Data Science and Machine Learning Essentials

Lab 4B – Working with Classification Models

By Stephen Elston and Graeme Malcolm

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 data set provided to categorize diabetes patients. The steps in this process include:

- Investigate relationships in the data set 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
- Python Anaconda or R and RStudio
- The lab files for this lab

Note: To set up the required environment for the lab, follow the instructions in the **Setup** document for this course. Then download and extract 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. In this lab you will 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 Data Set

The full diabetes data set 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:

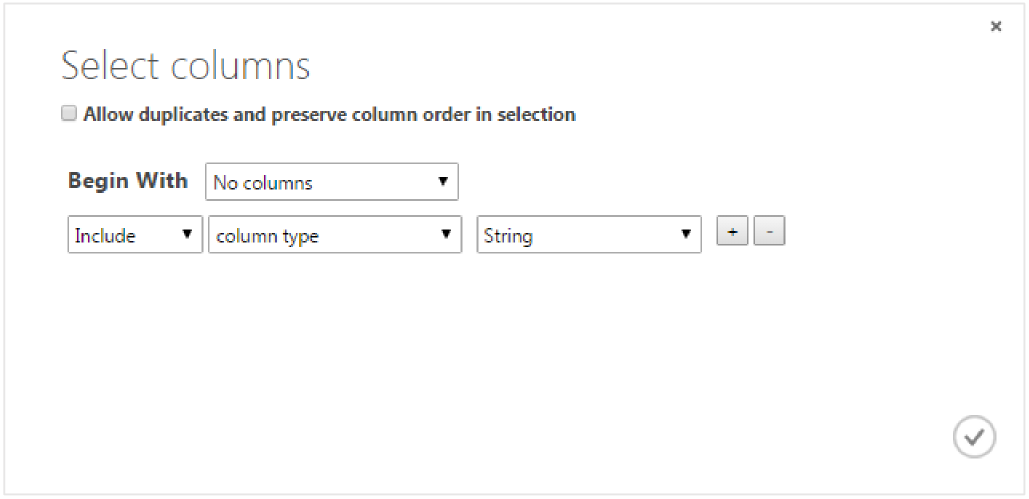
1. If you have not already done so, open a browser and browse to https://studio.azureml.net. Then sign in using the Microsoft account associated with your Azure ML account.
2. Create a new blank experiment, and give it the title **Diabetes Classification**.
3. With the **Diabetes Classification** experiment open, at the bottom left, click **NEW**. Then in the **NEW** dialog box, click the **DATASET** tab as shown in the following image.
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 as shown in the image below, 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 R

In this exercise you will create custom R code to visualize the data set and examine the relationships. This data set has 44 feature columns, with both numeric and categorical (string) features. The label column is named **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.

Note: If you prefer to work with Python, complete the *Visualize the Data with Python* exercise.

1. Drag the **Diabetes_Data (Clean)** dataset onto the canvas.
2. Search for the **Metadata Editor** and drag it onto the canvas. Connect the output of the **Diabetes_Data (Clean)** data set to the input of the **Metadata Editor**.
3. Click the **Metadata Editor** and on the properties pane launch the **Column Selector**. Select all string columns as shown:



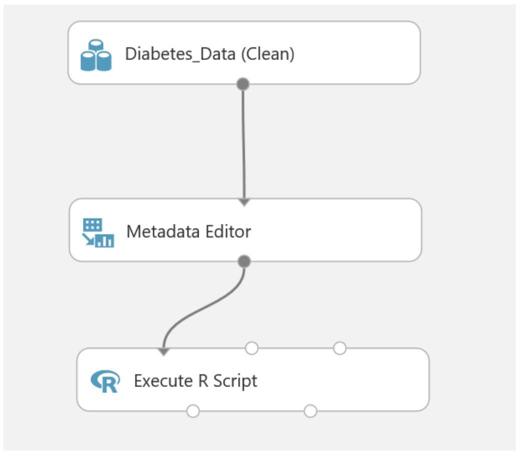
Select columns

Allow duplicates and preserve column order in selection

Begin With: No columns

Include | column type | String

4. In the **Categorical** drop down list, select **Make Categorical**.
5. Search for the **Execute R Script** module and drag it onto the canvas. Then connect the **Metadata Editor** module to the first input port of the **Execute R Script** module. At this point your experiment should resemble the following image.



6. Click the **Execute R Script** module, and in the Properties pane, replace the existing code with the following code, which you can copy from the **DiabetesVis.R** file:

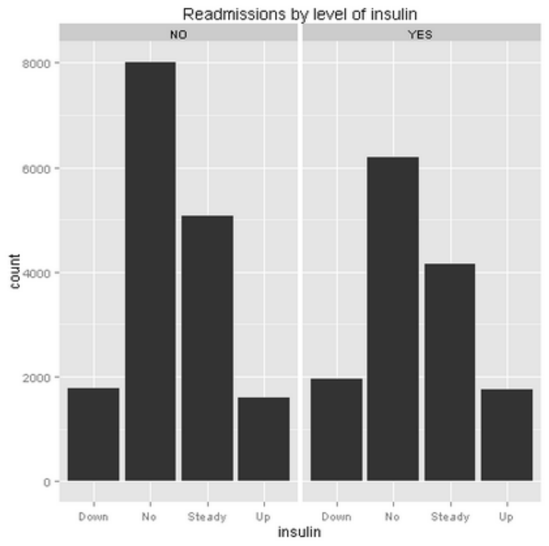
```
frame1 <- maml.mapInputPort(1)
library(ggplot2)
library(dplyr)

## Compare outcomes for various levels of
## factor (categorical) features.
bar.plot <- function(x){
  if(is.factor(frame1[, x])){
    sums <- summary(frame1[, x], counts = n())
    msk <- names(sums[which(sums > 100)])
    tmp <- frame1[frame1[, x] %in% msk, c('readmi_class', x)]
    if(strsplit(x, '[-]')[[1]][1] == x){
      g <- ggplot(tmp, aes_string(x)) +
        geom_bar() +
        facet_grid(. ~ readmi_class) +
        ggtitle(paste('Readmissions by level of', x))
      print(g)
    }
  }
}
cols <- names(frame1)
cols <- cols[1:(length(cols) - 1)]
lapply(cols, bar.plot)

## Make box plots of the numeric columns
box.plot <- function(x){
  if(is.numeric(frame1[, x])){
    ggplot(frame1, aes_string('readmi_class', x)) +
      geom_boxplot() +
      ggtitle(paste('Readmissions by', x))
  }
}
lapply(names(frame1), box.plot)
```

Tip: To copy code in a local code file to the clipboard, press **CTRL+A** to select all of the code, and then press **CTRL+C** to copy it. To paste copied code into the code editor in the Azure ML **Properties** pane, press **CTRL+A** to select the existing code, and then press **CTRL+V** to paste the code from the clipboard, replacing the existing code.

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

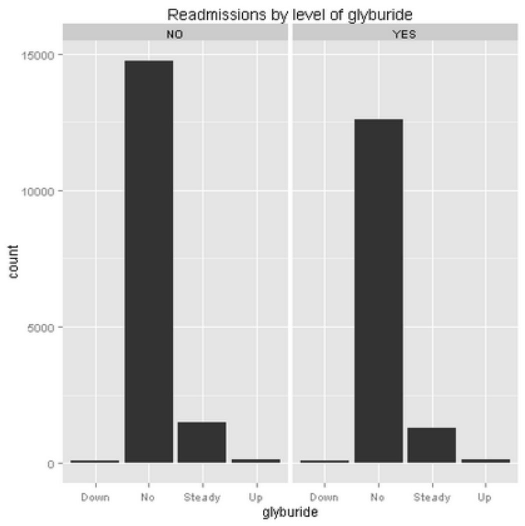
- Define a function to create bar plots. This function filters categorical (factor) variables with greater than 100 members (out of 30,000). The plots are conditioned on the label, **readmi_class**.
- The **bar.plot** function is applied to the columns of the data frame, less the label column.
- Define a function to create box plots of the numeric columns, conditioned by the label column, **readmi_class**.
- The **box.plot** function is applied to the columns of the data frame.

7. Save and run the experiment. When the experiment has finished running, visualize the output from the **R Device** port. You will see a conditioned bar plot for each of the categorical features. Examine these bar plots to develop and understanding of the relationship between these features and the label values.
8. Examine the bar plot of the **insulin** feature, as shown below:



Note, the vertical (frequency) scale is identical for each value of the label ('YES', 'NO'). There are more total cases not readmitted ('NO') than readmitted ('YES'). The frequency of Down and Up are proportionately more likely for patients who are latter readmitted ('YES'). This feature exhibits some difference 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.

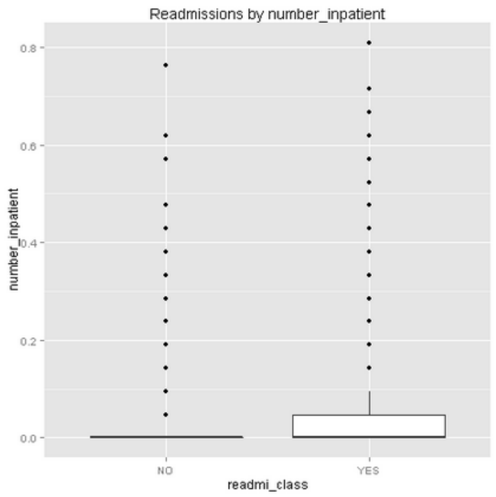
9. Locate the bar plot of the glyburide feature, as shown below:



Examine this plot and note the several reasons why this feature is unlikely to spate the label cases. 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 difference 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 less than 100 cases with any other value. These features are unlikely to separate the label cases.

