Microsoft

ImagineCup

Churn Prediction and Intervention

Azure Machine Learning and Azure App Service

Overview

Churn is a term employed when consumers stop using a good or service. It is seen across a number of industries, and in many cases, companies devote additional resources to stop a customer from leaving. With online computer games, for example, substantial data is available to analyze individual and collective player behavior to help predict when a particular gamer may leave. Game creators may even take their capabilities a step further and intervene in-game to incentivize players to continue gaming. Additional bonuses, power-ups, and numerous other tactics are common methods employed to keep such players involved.

The *Super Family* game is a basic web app about a family of super heroes. Heroes battle villains, but there is a twist—the game uses the results of a predictive model to account for potential churn, and it provides a new helmet to a player if a churn event is imminent. If churn is not likely, the game does not provide the bonus.

You will have the opportunity to alter your machine learning experiment to try to make your churn prediction as accurate as possible. In the real world, misclassification can be costly. Imagine if a company provided a free month of service or a hotel provided a free stay, when the customer was not actually considering alternative services. Money or other resources would have been employed for no reason. The goal in this case is to reduce such *false positives*—when a churn event is predicted, but in reality, the consumer is not likely to churn.

Objectives

This exercise is split into six activities that will help you gain an understanding of churn analysis and how to *intervene* to try to prevent churn. You will learn the basic process used to train, score and evaluate a predictive churn model. The first four activities utilize **Azure Machine Learning**

(Azure ML) to prepare a model to determine whether or not a player may churn. The final two activities use **Azure App Service** and involve deploying a simple ASP.NET *Super Family* game that invokes your Azure ML model. Based on the prediction returned by Azure ML, in-game behavior changes to incentivize a player to continue playing. Using different selection of features and changes to model parameters, how accurate can you become at predicting churn?

Requirements/Prerequisites

1. A Microsoft account is required to access an Azure Machine Learning workspace. If you do not already have a Microsoft account, you can obtain one for free by following the link below:

   https://www.microsoft.com/en-us/account/default.aspx

2. An Azure subscription is required to use Azure App Service and the associated publishing features of Visual Studio. Students can create a limited Azure subscription for free by signing up for the DreamSpark program using the relevant link below. If you are not a student or are unable to provide proof of education, you can obtain a free trial for Azure using the main Azure link:

   https://www.dreamspark.com/

   https://azure.microsoft.com/

3. Visual Studio 2015 is required for the Super Family ASP.NET web app project. The free Visual Studio Community version is available using the link below:

   https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx

4. The Azure SDK version 2.8.1 is required to publish your Super Family app to Azure App Service using Visual Studio. The Azure SDK can be installed using the Web Platform Installer.

   https://www.microsoft.com/web/downloads/platform.aspx

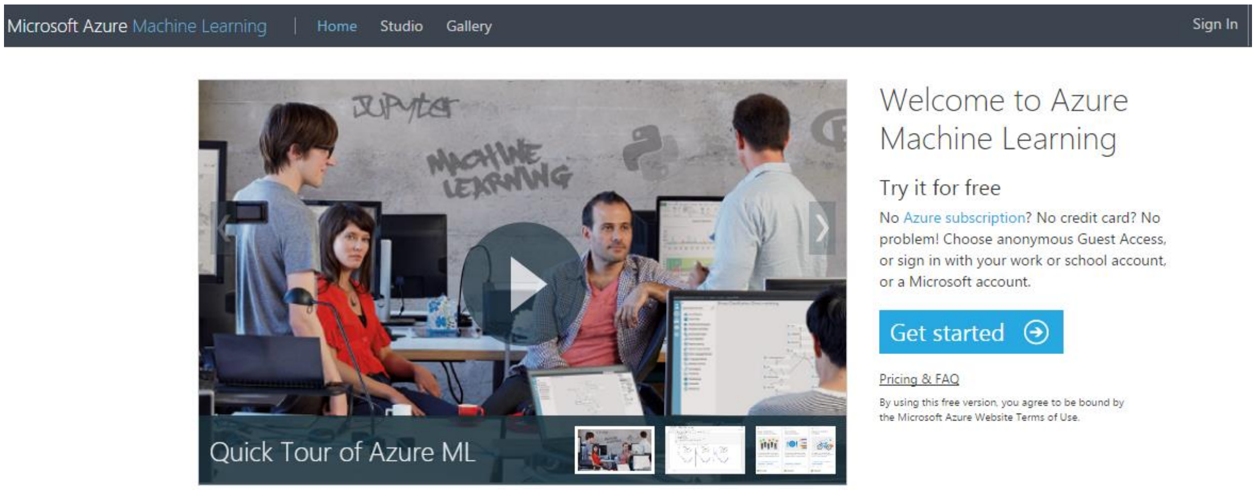
Acknowledgements

This lab was adapted from original content created by Wee Hyong Tok, Ph.D., Senior Program Manager on the Azure Machine Learning team at Microsoft.

Activity 1: Prepare Data for a Churn Experiment

In order to predict when a gamer may churn and provide incentive for them to stay, you will start by creating an experiment in Azure Machine Learning. This activity covers how to setup an experiment in **Azure ML Studio**, read a dataset, and perform basic data preparation before it can be used by a machine learning algorithm.

1. Go to https://studio.azureml.net and **Sign In** using your Microsoft Account. If you do not yet have a Microsoft Account, you can create one at https://www.microsoft.com/en-us/account/default.aspx.



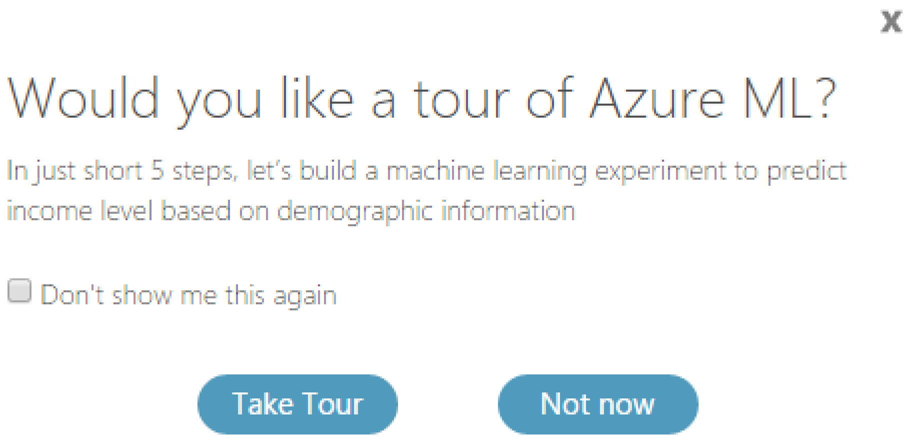
Welcome to Azure Machine Learning

Try it for free

No Azure subscription? No credit card? No problem! Choose anonymous Guest Access, or sign in with your work or school account, or a Microsoft account.

By using this free version, you agree to be bound by the Microsoft Azure Website Terms of Use.

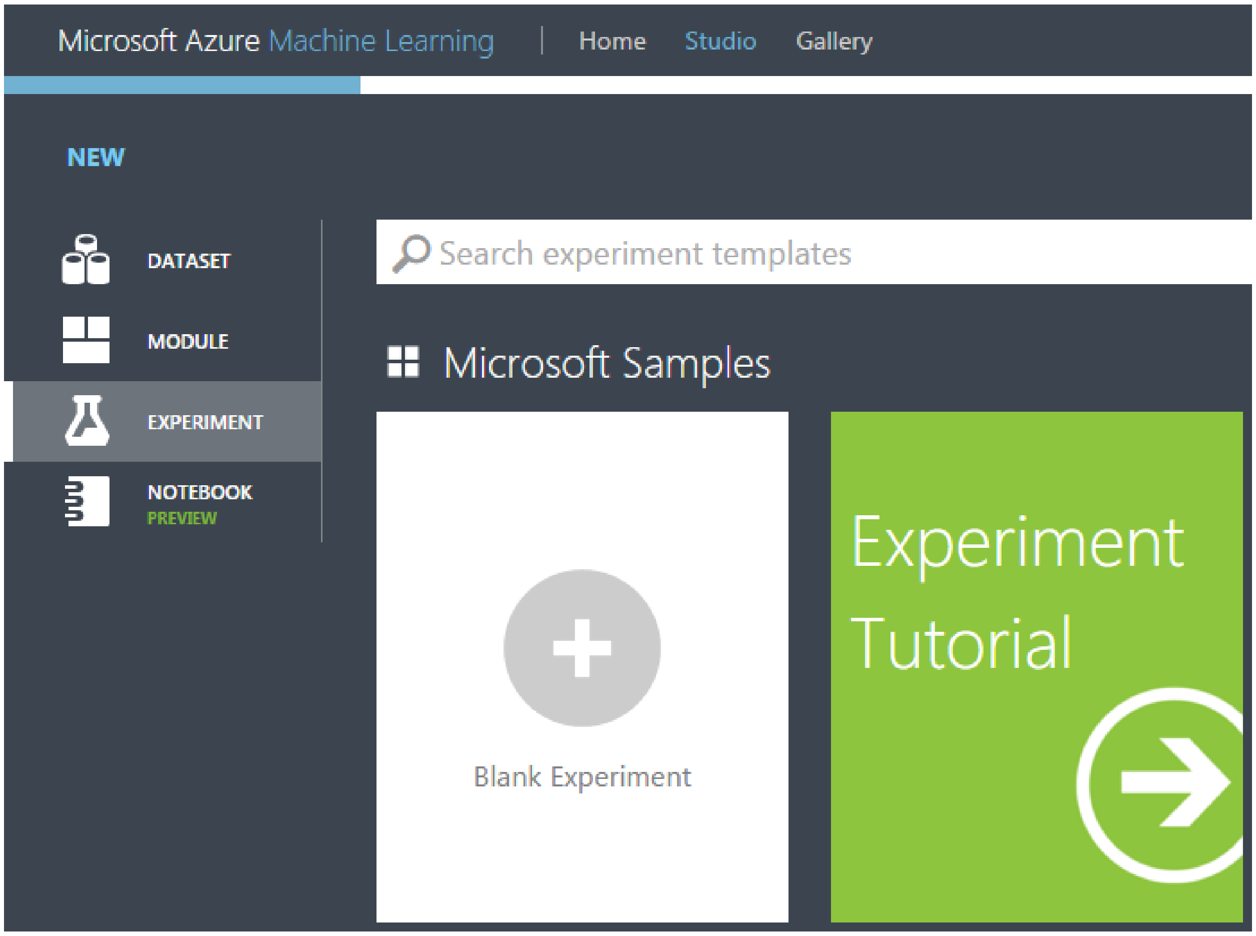
2. After signing in, if this is your first time accessing Azure ML, you may see an option to *Take a Tour*. Select **Not Now** if applicable.



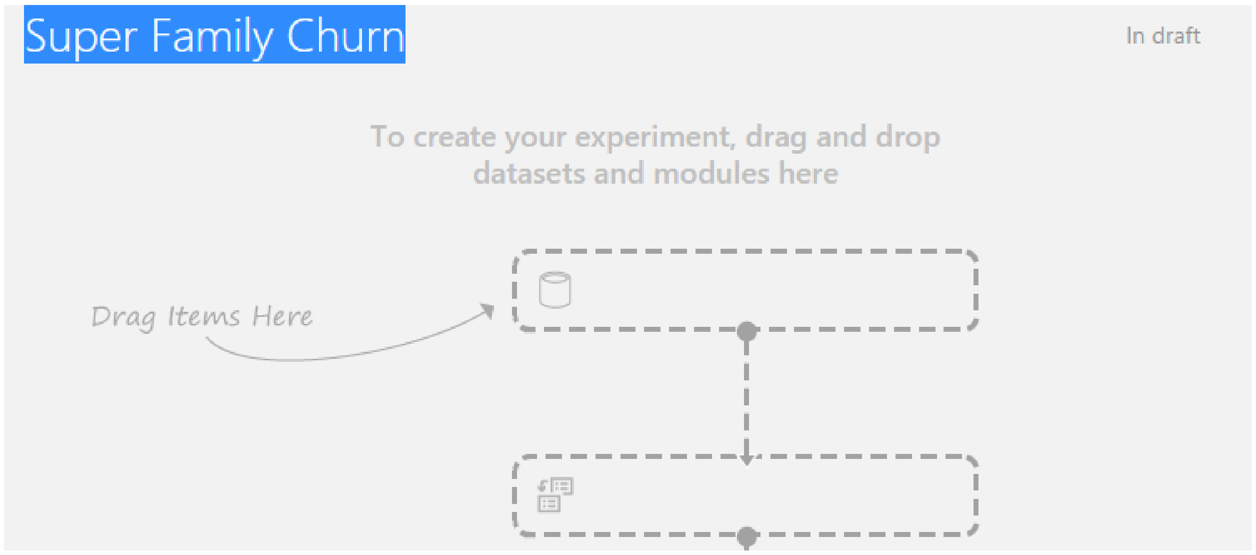
Would you like a tour of Azure ML?

In just short 5 steps, let's build a machine learning experiment to predict income level based on demographic information

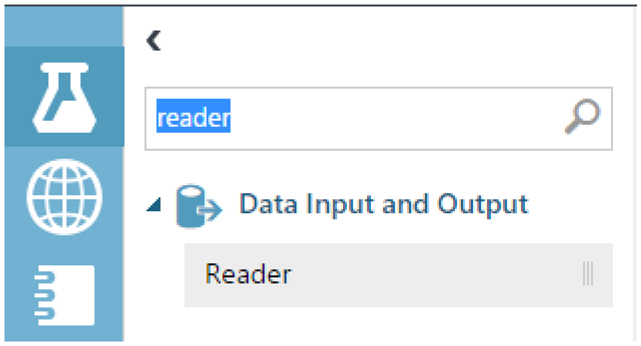
3. Select the option for **Blank Experiment**. NOTE: You may need to click **New** first.



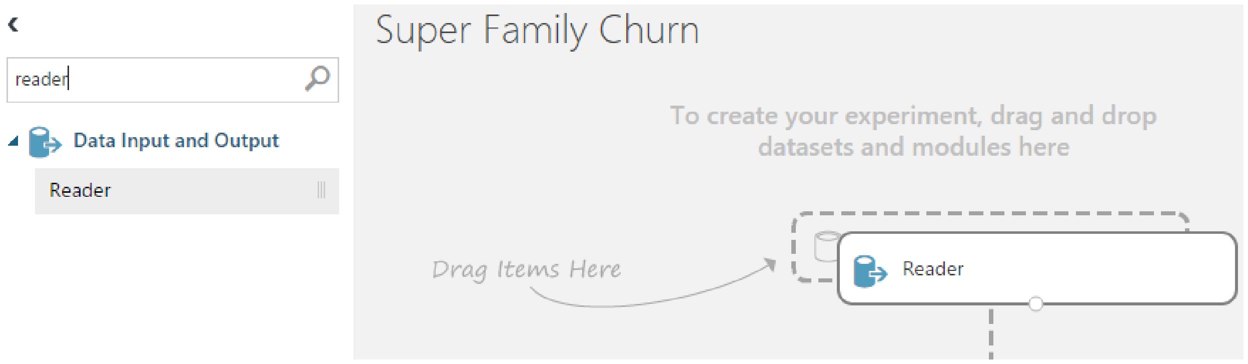
4. Click on the experiment title at the top and rename it from “Experiment created on [current date]” to **Super Family Churn**. The game scenario that you will create in a later activity will be *Super Family*.


Super Family Churn
In draft
To create your experiment, drag and drop datasets and modules here
Drag Items Here

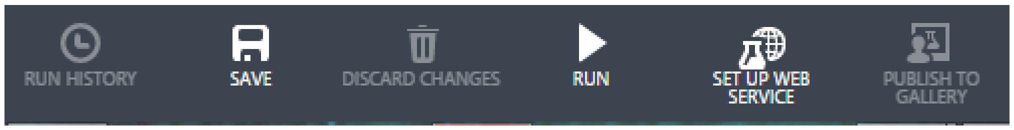
5. In the selection panel on the left, you will see a number of menu options relating to functionality in Azure ML. Many of the options will not be relevant for this experiment, so you will rely on the search functionality rather than browsing. In the search box, type **reader**, and you should see the **Reader** option appear under *Data Input and Output*.

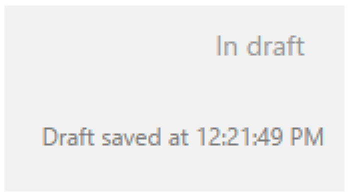


6. Click on the **Reader** option under *Data Input and Output*, then drag it into your experiment where it says *Drag Items Here*.



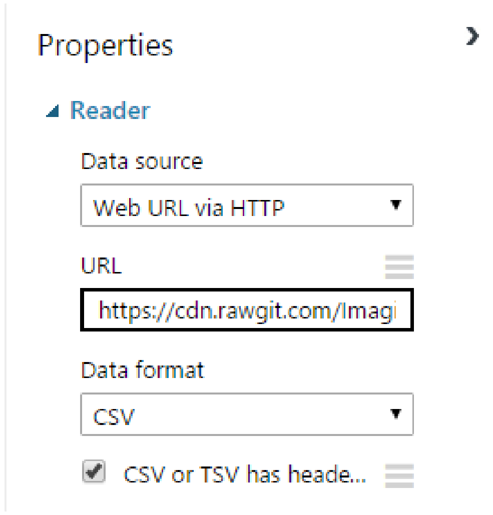
7. At this point, click the **Save** icon on the bottom bar, then select the **Save** option. After a few seconds, you should see a timestamp noting that the experiment has been saved.



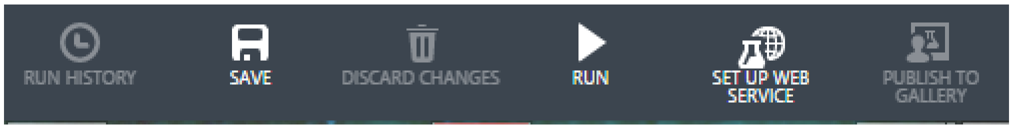


8. In the Properties panel for the Reader, you will add your data input. The input is available online in CSV (comma-separated) format. It contains historical attributes that can be used to train your machine learning models as well as a *Churn* column noting whether or not a gamer churned in reality.
   a. For *Data source*, select **Web URL via HTTP**.

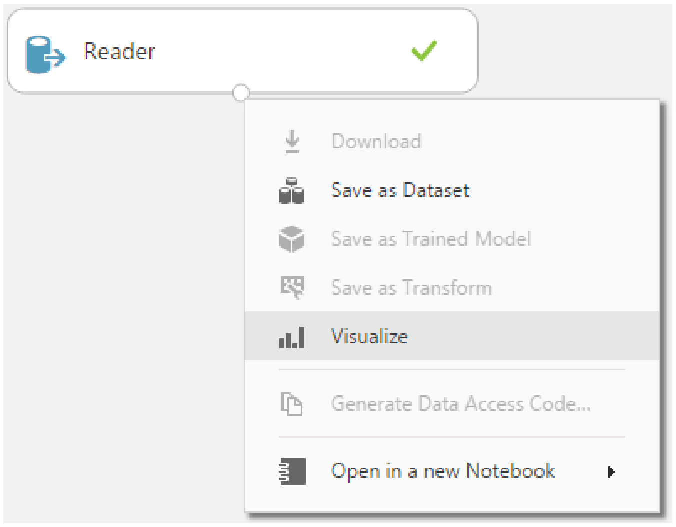
b. For *URL*, copy the following link and paste it into the *URL* field: **https://cdn.rawgit.com/ImagineCupGame/SuperFamilyAzureML/master/SuperFamilyTraining.csv**
c. For *Data format*, select **CSV.**
d. Check the box for **CSV or TSV has headers** (NOTE: this selection is not optional and having it checked is important for future steps).


