# Machine Learning Studio Lab

# A Predictive Analytics Solution for Credit Risk Assessment

In this walkthrough, we take an extended look at the process of developing a predictive analytics solution in Machine Learning Studio. We develop a simple model in Machine Learning Studio, and then deploy it as an Azure Machine Learning web service where the model can make predictions using new data.

If you're new to the field of machine learning in general, there's a video series that might be helpful to you. It's called [Data Science for Beginners](https://docs.microsoft.com/en-us/azure/machine-learning/studio/data-science-for-beginners-the-5-questions-data-science-answers) and it can give you a great introduction to machine learning using everyday language and concepts.

## The problem

Suppose you need to predict an individual's credit risk based on the information they gave on a credit application.

Credit risk assessment is a complex problem, but we can simplify it a bit for this walkthrough. We'll use it as an example of how you can create a predictive analytics solution using Microsoft Azure Machine Learning. To do this, we use Azure Machine Learning Studio and a Machine Learning web service.

## The solution

In this detailed walkthrough, we start with publicly available credit risk data and develop and train a predictive model based on that data. Then we deploy the model as a web service so it can be used by others for credit risk assessment.

To create this credit risk assessment solution, we follow these steps:

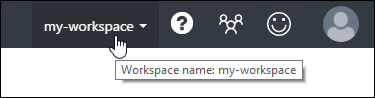
1. Create a Machine Learning workspace
2. Upload existing data
3. Create an experiment
4. Train and evaluate the models
5. Deploy the web service
6. Access the web service

### Step 1: Create a Machine Learning workspace

To use Machine Learning Studio, you need to have a Microsoft Azure Machine Learning workspace. This workspace contains the tools you need to create, manage, and publish experiments.

The administrator for your Azure subscription needs to create the workspace and then add you as an owner or contributor. For details, see Create and share an Azure Machine Learning workspace.

After your workspace is created, open Machine Learning Studio (https://studio.azureml.net/Home). If you have more than one workspace, you can select the workspace in the toolbar in the upper-right corner of the window.



|  |
| --- |
| Tip |
| If you were made an owner of the workspace, you can share the experiments you're working on by inviting others to the workspace. You can do this in Machine Learning Studio on the SETTINGS page. You just need the Microsoft account or organizational account for each user. |
|  |
| On the SETTINGS page, click USERS, then click INVITE MORE USERS at the bottom of the window. |

### Step 2: Upload existing data into an Azure Machine Learning experiment

We'll use the file named german.data.

The **german.data** dataset contains rows of 20 variables for 1000 past applicants for credit. These 20 variables represent the dataset's set of features (the feature vector), which provides identifying characteristics for each credit applicant. An additional column in each row represents the applicant's calculated credit risk, with 700 applicants identified as a low credit risk and 300 as a high risk.

The UCI website provides a description of the attributes of the feature vector for this data. This includes financial information, credit history, employment status, and personal information. For each applicant, a binary rating has been given indicating whether they are a low or high credit risk.

We'll use this data to train a predictive analytics model. When we're done, our model should be able to accept a feature vector for a new individual and predict whether he or she is a low or high credit risk.

Here's an interesting twist. The description of the dataset on the UCI website mentions what it costs if we misclassify a person's credit risk. If the model predicts a high credit risk for someone who is actually a low credit risk, the model has made a misclassification. But the reverse misclassification is five times more costly to the financial institution: if the model predicts a low credit risk for someone who is actually a high credit risk.

So, we want to train our model so that the cost of this latter type of misclassification is five times higher than misclassifying the other way. One simple way to do this when training the model in our experiment is by duplicating (five times) those entries that represent someone with a high credit risk. Then, if the model misclassifies someone as a low credit risk when they're actually a high risk, the model does that same misclassification five times, once for each duplicate. This will increase the cost of this error in the training results.

#### Convert the dataset format

The original dataset uses a blank-separated format. Machine Learning Studio works better with a comma-separated value (CSV) file, so we'll convert the dataset by replacing spaces with commas.

There are many ways to convert this data. One way is by using the following Windows PowerShell command:

*cat german.data | %{$\_ -replace " ",","} | sc german.csv*

Another way is by using the Unix sed command:

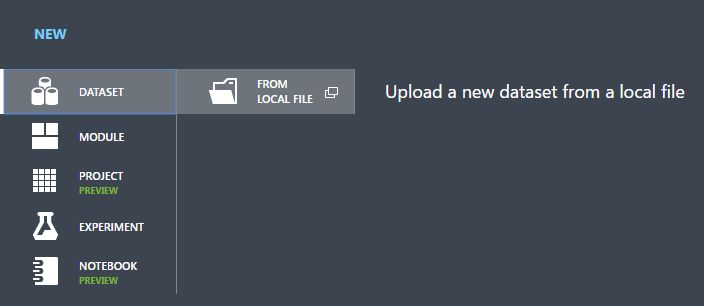
*sed 's/ /,/g' german.data > german.csv*

In either case, we have created a comma-separated version of the data in a file named german.csv that we can use in our experiment.

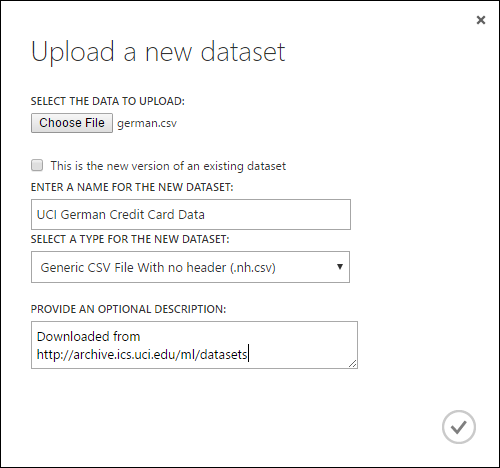
#### Upload the dataset to Machine Learning Studio

Once the data has been converted to CSV format, we need to upload it into Machine Learning Studio.

1. Open the Machine Learning Studio home page (https://studio.azureml.net).
2. Click the menu in the upper-left corner of the window, click **Azure Machine Learning**, select **Studio**, and sign in.
3. Click **+NEW** at the bottom of the window.
4. Select **DATASET**.
5. Select **FROM LOCAL FILE**.