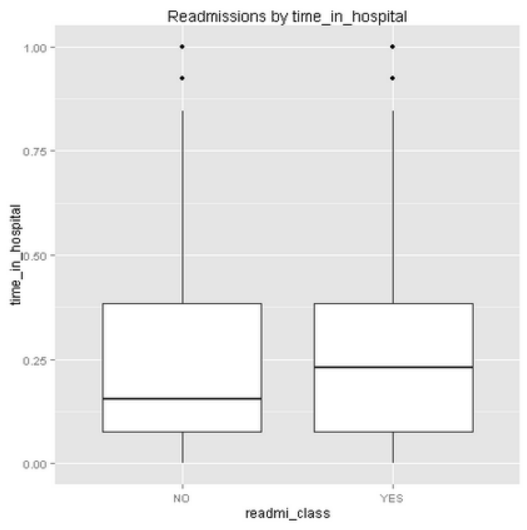
10. Locate the box plots of each numeric variable conditioned on the levels of the label. Examine these box plots to develop and understanding of the relationship between these features and the label values.
11. Locate the box plot of **number_inpatient** as shown below:



Examine this plot and note the differences between the values for patients who have been readmitted ('YES') and patients who have not been readmitted ('NO'). 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 ('YES'), the two upper quartiles (indicated by the box and the vertical line or whisker) shows little overlap with the other case

(‘NO’). Based on these observations of overlap and difference you can expect **number_inpatient** to separate some but not all cases.

12. Next, examine the box plots for the **time_in_hospital** feature as shown below:



Examine this plot and note the differences between the values for patients who have been readmitted (‘YES’) and patients who have not been not been readmitted (‘NO’). 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.

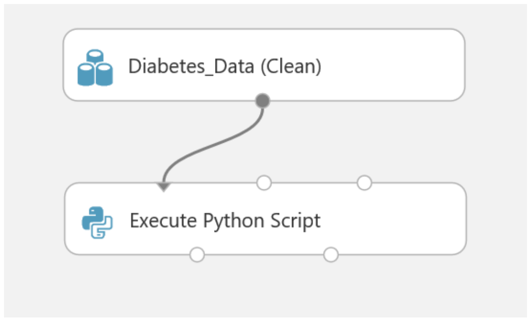
13. Close the R Device output

Visualize the Data with Python

In this exercise you will create custom Python code to visualize the data set and examine the relationships. This data set 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.

Note: If you prefer to work with R, complete the *Visualize the Data with R* exercise.

1. Drag the **Diabetes_Data (Clean)** and onto the canvas.
2. Search for the **Execute Python Script** module and drag it onto the canvas. Connect the output of the data set 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** file in the folder where you extracted the lab files:

```
def azureml_main(frame1):
    import matplotlib
    matplotlib.use('agg')

    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import statsmodels.graphics.boxplots as sm

    Azure = True

## Create a series of bar plots for the various levels of the
## string columns in the data frame by readmi_class.
    names = list(frame1)
    num_cols = frame1.shape[1]
   for indx in range(num_cols - 1):
        if(frame1.ix[:, indx].dtype not in [np.int64, np.int32,
np.float64]):
            temp1 = frame1.ix[frame1.readmi_class == 'YES',
indx].value_counts()
            temp0 = frame1.ix[frame1.readmi_class == 'NO',
indx].value_counts()

            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax1 = fig.add_subplot(1, 2, 1)
            ax0 = fig.add_subplot(1, 2, 2)
            temp1.plot(kind = 'bar', ax = ax1)
            ax1.set_title('Values of ' + names[indx] + '\n for
readmitted patients')
            temp0.plot(kind = 'bar', ax = ax0)
            ax0.set_title('Values of ' + names[indx] + '\n for
patients not readmitted')

            if(Azure == True): fig.savefig('bar_' + names[indx] +
'.png')


## Now make some box plots of the columbns with numerical values.
    for indx in range(num_cols):
        if(frame1.ix[:, indx].dtype in [np.int64, np.int32,
np.float64]):
            temp1 = frame1.ix[frame1.readmi_class == 'YES', indx]
            temp0 = frame1.ix[frame1.readmi_class == 'NO', indx]

            fig = plt.figure(figsize = (12,6))
            fig.clf()
            ax1 = fig.add_subplot(1, 2, 1)
            ax0 = fig.add_subplot(1, 2, 2)
            ax1.boxplot(temp1.as_matrix())
            ax1.set_title('Box plot of ' + names[indx] + '\n for
readmitted patients')
            ax0.boxplot(temp0.as_matrix())
            ax0.set_title('Box plot of ' + names[indx] + '\n for
patients not readmitted')

            if(Azure == True): fig.savefig('box_' + names[indx] +
'.png')

    return frame1
```

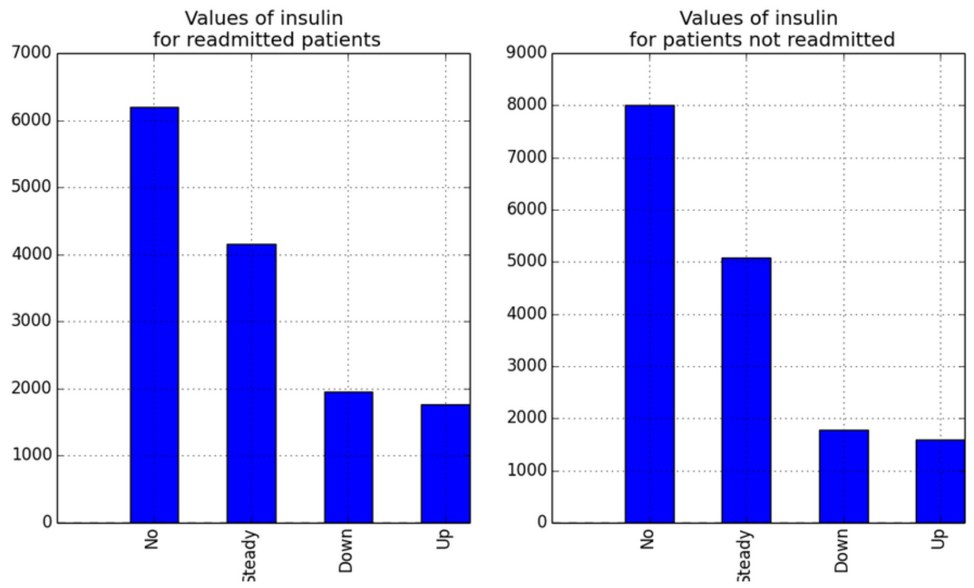
Tip: To copy code in a local code file to the clipboard, press **CTRL+A** to select all of the code, and then press **CTRL+C** to copy it. To paste copied code into the code editor in the Azure ML **Properties** pane, press **CTRL+A** to select the existing code, and then press **CTRL+V** to paste the code from the clipboard, replacing the existing code.

WARNING!: Ensure you have a Python `return` statement at the end of your `azureml_main` function; for example, `return frame1`. Failure to include a `return` statement will prevent your code from running and may produce an inconsistent error message.

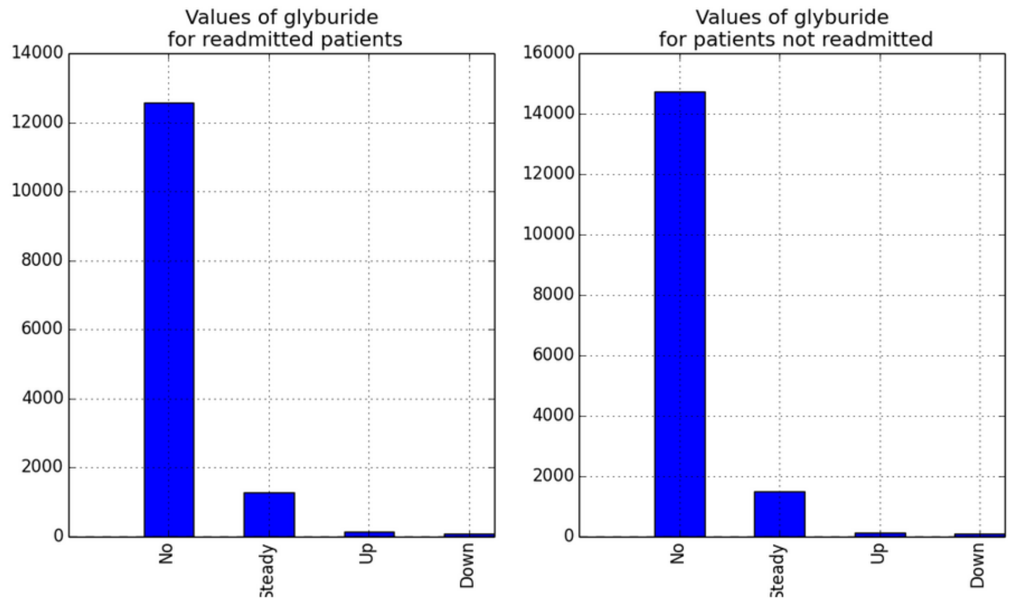
This code creates bar plots for the categorical variables and box plots for numeric variables in the data set with the following steps:

- 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 and understanding of the relationship between these features and the label values.
6. Examine the bar plots of the **insulin** feature, as shown below:



