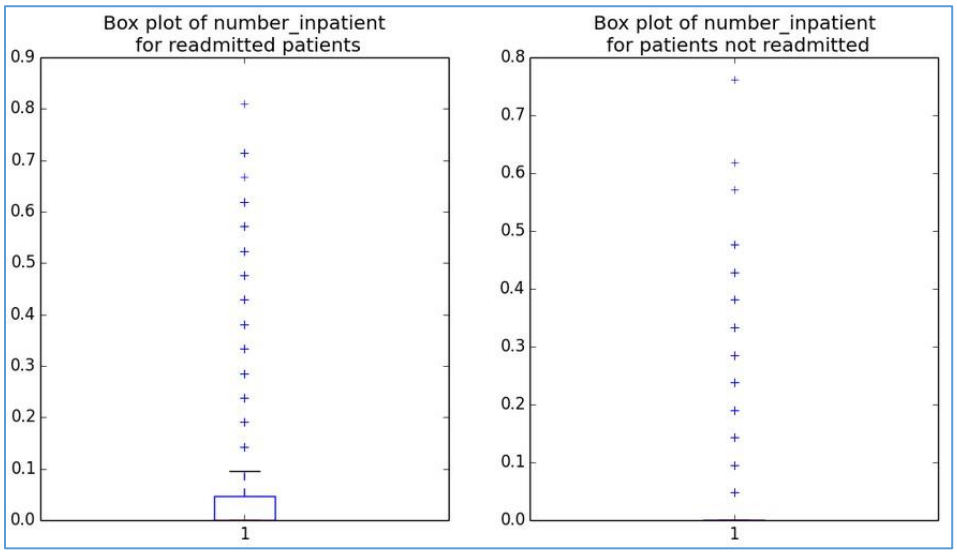


Examine this plot and note the reasons why this feature is unlikely to spate the label cases. Note, the vertical (frequency) scale is different for each value of the label readmitted or not readmitted. First, the relative frequencies of the four categories of the feature are nearly identical for the two label values. Second, some of the categories of the **glyburide** feature are fairly infrequent, meaning that even if there were significant differences, these values would only separate a minority of cases.

Further examination of the bar plots shows several features where only one category is plotted, indicating there are no other values in the data sample. These features cannot separate the label cases.

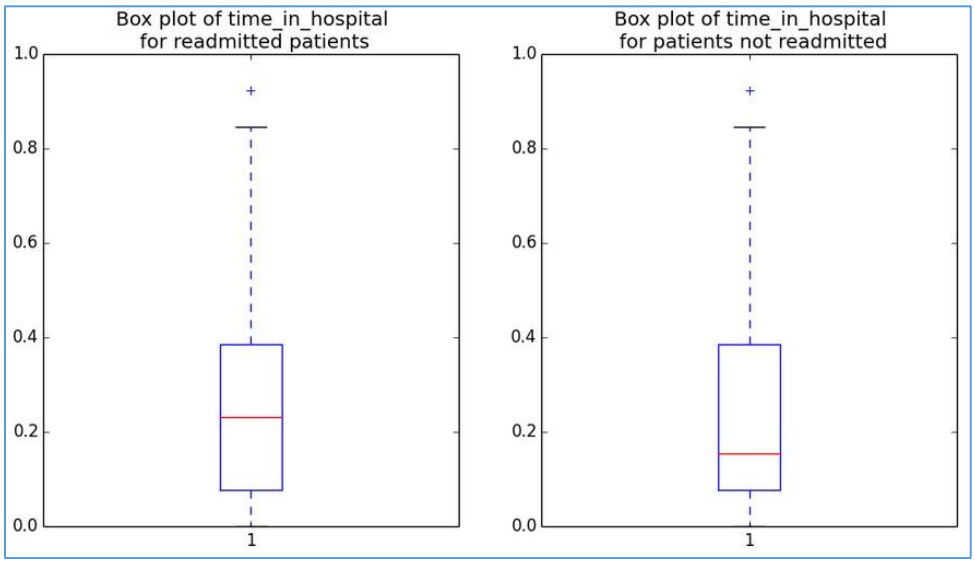
8. Locate the box plots of each numeric variable conditioned on the levels of the label. Examine these box plots to develop and understanding of the relationship between these features and the label values.

9. Locate the box plot of **number_inpatient**, which should resemble the following image.



Examine this plot and note the differences between the values for patients who have been readmitted and patients who have not been readmitted. These values have been normalized or scaled. In both cases, the median (black horizontal bar) is at zero, indicating significant overlap between the cases. For readmitted patients, the two upper quartiles (indicated by the box and the vertical line or whisker) shows little overlap with the other case. Based on these observations of overlap and difference, you can expect **number_inpatient** to separate some but not all cases.

10. Next, examine the box plots for the **time_in_hospital** feature as shown in the following image.



Examine this plot and note the differences between the values for patients who have been readmitted and patients who have not been not been readmitted. These values have been normalized or scaled. The median (black horizontal bar) is different for the two cases. However, the first upper and lower quartile (boxes) and the second upper and lower quartile (vertical line or whisker) show significant overlap. Based on these observations of overlap and difference, you can expect **time_in_hospital** to be a poor separator of the label cases.