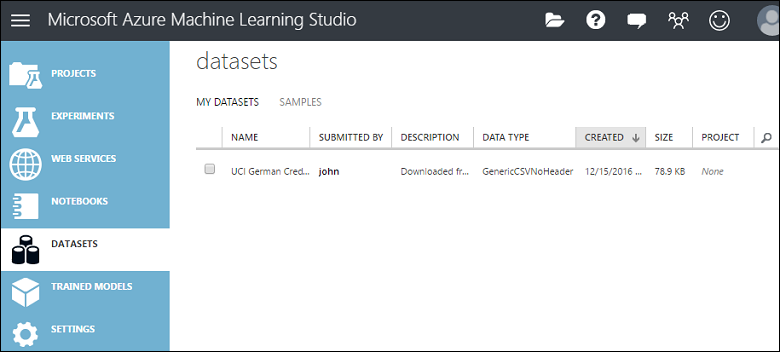


1. In the **Upload a new dataset** dialog, click **Browse** and find the **german.csv** file you created.
2. Enter a name for the dataset. For this walkthrough, call it "UCI German Credit Card Data".
3. For data type, select **Generic CSV File With no header (.nh.csv)**.
4. Add a description if you’d like.
5. Click the **OK** check mark.



This uploads the data into a dataset module that we can use in an experiment.

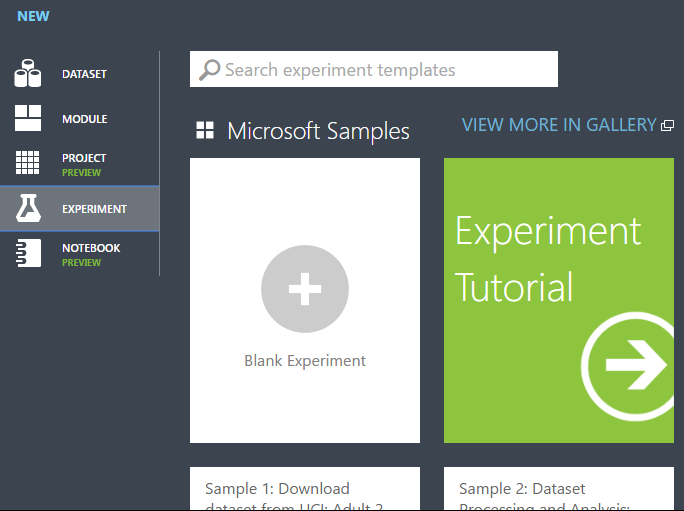
You can manage datasets that you've uploaded to Studio by clicking the **DATASETS** tab to the left of the Studio window.



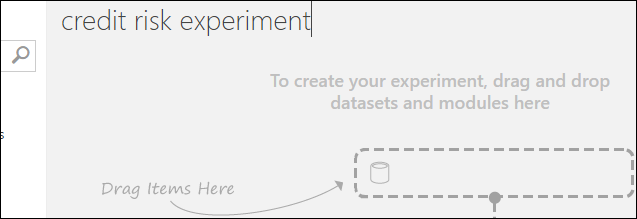
### Step 3: Create a new Azure Machine Learning experiment

The next step in this walkthrough is to create an experiment in Machine Learning Studio that uses the dataset we uploaded.

1. In Studio, click **+NEW** at the bottom of the window.
2. Select **EXPERIMENT**, and then select "Blank Experiment".

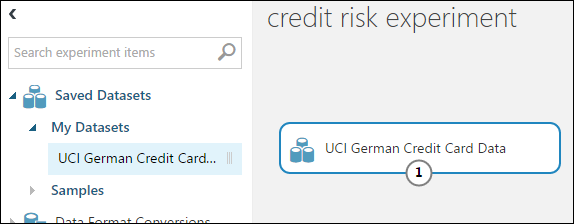


1. Select the default experiment name at the top of the canvas and rename it to something meaningful.



|  |
| --- |
| Tip |
| It's a good practice to fill in Summary and Description for the experiment in the Properties pane. These properties give you the chance to document the experiment so that anyone who looks at it later will understand your goals and methodology. |

1. In the module palette to the left of the experiment canvas, expand **Saved Datasets**.
2. Find the dataset you created under **My Datasets** and drag it onto the canvas. You can also find the dataset by entering the name in the **Search** box above the palette.



#### Prepare the data

You can view the first 100 rows of the data and some statistical information for the whole dataset: Click the output port of the dataset (the small circle at the bottom) and select **Visualize**.

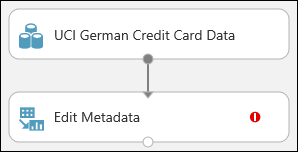
Because the data file didn't come with column headings, Studio has provided generic headings (Col1, Col2, etc.). Good headings aren't essential to creating a model, but they make it easier to work with the data in the experiment. Also, when we eventually publish this model in a web service, the headings help identify the columns to the user of the service.

We can add column headings using the Edit Metadata module. You use the Edit Metadata module to change metadata associated with a dataset. In this case, we use it to provide more friendly names for column headings.

To use Edit Metadata, you first specify which columns to modify (in this case, all of them.) Next, you specify the action to be performed on those columns (in this case, changing column headings.)

1. In the module palette, type "metadata" in the **Search** box. The Edit Metadata appears in the module list.
2. Click and drag the Edit Metadata module onto the canvas and drop it below the dataset we added earlier.
3. Connect the dataset to the Edit Metadata: click the output port of the dataset (the small circle at the bottom of the dataset), drag to the input port of Edit Metadata (the small circle at the top of the module), then release the mouse button. The dataset and module remain connected even if you move either around on the canvas.