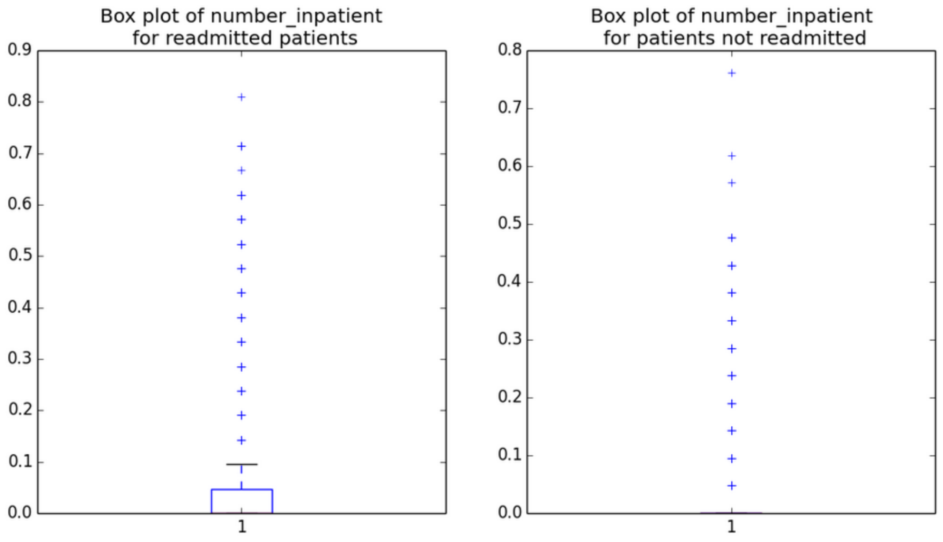
7. Note, the vertical (frequency) scale is different for each value of the label readmitted or not readmitted. There are more total cases not readmitted than readmitted. The frequency of Down and Up are proportionately more likely for patients who are latter readmitted. This feature exhibits some difference 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.
8. Locate the bar plot of the **glyburide** feature, as shown below:



Examine this plot and note the several 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 difference these values would only separate a minority of cases.

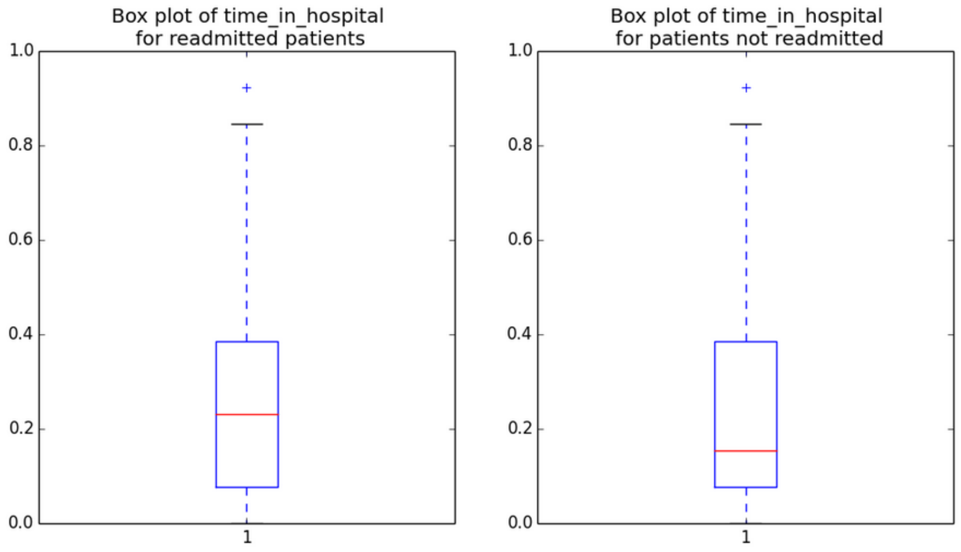
Further examination of the bar plots shows several features where only one category is plotted, indicating there are no other values in the data sample. These features cannot separate the label cases.

9. 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.
10. Locate the box plot of **number_inpatient** which should resemble the figure below.



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.

11. Next, examine the box plots for the **time_in_hospital** feature as shown below.



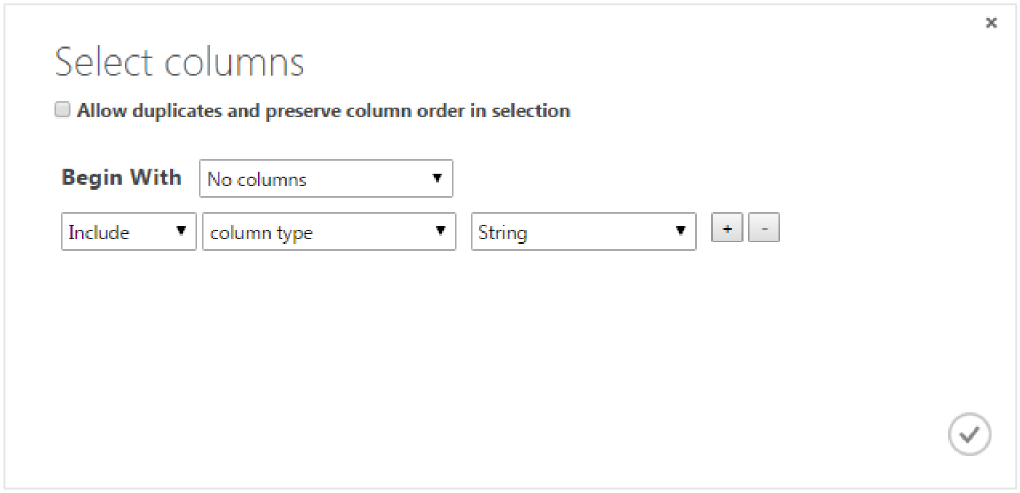
Examine this plot and note the differences between the values for patients who have been readmitted and patients who have not been not been readmitted. These values have been normalized or scaled. The median (black horizontal bar) is different for the two cases. However, the first upper and lower quartile (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.

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. If you are working with Python, you need to add a **Metadata Editor** module to your experiment by following steps a, b and c below. If you are working with R, you have already added this **Metadata Editor** module to your experiment, and you should proceed to step 2.
    a. Search for the **Metadata Editor** and drag it onto the canvas. Connect the output of the **Diabetes_Data (Clean)** data set to the input of the **Metadata Editor**.
    b. Click the **Metadata Editor** and in the properties pane, launch the **Column Selector**. Select all string columns as shown:



Select columns

Allow duplicates and preserve column order in selection

Begin With: No columns

Include | column type | String

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

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

   Note: With the **Project Columns** module, you will remove features from the data set 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.

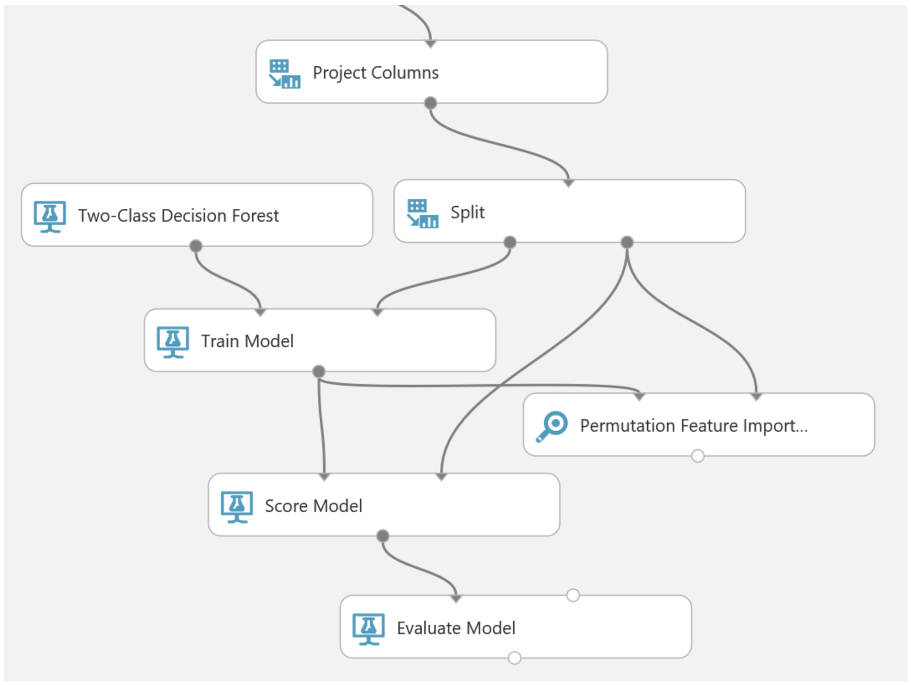
3. With the **Project Columns** module selected, in the properties pane, launch the column selector, and exclude the following columns:
    - acetohexamide

- glimepiride-pioglitazone
- glipizide-metformin
- metformin-pioglitazone
- metformin-rosiglitazone
- miglitol
- tolazamide
- troglitazone

4. Search for the **Split** module. Drag this module onto your experiment canvas. Connect the **Results dataset** output port of the **Project Columns** module to the **Dataset** input port of the **Split** module. Set the **Properties** of the **Split** 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