Properties
Reader
Data source
Web URL via HTTP
URL
https://cdn.rawgit.com/Imagi
Data format
CSV
☑ CSV or TSV has heade...

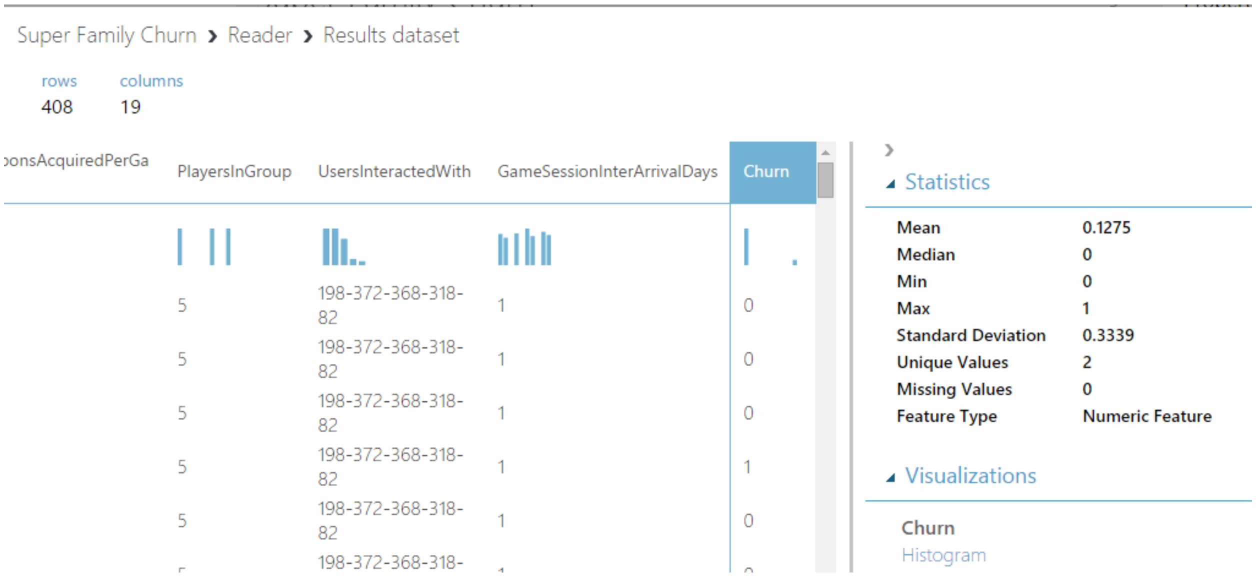
9. Select **Run** in the bottom bar to run the experiment. Running it now will allow you to visualize the new CSV dataset in Azure ML.



10. After the experiment runs, you should see a green checkmark next to your *Reader* module. Click on the small circle at the bottom of the *Reader* module and select the option to **Visualize**.



11. The window that opens shows a preview of all of the data that is available to use for predicting whether a gamer will churn. Note that the training dataset contains 408 rows and 19 columns. In each column, a small histogram appears showing how the data is distributed. Scroll to the right until you see the **Churn** column, and then select it. Note how more detailed *Statistics* are available. Because gamers will either churn or not churn, this data is represented as either a zero (false) or one (true). A "1" represents a churn event, and looking at the Mean of 0.1275, it appears that one in eight gamers churned in our training dataset. Browse the data as desired, and then click the **X** in the top right corner to return to the experiment when ready.



Super Family Churn › Reader › Results dataset

rows	columns
408	19

›onsAcquiredPerGa	PlayersInGroup	UsersInteractedWith	GameSessionInterArrivalDays	Churn
	5	198-372-368-318-82	1	0
	5	198-372-368-318-82	1	0
	5	198-372-368-318-82	1	0
	5	198-372-368-318-82	1	1
	5	198-372-368-318-82	1	0
		198-372-368-318-		

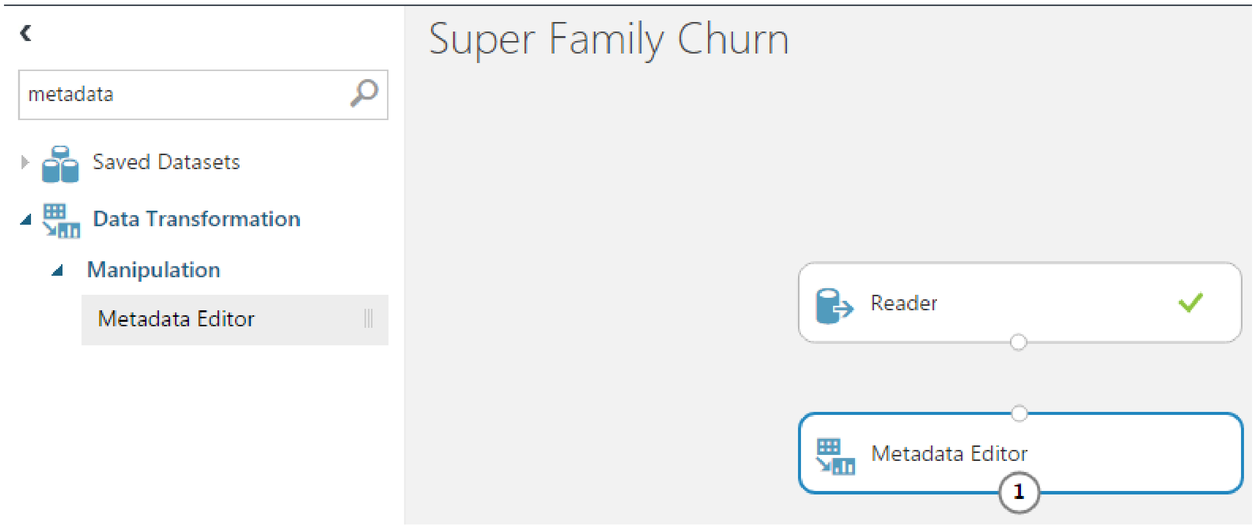
Statistics

Mean	0.1275
Median	0
Min	0
Max	1
Standard Deviation	0.3339
Unique Values	2
Missing Values	0
Feature Type	Numeric Feature

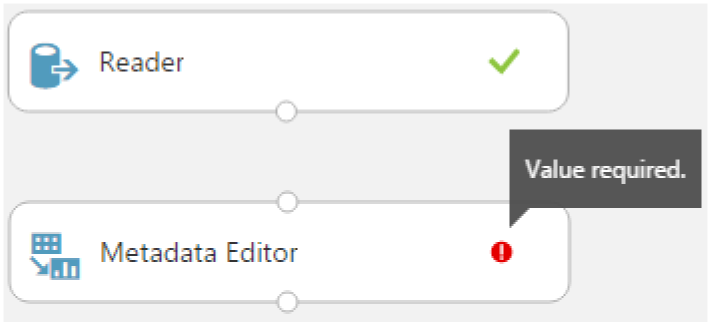
Visualizations

Churn
Histogram

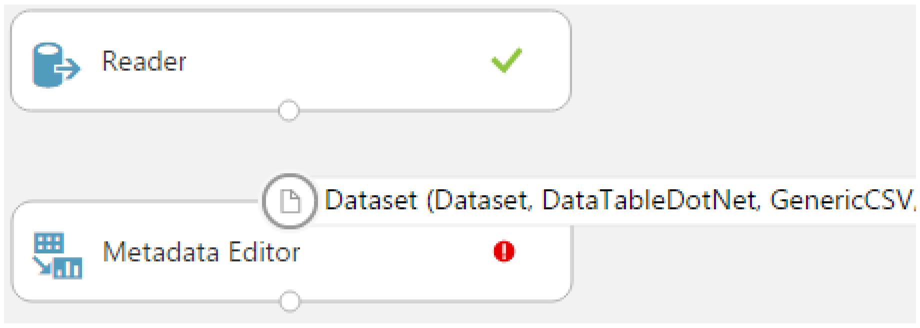
12. In the selection panel, search for **metadata**, then drag the **Metadata Editor** into the experiment. This module will allow you to manage how the training data is described.



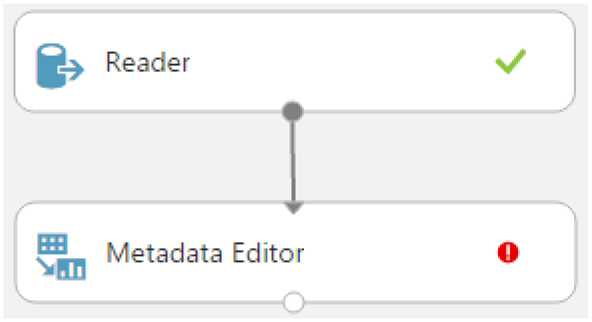
13. When you click off of the new Metadata Editor module, notice that it has a red exclamation point. Hovering over the warning allows you to see that a value is required. This warning appears because the Metadata Editor does not have any data input from the Reader!



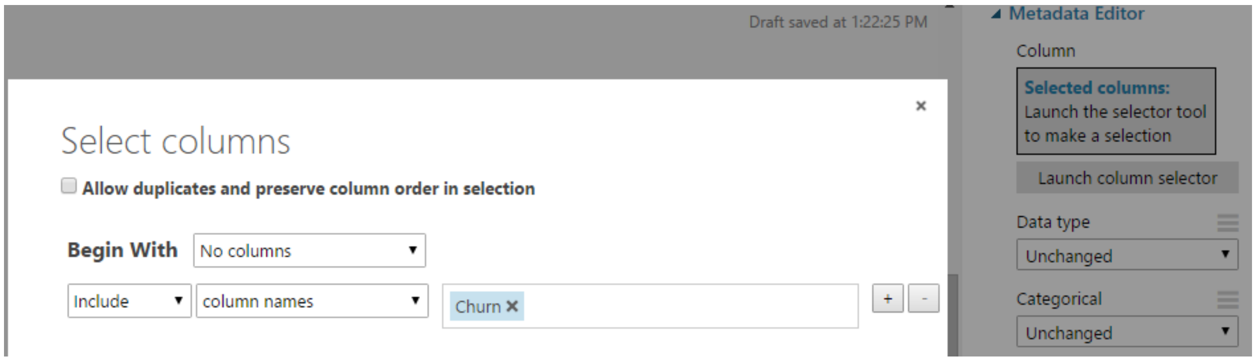
14. Hover over the circle atop the **Metadata Editor** and note that it is requesting a Dataset input.



15. Now, click on the circle under **Reader**, and drag the arrow to the circle above **Metadata Editor**. This takes the dataset as output from your Reader and makes it available to the Metadata Editor as input.



16. Select **Metadata Editor**, and in the Properties panel on the right, click **Launch column selector**. In the new window, select the **Churn** column from the list, then click the **OK** button.


Draft saved at 1:22:25 PM

Select columns

☐ Allow duplicates and preserve column order in selection

Begin With: No columns

Include | column names | Churn ✕

Metadata Editor

Column

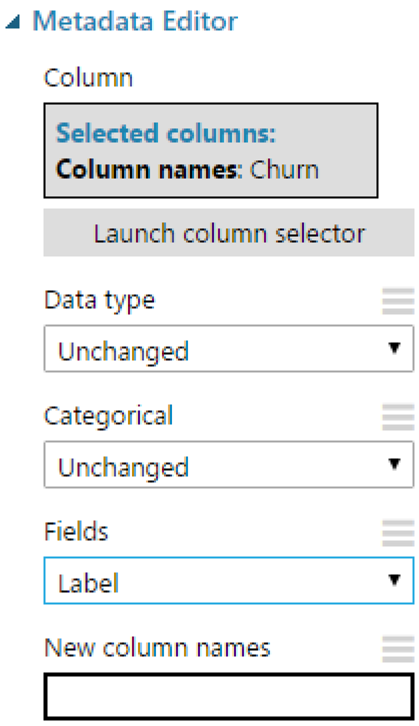
Selected columns:
Launch the selector tool to make a selection

Launch column selector

Data type: Unchanged

Categorical: Unchanged

17. Back in Properties, change the option for *Fields* to **Label**.


Metadata Editor

Column

Selected columns:
Column names: Churn

Launch column selector

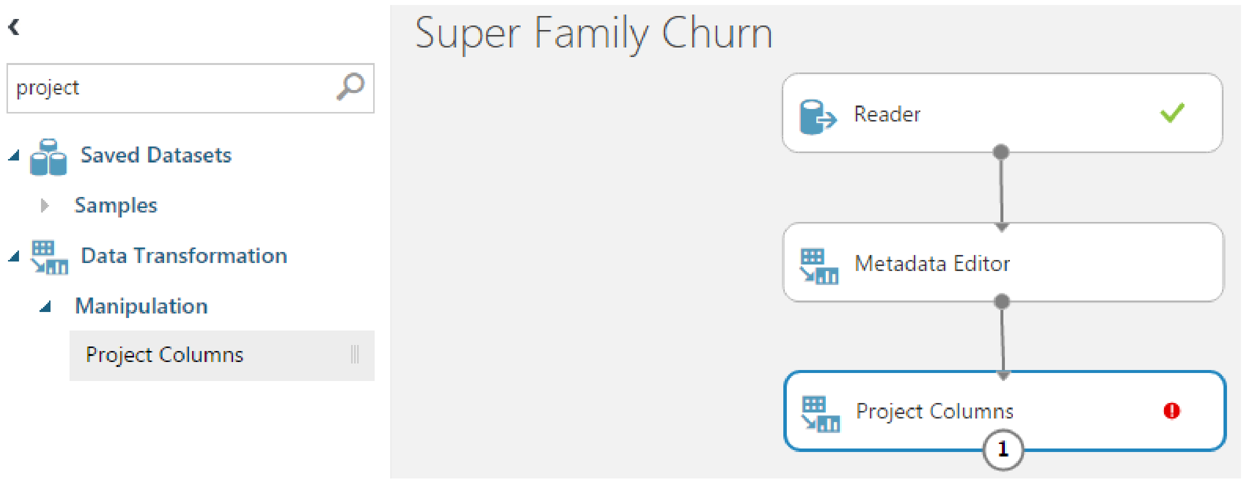
Data type: Unchanged

Categorical: Unchanged

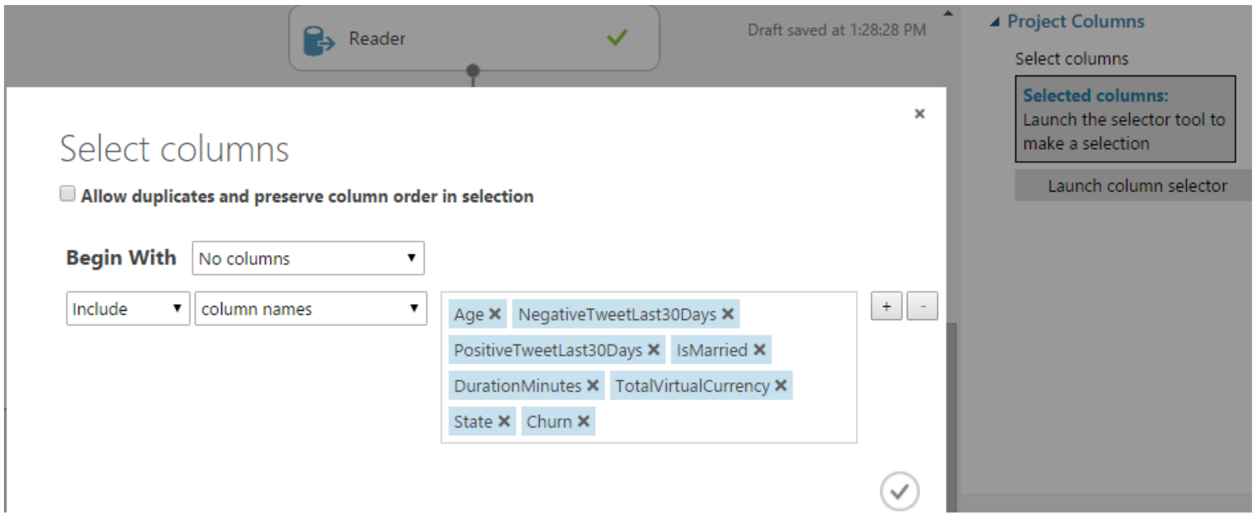
Fields: Label

New column names:

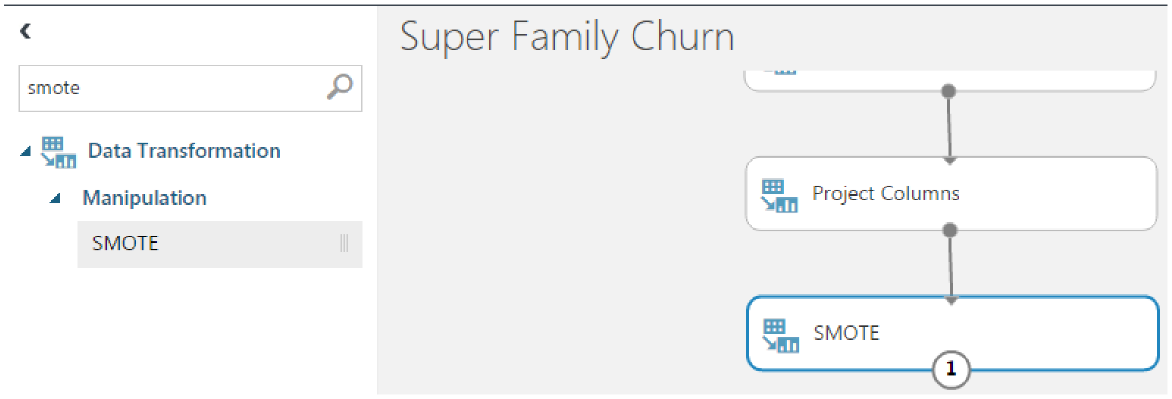
18. In the selection panel, search for **project**, drag the **Project Columns** module into the experiment, and drag the output from Metadata Editor into the input circle for Project Columns. This new module allows you to filter the dataset and select which of the attributes in the training dataset will be used (out of the 19 feature columns).



19. Select **Project Columns**, and in the Properties panel on the right, click **Launch column selector**. In the new window, select the **Age**, **NegativeTweetLast30Days**, **PositiveTweetLast30Days**, **IsMarried**, **DurationMinutes**, **TotalVirtualCurrency**, **State**, and **Churn** columns from the list. Click the **OK** button when done.