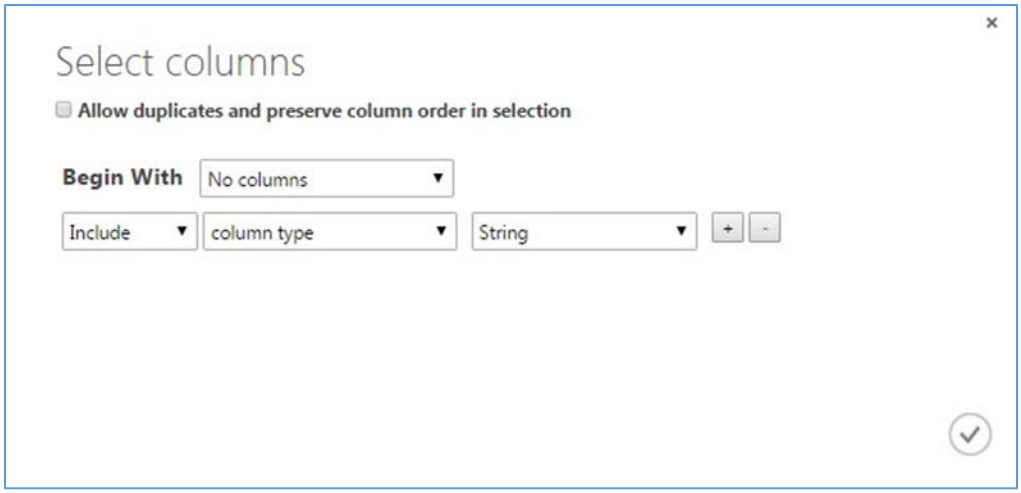
11 Can you draw any conclusions from the results?

Building a Classification Model

Now that you have investigated the relationships in the data you will build, improve and validate a machine learning model.

Create a New Model

1. Add a **Edit Metadata** module to your experiment
    a. Search for the **Edit Metadata** and drag it onto the canvas. Connect the output of the **Diabetes_Data (Clean)** dataset to the input of the **Edit Metadata**.
    b. Click the **Edit Metadata** and in the **Properties** pane, launch the **Column Selector**. Select all string columns as shown.



c. In the **Categorical** drop-down list, select **Make Categorical**.

2. Search for the **Select Columns in Dataset** module and drag it onto your canvas. Connect the **Results Dataset** output of the **Edit Metadata** module to the input port of the **Select Columns in Dataset** module.

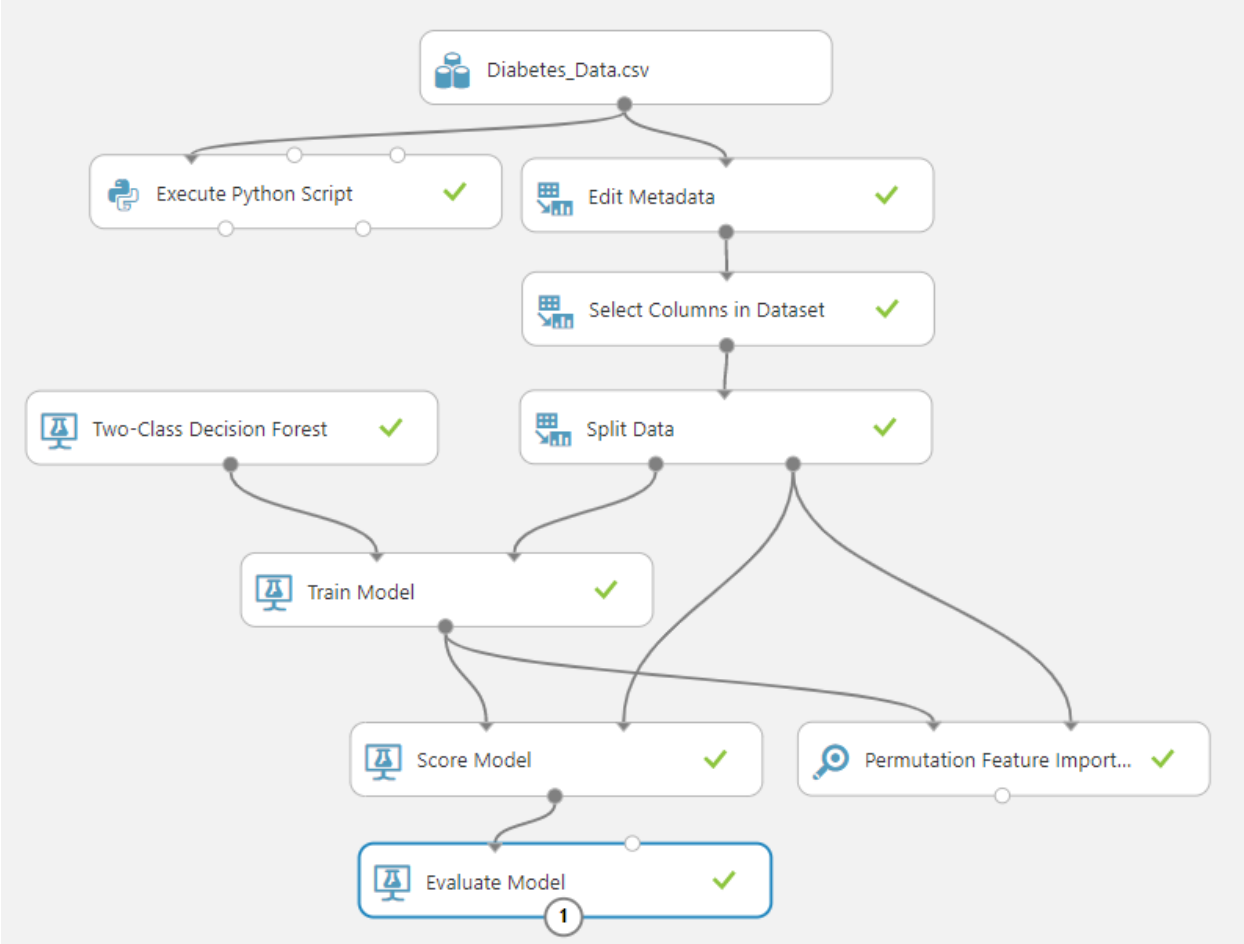
Note: With the **Select Columns in Dataset** module, you will remove features from the dataset found to be poor separators of the label cases during visualization of the dataset. For example, categorical features with only one category for both label cases are unable to separate these cases. Similarly, categorical features with one dominant category for both label cases are likewise unlikely to separate these cases. Such degenerate features can add noise or create generalization problems when the model is put into production.