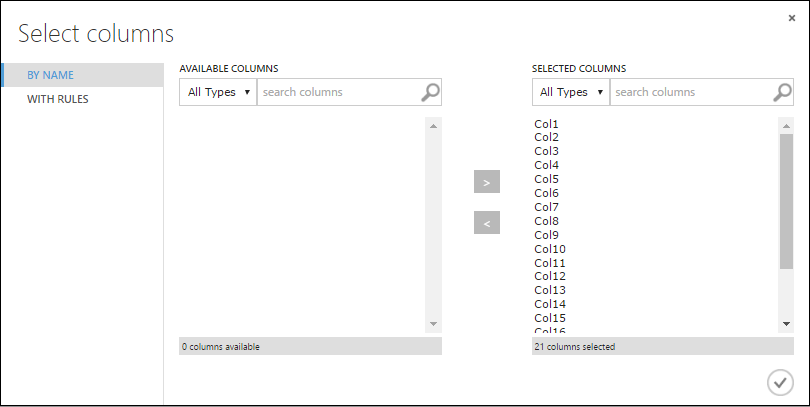
The experiment should now look something like this:



The red exclamation mark indicates that we haven't set the properties for this module yet. We'll do that next.

|  |
| --- |
| Tip |
| You can add a comment to a module by double-clicking the module and entering text. This can help you see at a glance what the module is doing in your experiment. In this case, double-click the Edit Metadata module and type the comment "Add column headings". Click anywhere else on the canvas to close the text box. To display the comment, click the down-arrow on the module. |

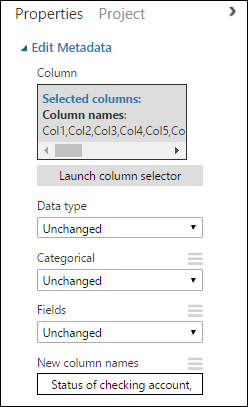
1. Select Edit Metadata, and in the **Properties** pane to the right of the canvas, click **Launch column selector**.
2. In the **Select columns** dialog, select all the rows in **Available Columns** and click > to move them to **Selected Columns**. The dialog should look like this:



1. Click the **OK** check mark.
2. Back in the **Properties** pane, look for the **New column names** parameter. In this field, enter a list of names for the 21 columns in the dataset, separated by commas and in column order. You can obtain the columns names from the dataset documentation on the UCI website, or for convenience you can copy and paste the following list:

*Status of checking account, Duration in months, Credit history, Purpose, Credit amount, Savings account/bond, Present employment since, Installment rate in percentage of disposable income, Personal status and sex, Other debtors, Present residence since, Property, Age in years, Other installment plans, Housing, Number of existing credits, Job, Number of people providing maintenance for, Telephone, Foreign worker, Credit risk*

The Properties pane looks like this:



|  |
| --- |
| Tip |
| If you want to verify the column headings, run the experiment (click RUN below the experiment canvas). When it finishes running (a green check mark appears on Edit Metadata), click the output port of the Edit Metadata module, and select Visualize. You can view the output of any module in the same way to view the progress of the data through the experiment. |

#### Create training and test datasets

We need some data to train the model and some to test it. So, in the next step of the experiment, we split the dataset into two separate datasets: one for training our model and one for testing it.

To do this, we use the Split Data module.

1. Find the Split Data module, drag it onto the canvas, and connect it to the Edit Metadata module.
2. By default, the split ratio is 0.5 and the **Randomized split** parameter is set. This means that a random half of the data is output through one port of the Split Data module, and half through the other. You can adjust these parameters, as well as the Random seed parameter, to change the split between training and testing data. For this example, we leave them as-is.

|  |
| --- |
| Tip |
| The property Fraction of rows in the first output dataset determines how much of the data is output through the left output port. For instance, if you set the ratio to 0.7, then 70% of the data is output through the left port and 30% through the right port. |

1. Double-click the Split Data module and enter the comment, "Training/testing data split 50%".

We can use the outputs of the Split Data module however we like, but let's choose to use the left output as training data and the right output as testing data.

As mentioned in the previous step, the cost of misclassifying a high credit risk as low is five times higher than the cost of misclassifying a low credit risk as high. To account for this, we generate a new dataset that reflects this cost function. In the new dataset, each high-risk example is replicated five times, while each low risk example is not replicated.

We can do this replication using R code:

1. Find and drag the Execute R Script module onto the experiment canvas.
2. Connect the left output port of the Split Data module to the first input port ("Dataset1") of the Execute R Script module.
3. Double-click the Execute R Script module and enter the comment, "Set cost adjustment".
4. In the **Properties** pane, delete the default text in the **R Script** parameter and enter this script:

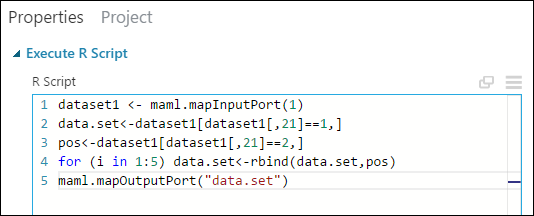
*dataset1 <- maml.mapInputPort(1)*

*data.set<-dataset1[dataset1[,21]==1,]*

*pos<-dataset1[dataset1[,21]==2,]*

*for (i in 1:5) data.set<-rbind(data.set,pos)*

*maml.mapOutputPort("data.set")*

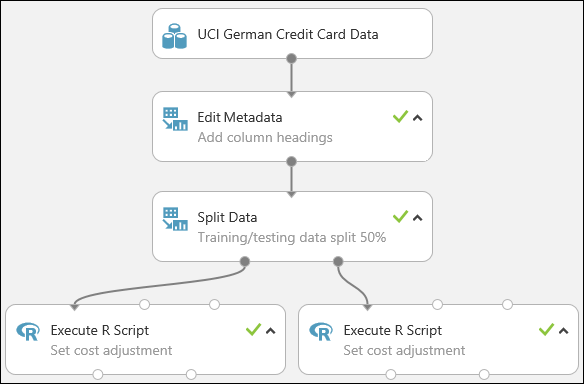


We need to do this same replication operation for each output of the Split Data module so that the training and testing data have the same cost adjustment. The easiest way to do this is by duplicating the Execute R Script module we just made and connecting it to the other output port of the Split Data module.

1. Right-click the Execute R Script module and select Copy.
2. Right-click the experiment canvas and select Paste.
3. Drag the new module into position, and then connect the right output port of the Split Data module to the first input port of this new Execute R Script module.
4. At the bottom of the canvas, click Run.

|  |
| --- |
| Tip |
| The copy of the Execute R Script module contains the same script as the original module. When you copy and paste a module on the canvas, the copy retains all the properties of the original. |

Our experiment now looks something like this:



### Step 4: Train and evaluate the predictive analytic models

One of the benefits of using Azure Machine Learning Studio for creating machine learning models is the ability to try more than one type of model at a time in a single experiment and compare the results. This type of experimentation helps you find the best solution for your problem.