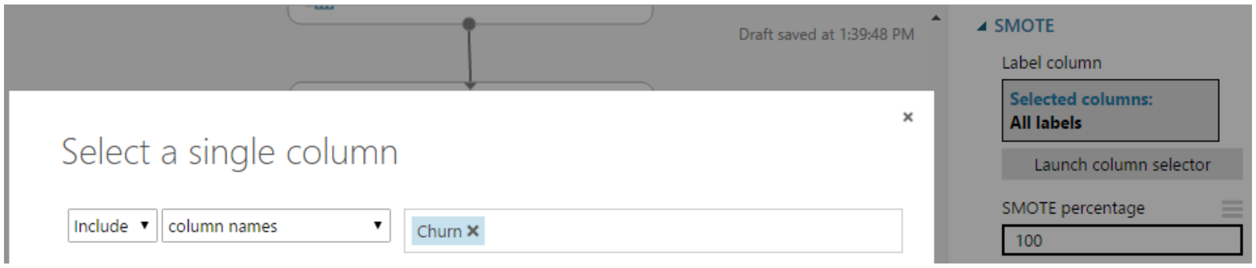


20. In the selection panel, search for **smote**, drag the **SMOTE** module into the experiment, and drag the output from **Project Columns** into the input circle for **SMOTE**.



21. Select the **SMOTE** module, and in the Properties panel on the right, select **Launch column selector**. Change the included columns option from *all labels* to **column names**, select the **Churn** column, and click **OK**.

"SMOTE" stands for *Synthetic Minority Oversampling Technique* and is a way to strategically increase your sample size (http://bit.ly/1PT2Vqt)


Draft saved at 1:39:48 PM

Select a single column

Include column names Churn

SMOTE

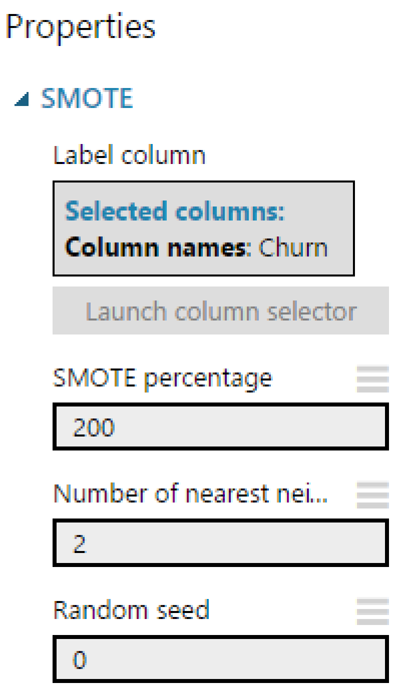
Label column

Selected columns: All labels

Launch column selector

SMOTE percentage 100

22. Back in the Properties panel, change the *SMOTE percentage* to **200** and *Number of nearest neighbors* to **2.** Leave *Random seed* as 0.


Properties

SMOTE

Label column

Selected columns: Column names: Churn

Launch column selector

SMOTE percentage 200

Number of nearest nei... 2

Random seed 0

23. At this stage, you took an initial dataset containing past player behavior and selected a subset of features that will be used later for prediction. You have prepared your data and are ready to work with machine learning models. **Save** the experiment.

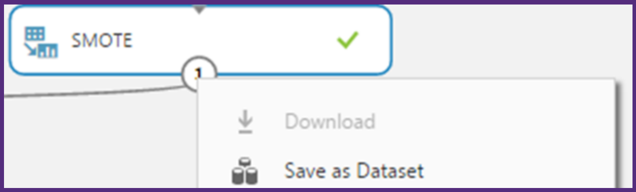
WANT TO LEARN MORE?

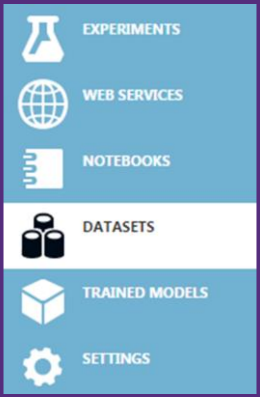
Before using machine learning algorithms in Azure ML, you need to make good initial decisions about your data. The sample game data is fairly straightforward, but in reality, it may take a lot of work to obtain and clean your dataset. Beyond data cleanliness, you also need to decide what is relevant to your analysis. In the **Project Columns** step in Activity 1, did you notice that you did not use every available column in the CSV file? In order to increase the predictive power of your model, you narrowed down the number of features. This is a key part of what is called *feature selection* in machine learning. Additionally, through *feature engineering*, you can even create new data that may help enhance your model.

Read more about feature selection and engineering in the Microsoft Azure documentation: https://azure.microsoft.com/en-us/documentation/articles/machine-learning-feature-selection-and-engineering/

ADDITIONAL CHALLENGE

Click on the output for the **SMOTE** module and select **Save as Dataset**. Enter a name and click **OK**. Click on the **DATASETS** icon to the left in the Azure ML Studio menu, and your dataset should appear. If you create a *different* experiment, can you determine how to read from this saved dataset instead of the CSV file from Activity 1?

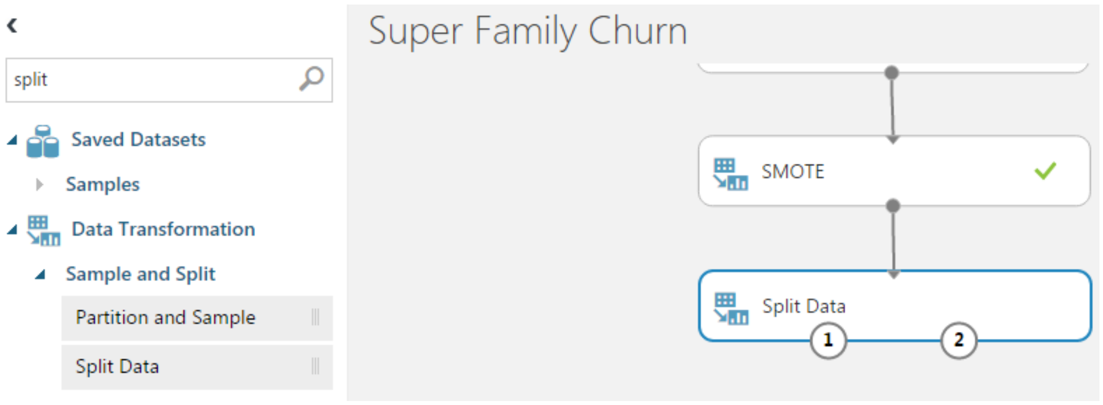




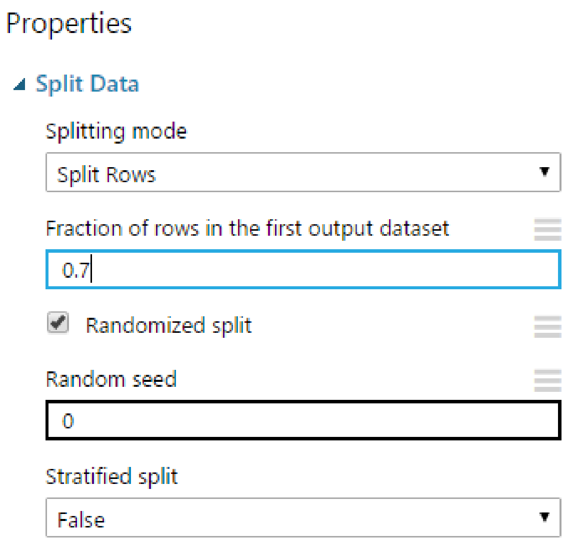
Activity 2: Train and Score a Model Using Support Vector Machines (SVM)

This activity introduces the process to split data into separate *train* and *test* datasets, select a model, and train that model using the train dataset. It also covers scoring the model using the test dataset and finally looks at different ways to evaluate model performance. A classification algorithm called *Support Vector Machines* (SVM) is used for this activity. The following steps do not require knowledge of the algorithm, but for more technical detail, see http://bit.ly/1m8U7jo.

1. In the selection panel, search for **split**, drag the **Split Data** module into the experiment, and drag the output from **SMOTE** into the input circle for **Split Data**.



2. Select the **Split Data** module, and in the Properties panel on the right, change the value for *Fraction of rows in the first output dataset* from 0.5 to **0.7**. This assigns 70% of the data to your train dataset.



Splitting mode: Split Rows

Fraction of rows in the first output dataset: 0.7

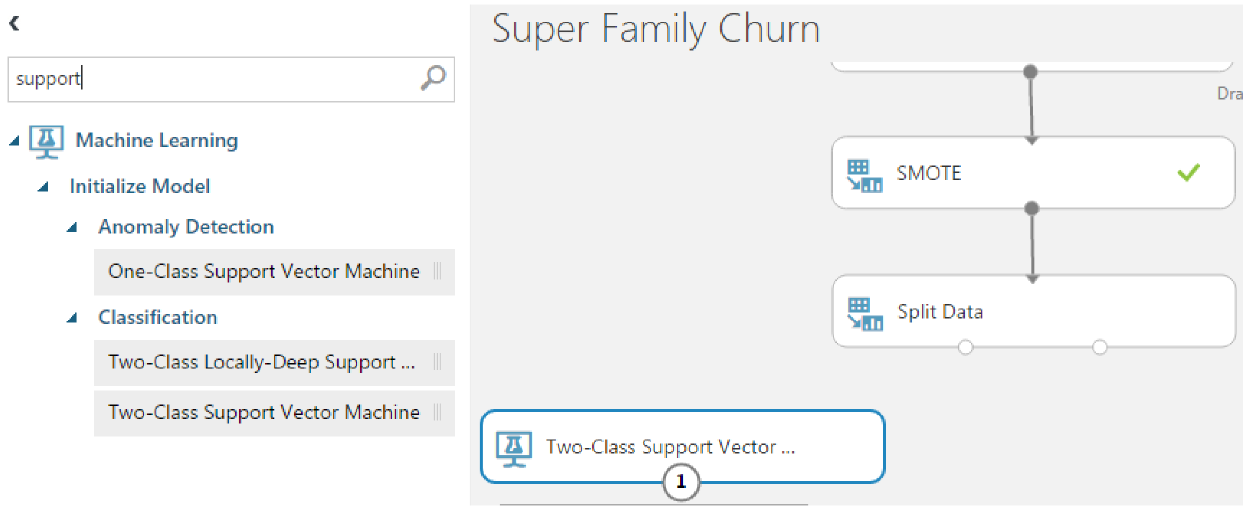
Randomized split: ✓

Random seed: 0

Stratified split: False

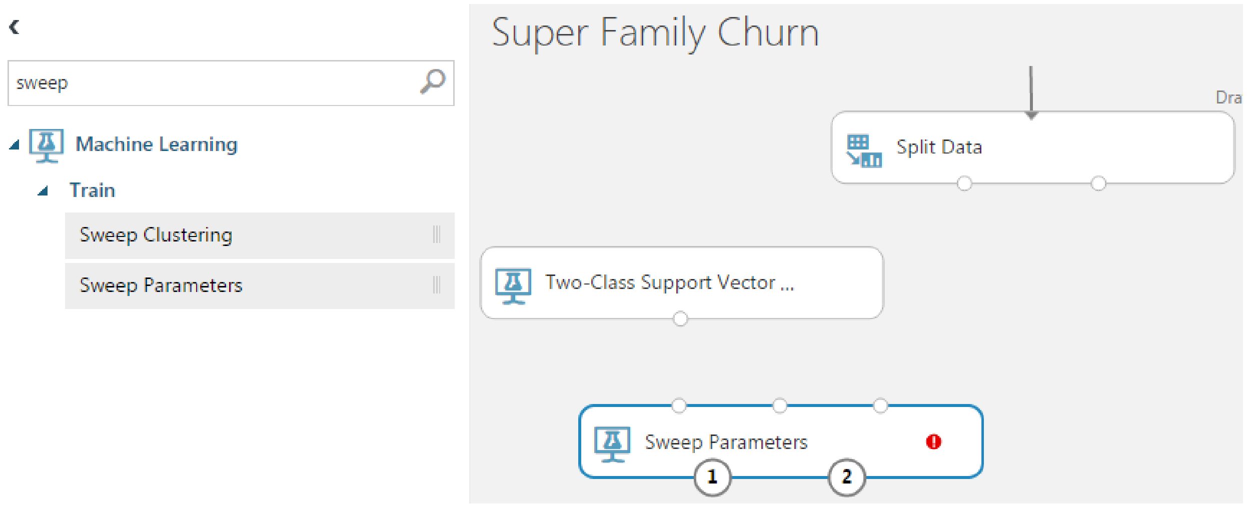
3. In the selection panel, search for **support**. A few results will appear. Find **Two-Class Support Vector Machine** under *Classification*. Drag the module into the experiment, but *do not* attempt to connect the Split Data module with the new SVM module. Leave all of the default Properties for the SVM module.

Classification algorithms group new observations based on features in your training dataset. They are used with discrete data.

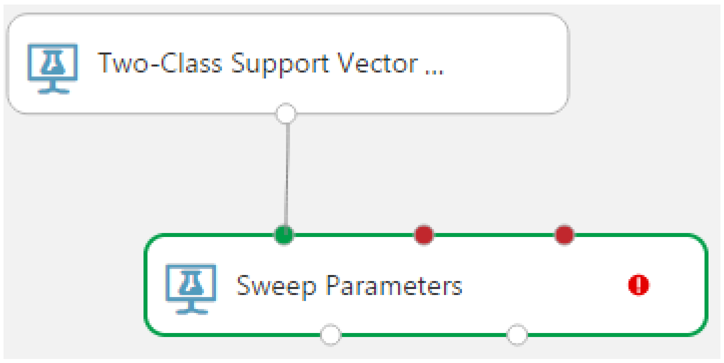

support
Machine Learning
Initialize Model
Anomaly Detection
One-Class Support Vector Machine
Classification
Two-Class Locally-Deep Support ...
Two-Class Support Vector Machine
Super Family Churn
Dra
SMOTE
Split Data
Two-Class Support Vector ...
1

4. In the selection panel, search for **sweep**. Drag the **Sweep Parameters** module into the experiment, but *do not* attempt to connect anything to the module yet.

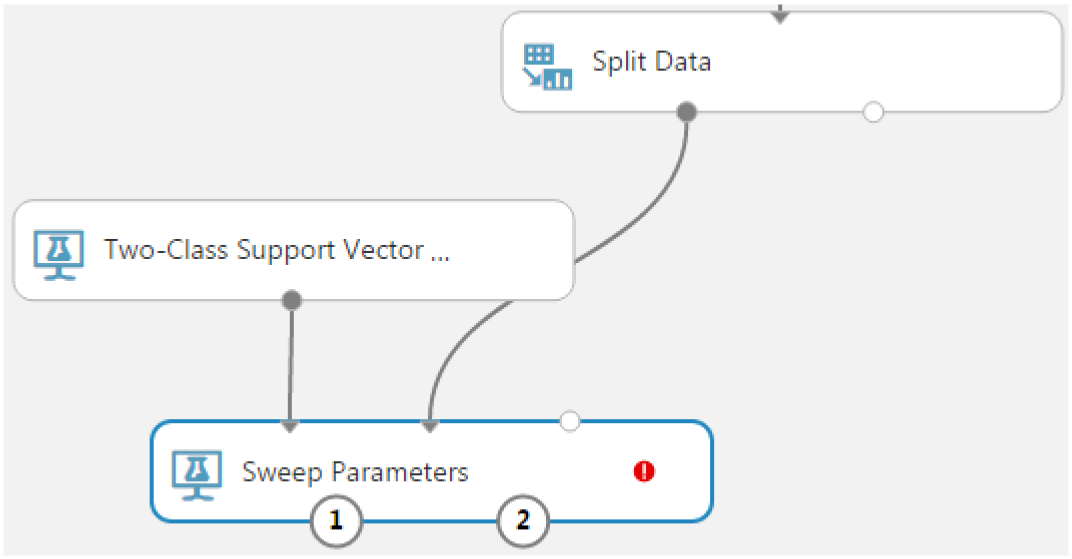
Parameter sweeping uses different variations of parameters and selects the optimal combination for training your model (http://bit.ly/1HRVWer)



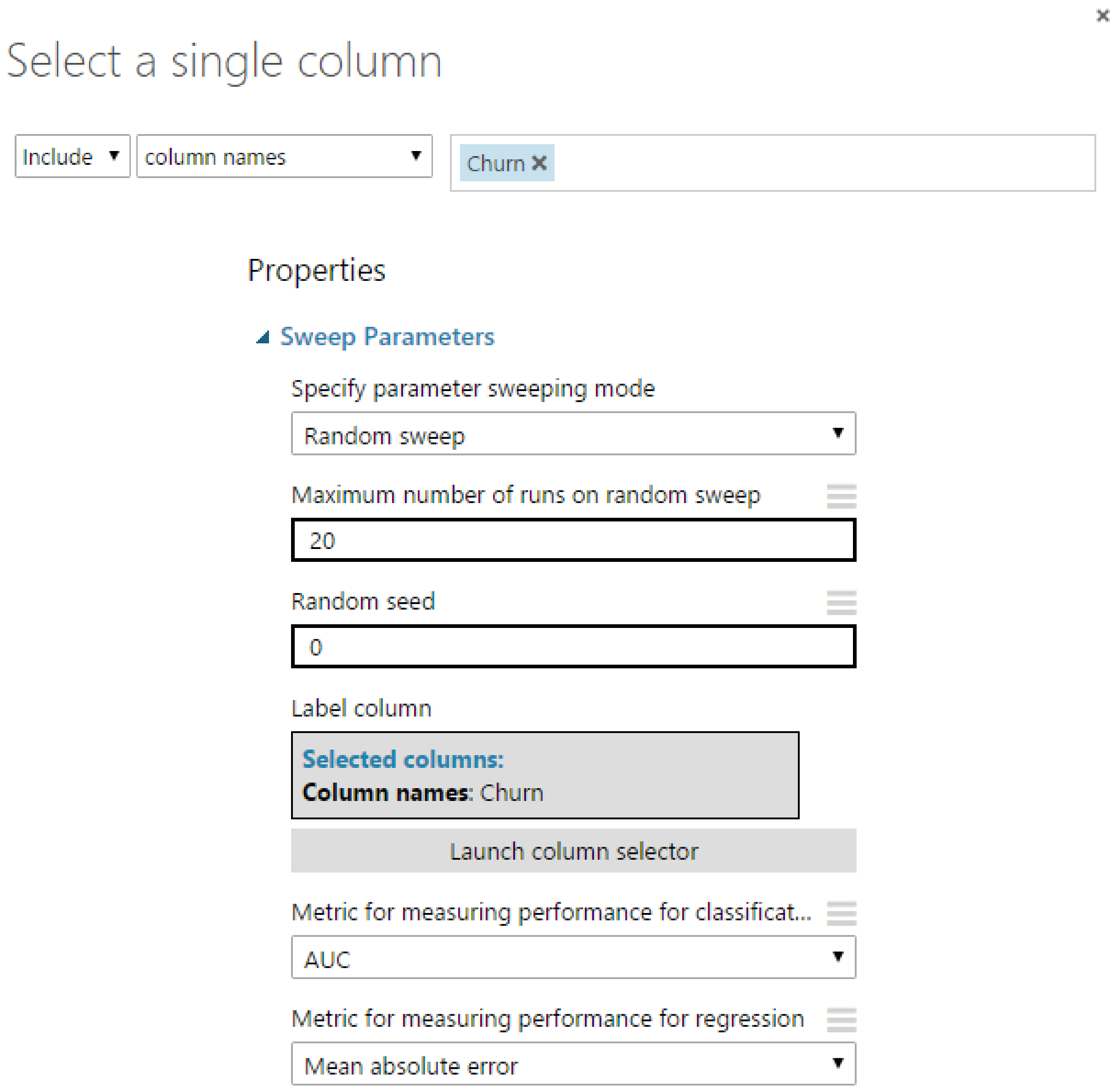
5. Connect the untrained model output from **Two-Class Support Vector Machine** to the leftmost input of **Sweep Parameters.** Note as you drag the arrow how the appropriate input is highlighted in green while the other two are red. Only the appropriate connection is allowed.