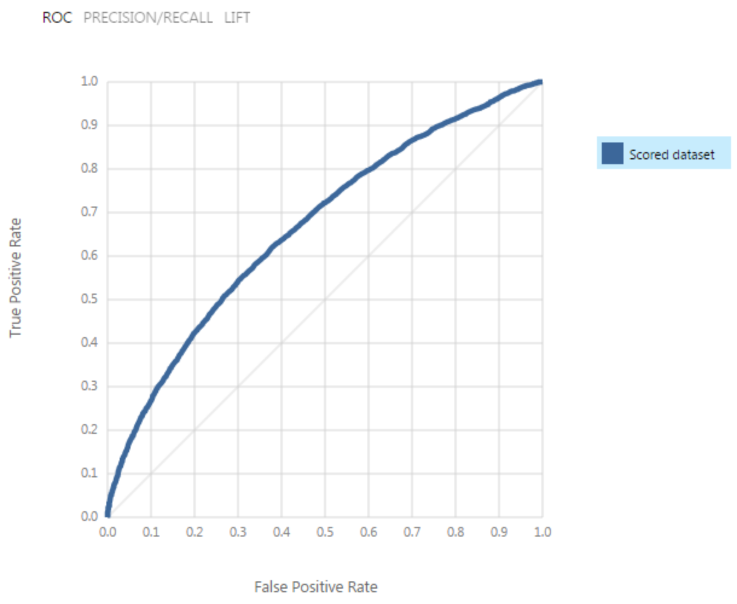
5. Search for the **Two Class Decision Forest** module. Make sure you have selected the regression model version of this algorithm. Drag this module onto the canvas. Set the Properties if this module as follows:
   - **Resampling method:** Bagging
   - **Create trainer mode:** Single Parameter
   - **Number 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** 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** 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** module to the **Dataset** port of the **Test data** input port of the **Permutation Feature Importance** module.
4. 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. New portion of your experiment, below the **Metadata Editor**, should now look like the following:

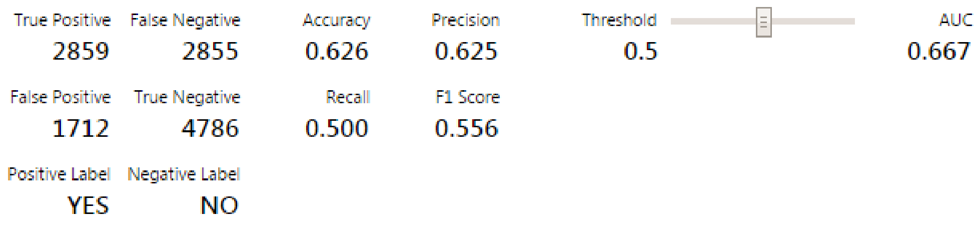


5. Save and run the experiment. When the experiment is finished, visualize the **Evaluation Result** port of the **Evaluate Model** module and review the ROC curve and performance statistics for the model as shown below.

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 which 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.



6. 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.
7. Next, examine the performance statistics provided, as shown here:



True Positive	False Negative	Accuracy	Precision	Threshold	AUC
2859	2855	0.626	0.625	0.5	0.667
False Positive	**True Negative**	**Recall**	**F1 Score**		
1712	4786	0.500	0.556		
Positive Label	**Negative Label**				
YES	NO				

Notice the following:

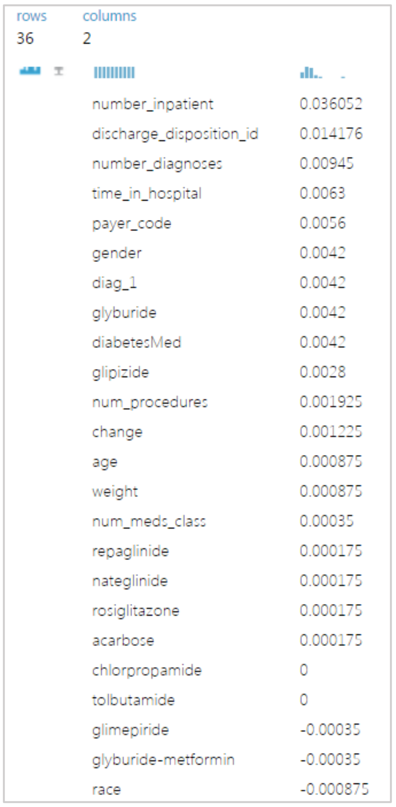
- In the confusion matrix, there are nearly equal numbers of **True Positive** and **False Negative** scores.
- The equal numbers of **True Positive** and **False Negative** scores gives rise to a recall value of 0.5.
- Overall **Accuracy** is 0.626, indicating the scores are correct more often than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

8. Close the evaluation 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:



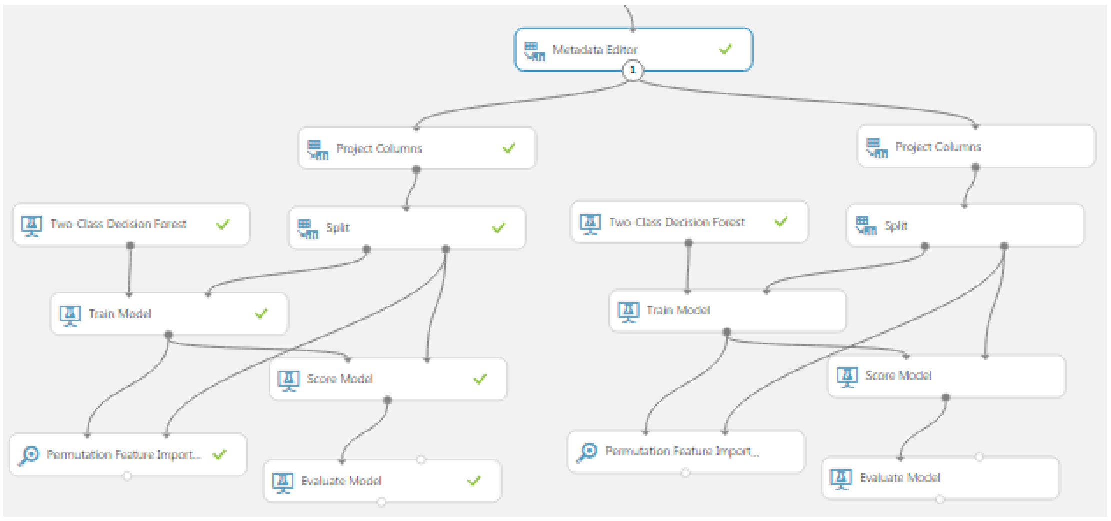
rows 36, columns 2

number_inpatient	0.036052
discharge_disposition_id	0.014176
number_diagnoses	0.00945
time_in_hospital	0.0063
payer_code	0.0056
gender	0.0042
diag_1	0.0042
glyburide	0.0042
diabetesMed	0.0042
glipizide	0.0028
num_procedures	0.001925
change	0.001225
age	0.000875
weight	0.000875
num_meds_class	0.00035
repaglinide	0.000175
nateglinide	0.000175
rosiglitazone	0.000175
acarbose	0.000175
chlorpropamide	0
tolbutamide	0
glimepiride	-0.00035
glyburide-metformin	-0.00035
race	-0.000875

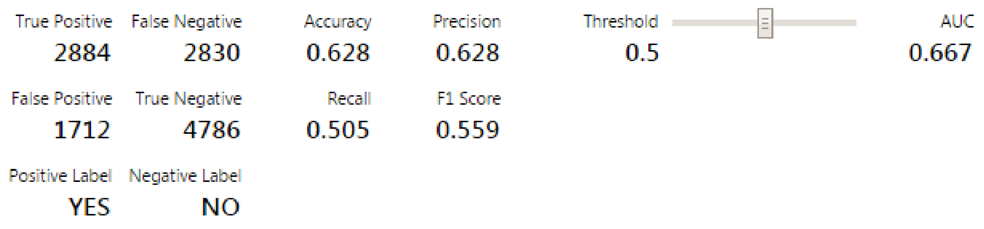
Notice that several of these features have a 0 or very nearly zero importance.

2. Copy all of the modules in the experiment from the **Project Columns** module onwards. Paste these modules onto the experiment canvas.
3. Connect the output of the **Metadata Editor** module to the input port of the new **Project Columns** module.
4. Select the new **Project Columns** 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-metformine.

5. Verify that your experiment from the Metadata Editor down resembles the following figure:



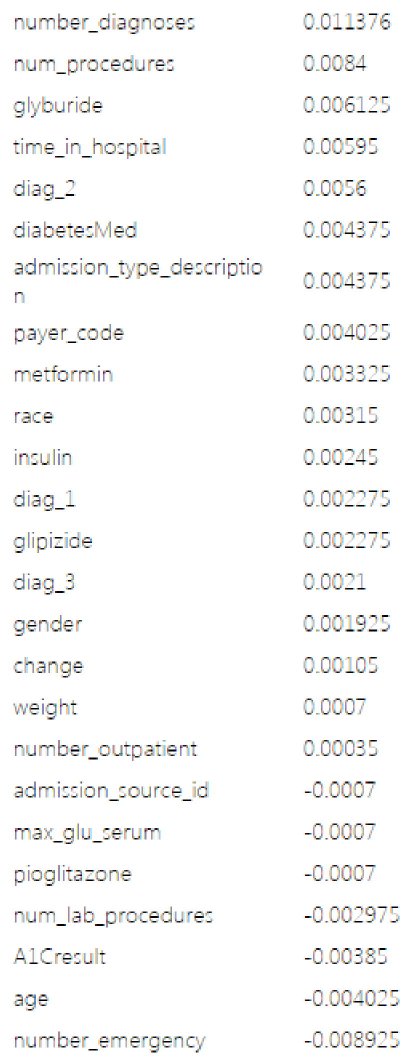
6. Save and run the experiment. When the experiment is finished, visualize the output of the new **Evaluate Model** module.
7. Examine the summary statistics as shown below:



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 list below:



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.