In the experiment we're developing in this walkthrough, we'll create two different types of models and then compare their scoring results to decide which algorithm we want to use in our final experiment.

There are various models we could choose from. To see the models available, expand the **Machine Learning** node in the module palette, and then expand **Initialize Model** and the nodes beneath it. For the purposes of this experiment, we'll select the Two-Class Support Vector Machine (SVM) and the Two-Class Boosted Decision Tree modules.

|  |
| --- |
| Tip |
| To get help deciding which Machine Learning algorithm best suits the particular problem you're trying to solve, see How to choose algorithms for Microsoft Azure Machine Learning. |

#### Train the models

We'll add both the Two-Class Boosted Decision Tree module and Two-Class Support Vector Machine module in this experiment.

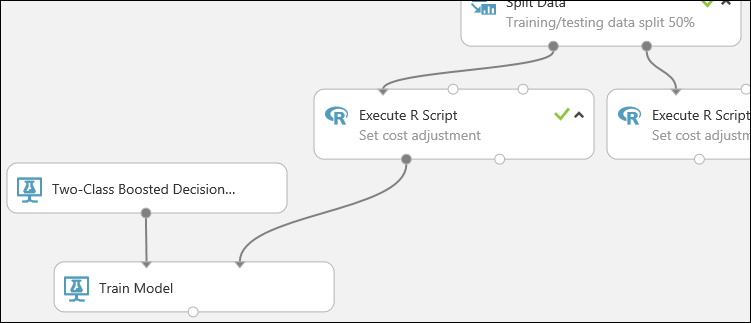
##### Two-Class Boosted Decision Tree

First, let's set up the boosted decision tree model.

1. Find the Two-Class Boosted Decision Tree module in the module palette and drag it onto the canvas.
2. Find the Train Model module, drag it onto the canvas, and then connect the output of the Two-Class Boosted Decision Tree module to the left input port of the Train Model module. The Two-Class Boosted Decision Tree module initializes the generic model, and Train Model uses training data to train the model.
3. Connect the left output of the left Execute R Script module to the right input port of the Train Model module (we decided in Step 3 of this walkthrough to use the data coming from the left side of the Split Data module for training).

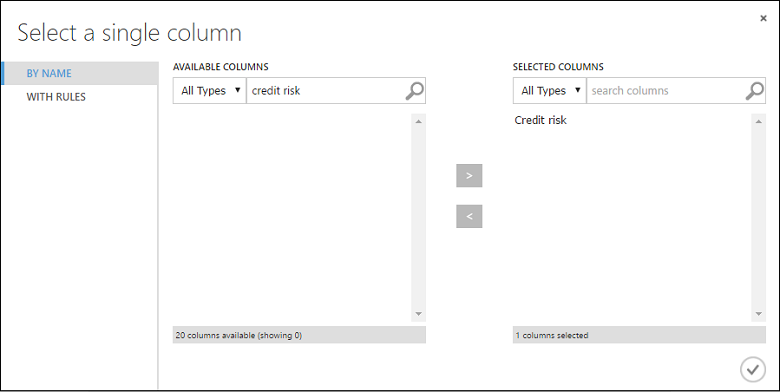
|  |
| --- |
| Tip |
| We don't need two of the inputs and one of the outputs of the Execute R Script module for this experiment, so we can leave them unattached. |

This portion of the experiment now looks something like this:



Now we need to tell the Train Model module that we want the model to predict the Credit Risk value.

1. Select the Train Model module. In the **Properties** pane, click **Launch column selector**.
2. In the **Select a single column** dialog, type "credit risk" in the search field under **Available Columns**, select "Credit risk" below, and click the right arrow button (>) to move "Credit risk" to **Selected** Columns.



1. Click the **OK** check mark.

##### Two-Class Support Vector Machine

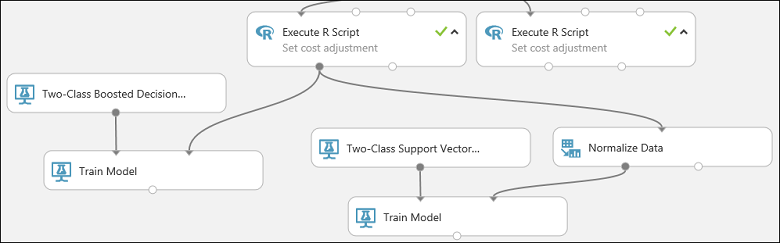
Next, we set up the SVM model.

First, a little explanation about SVM. Boosted decision trees work well with features of any type. However, since the SVM module generates a linear classifier, the model that it generates has the best test error when all numeric features have the same scale. To convert all numeric features to the same scale, we use a "Tanh" transformation (with the Normalize Data module). This transforms our numbers into the [0,1] range. The SVM module converts string features to categorical features and then to binary 0/1 features, so we don't need to manually transform string features. Also, we don't want to transform the Credit Risk column (column 21) - it's numeric, but it's the value we're training the model to predict, so we need to leave it alone.

To set up the SVM model, do the following:

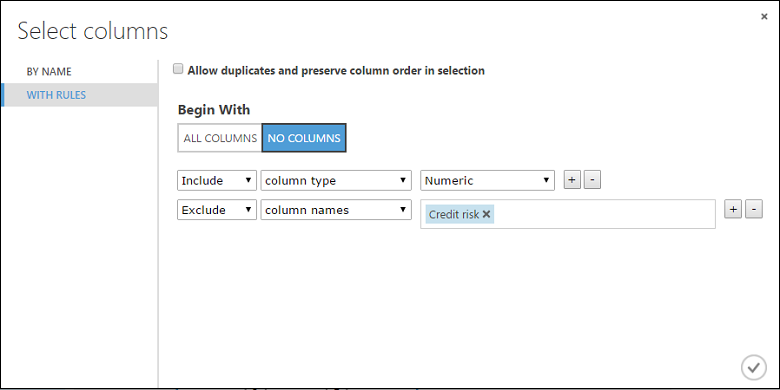
1. Find the Two-Class Support Vector Machine module in the module palette and drag it onto the canvas.
2. Right-click the Train Model module, select Copy, and then right-click the canvas and select Paste. The copy of the Train Model module has the same column selection as the original.
3. Connect the output of the Two-Class Support Vector Machine module to the left input port of the second Train Model module.
4. Find the Normalize Data module and drag it onto the canvas.
5. Connect the left output of the left Execute R Script module to the input of this module (notice that the output port of a module may be connected to more than one other module).
6. Connect the left output port of the Normalize Data module to the right input port of the second Train Model module.