6. Drag the leftmost output from the **Split Data** module to the second input of **Sweep Parameters**.



7. In the Properties panel for **Sweep Parameters** to the right, change the value of *Maximum number of runs on random sweep* from 5 to **20.** Change the *Metric for measuring performance for classification* from Accuracy to **AUC** (AUC stands for "area under curve", which will be explained in more depth later when the model is evaluated). Finally, click **Launch column selector** and select the **Churn** column, then click **OK**.


Select a single column

Include column names Churn

Properties

Sweep Parameters

Specify parameter sweeping mode: Random sweep

Maximum number of runs on random sweep: 20

Random seed: 0

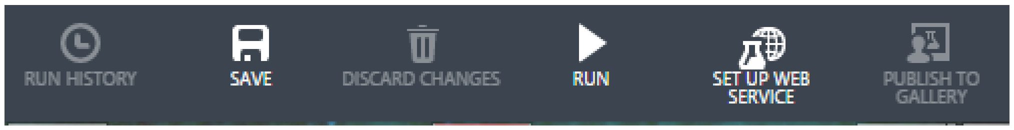
Label column

Selected columns:
Column names: Churn

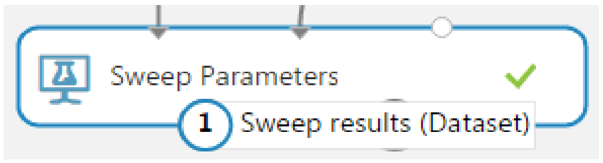
Metric for measuring performance for classificat...: AUC

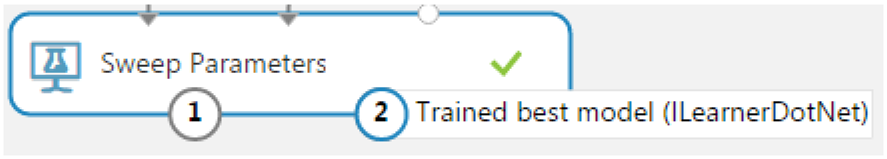
Metric for measuring performance for regression: Mean absolute error

8. Select **Run** in the bottom bar to run the experiment. It will take a minute for the experiment to run. After it finishes, you should see a green check mark in all of the modules.

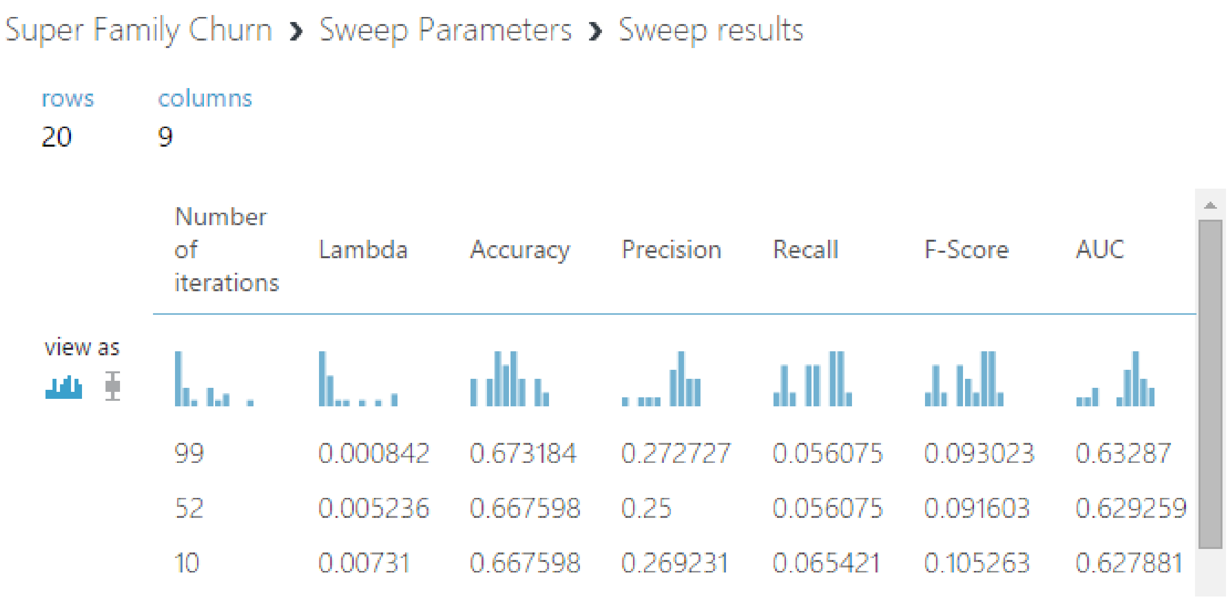


9. Click on the **Sweep Parameters** module. Hover over the **1** output and note how it is labeled *Sweep results*. Now, hover over the **2** output and note how it is labeled *Trained best model*.





10. Click on the **1 (Sweep results)** output and select **Visualize**. Note how the columns do not reflect the original features from the dataset. There are nine columns associated with the sweep action and twenty rows that correspond to the *Maximum number of runs on random sweep* property. Click the **X** in the corner when done to return to the experiment.



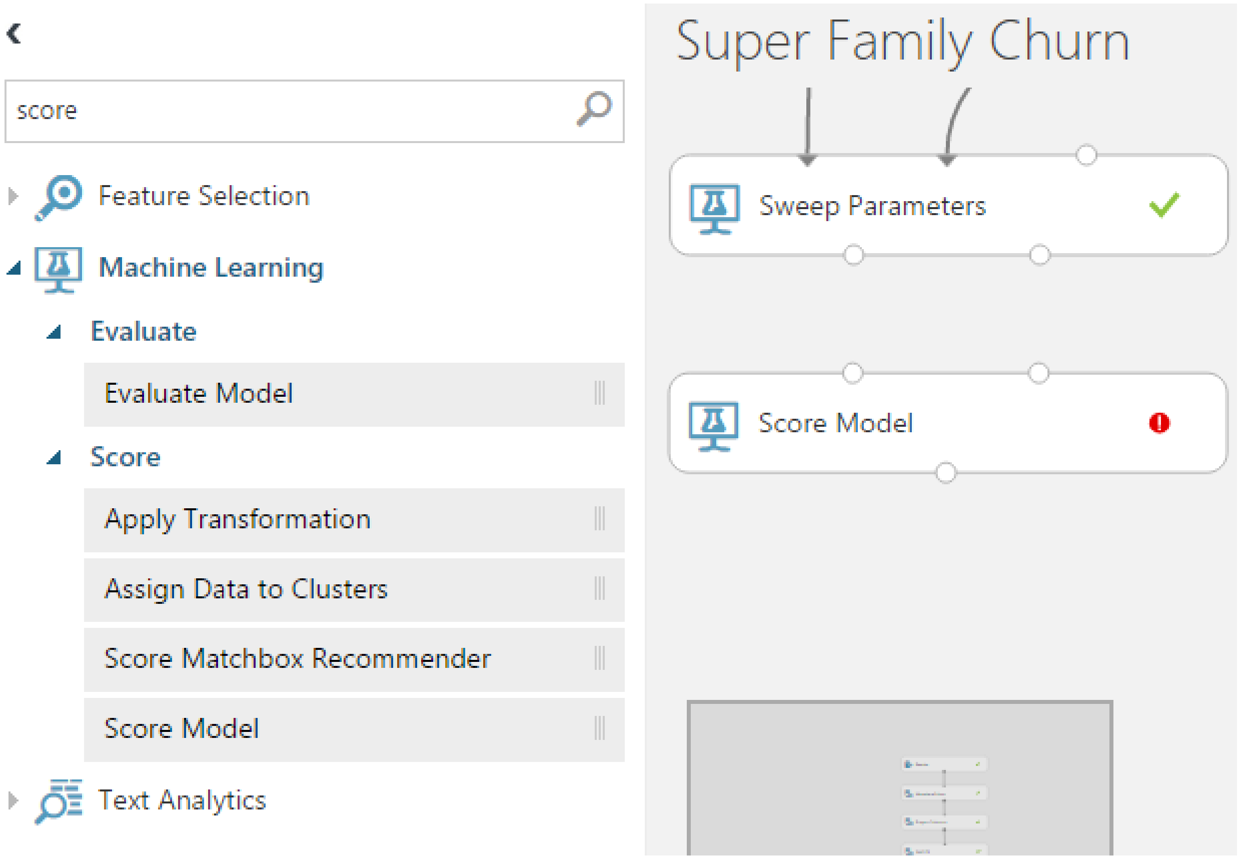
Super Family Churn › Sweep Parameters › Sweep results

rows	columns
20	9

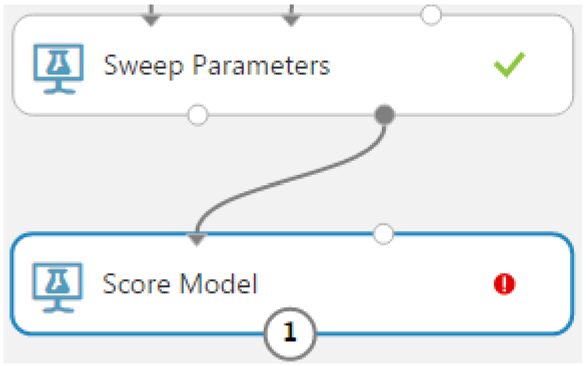
Number of iterations	Lambda	Accuracy	Precision	Recall	F-Score	AUC
99	0.000842	0.673184	0.272727	0.056075	0.093023	0.63287
52	0.005236	0.667598	0.25	0.056075	0.091603	0.629259
10	0.00731	0.667598	0.269231	0.065421	0.105263	0.627881

Once the model has been trained, the next step is to score it. For classification, scoring provides a prediction and the probability of it occurring. (http://bit.ly/1lpX2Ed)

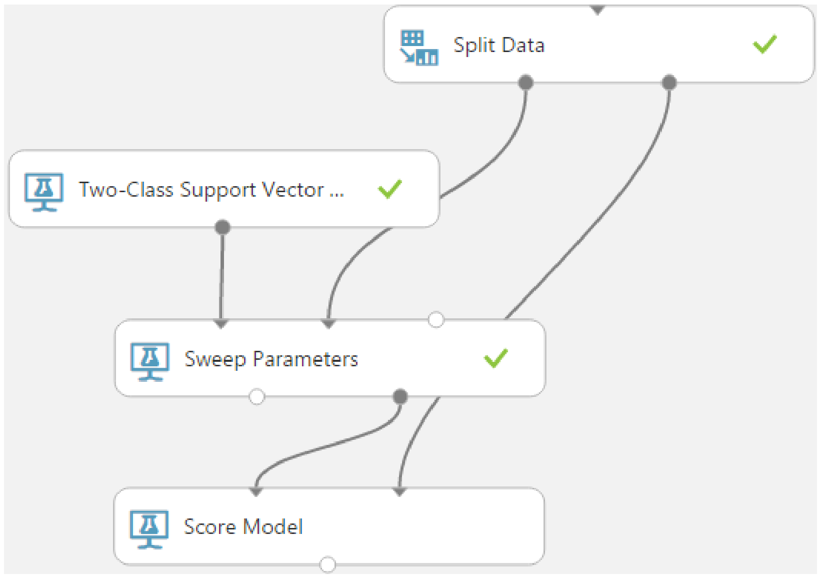
11. In the selection panel, search for **score**. Find the **Score Model** module under *Machine Learning—Score* and drag it into the experiment.



12. Connect the *second* output from **Sweep Parameters** (trained best model) to the *first* input of **Score Model**.

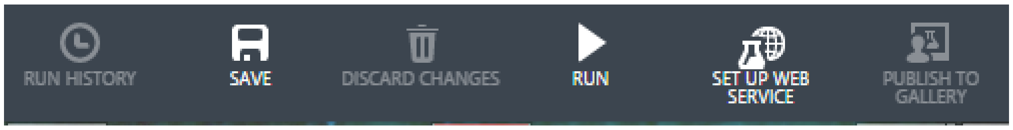


13. Connect the *second* output from Split Data to the *second* input of **Score Model**.

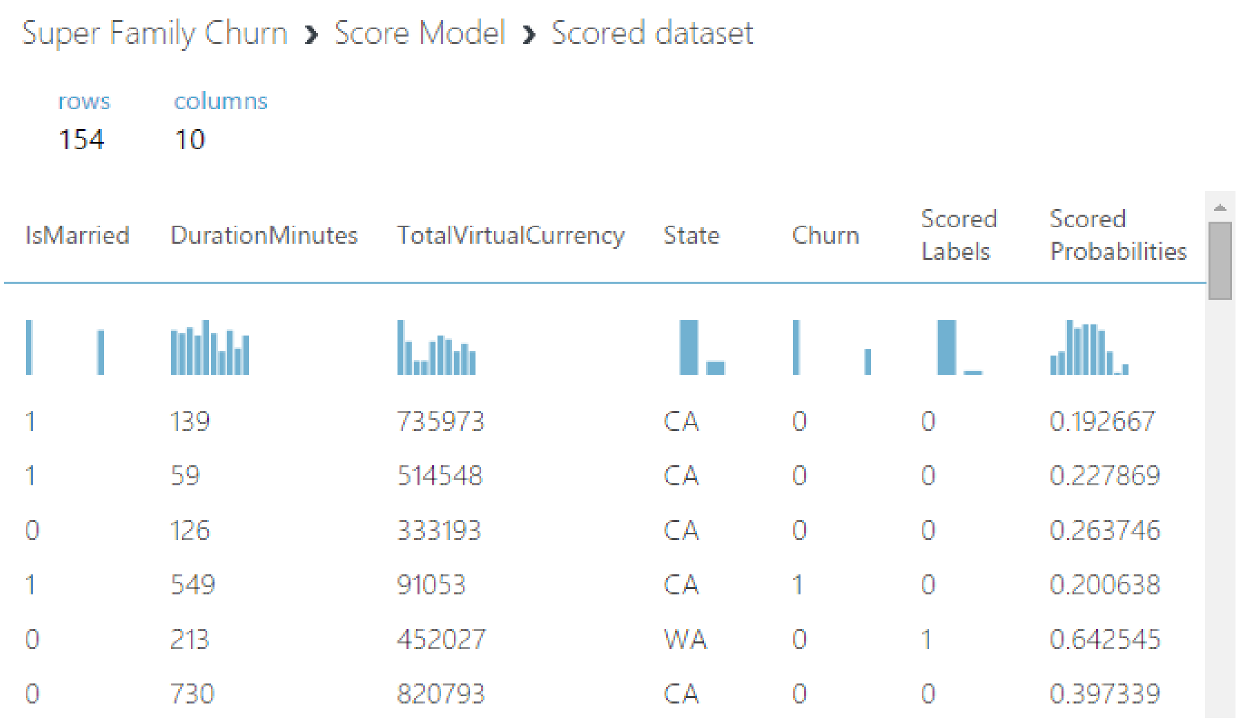


14. Select **Run** in the bottom bar to run the experiment. It will take a minute for the experiment to run. After it finishes, you should see a green check mark in all of the modules.

Note how Azure ML uses the split dataset. The first output is used to train the model, and the second is used to score it with the test dataset.



15. Click on the output circle of **Score Model** and select **Visualize**. Scroll to the right and note the addition of two new columns labeled **Scored Labels** and **Scored Probabilities**. The SVM model predicts the probability that a gamer is going to churn based on the selected features, and anything with a Scored Probability greater than 0.5 will have a Scored Label of 1. Any Scored Probability less than 0.5 will be predicted not to churn and have a Scored Label of 0. This trained model can be used later in time with new data to help predict whether gamers in the new dataset may churn as well. If so, incentives can be provided in-game in real-time to try to keep such gamers from leaving. When done, click **X** in the corner to close the Visualize window.

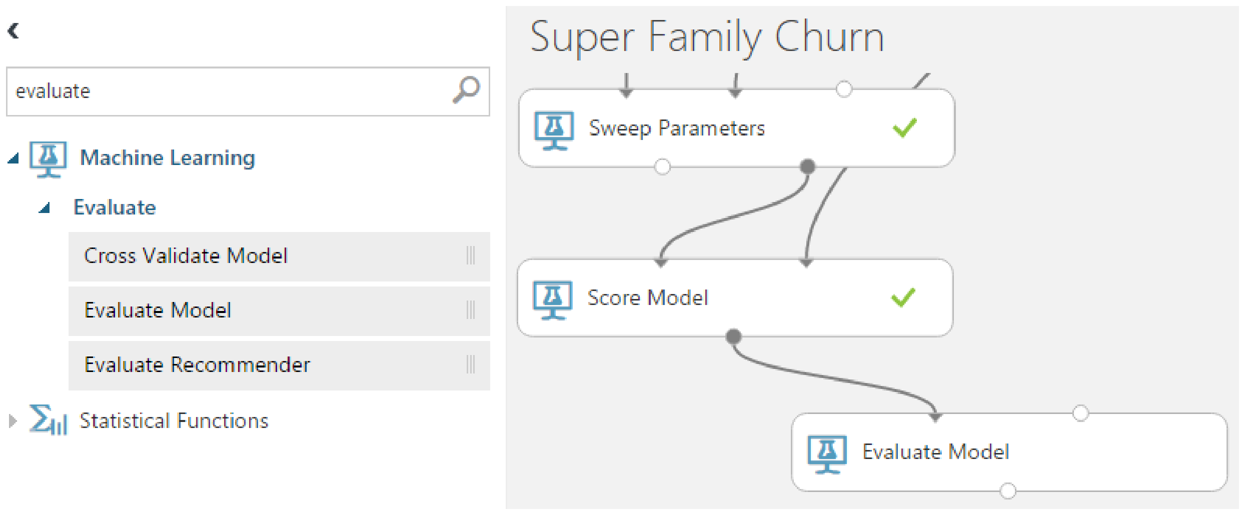


Super Family Churn › Score Model › Scored dataset

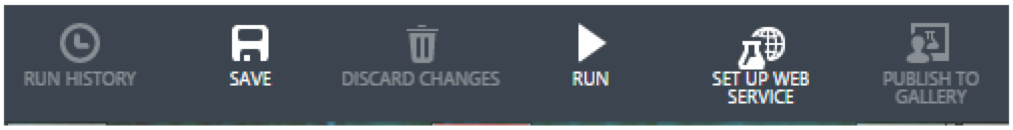
rows	columns
154	10

IsMarried	DurationMinutes	TotalVirtualCurrency	State	Churn	Scored Labels	Scored Probabilities
1	139	735973	CA	0	0	0.192667
1	59	514548	CA	0	0	0.227869
0	126	333193	CA	0	0	0.263746
1	549	91053	CA	1	0	0.200638
0	213	452027	WA	0	1	0.642545
0	730	820793	CA	0	0	0.397339

16. In the selection panel, search for **evaluate**. Find the **Evaluate Model** module under *Machine Learning—Evaluate* and drag it into the experiment. Connect the output of the **Score Model** module to the new **Evaluate Model**.