---

3. With the **Select Columns in Dataset** module selected, in the **Properties** pane, launch the column selector, and exclude the following columns:
   - acetohexamide
   - glimepiride-pioglitazone
   - glipizide-metformin
   - metformin-pioglitazone
   - metformin-rosiglitazone
   - miglitol
   - tolazamide
   - troglitazone
4. Search for the **Split Data** module. Drag this module onto your experiment canvas. Connect the **Results dataset** output port of the **Select Columns in Dataset** module to the **Dataset** input port of the **Split Data** module. Set the **Properties** of the **Split Data** module as follows:
   - **Splitting mode:** Split Rows
   - **Fraction of rows in the first output:** 0.6

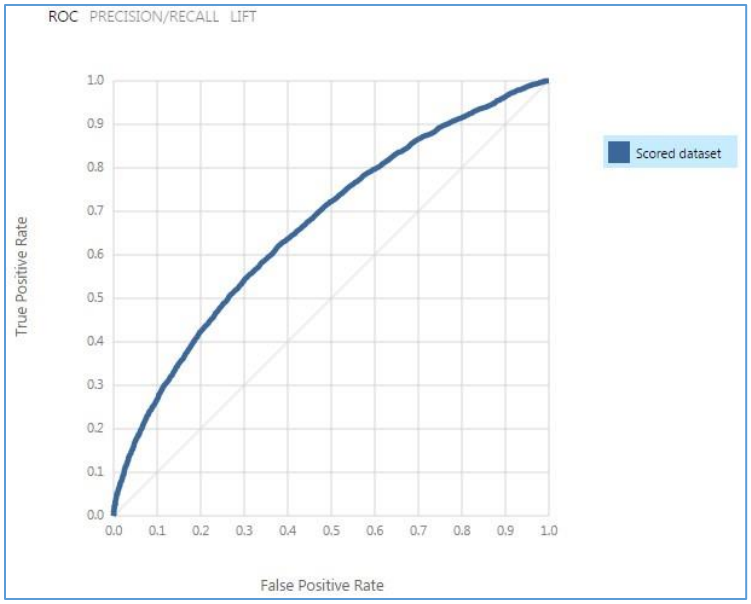
- **Randomized split:** Checked
- **Random seed:** 6789
- **Stratified split:** False

5. Search for the **Two Class Decision Forest** module. Make sure you have selected the regression model version of this algorithm. Drag this module onto the canvas. Set the Properties of this module as follows:
   - **Resampling method:** Bagging
   - **Create trainer mode:** Single Parameter
   - **Number of decision trees:** 40
   - **Maximum depth of the decision trees:** 32
   - **Number of random splits per node:** 128
   - **Minimum number of samples per leaf node:** 4
6. Search for the **Train Model** module. Drag this module onto the canvas.
7. Connect the **Untrained Model** output port of the **Two Class Decision Forest** module to the **Untrained Model** input port of the **Train Model** module. Connect the **Results dataset1** output port of the **Split Data** module to the **Dataset input** port of the **Train model** module. On the **Properties** pane, launch the column selector and select the **readmi_class** column.
8. Search for the **Score Model** module and drag it onto the canvas.
9. Connect the **Trained Model** output port of the of the **Train Model** module to the **Trained Model** input port of the **Score Model** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **Score Model** module.
10. Search for the **Permutation Feature Importance** module and drag it onto the canvas. Connect the Trained **Model** output port of the **Train Model** module to the **Trained model** input port of the **Permutation Feature Importance** module. Connect the **Results dataset2** output port of the **Split Data** module to the **Dataset** port of the **Test data** input port of the **Permutation Feature Importance** module.
11. Search for the **Evaluate Model** module and drag it onto the canvas. Connect the **Scored Dataset** output port of the **Score Model** module to the left hand **Scored dataset** input port of the **Evaluate Model** module. The new portion of your experiment, below the **Edit Metadata**, should now look like the following.

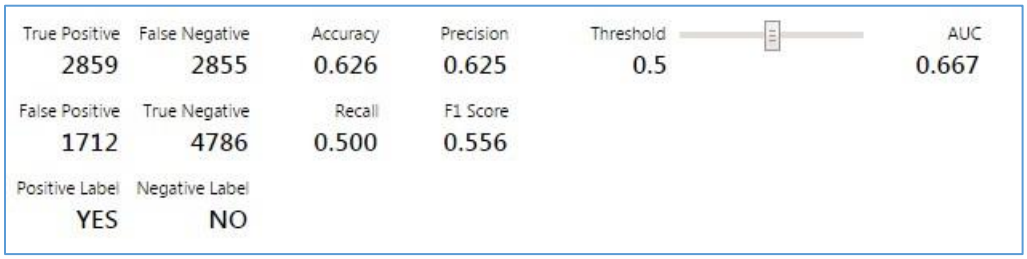


12. Save and run the experiment. When the experiment is finished, visualize the **Evaluation Result** port of the **Evaluate Model** module and review the ROC curve and performance statistics for the model as shown in the following image.

Reminder: The goal of this classification problem is to prevent patients from requiring readmission to a hospital for their diabetic condition. Correctly identifying patients who are likely to require further treatment allows medical care providers to take actions that can prevent this situation. Patients with a false negative score will not be properly classified, and subsequently will require readmission to a hospital. Therefore, the recall statistic is important in this problem since maximizing recall minimizes the number of false negative scores.



13. Examine this ROC curve. Notice that the bold blue line is well above the diagonal grey line, indicating the model is performing significantly better than random guessing. The AUC is 0.667 (see below), which is significantly more than 0.5 obtained by random guessing.
14. Next, examine the performance statistics provided, as shown here.