This portion of our experiment should now look something like this:



Now configure the Normalize Data module:

1. Click to select the Normalize Data module. In the **Properties** pane, select **Tanh** for the **Transformation method** parameter.
2. Click **Launch column selector**, select "No columns" for **Begin With**, select **Include** in the first dropdown, select **column type** in the second dropdown, and select **Numeric** in the third dropdown. This specifies that all the numeric columns (and only numeric) are transformed.
3. Click the plus sign (+) to the right of this row - this creates a row of dropdowns. Select **Exclude** in the first dropdown, select **column names** in the second dropdown, and enter "Credit risk" in the text field. This specifies that the Credit Risk column should be ignored (we need to do this because this column is numeric and so would be transformed if we didn't exclude it).
4. Click the **OK** check mark.



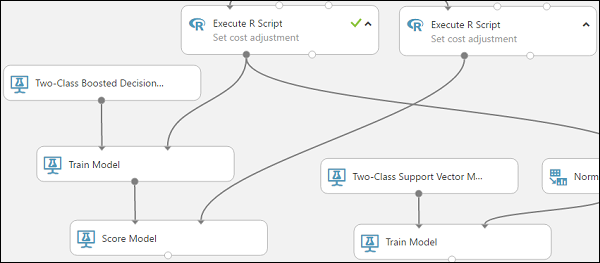
The Normalize Data module is now set to perform a Tanh transformation on all numeric columns except for the Credit Risk column.

##### Score and evaluate the models

We use the testing data that was separated out by the Split Data module to score our trained models. We can then compare the results of the two models to see which generated better results.

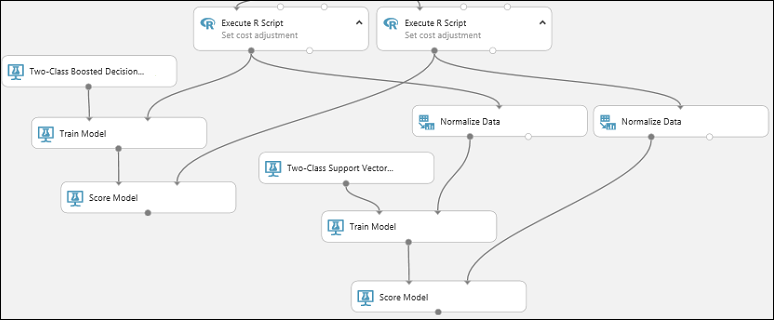
**Add the Score Model modules**

1. Find the Score Model module and drag it onto the canvas.
2. Connect the Train Model module that's connected to the Two-Class Boosted Decision Tree module to the left input port of the Score Model module.
3. Connect the right Execute R Script module (our testing data) to the right input port of the Score Model module.



The Score Model module can now take the credit information from the testing data, run it through the model, and compare the predictions the model generates with the actual credit risk column in the testing data.

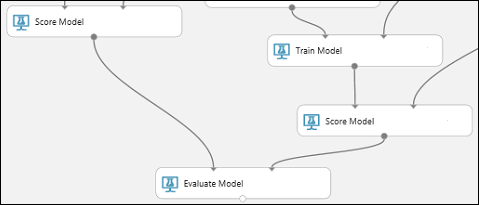
1. Copy and paste the Score Model module to create a second copy.
2. Connect the output of the SVM model (that is, the output port of the Train Model module that's connected to the Two-Class Support Vector Machine module) to the input port of the second Score Model module.
3. For the SVM model, we have to do the same transformation to the test data as we did to the training data. So copy and paste the Normalize Data module to create a second copy and connect it to the right Execute R Script module.
4. Connect the left output of the second Normalize Data module to the right input port of the second Score Model module.



**Add the Evaluate Model module**

To evaluate the two scoring results and compare them, we use an Evaluate Model module.

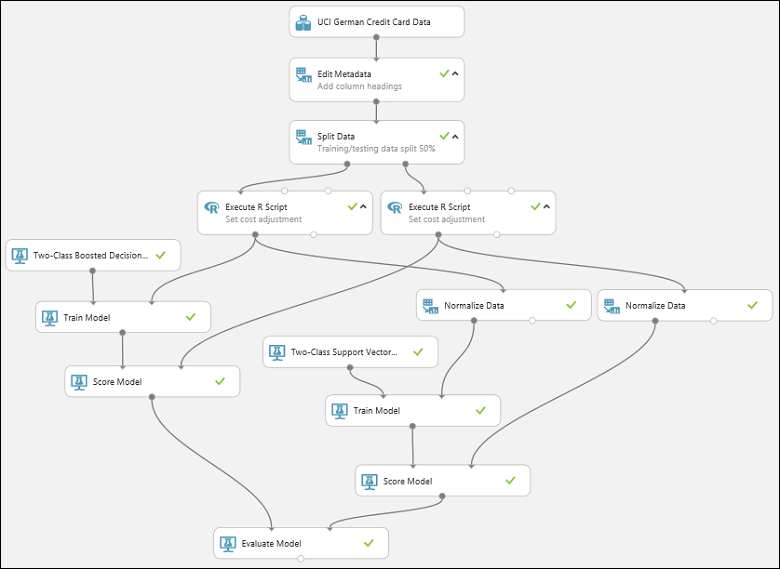
1. Find the Evaluate Model module and drag it onto the canvas.
2. Connect the output port of the Score Model module associated with the boosted decision tree model to the left input port of the Evaluate Model module.
3. Connect the other Score Model module to the right input port.



**Run the experiment and check the results**

To run the experiment, click the RUN button below the canvas. It may take a few minutes. A spinning indicator on each module shows that it's running, and then a green check mark shows when the module is finished. When all the modules have a check mark, the experiment has finished running.