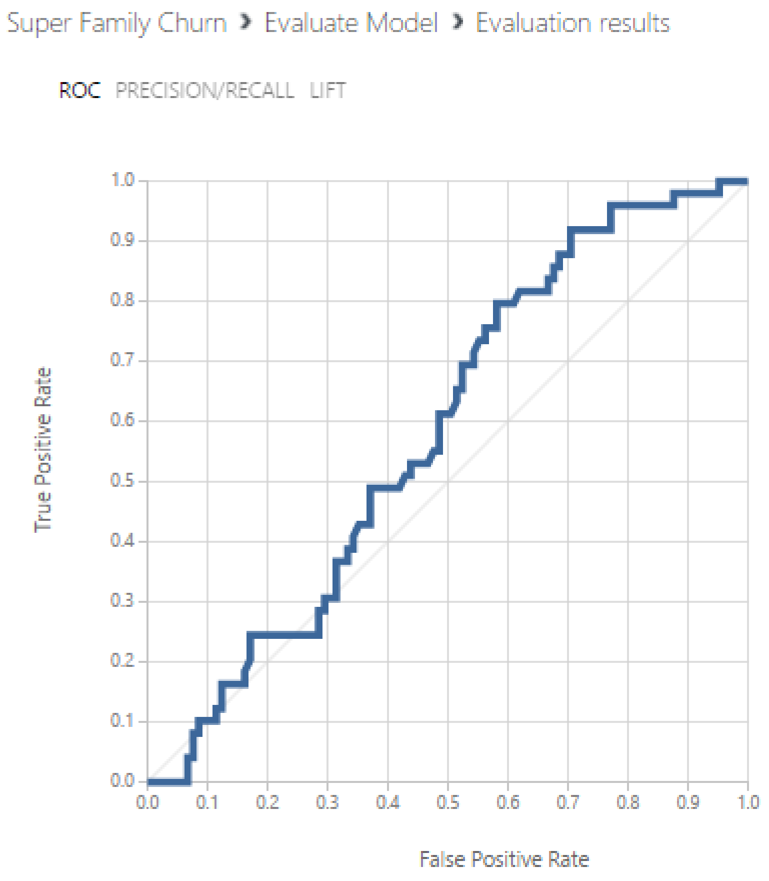


17. Select **Run** in the bottom bar to run the experiment. It will take a minute for the experiment to run. After it finishes, you should see a green check mark in all of the modules.

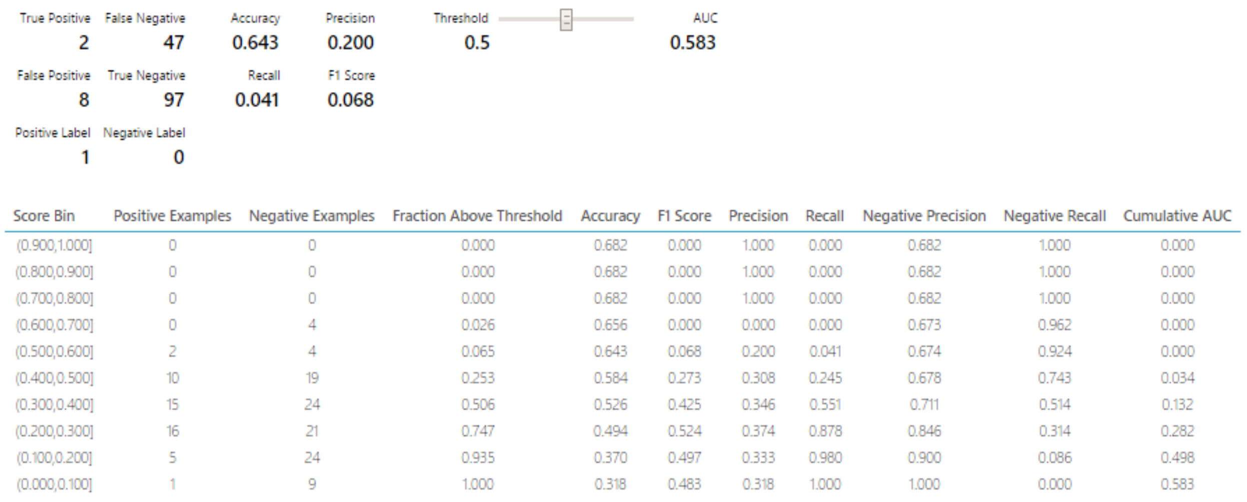


18. Click the output under **Evaluate Model** and select **Visualize** to see the evaluation results. The default chart is *ROC*, but you can select either *Precision/Recall* or *Lift* as well. Scrolling down, you see statistics related to how well the model should perform at different thresholds. Drag the *threshold* slider to see how the accuracy would be affected at different levels.

Evaluating the model provides details about its accuracy. Azure ML uses different methods to determine accuracy such as ROC, recall, and lift. For a more detailed look at how to interpret these charts, see the Azure documentation (http://bit.ly/1SL05By)



A True Positive in this case is a predicted churn when the gamer is likely to churn in reality. A False Positive is a predicted churn when the gamer is *not* likely to churn. Looking at the ROC curve, a larger area under the curve (AUC) reflects higher accuracy. The Precision/Recall and Lift charts are two alternative methods to measure accuracy. For a deeper understanding of ROC and how to interpret the ROC chart, see http://bit.ly/1NJzvEp.



True Positive	False Negative	Accuracy	Precision	Threshold	AUC
2	47	0.643	0.200	0.5	0.583

False Positive	True Negative	Recall	F1 Score
8	97	0.041	0.068

Positive Label	Negative Label
1	0

Score Bin	Positive Examples	Negative Examples	Fraction Above Threshold	Accuracy	F1 Score	Precision	Recall	Negative Precision	Negative Recall	Cumulative AUC
(0.900,1.000]	0	0	0.000	0.682	0.000	1.000	0.000	0.682	1.000	0.000
(0.800,0.900]	0	0	0.000	0.682	0.000	1.000	0.000	0.682	1.000	0.000
(0.700,0.800]	0	0	0.000	0.682	0.000	1.000	0.000	0.682	1.000	0.000
(0.600,0.700]	0	4	0.026	0.656	0.000	0.000	0.000	0.673	0.962	0.000
(0.500,0.600]	2	4	0.065	0.643	0.068	0.200	0.041	0.674	0.924	0.000
(0.400,0.500]	10	19	0.253	0.584	0.273	0.308	0.245	0.678	0.743	0.034
(0.300,0.400]	15	24	0.506	0.526	0.425	0.346	0.551	0.711	0.514	0.132
(0.200,0.300]	16	21	0.747	0.494	0.524	0.374	0.878	0.846	0.314	0.282
(0.100,0.200]	5	24	0.935	0.370	0.497	0.333	0.980	0.900	0.086	0.498
(0.000,0.100]	1	9	1.000	0.318	0.483	0.318	1.000	1.000	0.000	0.583

19. Click **X** in the corner of the evaluation window to close, then click **Save** in the bottom bar to save your experiment.

WANT TO LEARN MORE?

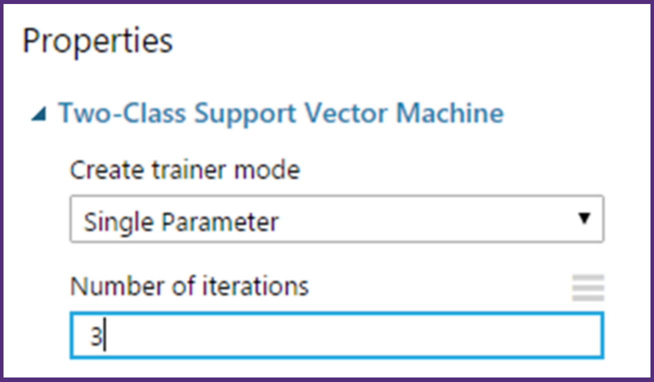
Support Vector Machines (SVM) is only one algorithm among many available in Azure ML Studio. Depending on the goal of your analysis, the data available, and other factors, you may need either *supervised* or *unsupervised* learning. What is the difference between the two? At a high level, supervised learning involves *labeled* data and looks for patterns in what is known. Unsupervised learning helps provide structure to *unlabeled* data.

SVM is used for classification—a supervised learning problem.

Read more about the different types of algorithms and how to choose between them: https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-choice/

ADDITIONAL CHALLENGE

Return to your experiment and click on the **Two-Class Support Vector Machine** module. In the **Properties** window, change the number of iterations from *1* to **3**. Run the experiment again (it may take longer this time), then click the output under **Evaluate Model** and select **Visualize**. Do the *ROC* and *Lift* graphs differ from what you saw with one iteration? Has the accuracy changed?


Properties

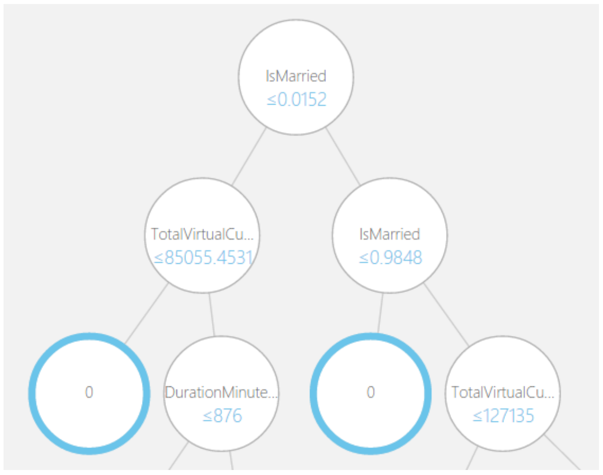
Two-Class Support Vector Machine

Create trainer mode: Single Parameter

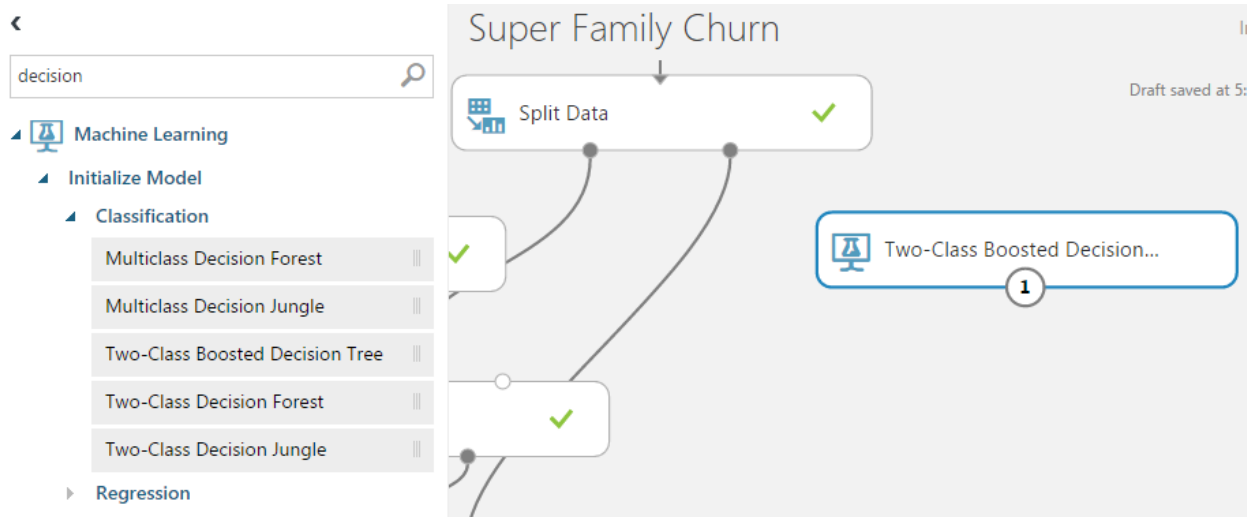
Number of iterations: 3

Activity 3: Train and Score a Model Using Boosted Decision Trees

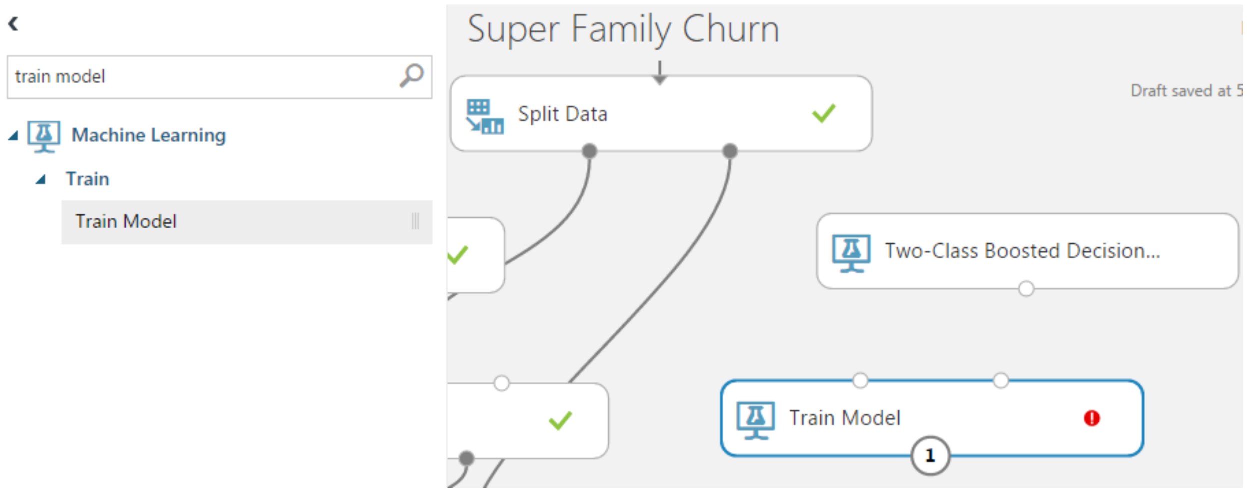
This activity resembles the prior one, but an alternative classification algorithm called *Boosted Decision Trees* is employed instead of SVM. Azure ML allows multiple machine learning algorithms to be trained and scored in the same experiment. A decision tree models how various combinations of features may contribute to a result. *Boosted* decision trees use multiple individual decision trees to arrive at a better prediction. For more technical detail, see http://bit.ly/1ID0rd5.



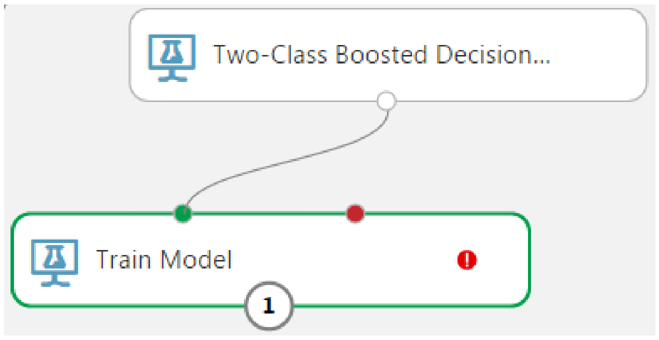
1. In the selection panel, search for **decision**. A few results will appear, and find **Two-Class Boosted Decision Tree** under *Classification*. Drag the module into the experiment, but *do not* attempt to connect the Split Data module with the new Boosted Decision Tree module. Leave all of the default Properties for the Boosted Decision Tree module.


decision
Machine Learning
Initialize Model
Classification
Multiclass Decision Forest
Multiclass Decision Jungle
Two-Class Boosted Decision Tree
Two-Class Decision Forest
Two-Class Decision Jungle
Regression
Super Family Churn
Draft saved at 5:
Split Data
Two-Class Boosted Decision...
1

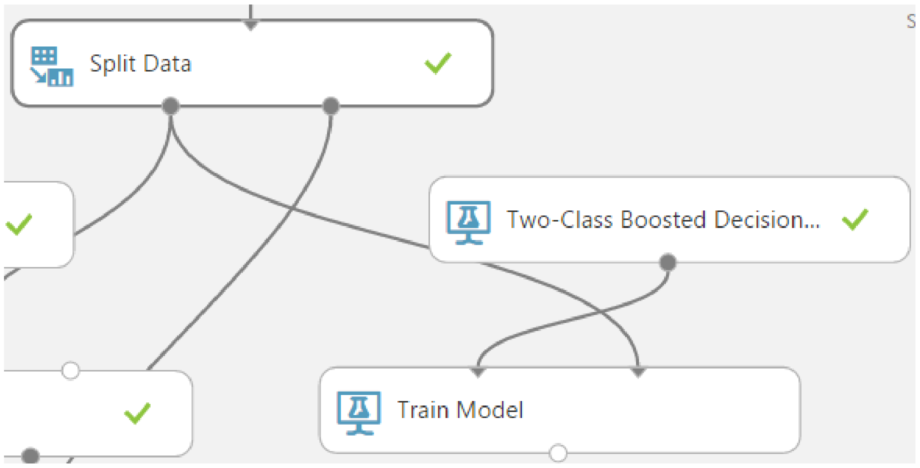
2. In the selection panel, search for **train model**. Drag the **Train Model** module into the experiment.



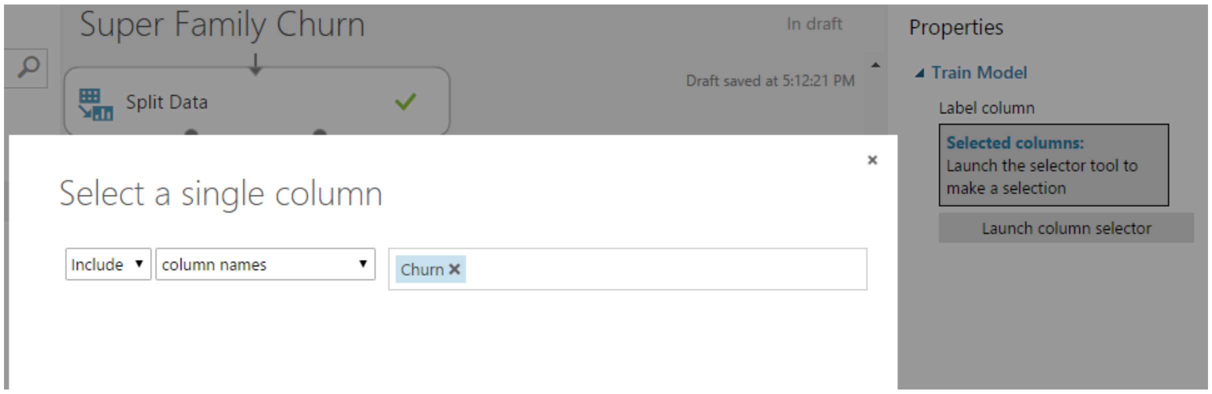
3. Connect the untrained model output from **Two-Class Boosted Decision Tree** to the leftmost input of **Train Model.** Note as you drag the arrow how the appropriate input is highlighted in green while the other input is red. Only the appropriate connection is allowed.



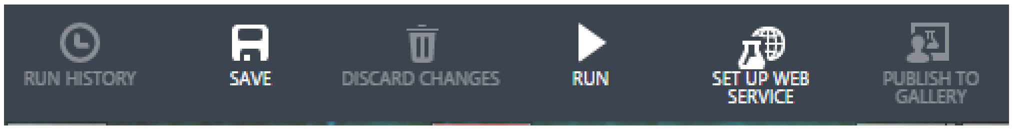
4. Drag the leftmost output from the **Split Data** module to the second input of **Train Model**.