Notice the following:

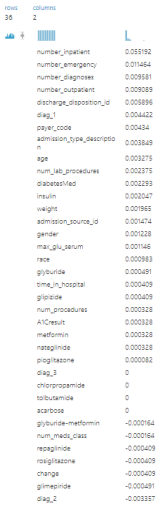
- In the confusion matrix, there are nearly equal numbers of **True Positive** and **False Negative** scores.
- The equal numbers of **True Positive** and **False Negative** scores gives rise to a recall value of 0.5.
- Overall **Accuracy** is 0.626, indicating the scores are correct more often than not.

15. Close the evaluation results.
16. Can you draw any conclusions from the results?

Prune Features

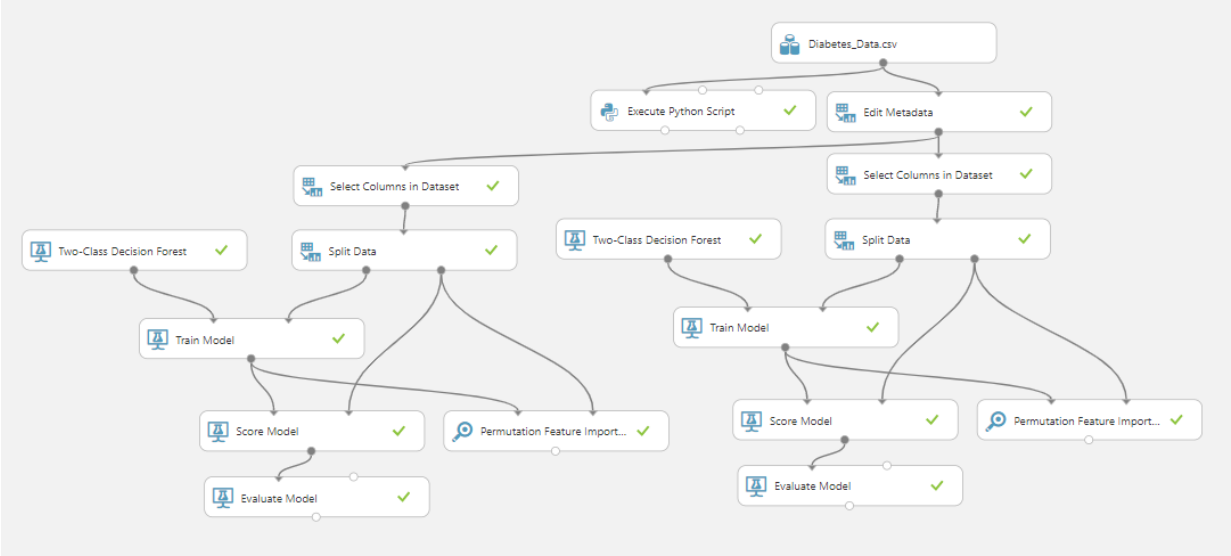
You will now improve model performance by pruning less important features by following these steps:

1. Visualize the output of the **Permutation Feature Importance** module. The upper portion of the list produced should resemble the following.



Notice that several of these features have a 0 or very nearly zero importance.

2. Copy all of the modules in the experiment from the **Select Columns in Dataset** module onwards. Paste these modules onto the experiment canvas.
3. Connect the output of the **Edit Metadata** module to the input port of the new **Select Columns in Dataset** module.
4. Select the new **Select Columns in Dataset** module, and in the **Properties** pane, launch the column selector. Modify the module to exclude the following columns, in addition to the ones already excluded:
    - chlorpropamide
    - tolbutamide
    - repaglinide
    - num_meds_class
    - nateglinide
    - rosiglitazone
    - acarbose
    - glimepiride
    - glyburide-metformin.
5. Verify that your experiment from the Edit Metadata down resembles the following image.



6. Save and run the experiment. When the experiment is finished, visualize the output of the new **Evaluate Model** module.

7. Examine the summary statistics, as shown in the following image.

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
2884	2830	0.628	0.628	0.5	0.667
False Positive	**True Negative**	**Recall**	**F1 Score**		
1712	4786	0.505	0.559		
Positive Label	**Negative Label**				
YES	NO				

Note that the **Accuracy** and **Recall** have improved slightly, while **AUC** is unchanged. Pruning these features was a good decision.

8. Visualize the output of the **Permutation Feature Importance** module. The lower part of the list should resemble the following list.

number_diagnoses	0.011376
num_procedures	0.0084
glyburide	0.006125
time_in_hospital	0.00595
diag_2	0.0056
diabetesMed	0.004375
admission_type_description	0.004375
payer_code	0.004025
metformin	0.003325
race	0.00315
insulin	0.00245
diag_1	0.002275
glipizide	0.002275
diag_3	0.0021
gender	0.001925
change	0.00105
weight	0.0007
number_outpatient	0.00035
admission_source_id	-0.0007
max_glu_serum	-0.0007
pioglitazone	-0.0007
num_lab_procedures	-0.002975
A1Cresult	-0.00385
age	-0.004025
number_emergency	-0.008925

Note the several features with near-zero importance. You could continue an iterative pruning process, testing the results until model performance begins to worsen.