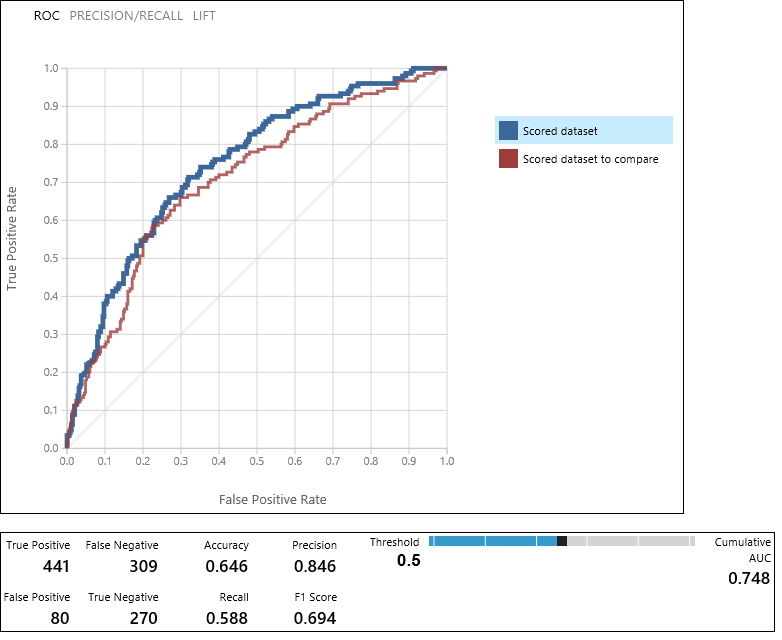
The experiment should now look something like this:



To check the results, click the output port of the Evaluate Model module and select Visualize.

The Evaluate Model module produces a pair of curves and metrics that allow you to compare the results of the two scored models. You can view the results as Receiver Operator Characteristic (ROC) curves, Precision/Recall curves, or Lift curves. Additional data displayed includes a confusion matrix, cumulative values for the area under the curve (AUC), and other metrics. You can change the threshold value by moving the slider left or right and see how it affects the set of metrics.

To the right of the graph, click Scored dataset or Scored dataset to compare to highlight the associated curve and to display the associated metrics below. In the legend for the curves, "Scored dataset" corresponds to the left input port of the Evaluate Model module - in our case, this is the boosted decision tree model. "Scored dataset to compare" corresponds to the right input port - the SVM model in our case. When you click one of these labels, the curve for that model is highlighted and the corresponding metrics are displayed, as shown in the following graphic.



By examining these values, you can decide which model is closest to giving you the results you're looking for. You can go back and iterate on your experiment by changing parameter values in the different models.

|  |
| --- |
| Tip |
| Each time you run the experiment a record of that iteration is kept in the Run History. You can view these iterations, and return to any of them, by clicking VIEW RUN HISTORY below the canvas. You can also click Prior Run in the Properties pane to return to the iteration immediately preceding the one you have open. |
| You can make a copy of any iteration of your experiment by clicking SAVE AS below the canvas. Use the experiment's Summary and Description properties to keep a record of what you've tried in your experiment iterations. |

### Step 5: Deploy the Azure Machine Learning web service

To give others a chance to use the predictive model we've developed in this walkthrough, we can deploy it as a web service on Azure.

Up to this point we've been experimenting with training our model. But the deployed service is no longer going to do training - it's going to generate new predictions by scoring the user's input based on our model. So we're going to do some preparation to convert this experiment from a **training** experiment to a **predictive** experiment.

This is a three-step process:

1. Remove one of the models
2. Convert the training experiment we've created into a predictive experiment
3. Deploy the predictive experiment as a web service

#### Remove one of the models

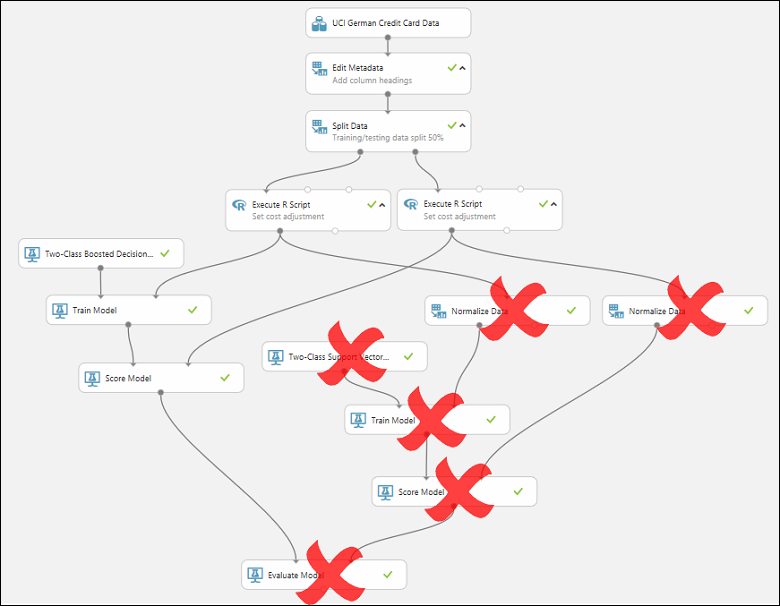
First, we need to trim this experiment down a little. We currently have two different models in the experiment, but we only want to use one model when we deploy this as a web service.

Let's say we've decided that the boosted tree model performed better than the SVM model. So, the first thing to do is remove the Two-Class Support Vector Machine module and the modules that were used for training it. You may want to make a copy of the experiment first by clicking Save As at the bottom of the experiment canvas.

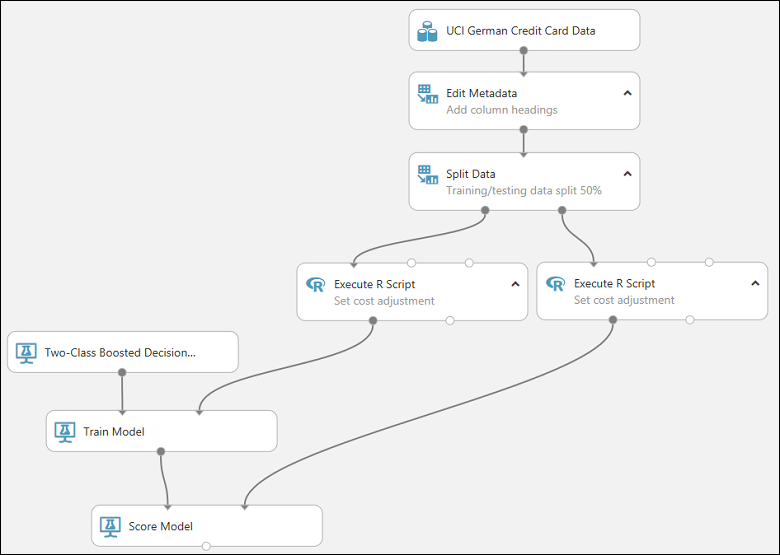
We need to delete the following modules:

* Two-Class Support Vector Machine
* Train Model and Score Model modules that were connected to it
* Normalize Data (both of them)
* Evaluate Model (because we're finished evaluating the models)

Select each module and press the Delete key, or right-click the module and select Delete.