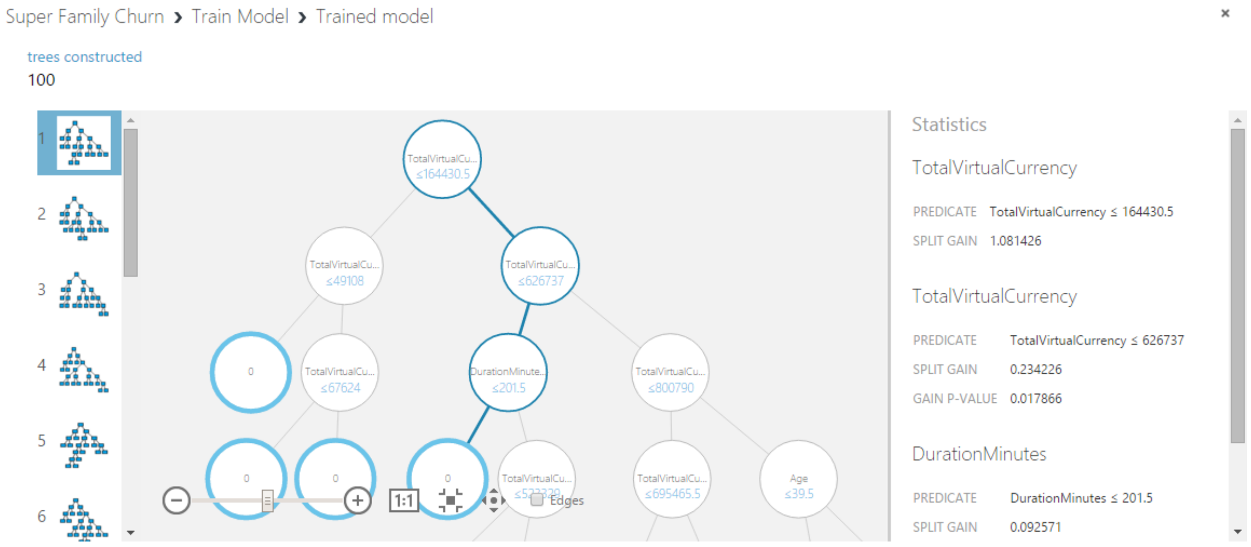
5. Select the **Train Model** module, and in the Properties panel to the right, select **Launch column selector**. Select the **Churn** column and click **OK** when done.



6. Select **Run** in the bottom bar to run the experiment. After it finishes, you should see a green check mark in all of the modules.



7. Select the **Train Model** module, click on the output circle, and select **Visualize**. Note that there are one hundred decision trees that correspond with the defaults for the Boosted Decision Trees module. The algorithm is labeled “boosted” since it considers the results from one hundred trees instead of only using one decision tree. You can explore each of the trees visually. In addition, you can see the weight given to different features within each tree by clicking on each of the leaves labeled 0. Click **X** in the corner when done.



Statistics

TotalVirtualCurrency

PREDICATE	TotalVirtualCurrency ≤ 164430.5
SPLIT GAIN	1.081426

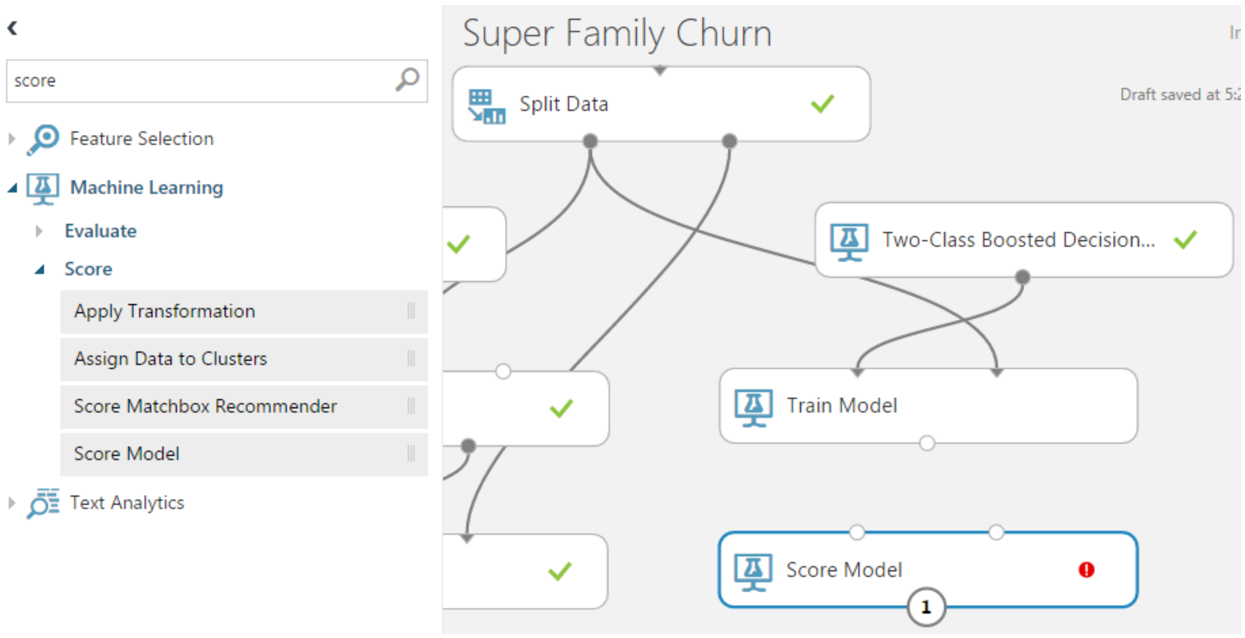
TotalVirtualCurrency

PREDICATE	TotalVirtualCurrency ≤ 626737
SPLIT GAIN	0.234226
GAIN P-VALUE	0.017866

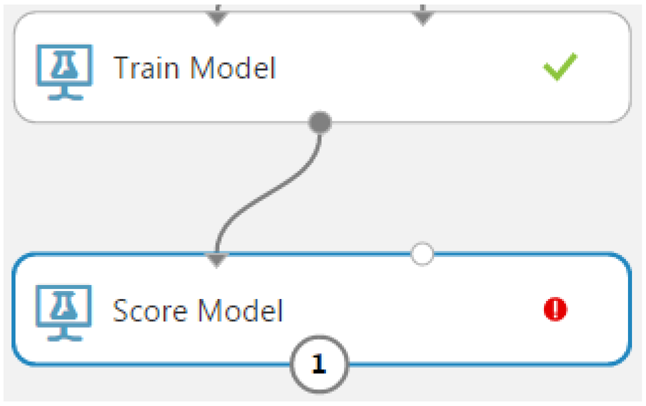
DurationMinutes

PREDICATE	DurationMinutes ≤ 201.5
SPLIT GAIN	0.092571

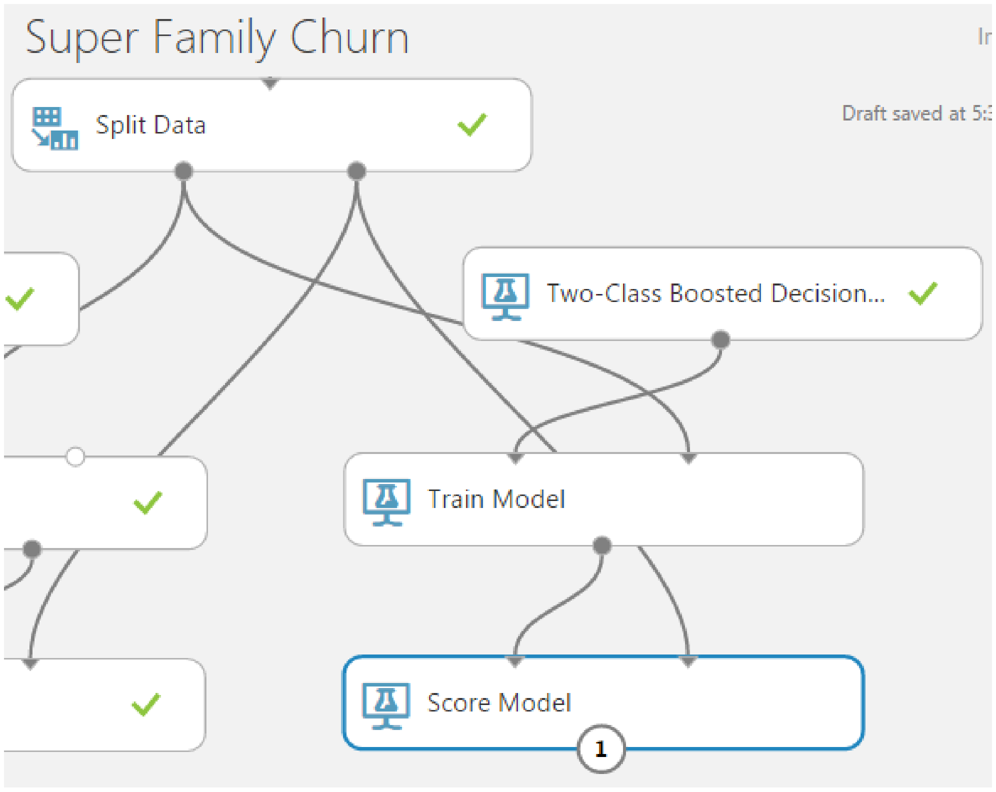
8. In the selection panel, search for **score**. Find the **Score Model** module under *Machine Learning—Score* and drag it into the experiment. This creates a second module labeled **Score Model** in the experiment, but it is on a separate path from the one used for the SVM in the earlier activity.



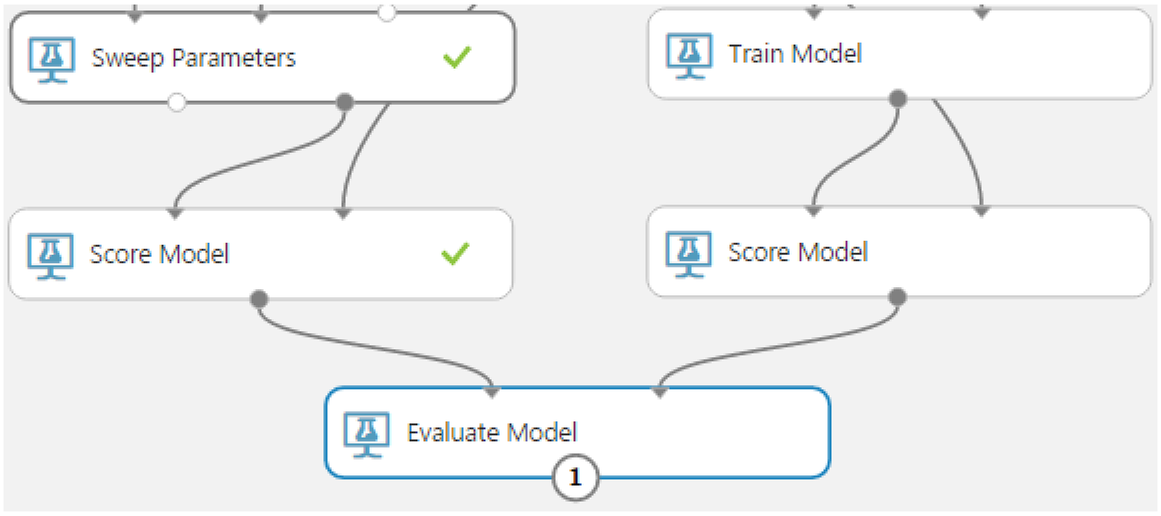
9. Connect the output from **Train Model** to the *first* input of **Score Model** in the Boosted Decision Tree path.



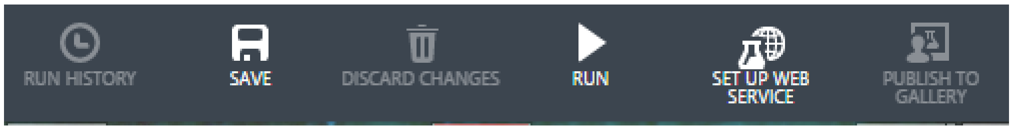
10. Connect the *second* output from Split Data to the *second* input of **Score Model** in the Boosted Decision Tree path.



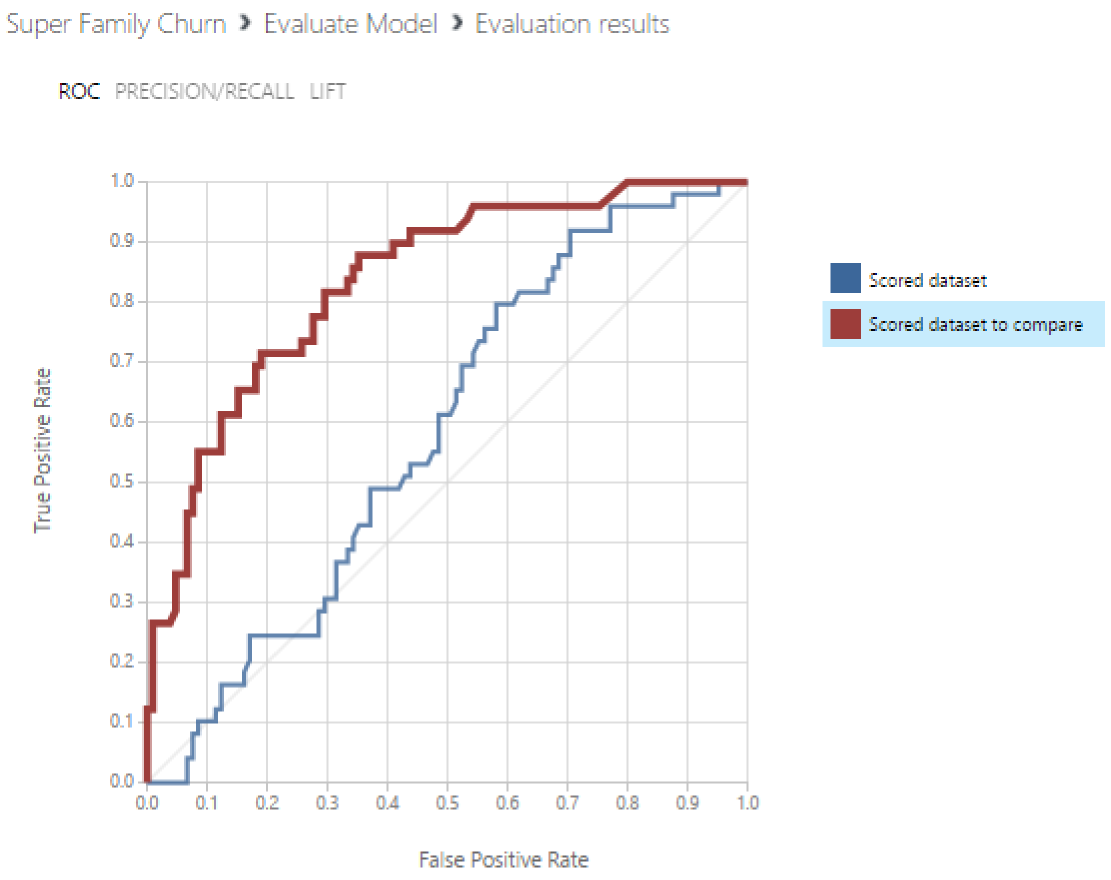
11. The two paths from the separate SVM and Boosted Decision Tree algorithms will now converge at the **Evaluate Model** module. Connect the output of the **Score Model** in the Boosted Decision Tree path to the **Evaluate Model** module.



12. Select **Run** in the bottom bar to run the experiment. After it finishes, you should see a green check mark in all of the modules.



13. Click the output under **Evaluate Model** and select **Visualize** to see the evaluation results. Since there are two inputs, you can see both results together. The SVM model should appear as the *Scored dataset*, and the new Boosted Decision Tree model should appear as the *Scored dataset to compare*. In the screenshot below, the *ROC* curve for boosted decision trees shows that it performed much better than the SVM model in this case. The area under the curve (AUC) is closer to 1.0 than the corresponding SVM curve.



14. Click **X** in the corner of the evaluation window to close, then click **Save** in the bottom bar to save your experiment.

WANT TO LEARN MORE?

One of the primary goals of churn analysis is reducing the misclassification rate, or the number of false positives. In Activity 3, you saw how changing the classification algorithm resulted in a better prediction in this case. Why not always use Boosted Decision Trees instead of Support Vector Machines? A given model may be better in certain circumstances, but you learned in the *Want to Learn More?* section in Activity 2 how to choose between models to support different scenarios. In Azure ML, you can add many models to the experiment and pick the most appropriate one for your data. In addition, as you become more involved in machine learning, your algorithm parameter choices also play a large role. In this experiment, SVM performed more poorly when the number of iterations was set to one, while the number of decision trees was set to one hundred!

Read more about how to optimize parameters in Azure ML:
https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-parameters-optimize/

ADDITIONAL CHALLENGE

1. Try to raise the accuracy of both SVM and Boosted Decision Trees by altering parameters. Keep in mind that there can be a large tradeoff between performance (time and compute resources) versus accuracy. Especially when trying to predict churn in near-real time, lower accuracy may be acceptable if it means the difference between milliseconds and minutes.

2. Select a third classification algorithm using the Cheat Sheet (link below). Add it to your experiment, then train, score, and evaluate it. How does it compare to the accuracy of SVM and Boosted Decision Trees with the *Super Family* training data?
   https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-cheat-sheet/

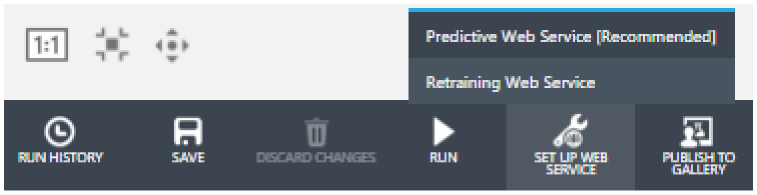
Activity 4: Publish Your Churn Experiment as a Web Service

In prior activities, you prepared a sample churn experiment that trained, scored, and evaluated Support Vector Machine and Boosted Decision Tree models. It is beneficial to see machine learning in action in the Azure ML Studio, but how can an app such as your future *Super Family* game take advantage of the churn model? In order to provide access to your model's capabilities, Azure ML allows you to create a web service that outside applications invoke.

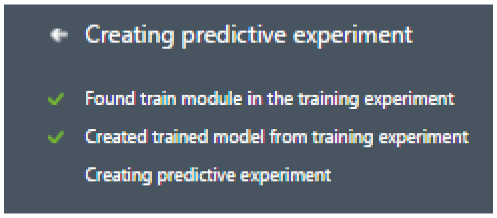
1. Open your *Super Family Churn* experiment in Azure ML Studio.

2. Click on the **Train Model** module in the Boosted Decision Tree path.

3. In the bottom bar, hover over **Set Up Web Service**, then select **Predictive Web Service**.
    a. NOTE: If you do not have the option to setup a web service, you may need to **Run** your experiment again.
    b. NOTE: If you see a small window appear that prompts you to select a train model, click **Finish**, verify that you have selected the **Train Model** module associated with Boosted Decision Trees as directed in step 2, then repeat this step.