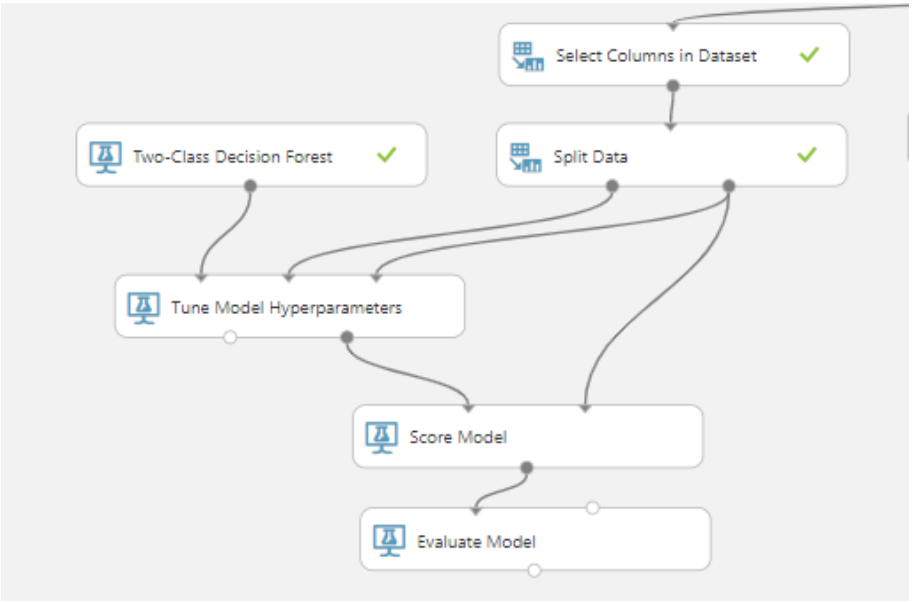
Sweep Model Parameters

You will now improve the machine learning model by sweeping the parameter space.

1. Remove the second (newest) **Train Model** and **Permutation Feature Importance** modules from the experiment.
2. Search for the **Tune Model Hyperparameters** module. Drag this module onto the canvas in place of the **Train Model** module you removed, and reconnect the experiment modules as follows:
    a. Connect the **Untrained model** output port of the **Two-Class Decision Forest** module to the **Untrained model** input port of the **Tune Model Hyperparameters** module.
    b. Connect the **Results dataset1** output port of the **Split Data** module to the **Training dataset** input port of the **Tune Model Hyperparameters** module.
    c. Connect the **Results dataset2** output port of the **Split Data** module to the **Optional test dataset** input port of the **Tune Model Hyperparameters** module.

d. Connect the **Trained best model** (right-hand) output of the **Tune Model Hyperparameters** module to the **Trained Model** input of the **Score Model** module.

3. Click the **Tune Model Hyperparameters** module to expose the **Properties** pane. Set the properties as follows so that 30 combinations of parameters are randomly tested to predict the **readmi_class** variable:
   - **Specify parameter sweeping mode:** Random sweep
   - **Maximum number of runs on random sweep:** 30
   - **Random seed:** 4567
   - **Column Selector:** readmi_class
   - **Metric for measuring performance for classification:** Recall
   - **Metric for measuring performance for regression:** Mean absolute error (this doesn't matter for a classification model)
4. Verify that the new portion of you experiment resembles the following image.



5. Save and run the experiment. With 30 iterations, the experiment may take a while to run.
6. Visualize the output of the **Evaluate Model** module. The performance statistics should appear as shown here.

You should note that **Recall** and **AUC** have been improved at the expense of accuracy.

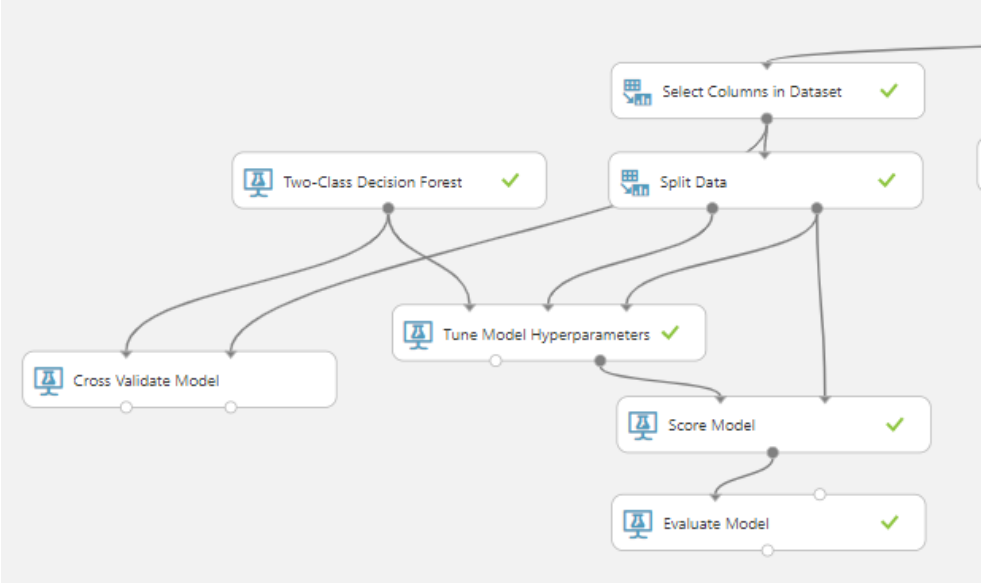
7 Can you draw any conclusions from the results?

Cross Validate the Model

You will cross validate your model by following these steps:

1. Search for the **Cross Validate Model** module. Drag this module onto the canvas. Connect the **Untrained model** output from the **Two-Class Decision Forest** module to the **Untrained model** input port of the **Cross Validate Model** module. Connect the **Results dataset** output port of the **Select Columns in Dataset** module to the **Dataset input** port of the **Cross Validate Model** module.
2. Click the **Cross Validate Model** module to expose the **Properties** pane. Set the properties as follows:
   - **Column Selector:** readmi_class
   - **Random seed:** 3467

   This portion of your experiment should resemble the following.