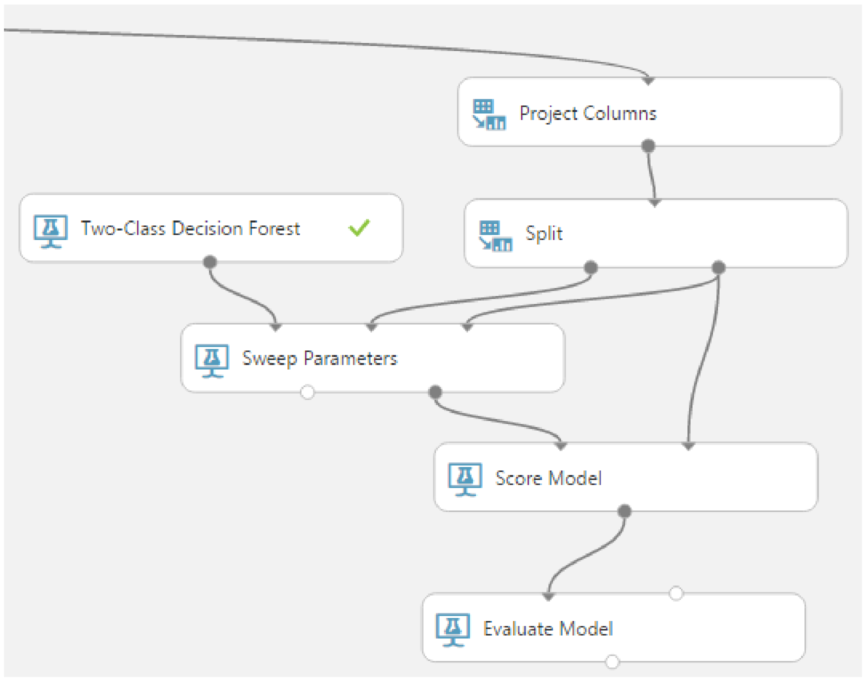
Sweep Model Parameters

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

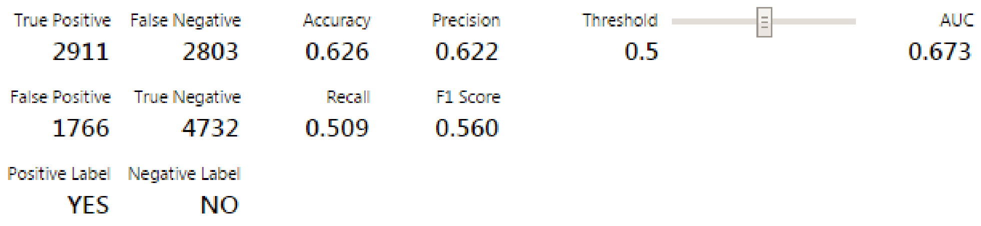
1. Remove the second (newest) **Train Model** and **Permutation Feature Importance** modules from the experiment.
2. Search for the **Sweep Parameters** module. Drag this module onto the canvas in place of the **Train Model** module you removed, and reconnect the experiment modules as follows:
    a.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

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

3. Click the **Sweep Parameters** 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)
9. Verify that the new portion of you experiment resembles the following figure:



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



True Positive	False Negative	Accuracy	Precision	Threshold	AUC
2911	2803	0.626	0.622	0.5	0.673
False Positive	True Negative	Recall	F1 Score		
1766	4732	0.509	0.560		
Positive Label	Negative Label				
YES	NO				

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

Cross Validate the Model

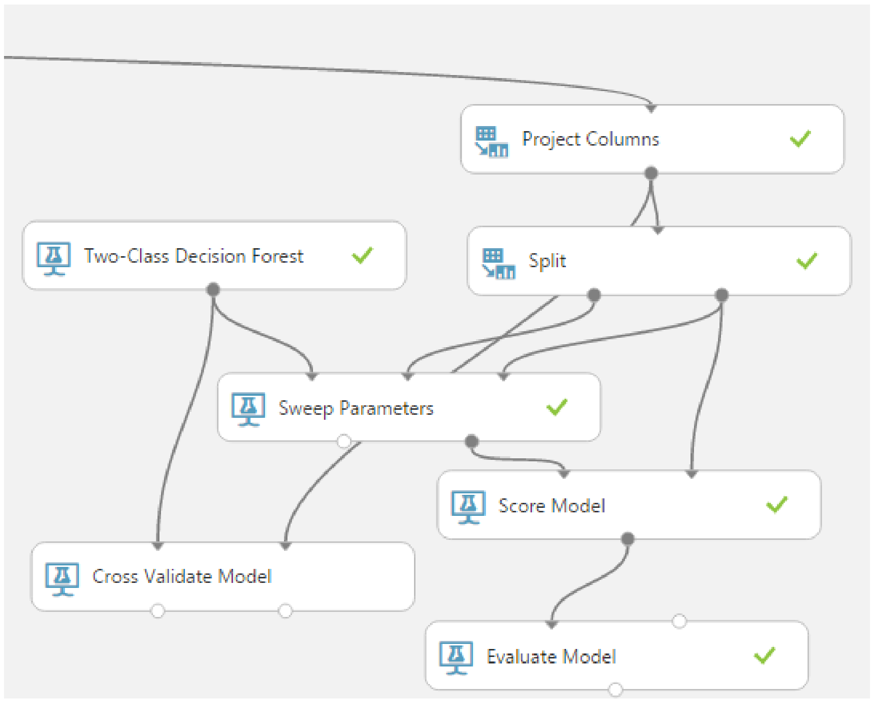
You will cross validate your model by following these steps:

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**

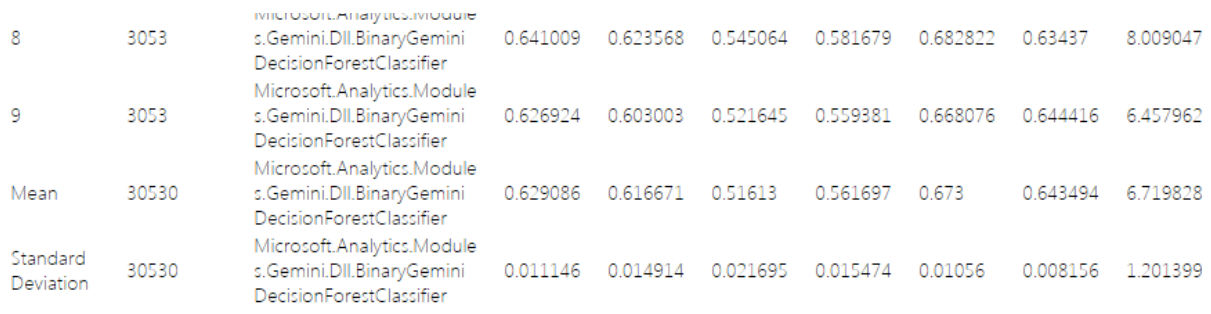
input port of the **Cross Validate Model** module. Connect the **Results dataset** output port of the **Project Columns** module to the **Dataset input** port of the **Cross Validate Model** module.

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:



12. Save and run the experiment.
13. When the experiment has run click on 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, passed the results of the 10 folds of the cross validation and examine the **Mean** value row. These results look like the following:



8	3053	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.641009	0.623568	0.545064	0.581679	0.682822	0.63437	8.009047
9	3053	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.626924	0.603003	0.521645	0.559381	0.668076	0.644416	6.457962
Mean	30530	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.629086	0.616671	0.51613	0.561697	0.673	0.643494	6.719828
Standard Deviation	30530	Microsoft.Analytics.Modules.Gemini.Dll.BinaryGeminiDecisionForestClassifier	0.011146	0.014914	0.021695	0.015474	0.01056	0.008156	1.201399

   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.

Visualizing Errors

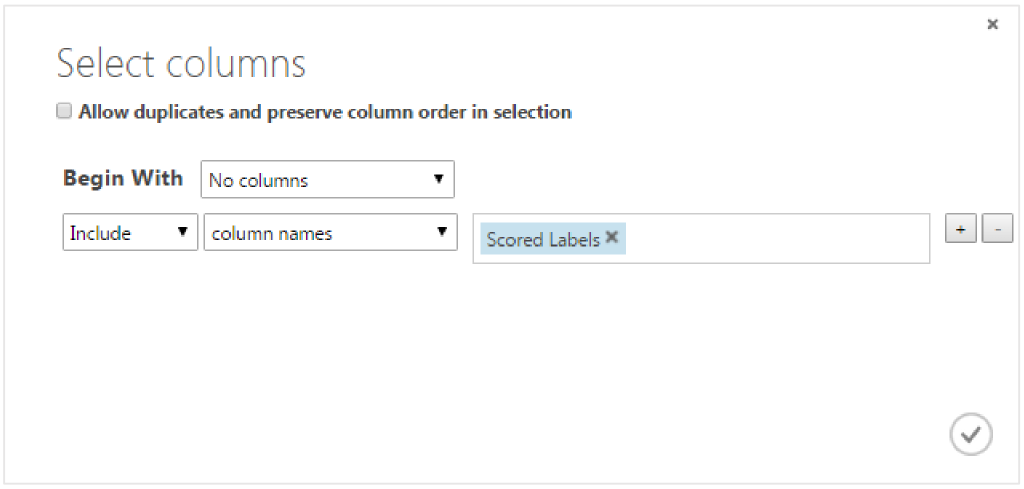
In this exercise you will evaluate the model errors using custom code.

Visualize Errors with R

Note: If you prefer to work with Python, skip this procedure and complete the following procedure, *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 R code.