4. A progress bar should appear at the bottom, and the predictive experiment should be available after a few seconds.



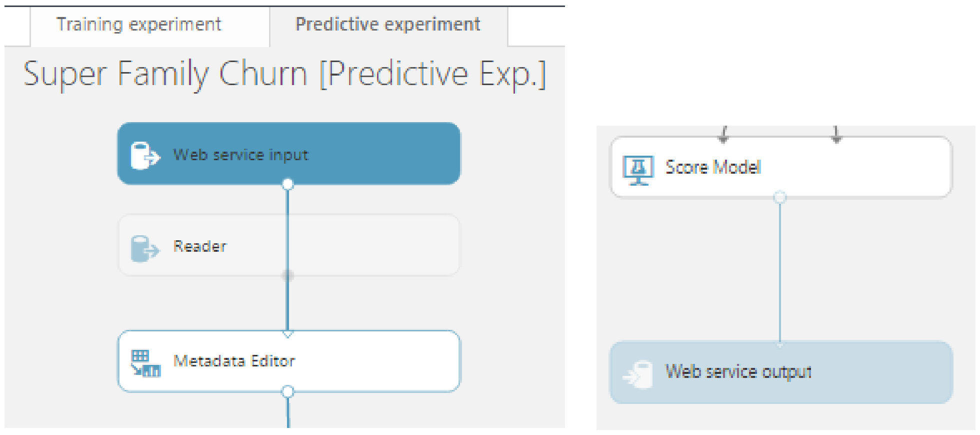
Creating predictive experiment

✓ Found train module in the training experiment

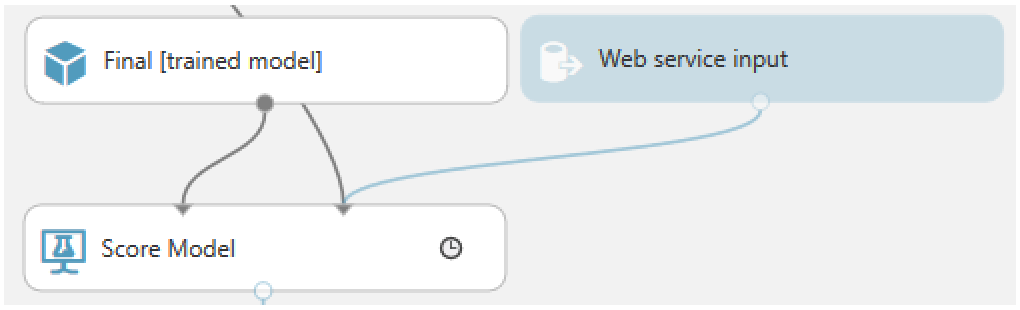
✓ Created trained model from training experiment

Creating predictive experiment

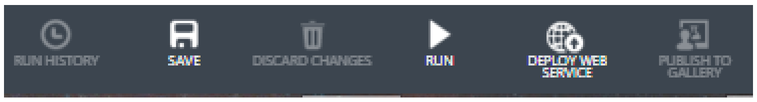
5. Note how two tabs now appear at the top: one for the original *Training experiment*, and one that contains a modified *Predictive* experiment. At the top of the predictive experiment, you should see a new **Web service input** module. At the bottom, you should see a **Web service output**.



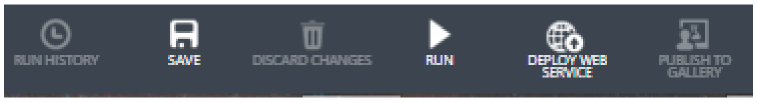
6. The default **Web service input** incorporates *all* features and is linked to the Metadata Editor module, but you are only concerned with the subset of features from the Project Columns module. Click on the arrow output from the **Web service input** module and hit the **Delete** key. Drag an arrow from the output of **Web service input** to the second input of the **Score Model** module. This will allow your web app to bypass the earlier data processing steps.



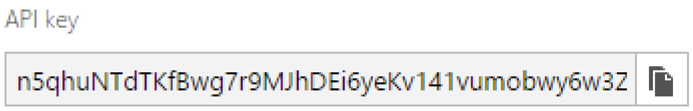
7. In the bottom bar, select the option to **Run** the predictive experiment.



8. After the experiment runs, select the option to **Deploy Web Service** in the bottom bar.



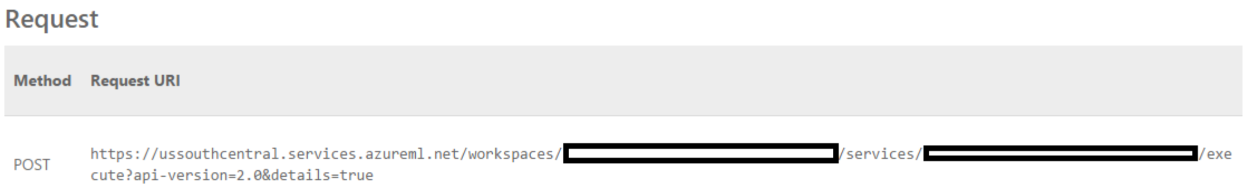
9. Once deployed, a new window appears that displays an **API Key** as well as **Default Endpoint** for using the predictive web service. **Copy and retain the API Key** since this will later be used in Activity 5 to provide authentication to the service.



API key

n5qhuNTdTKfBwg7r9MJhDEi6yeKv141vumobwy6w3Z

10. Click the **REQUEST/RESPONSE** link under *Default Endpoint*, and a new tab labeled *Request Response API Documentation* will open. **Copy and retain the Request URI**, but omit the *&details=true* portion. This address is what your Super Family web app in Activity 5 will use to make requests.

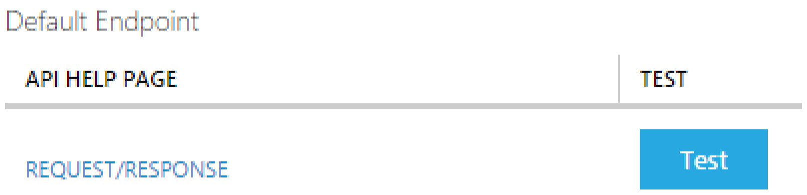


Request

Method	Request URI
POST	https://ussouthcentral.services.azureml.net/workspaces/[redacted]/services/[redacted]/execute?api-version=2.0&details=true

11. Optionally, test the service by clicking on the **Test** button next to *Request/Response*. A new window appears, and you can enter sample data that imitates what your actual data may contain.

The Request/Response and Batch Execution formats are two ways to operationalize your Azure ML experiment. While not covered here, Batch Execution allows you to create jobs that handle multiple predictions.



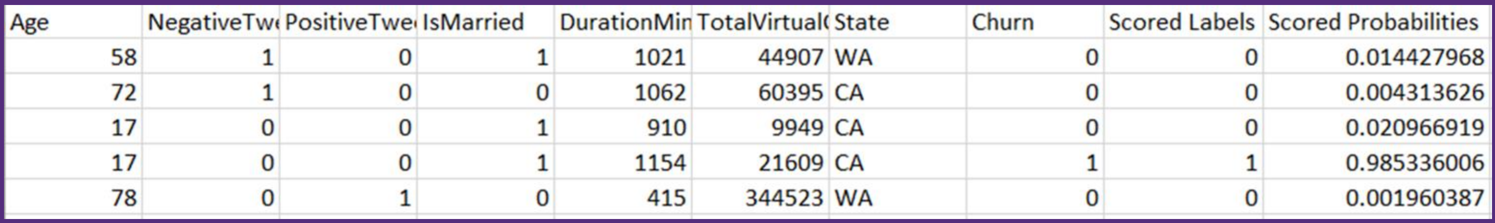
WANT TO LEARN MORE?

A web service is the key way that your *Super Family* app will interact with your experiment. Operationalizing your experiment by exposing it as a service takes your predictive capability and extends it to the outside world.

Read more about web services in Azure ML: https://azure.microsoft.com/en-us/documentation/articles/machine-learning-model-progression-experiment-to-web-service/

ADDITIONAL CHALLENGE

The *Super Family* ASP.NET web app is only one way to interact with your web service. You can also use the service with any application that makes HTTP requests—such as Microsoft Excel. If you looked carefully at the *Default Endpoint* section on the web service *Dashboard*, there was a link to *Download Excel Workbook* next to the Test button referenced in Step 9. Try downloading the workbook to see how you can test the web service in Excel. If you create sample data in Excel, can you score that data by making a request to your web service?



Age	NegativeTw	PositiveTwe	IsMarried	DurationMin	TotalVirtual	State	Churn	Scored Labels	Scored Probabilities
58	1	0	1	1021	44907	WA	0	0	0.014427968
72	1	0	0	1062	60395	CA	0	0	0.004313626
17	0	0	1	910	9949	CA	0	0	0.020966919
17	0	0	1	1154	21609	CA	1	1	0.985336006
78	0	1	0	415	344523	WA	0	0	0.001960387

Activity 5: Create Your *Super Family* Game

In this activity, you will utilize the predictive web service that you published in the Azure ML Studio in the *Super Family* ASP.NET web app. While it is only a very basic game, it illustrates how to request a churn score from your service as well as how to use that response to potentially intervene and change the resulting app behavior. In the latter portion of this activity, you can review at a high level how this code works.

Screenshots in both Activity 5 and Activity 6 reference the user experience in Visual Studio Community and Azure SDK 2.8.1.

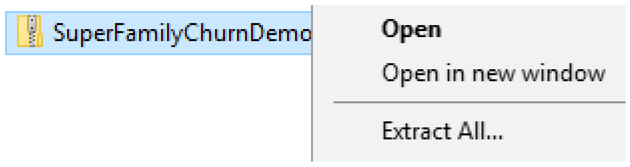
1. Verify that you have Visual Studio 2015 and the Azure SDK version 2.8.1 (or greater) installed.

   You can download **Visual Studio Community** for free at https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx

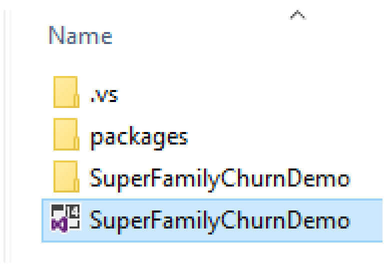
   You can install the Azure SDK using the **Microsoft Web Platform Installer**.

2. In a browser, navigate to the following link to download a zip file that contains the Super Family web app solution. You will modify this version prior to publishing to a website. https://github.com/ImagineCupGame/SuperFamilyAzureML/raw/master/SuperFamilyChurnDemo.zip (alternatively http://bit.ly/1I3LIrG)

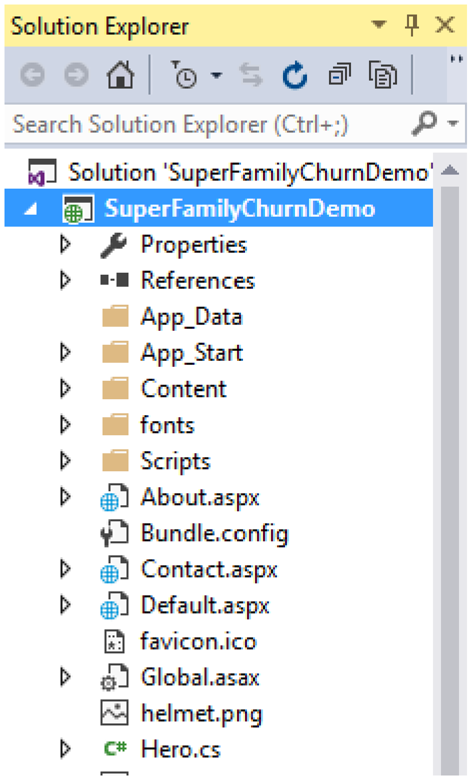
3. After the *SuperFamilyChurnDemo.zip* download completes, extract the files by right-clicking on the zip file and selecting **Extract All**. Extract to an appropriate location.



4. In the extracted folder, open the *SuperFamilyChurnDemo* solution in Visual Studio by double-clicking it. Alternatively, open Visual Studio, select **Open Project**, and browse for the solution file.



5. The *Solution Explorer* pane displays files related to the ASP.NET web app.



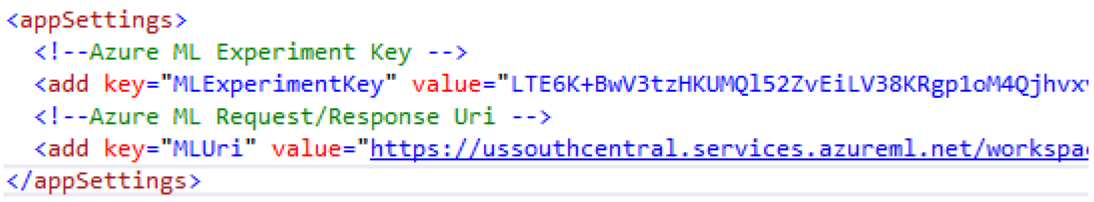
6. In the *Solution Explorer*, scroll to find the **Web.config** file, then double-click to open the file. This file contains custom settings for the web application. The last section of the file is *appSettings*, which has two key/value pairs for *MLExperimentKey* and *MLUri*.

```
<appSettings>
  <!--Azure ML Experiment Key -->
  <add key="MLExperimentKey" value="REPLACE CONTENT WITH AZURE ML KEY"/>
  <!--Azure ML Request/Response Uri -->
  <add key="MLUri" value="REPLACE CONTENT WITH AZURE ML URI"/>
</appSettings>
```

7. Replace the **value** for *MLExperimentKey* using the **API Key** obtained from the Azure ML web service in Activity 4. The value should maintain the quotation marks.

```
<appSettings>
  <!--Azure ML Experiment Key -->
  <add key="MLExperimentKey" value="LTE6K+BwV3tzHKUMQl52ZvEiLV38
  <!--Azure ML Request/Response Uri -->
  <add key="MLUri" value="REPLACE CONTENT WITH AZURE ML URI"/>
</appSettings>
```

8. Replace the **value** for *MLUri* using the **Request Uri** obtained from the Azure ML web service in Activity 4. Verify that *&details=true* has been omitted from the end of the URI since including it may cause issues later during the build process. The value should maintain the quotation marks.

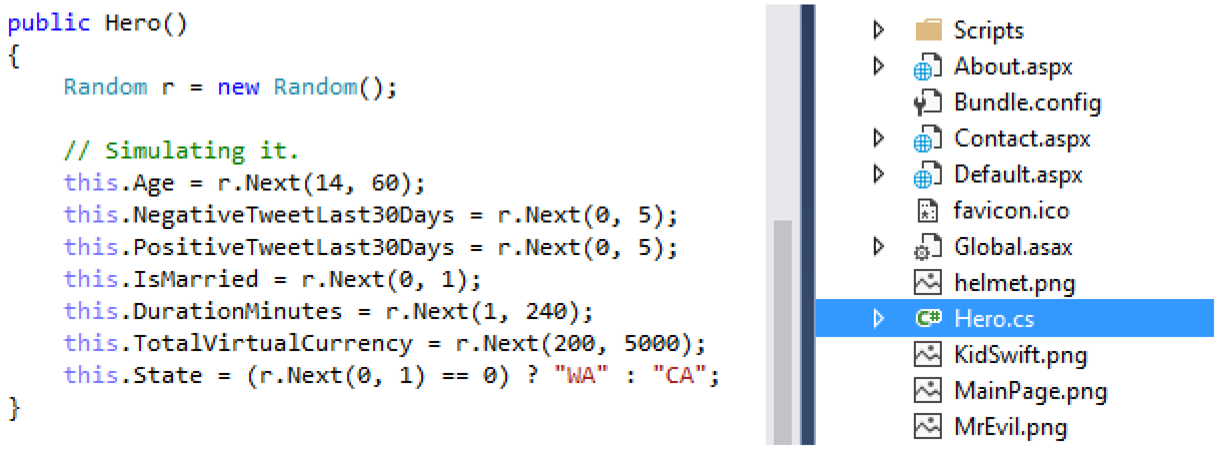


```
<appSettings>
  <!--Azure ML Experiment Key -->
  <add key="MLExperimentKey" value="LTE6K+BwV3tzHKUMQl52ZvEiLV38KRgp1oM4Qjhvx
  <!--Azure ML Request/Response Uri -->
  <add key="MLUri" value="https://ussouthcentral.services.azureml.net/workspa
</appSettings>
```

9. Click the **Save** icon or use **Ctrl+S** to save the Web.config file after making the changes.

10. Find the **Hero.cs** file in *Solution Explorer* and double-click to open it. Do not make any changes to this file.

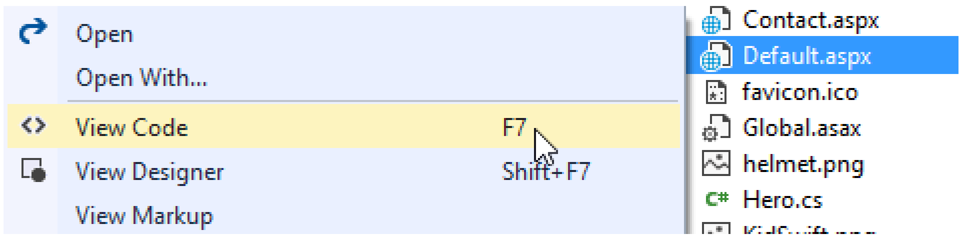
    The Hero class contains code that creates random data for churn prediction. These random values for *Age*, *NegativeTweetLast30Days, etc*. correspond with the feature selection from Azure Machine Learning. In a real game, all of this data would be based on actual players, but for your demo Super Family game, this simulated data will be sent to the Azure ML web service to predict churn.



```
public Hero()
{
    Random r = new Random();

    // Simulating it.
    this.Age = r.Next(14, 60);
    this.NegativeTweetLast30Days = r.Next(0, 5);
    this.PositiveTweetLast30Days = r.Next(0, 5);
    this.IsMarried = r.Next(0, 1);
    this.DurationMinutes = r.Next(1, 240);
    this.TotalVirtualCurrency = r.Next(200, 5000);
    this.State = (r.Next(0, 1) == 0) ? "WA" : "CA";
}
```

11. Find the **Default.aspx** file in *Solution Explorer*, right-click and select **View Code** (do not double-click to open the file since that will open the *markup* and not the *code* view). Do not make any changes to this file.



12. Find the **InvokeRequestResponseService** method in the code. Note the variables *mlUri* and *apiKey* that reference the *MLUri* and *MLExperimentKey* that you changed earlier in the appSettings section of the Web.config file.

```
protected async Task InvokeRequestResponseService(Hero hero)
{

    string mlUri = ConfigurationManager.AppSettings["MLUri"]; ;

    string apiKey = ConfigurationManager.AppSettings["MLExperimentKey"];
```

It is not necessary to replace the URI or Key in Default.aspx.cs since it already utilizes the values in Web.config. If you run into errors surrounding Web.config during the Build or Publish process, first verify that you have removed "&details=true" from the URI so that it ends with "version=2.0". If you still encounter issues, hardcoding the values in Default.aspx.cs may be another workaround.