Our model should now look something like this:



Now we're ready to deploy this model using the Two-Class Boosted Decision Tree.

#### Convert the training experiment to a predictive experiment

To get this model ready for deployment, we need to convert this training experiment to a predictive experiment. This involves three steps:

1. Save the model we've trained and then replace our training modules
2. Trim the experiment to remove modules that were only needed for training
3. Define where the web service will accept input and where it generates the output

We could do this manually, but fortunately all three steps can be accomplished by clicking Set Up Web Service at the bottom of the experiment canvas (and selecting the Predictive Web Service option).

|  |
| --- |
| Tip |
| If you want more details on what happens when you convert a training experiment to a predictive experiment, see How to prepare your model for deployment in Azure Machine Learning Studio. |

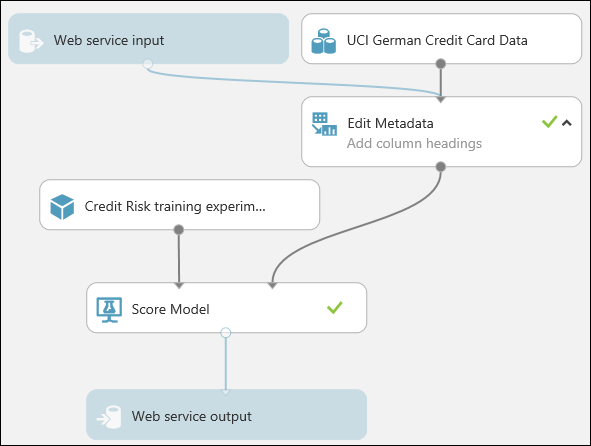
When you click Set Up Web Service, several things happen:

* The trained model is converted to a single Trained Model module and stored in the module palette to the left of the experiment canvas (you can find it under Trained Models)
* Modules that were used for training are removed; specifically:
  + Two-Class Boosted Decision Tree
  + Train Model
  + Split Data
  + The second Execute R Script module that was used for test data
* The saved trained model is added back into the experiment
* Web service input and Web service output modules are added (these identify where the user's data will enter the model, and what data is returned, when the web service is accessed)

|  |
| --- |
| Note |
| You can see that the experiment is saved in two parts under tabs that have been added at the top of the experiment canvas. The original training experiment is under the tab Training experiment, and the newly created predictive experiment is under Predictive experiment. The predictive experiment is the one we'll deploy as a web service. |

We need to take one additional step with this particular experiment. We added two Execute R Script modules to provide a weighting function to the data. That was just a trick we needed for training and testing, so we can take out those modules in the final model. Machine Learning Studio removed one Execute R Script module when it removed the Split module. Now we can remove the other and connect Metadata Editor directly to Score Model.

Our experiment should now look like this:



|  |
| --- |
| Note |
| You may be wondering why we left the UCI German Credit Card Data dataset in the predictive experiment. The service is going to score the user's data, not the original dataset, so why leave the original dataset in the model? |
| It's true that the service doesn't need the original credit card data. But it does need the schema for that data, which includes information such as how many columns there are and which columns are numeric. This schema information is necessary to interpret the user's data. We leave these components connected so that the scoring module has the dataset schema when the service is running. The data isn't used, just the schema. |
| One important thing to note is that if your original dataset contained the label, then the expected schema from the web input will also expect a column with the label! A way around this is to remove the label, and any other data that was in the training dataset, but will not be in the web inputs, before connecting the web input and training dataset into a common module. |

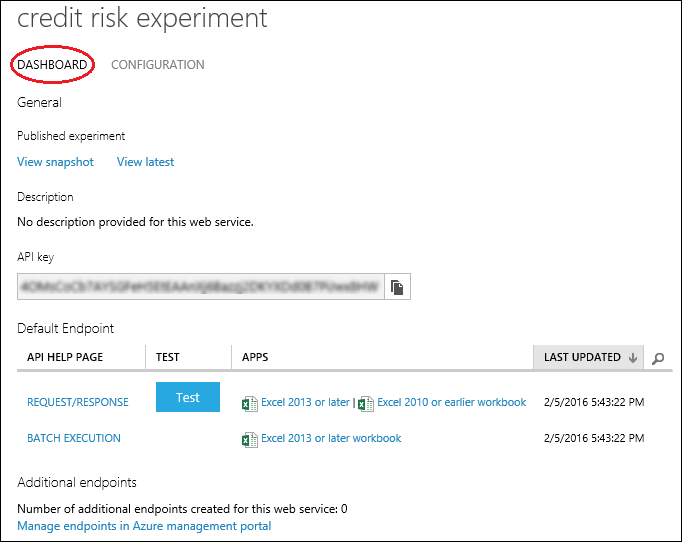
Run the experiment one last time (click Run.) If you want to verify that the model is still working, click the output of the Score Model module and select View Results. You can see that the original data is displayed, along with the credit risk value ("Scored Labels") and the scoring probability value ("Scored Probabilities".)