3. Save and run the experiment.
4. When the experiment has run, click the **Evaluation Results by Fold** output port of the **Cross Validation Model** and select **Visualize**. Scroll to the right and note the **Accuracy, Recall**, and **AUC** columns (the first, third, and fifth numeric columns from the left). Scroll to the bottom of the page, past the results of the 10 folds of the cross validation, and examine the **Mean** value row. These results look like the following.

rows	columns							
12	10							
	5	3053	es.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.622011	0.601516	0.511828	0.55306	0
	6	3053	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.628235	0.633921	0.519257	0.570888	0
	7	3053	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.641336	0.617421	0.527737	0.569067	0
	8	3053	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.612185	0.586118	0.493863	0.53605	0
	9	3053	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.653128	0.652209	0.556953	0.600829	0
	Mean	30530	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.631182	0.618198	0.521778	0.565824	0
	Standard Deviation	30530	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.012367	0.019262	0.020428	0.018807	0

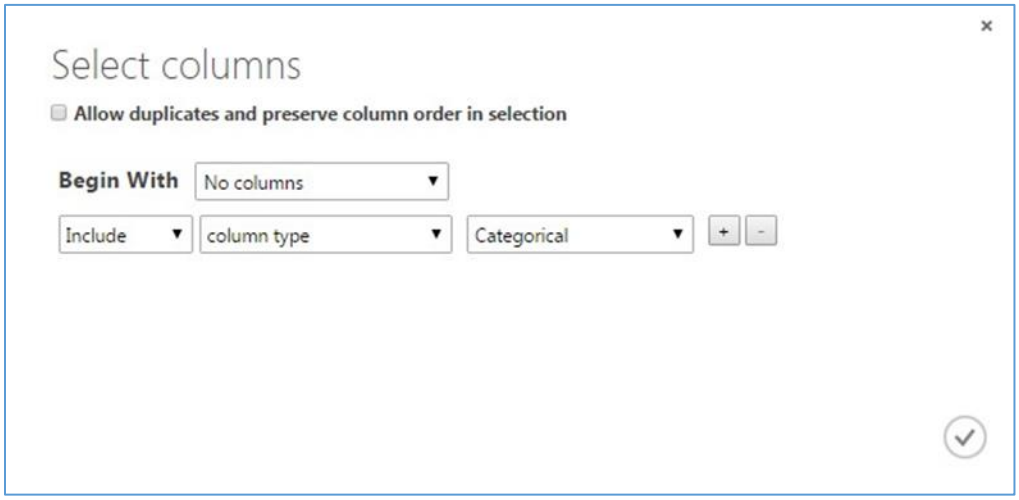
Notice that the **Accuracy, Recall** and **AUC** values in the folds are not that different from each other. The values in the folds are close to the **Mean**. Further, the **Standard Deviation** is much smaller than the **Mean**. These consistent results across the folds indicate that the model is insensitive to the training and test data chosen and should generalize well.

5 Can you draw any conclusions from the results?

Visualize Errors with Python

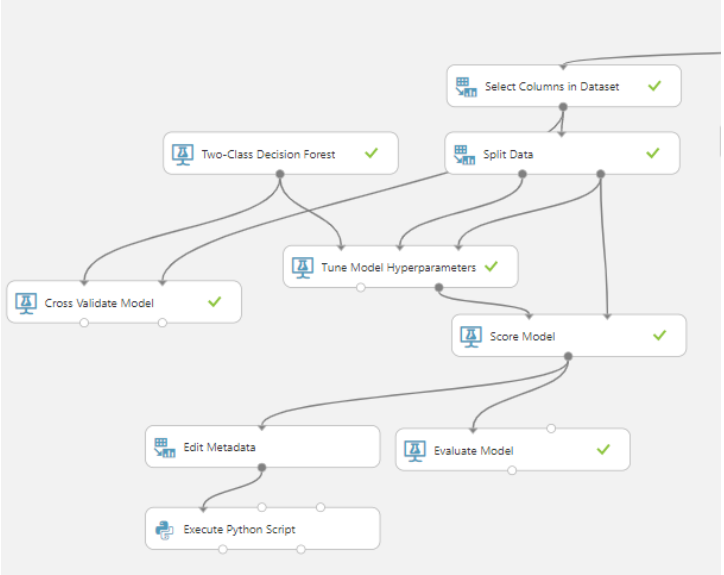
The summary performance statistics for the **Two-Class Decision Forest** model look fairly good. However, summary statistics can hide some significant problems one should be aware of. To investigate the model errors, you will use some custom Python code. Now, preform the following steps:

1. Search for the **Edit Metadata** module and drag it onto your canvas. Connect the **Scored Dataset** output of the most recently added **Score Model** module to the input of the **Edit Metadata** module.
2. Click the **Edit Metadata** model. In the **Properties** pane, click **Launch Column Selector**. Choose all columns of **Categorical** type as shown in the following image.



3. In the **Categorical** list, select **Make non-categorical**. The output from this **Edit Metadata** model will now have features of string type rather than categorical type.

4. Search for the **Execute Python Script** module. Drag this module onto your canvas. Connect the **Results Dataset** output of the **Edit Metadata** module to the **Dataset1** (left hand) input of the **Execute Python Script** module. Your experiment should resemble the following image.



5. With the **Execute Python Script** module selected, in the Properties pane, replace the existing Python script with the code from **DiabetesEval.py**.