1. Search for the **Metadata Editor** 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 **Metadata Editor** module.
2. Click the **Metadata Editor** model. From the properties pane click on **Launch Column Selector**. Choose the **Scored Labels** column as shown in the figure below.



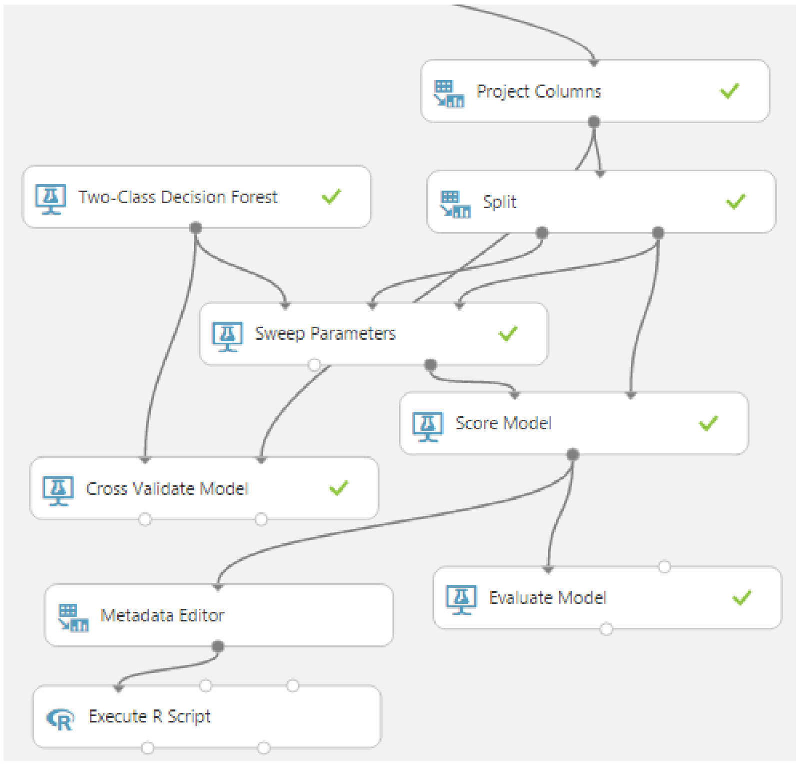
Select columns

Allow duplicates and preserve column order in selection

Begin With: No columns

Include | column names | Scored Labels

3. In the **New column names** box type **ScoredLabels**, with no space. The output from this **Metadata Editor** model will now have a column name with no spaces, compatible with R data frame column names.
4. Search for the **Execute R Script** module. Drag this module onto your canvas. Connect the **Results Dataset** output of the **Metadata Editor** module you just added to the **Dataset1** (left hand) input of the **Execute R Script** module. Your experiment should resemble the figure below:



5. With the **Execute R Script** module selected, in the properties pane, replace the existing R script with the following code. You can copy this code from **DiabetesEval.R** in the folder where you extracted the lab files:

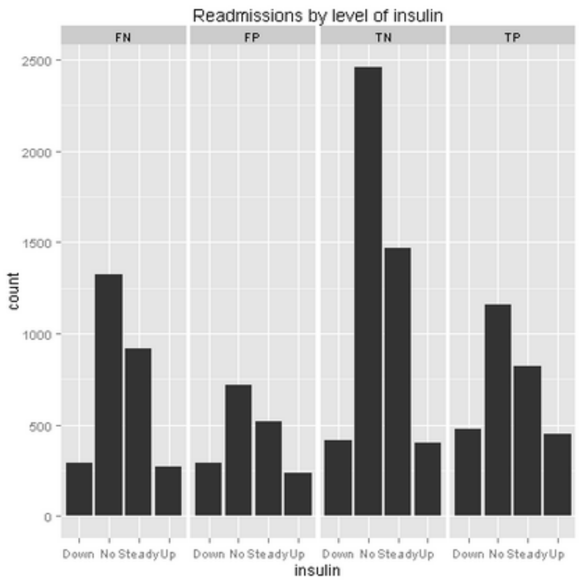
```
frame1 <- maml.mapInputPort(1)

## Assign scores to the cases.
frame1$Score = ifelse(frame1$readmi_class == 'YES' &
frame1$ScoredLabels == 'YES', 'TP',
          ifelse(frame1$readmi_class == 'NO' &
frame1$ScoredLabels == "YES", 'FP',
          ifelse(frame1$readmi_class == 'NO' &
frame1$ScoredLabels == 'NO', 'TN', 'FN')))

## Compair outcomes for various levels of
## factor (categorical) features.
library(ggplot2)
library(dplyr)
bar.plot <- function(x){
  if(is.factor(frame1[, x])){
    sums <- summary(frame1[, x], counts = n())
    msk <- names(sums[which(sums > 100)])
    tmp <- frame1[frame1[, x] %in% msk, c('Score', x)]
    if(strsplit(x, '[-]')[[1]][1] == x){
      g <- ggplot(tmp, aes_string(x)) +
        geom_bar() +
        facet_grid(. ~ Score) +
        ggtitle(paste('Readmissions by level of', x))
      print(g)
    }
  }
}
cols <- names(frame1)
cols <- cols[1:(length(cols) - 1)]
lapply(cols, bar.plot)

## Box plot the numeric features
box.plot <- function(x){
  if(is.numeric(frame1[, x])){
    ggplot(frame1, aes_string('Score', x)) +
      geom_boxplot(alpha = 0.1) +
      ggtitle(paste('Readmissions by', x))
  }
}
dropCols <- ncol(frame1)
dropCols <- c((dropCols - 2):dropCols)
lapply(names(frame1[, -c(dropCols)]), box.plot)
```

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.

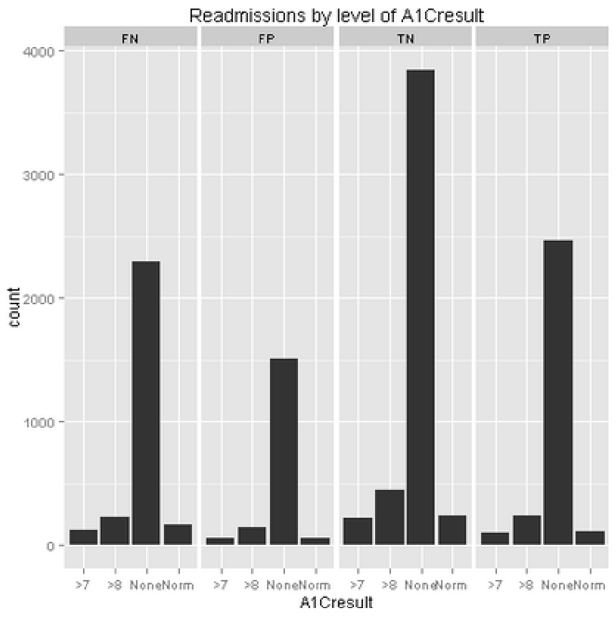
Note: By visualizing scored cases you can determine which features may have unexploited information which could be used to improve the model. In this case, minimizing false negative (**FN**) scores is a priority. **FN** scores result in a patient being readmitted to a hospital for their diabetic condition. Looking for features where there is a difference between the cases scored as **FN** vs. 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 **R Device port** of the **Execute R Script** module.
7. Examine the bar plots, noting the differences between the **TP** and **FN** scores. Note the plot of **Readmission by level of insulin**.



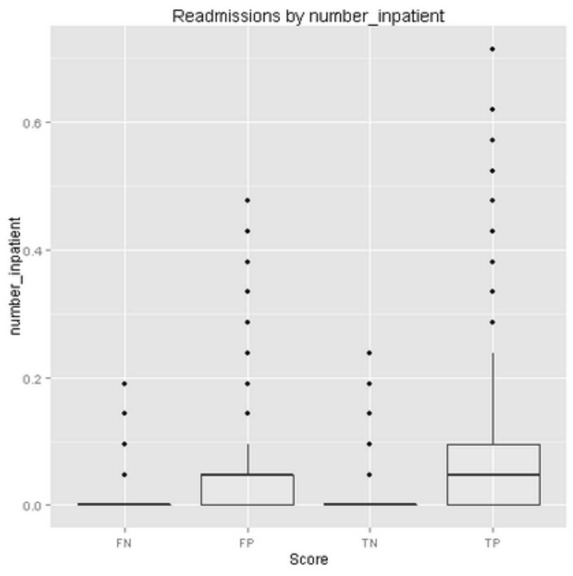
Examine this plot and notice the difference between the **TP** and **FN** scores. The **Down** and **Up** categories are proportionately less likely in the **FN** case. This information might be more fully exploited to improve model performance.

8. Also, note the differences in the plot of Readmission by level of **A1Cresult**.



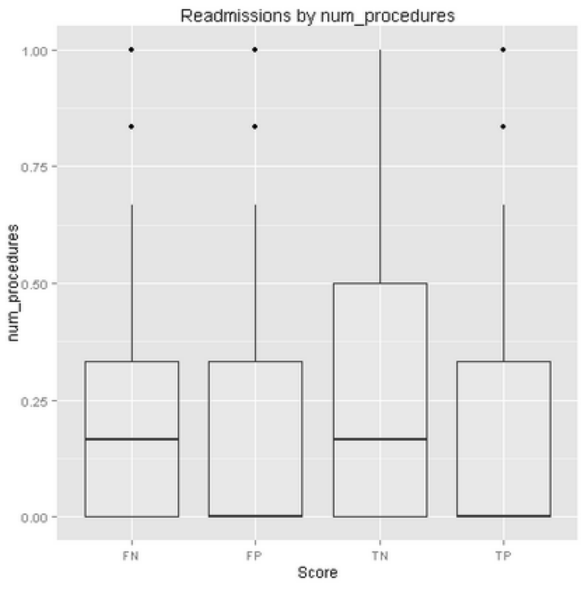
The proportion of the categories for the **TP** and **FN** scores are nearly identical. It is unlikely there is any additional information in this feature which can be exploited.