#### Deploy the web service

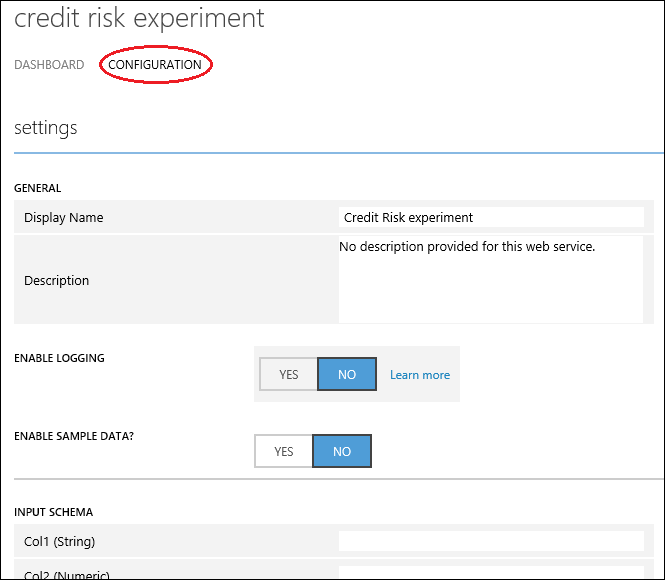
You can deploy the experiment as either a Classic web service, or as a New web service that's based on Azure Resource Manager.

##### Deploy as a Classic web service

To deploy a Classic web service derived from our experiment, click Deploy Web Service below the canvas and select Deploy Web Service [Classic]. Machine Learning Studio deploys the experiment as a web service and takes you to the dashboard for that web service. From this page you can return to the experiment (View snapshot or View latest) and run a simple test of the web service (see Test the web service below). There is also information here for creating applications that can access the web service (more on that in the next step of this walkthrough).



You can configure the service by clicking the CONFIGURATION tab. Here you can modify the service name (it's given the experiment name by default) and give it a description. You can also give more friendly labels for the input and output data.



##### Deploy as a New web service

|  |
| --- |
| Note |
| To deploy a New web service, you must have sufficient permissions in the subscription to which you are deploying the web service. For more information, see Manage a web service using the Azure Machine Learning Web Services portal. |

To deploy a New web service derived from our experiment:

1. Click Deploy Web Service below the canvas and select Deploy Web Service [New]. Machine Learning Studio transfers you to the Azure Machine Learning web services Deploy Experiment page.
2. Enter a name for the web service.
3. For Price Plan, you can select an existing pricing plan, or select "Create new" and give the new plan a name and select the monthly plan option. The plan tiers default to the plans for your default region and your web service is deployed to that region.
4. Click Deploy.

After a few minutes, the Quickstart page for your web service opens.

You can configure the service by clicking the Configure tab. Here you can modify the service title and give it a description.

To test the web service, click the Test tab (see Test the web service below). For information on creating applications that can access the web service, click the Consume tab (the next step in this walkthrough will go into more detail).

|  |
| --- |
| Tip |
| You can update the web service after you've deployed it. For example, if you want to change your model, then you can edit the training experiment, tweak the model parameters, and click Deploy Web Service, selecting Deploy Web Service [Classic] or Deploy Web Service [New]. When you deploy the experiment again, it replaces the web service, now using your updated model. |

#### Test the web service

When the web service is accessed, the user's data enters through the Web service input module where it's passed to the Score Model module and scored. The way we've set up the predictive experiment, the model expects data in the same format as the original credit risk dataset. The results are returned to the user from the web service through the Web service output module.

|  |
| --- |
| Tip |
| The way we have the predictive experiment configured, the entire results from the Score Model module are returned. This includes all the input data plus the credit risk value and the scoring probability. But you can return something different if you want - for example, you could return just the credit risk value. To do this, insert a Project Columns module between Score Model and the Web service output to eliminate columns you don't want the web service to return. |

You can test a Classic web service either in Machine Learning Studio or in the Azure Machine Learning Web Services portal. You can test a New web service only in the Machine Learning Web Services portal.

|  |
| --- |
| Tip |
| When testing in the Azure Machine Learning Web Services portal, you can have the portal create sample data that you can use to test the Request-Response service. On the Configure page, select "Yes" for Sample Data Enabled?. When you open the Request-Response tab on the Test page, the portal fills in sample data taken from the original credit risk dataset. |

##### Test a Classic web service

You can test a Classic web service in Machine Learning Studio or in the Machine Learning Web Services portal.

**Test in Machine Learning Studio**

1. On the DASHBOARD page for the web service, click the Test button under Default Endpoint. A dialog pops up and asks you for the input data for the service. These are the same columns that appeared in the original credit risk dataset.
2. Enter a set of data and then click OK.

**Test in the Machine Learning Web Services portal**

1. On the DASHBOARD page for the web service, click the Test preview link under Default Endpoint. The test page in the Azure Machine Learning Web Services portal for the web service endpoint opens and asks you for the input data for the service. These are the same columns that appeared in the original credit risk dataset.
2. Click Test Request-Response.

##### Test a New web service

You can test a New web service only in the Machine Learning Web Services portal.

1. In the Azure Machine Learning Web Services portal, click Test at the top of the page. The Test page opens and you can input data for the service. The input fields displayed correspond to the columns that appeared in the original credit risk dataset.
2. Enter a set of data and then click Test Request-Response.

The results of the test are displayed on the right-hand side of the page in the output column.

### Manage the web service

Once you've deployed your web service, whether Classic or New, you can manage it from the Microsoft Azure Machine Learning Web Services portal.

To monitor the performance of your web service:

1. Sign in to the Microsoft Azure Machine Learning Web Services portal
2. Click Web services
3. Click your web service
4. Click the Dashboard

### Step 6: Access the Azure Machine Learning web service

In the previous step in this walkthrough we deployed a web service that uses our credit risk prediction model. Now users are able to send data to it and receive results.

The Web service is an Azure web service that can receive and return data using REST APIs in one of two ways:

* **Request/Response** - The user sends one or more rows of credit data to the service by using an HTTP protocol, and the service responds with one or more sets of results.
* **Batch Execution** - The user stores one or more rows of credit data in an Azure blob and then sends the blob location to the service. The service scores all the rows of data in the input blob, stores the results in another blob, and returns the URL of that container.

The quickest and easiest way to access a Classic web service is through the Azure ML Request-Response Service Web App or Azure ML Batch Execution Service Web App Template.

These web app templates can build a custom web app that knows your web service's input data and what it will return. All you need to do is provide access to your web service and data, and the template does the rest.

For more information on using the web app templates, see [Consume an Azure Machine Learning Web service with a web app template](https://docs.microsoft.com/en-us/azure/machine-learning/studio/consume-web-service-with-web-app-template).

You can also develop a custom application to access the web service using starter code provided for you in R, C#, and Python programming languages.