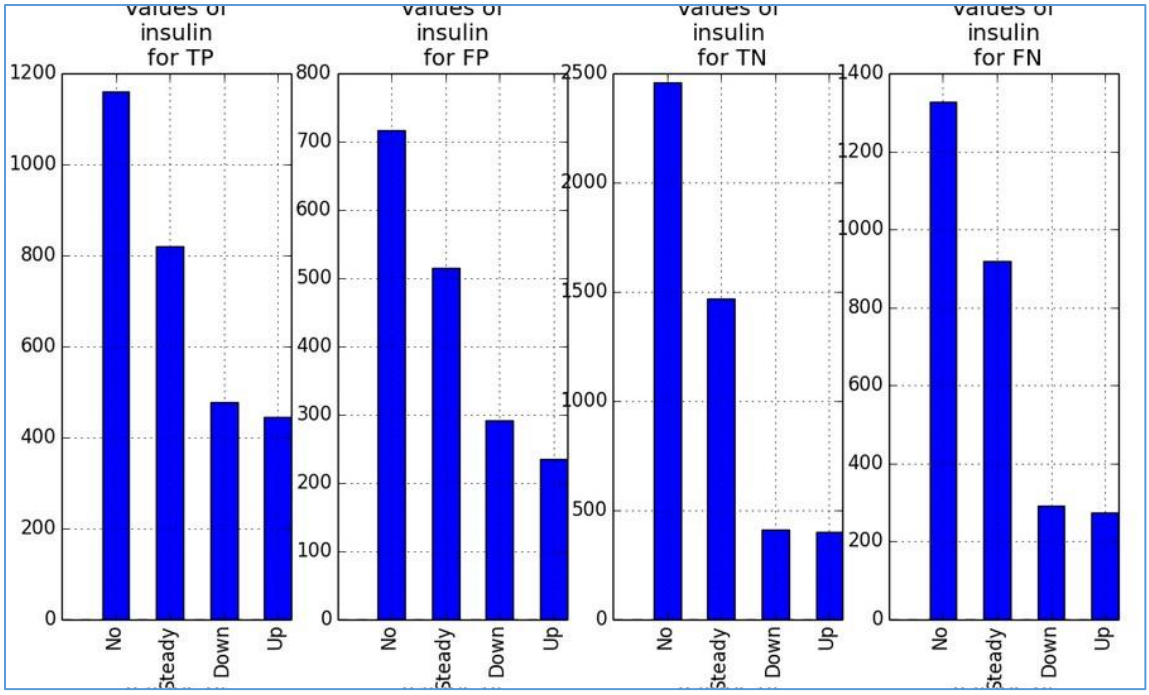
   This code does the following:

   - Compute scores for each case; TP, FP, TN, FN.
   - Make bar plots of the categorical features conditioned on the score.
   - Make box plots of the numeric features conditioned on the score.

   Note: By visualizing scored cases, you can determine which features may have unexploited information that could be used to improve the model. In this case, minimizing false negative (**FN**) scores is a priority. **FN** scores result in a patient being readmitted to a hospital for their diabetic condition. Looking for features where there is a difference between the cases scored as **FN** versus cases scored as true positives (**TP**) can guide further machine learning improvements.

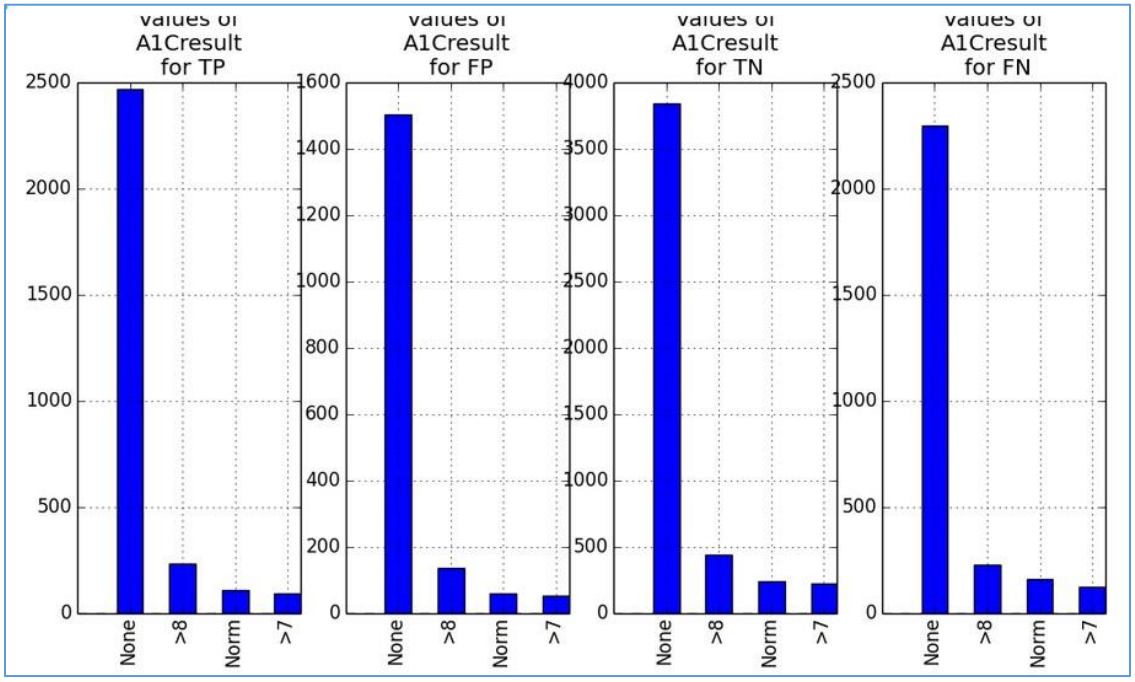
6. Save and run the experiment. Then, when the experiment is finished, visualize the **Python Device port** of the **Execute Python Script** module.