9. Next examine the box plots, noting the differences between the **TP** and **FN** scores. Note the plot of **Readmission by 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 which can be used to improve the model.

10. Next, examine the box plot of Readmission by **num_procedures** as shown here.



While 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 which can be exploited to improve the model.

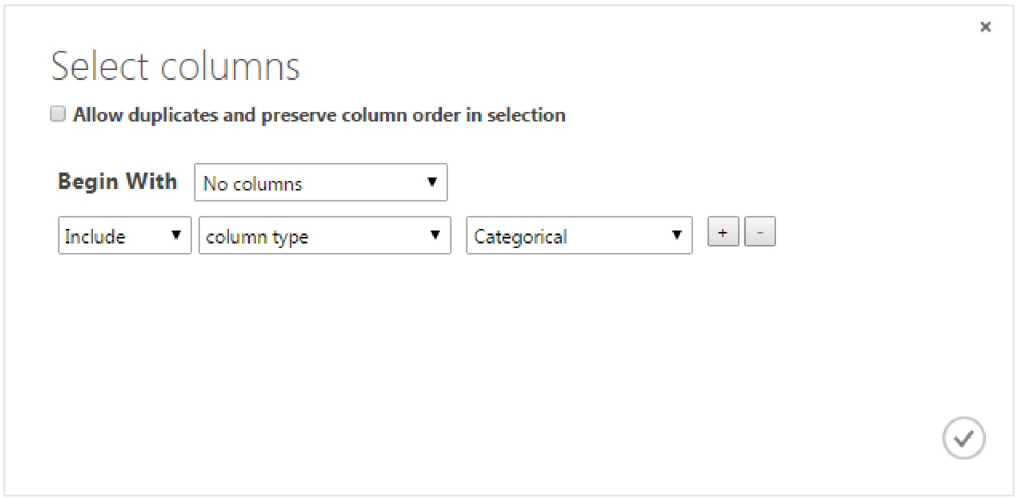
11. Close the R device output.

Visualize Errors with Python

Note: If you prefer to work with R, skip this procedure and complete the preceding procedure, *Visualize Errors with R*.

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 **Metadata Editor** 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 **Metadata Editor** module.
2. Click the **Metadata Editor** model. In the Properties pane click **Launch Column Selector**. Choose the all columns of **Categorical** type as shown in the figure below.



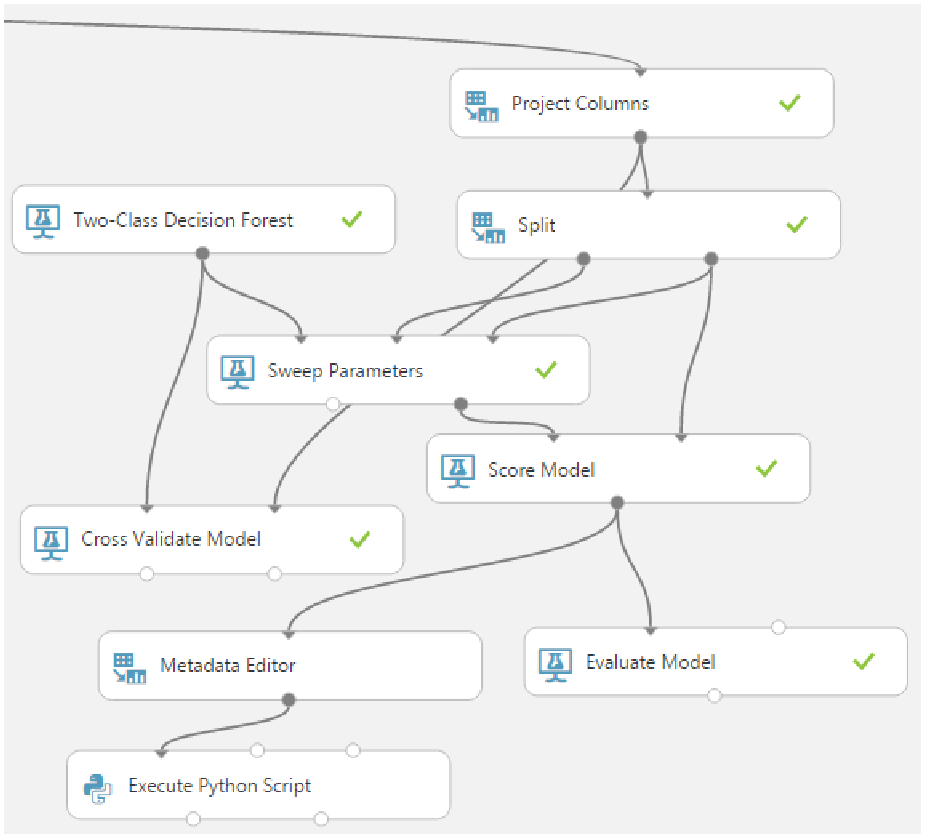
Select columns

Allow duplicates and preserve column order in selection

Begin With: No columns

Include | column type | Categorical

3. In the **Categorical** list, select **Make non-categorical**. The output from this **Metadata Editor** 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 **Metadata Editor** module to the **Dataset1** (left hand) input of the **Execute Python Script** module. Your experiment should resemble the figure below:



5. With the **Execute Python Script** module selected, in the properties pane, replace the existing Python script with the following code. You can copy this code from **DiabetesEval.py** in the folder where you extracted the lab files:

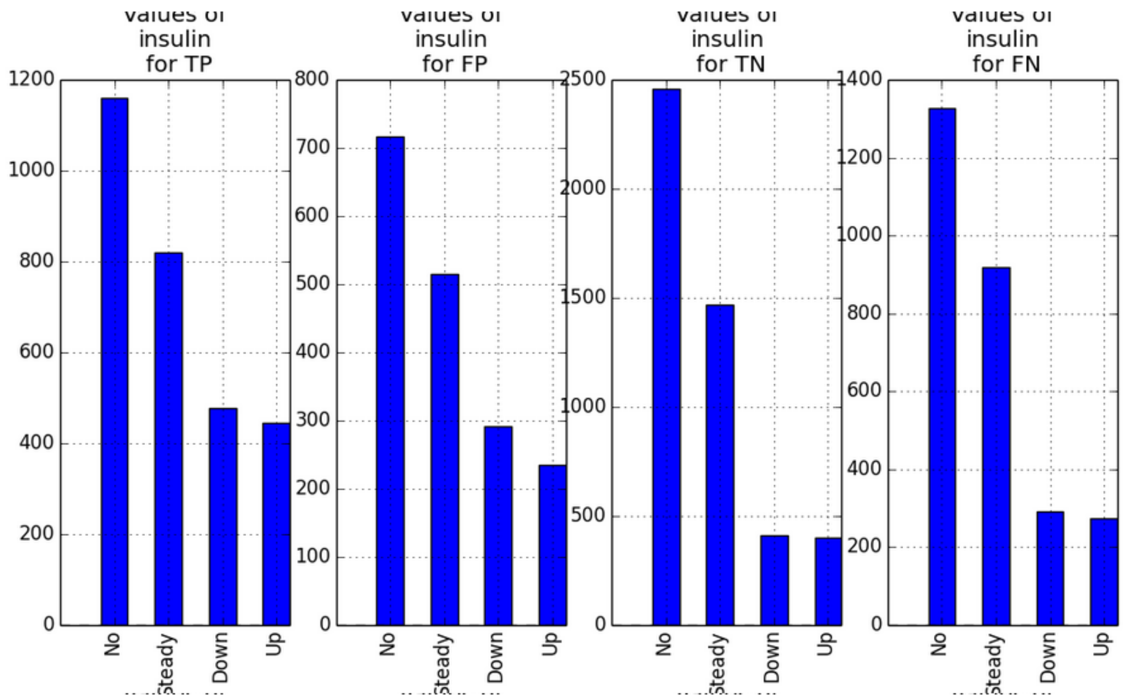
```
def azureml_main(frame1):
    import matplotlib
    matplotlib.use('agg')   # Set backend

    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import statsmodels.graphics.boxplots as sm

    Azure = True

## Compute a column with the score accruacy for each row.
    num_rows = frame1.shape[0]
    frame1['Score'] = pd.DataFrame({'Score': ['II'] * 
num_rows}).astype(str)
    for indx in range(num_rows):
        if((frame1.ix[indx, 'readmi_class'] == 'YES') & 
(frame1.ix[indx, 'Scored Labels'] == 'YES')): \
          frame1.ix[indx, 'Score'] = 'TP'
        elif((frame1.ix[indx, 'readmi_class'] == 'NO') & 
(frame1.ix[indx, 'Scored Labels'] == 'NO')): \
          frame1.ix[indx, 'Score'] = 'TN'
        elif((frame1.ix[indx, 'readmi_class'] == 'YES') &
(frame1.ix[indx, 'Scored Labels'] == 'NO')): \
          frame1.ix[indx, 'Score'] = 'FN'
        else: frame1.ix[indx, 'Score'] = 'FP'

## Create a series of bar plots for the various levels of the
## string columns in the data frame by readmi_class.
    names = list(frame1)
    num_cols = frame1.shape[1]
    err_list = ['TP', 'FP', 'TN', 'FN']
    for indx in range(num_cols):
        if(frame1.ix[:, indx].dtype not in [np.int64, np.int32,
np.float64]):
            fig = plt.figure(figsize = (12,6))
            fig.clf()
            i = 1
            for err in err_list:
                temp = frame1.ix[frame1.Score == err,
indx].value_counts()
                ax = fig.add_subplot(1, 4, i)
                temp.plot(kind = 'bar', ax = ax)
                ax.set_title('Values of \n' + names[indx] + '\n
for ' + err)
                i += 1

            if(Azure == True): fig.savefig('bar_' + names[indx] +
'.png')

## Now make some box plots of the columns with numerical values.
    for indx in range(num_cols):
        if(frame1.ix[:, indx].dtype in [np.int64, np.int32,
np.float64]):
            fig = plt.figure(figsize = (12,6))
            fig.clf()
            i = 1
            for err in err_list:
                temp = frame1.ix[frame1.Score == err, indx]
                ax = fig.add_subplot(1, 4, i)
                ax.boxplot(temp.as_matrix())
                ax.set_title('Values of \n' + names[indx] + '\n
for ' + err)
                i += 1

            if(Azure == True): fig.savefig('box_' + names[indx] +
'.png')

    return frame1
```

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.