7. Examine the bar plots, noting the differences between the **TP** and **FN** scores. Note the plot for **insulin**.



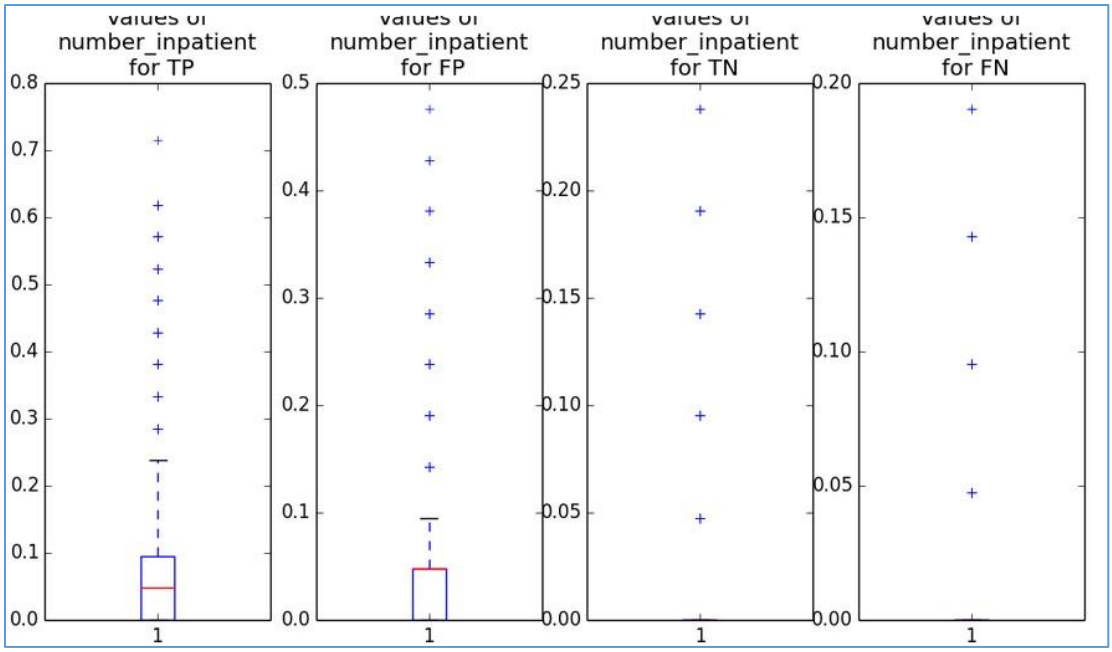
Examine this plot and notice the difference between the **TP** and **FN** scores. The **Down** and **Up** categories are proportionately less likely in the **FN** case. This information might be more fully exploited to improve model performance.

8. Note the differences in the plot for **A1Cresult**.



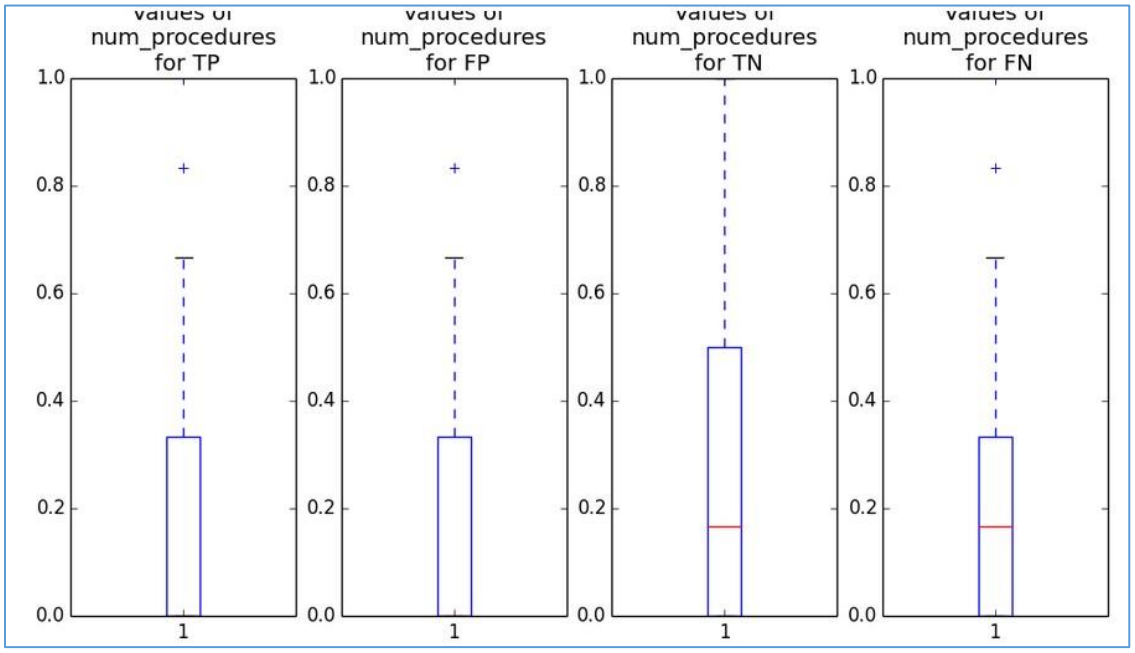
The proportion of the categories for the **TP** and **FN** scores are nearly identical. It is unlikely there is any additional information in this feature which can be exploited.

9. Next examine the box plots, noting the differences between the **TP** and **FN** scores. Note the plot for **number_inpatient** as shown here.



There is little overlap between the distribution of values for the **TP** and **FN** scores. It is likely that this feature contains additional information that can be used to improve the model.

10. Next, examine the box plot for **num_procedures** as shown here.



Although there are differences between the **TP** and **FN** case, there is considerable overlap in the distribution of these data. It is unlikely that this feature will yield any additional information that can be exploited to improve the model.

11. Close the Python device output.
12. Can you draw any conclusions from the results?

Summary

In this lab, you have constructed and evaluated a two-class or binary classification model. Highlights from the results of this lab are:

- Visualization of the dataset can help differentiate features, which separate the cases from those that are unlikely to do so.
- Feature pruning and parameter sweeping can improve model performance.
- Cross validation can indicate how well a model will generalize.
- Examining the classification behavior of features can highlight potential performance problems or provide guidance on improving a model.