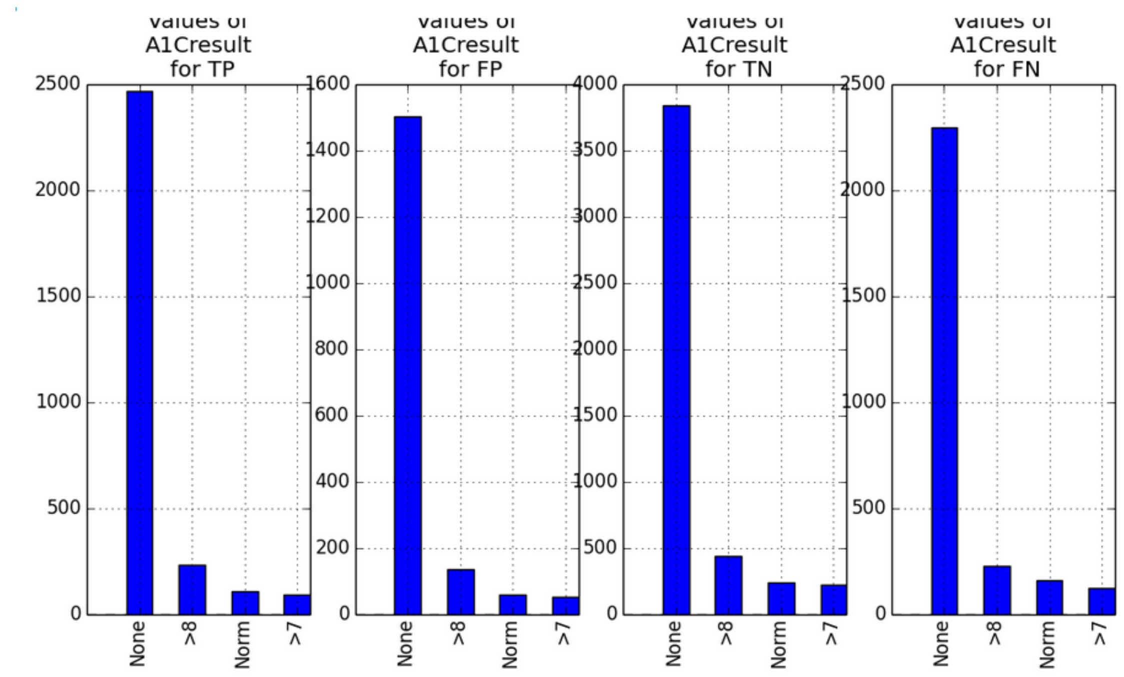
Note: By visualizing scored cases you can determine which features may have unexploited information which could be used to improve the model. In this case, minimizing false negative (**FN**) scores is a priority. **FN** scores result in a patient being readmitted to a hospital for their diabetic condition.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



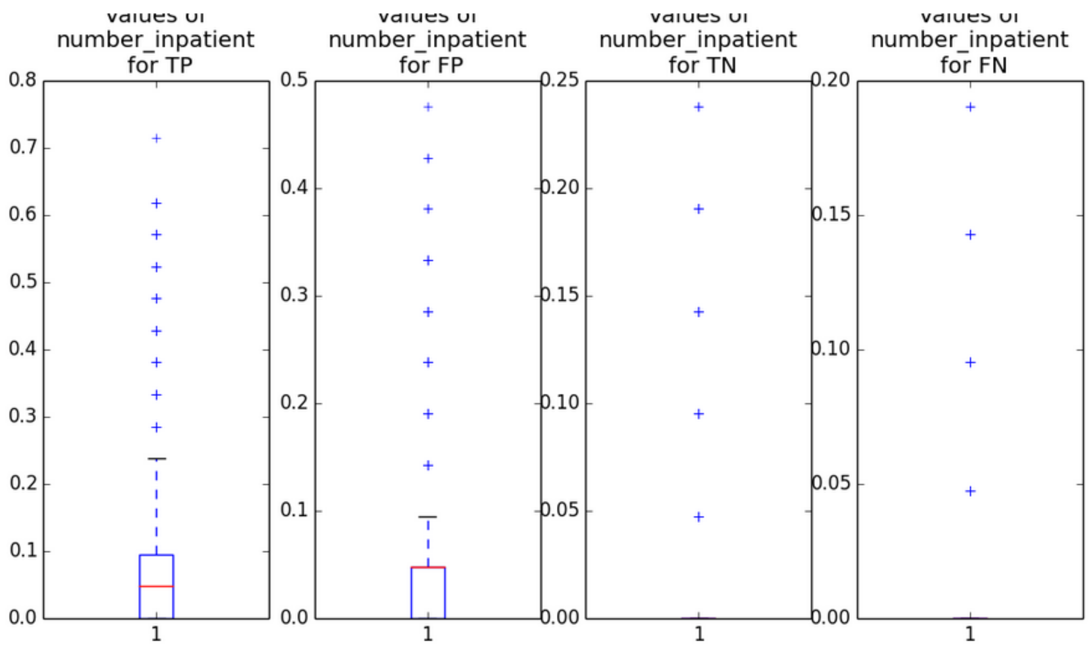
Examine this plot and notice the difference between the **TP** and **FN** scores. The **Down** and **Up** categories are proportionately less likely in the **FN** case. This information might be more fully exploited to improve model performance.

8. Also, note the differences in the plot for **A1Cresult**..



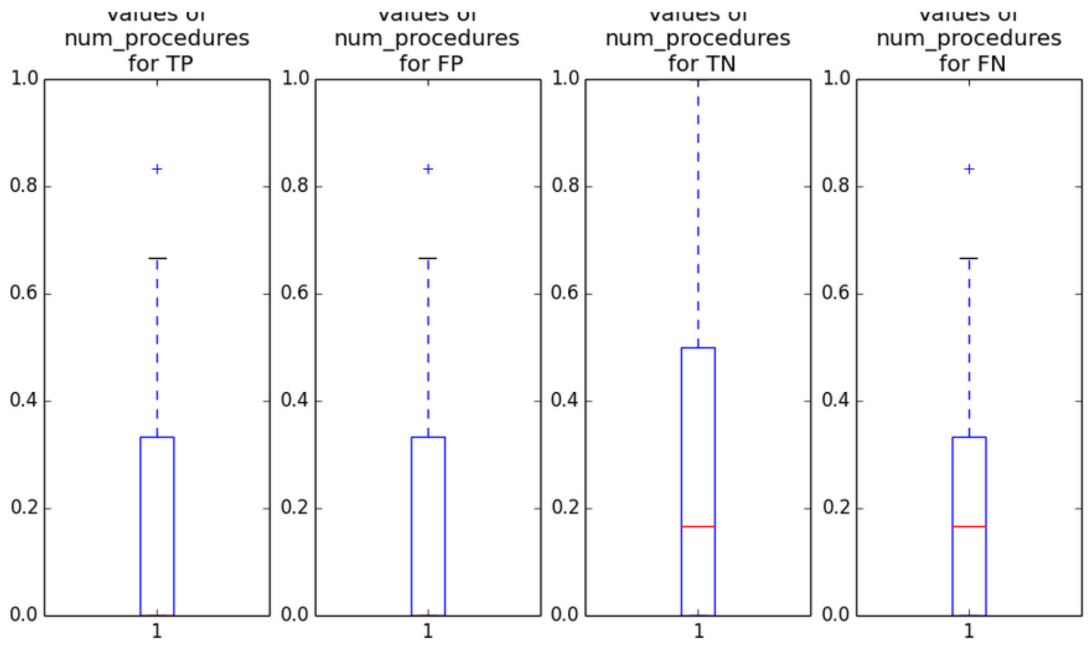
The proportion of the categories for the **TP** and **FN** scores are nearly identical. It is unlikely there is any additional information in this feature which can be exploited.

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 which can be used to improve the model.

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



While 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 which can be exploited to improve the model.

11. Close the Python device output.

Summary

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

- Visualization of the data set 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.

Note: The experiment created in this lab is available in the Cortana Analytics library at http://gallery.cortanaanalytics.com/Collection/5bfa7c8023724a29a41a4098d3fc3df9.