This portion of code prepares the input for the Azure ML web service request using the simulated Hero values:

```
var scoreRequest = new
{

    Inputs = new Dictionary<string, StringTable>() {
        {
            "input1",
            new StringTable()
            {
                ColumnNames = new string[] {"Age", "NegativeTweetLast30Days",
                Values = new string[,] {  { hero.Age.ToString(), hero.Negative
            }
        },
```

This portion of code specifies the address of your Azure ML web service using your Uri (request/response endpoint) and authenticates using your Key:

```
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);
client.BaseAddress = new Uri(mlUri);
```

This portion of code makes the request:

```
response = await client.PostAsJsonAsync("", scoreRequest).ConfigureAwait(false);
```

Once your app receives a response from the web service, this portion of code interprets the response. If the response indicates the probability of churn above the stated threshold, the *churn* variable is assigned the value of “1” (true).

```
if (response.IsSuccessStatusCode)
{
    string json = await response.Content.ReadAsStringAsync().ConfigureAwait(false);

    dynamic jsonDe = JsonConvert.DeserializeObject(json);
    JArray items = jsonDe.Results.output1.value.Values[0];

    int label = Int32.Parse(items[8].ToString());
    double scoreProb = Double.Parse(items[9].ToString());

    //  Indicate whether the user churn
    if (label == 1 || scoreProb > 0.1)
        this.churn = 1;
    else
        this.churn = 0;

}
```

In the **CreateHero** method found elsewhere in the *Default.aspx.cs* file, this churn value influences game behavior and reflects intervention to try to prevent a player from churning. If the churn prediction is “1” (true), the game displays a new helmet and a congratulatory message to the player. If the churn prediction is false, the game displays a message that the monster is too powerful and that the player has lost.

```
if (this.churn == 1)
{
    LiteralSpecial.Text = "Congratulations, you found the ultimate helmet! <BR> Using this, you won the fight!";
    ImageSpecial.Visible = true;
}
else
{
    LiteralSpecial.Text = "The monster is too powerful. You lost!";
    ImageSpecial.Visible = false;

}
```

WANT TO LEARN MORE?

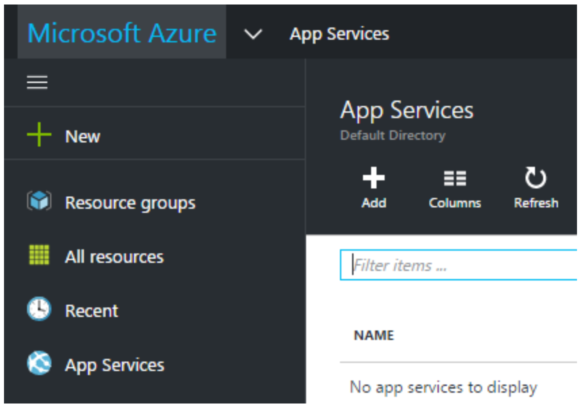
In the *Super Family* web application, the game code was provided for you. You replaced some default values with your own and reviewed the portion of the code that consumed the Azure ML web service in the *InvokeRequestResponseService* method. If you want to learn more about the Request-Response Service and get a deeper understanding of what happens when the service is invoked, more information is available in the Azure documentation.

Read more about consuming web services:
https://azure.microsoft.com/en-us/documentation/articles/machine-learning-consume-web-services/

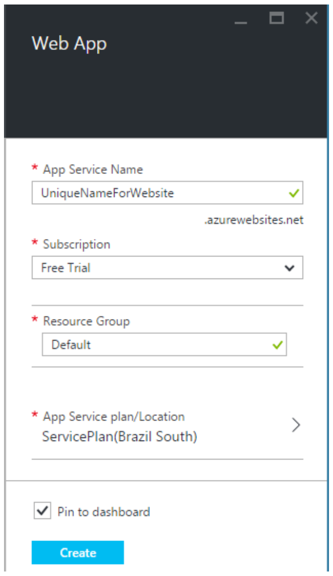
Activity 6: Publish and Play your *Super Family* Game

After changing the web application code to reference your own Azure Machine Learning web service, you are ready to publish your *Super Family* game to Azure App Service. This activity covers the steps required to move your app from the Visual Studio environment to Azure so that you can play the game online as your own Azure Web App.

1. In a browser, navigate to **https://portal.azure.com/#create/Microsoft.WebSite** (alternatively, go to https://portal.azure.com, select **App Services** and then click **Add**.



2. Enter a name for your new web app in **App Service Name**. This name will be included in a URL and must be unique. Verify that a Subscription, Resource Group, and Location are selected, then click **Create**. You should see a notification stating that Azure is "Deploying Web App".


Web App

* App Service Name: UniqueNameForWebsite .azurewebsites.net

* Subscription: Free Trial

* Resource Group: Default

* App Service plan/Location: ServicePlan(Brazil South)

✓ Pin to dashboard

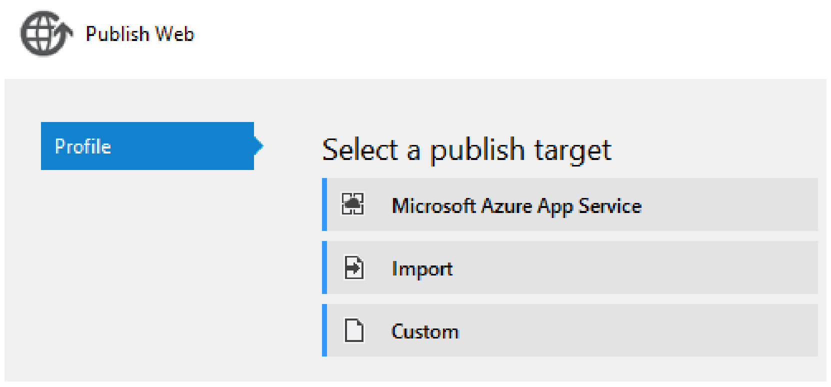
Create

It is possible to create a new web app directly in Visual Studio. If you are using a new Azure subscription and do not have an existing resource group setup, however, it may cause Visual Studio to hang. To account for this behavior, these steps use the Azure Portal to create the new web app.

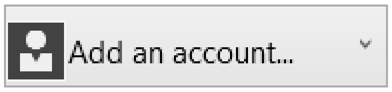
3. Once the web app has been deployed, return to Visual Studio and open the **SuperFamilyChurnDemo** project from Activity 5 if needed.

4. Right-click on the **SuperFamilyChurnDemo** project in the Solution Explorer, then select **Publish**.

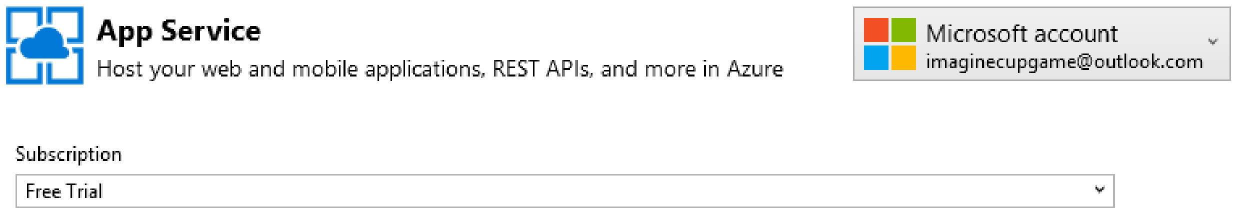
5. On the *Profile* screen, select **Microsoft Azure App Service**.



6. If you have never signed into Visual Studio with your Microsoft account, you will see an option to **Add an account** in the top right corner. If you need to add your account, click the button to login.



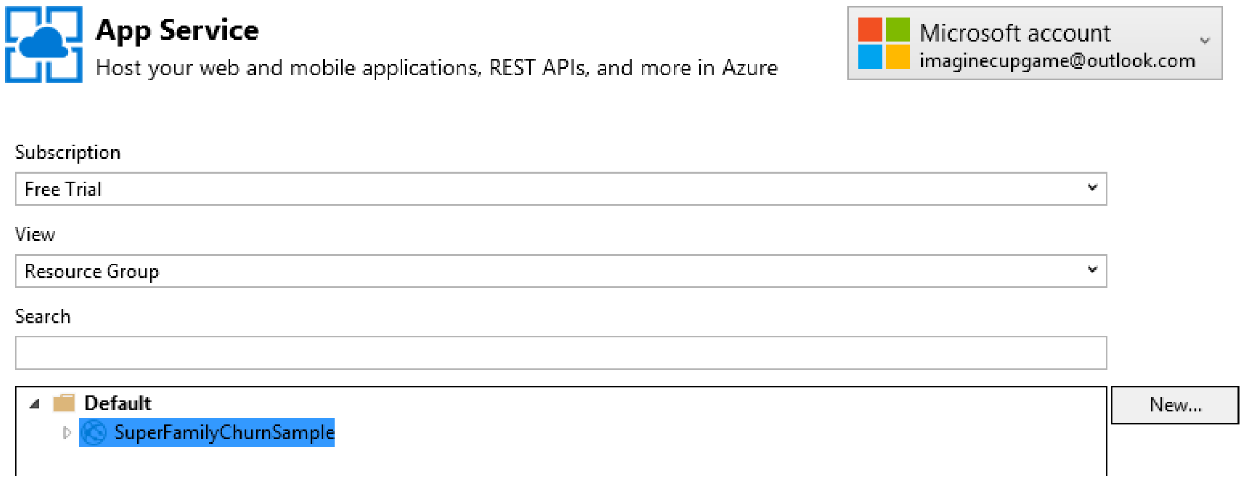
Otherwise, you should see your account listed with your Azure subscription.


App Service
Host your web and mobile applications, REST APIs, and more in Azure
Microsoft account
imaginecupgame@outlook.com

Subscription
Free Trial

7. Expand the **Default** resource group, and you should see the name of the web app that you created in the Azure Portal. NOTE: If you had created your web app under an alternative group name in the Azure Portal, you should expand that group instead.

8. Select the web app under the **Default** resource group and click **OK**.



App Service

Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account
imaginecupgame@outlook.com

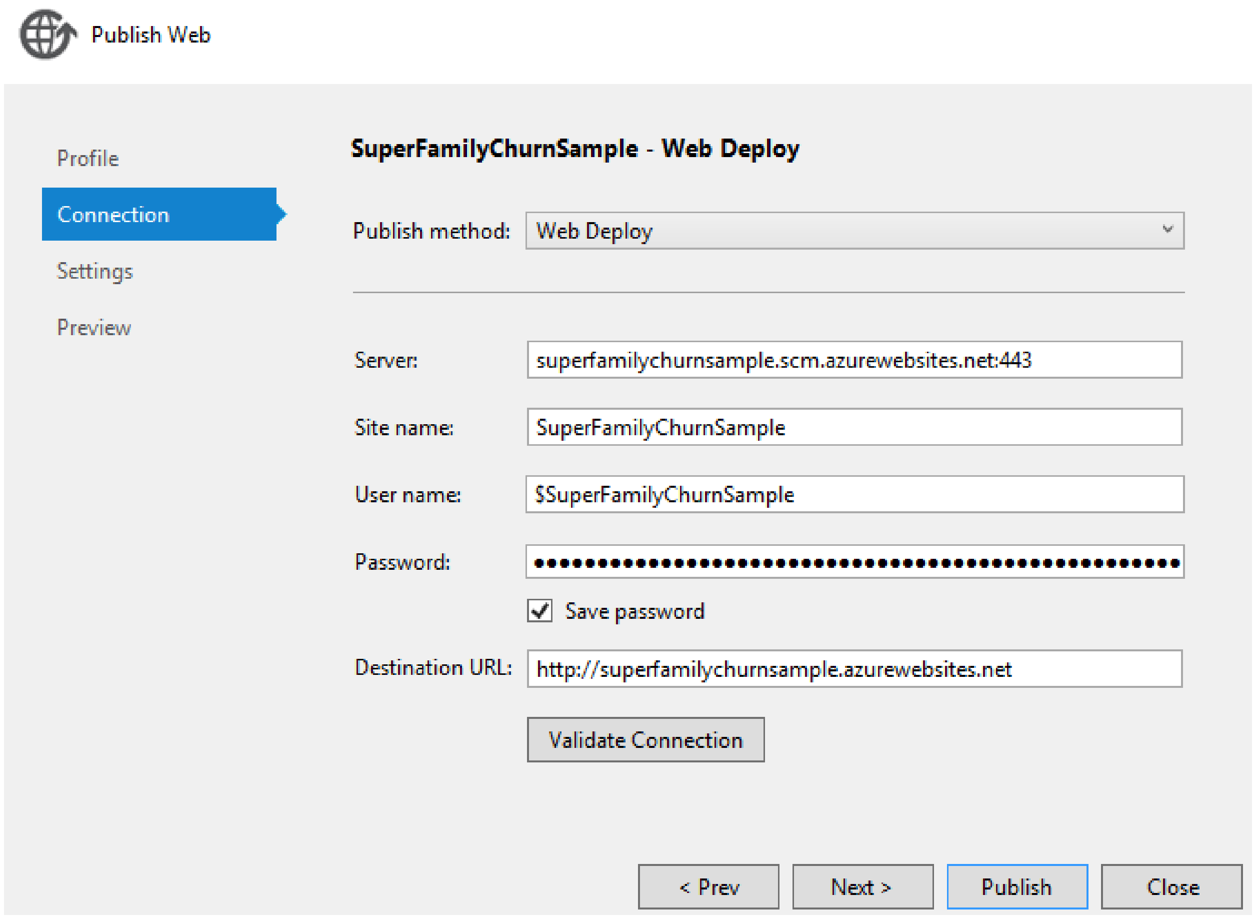
Subscription: Free Trial

View: Resource Group

Search:

Default
SuperFamilyChurnSample

9. On the *Connection* screen, you will see your web app Server, Site name, and Destination URL. Take note of the *Destination URL* since that will be used to access your website later. Click **Publish** to publish your web app to Azure.



Publish Web

SuperFamilyChurnSample - Web Deploy

Publish method: Web Deploy

Server: superfamilychurnsample.scm.azurewebsites.net:443

Site name: SuperFamilyChurnSample

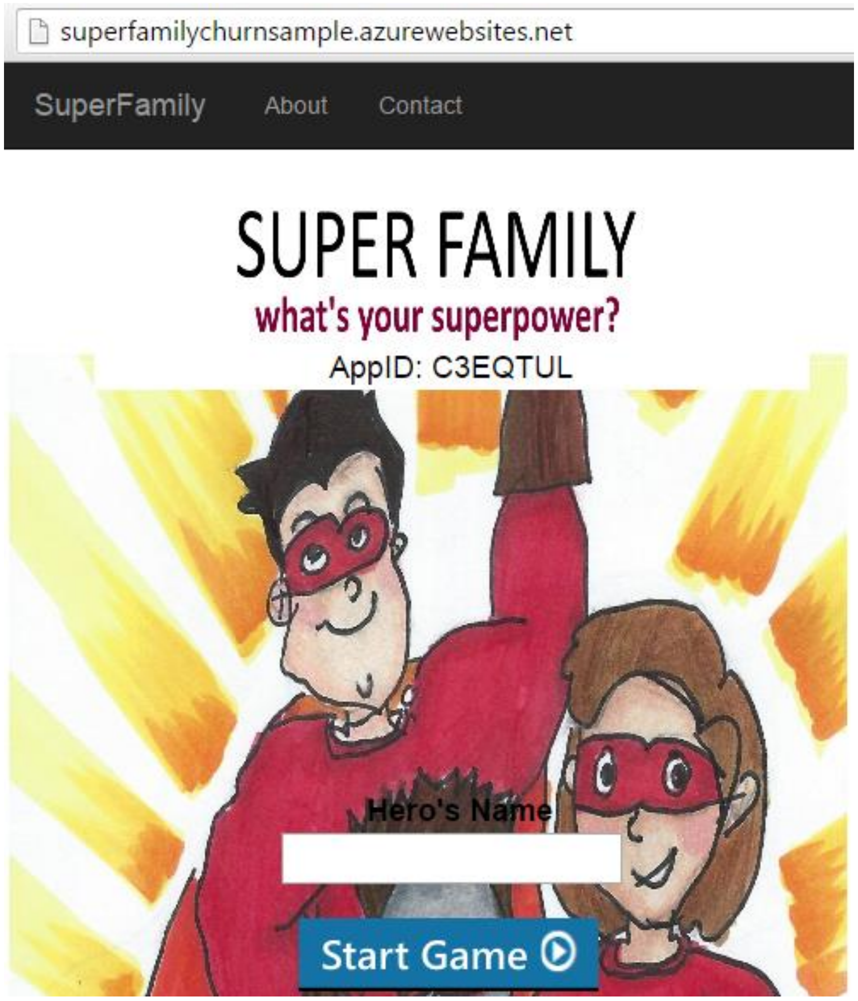
User name: $SuperFamilyChurnSample

Password: ••

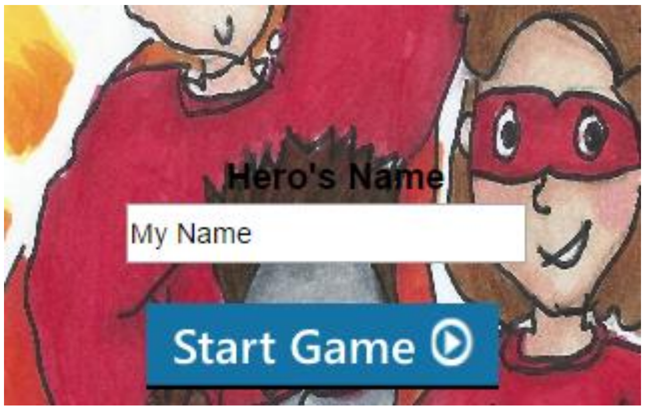
✓ Save password

Destination URL: http://superfamilychurnsample.azurewebsites.net

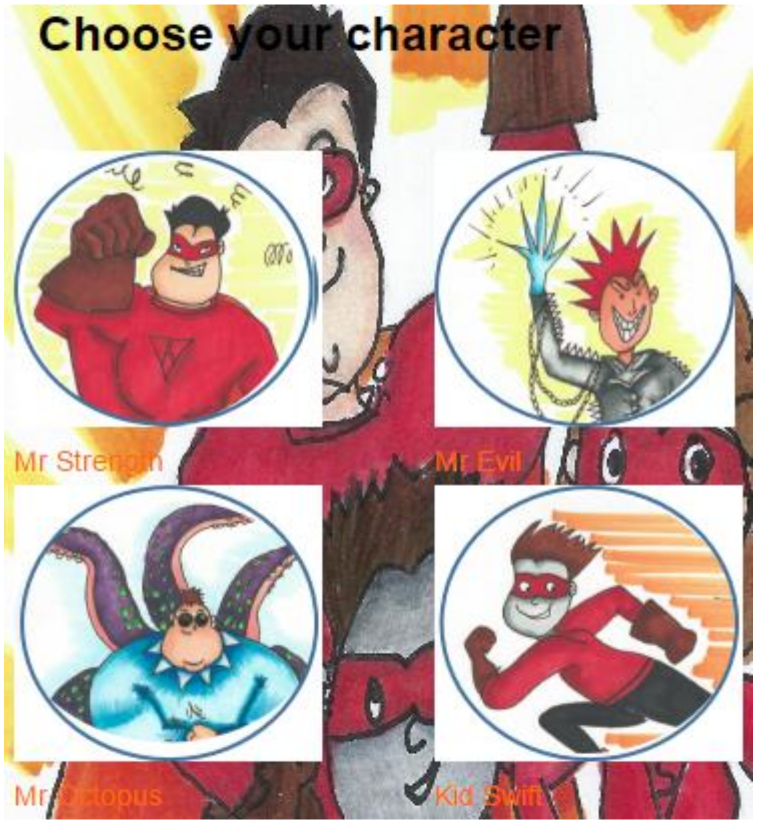
10. Once the web app has been published, the Super Family game should open in your default web browser. If you do not see it after Visual Studio publishes the app, try to navigate manually to the *Destination URL* from the prior step.



11. In order to play your game, enter a **Hero's Name** and click **Start Game**.

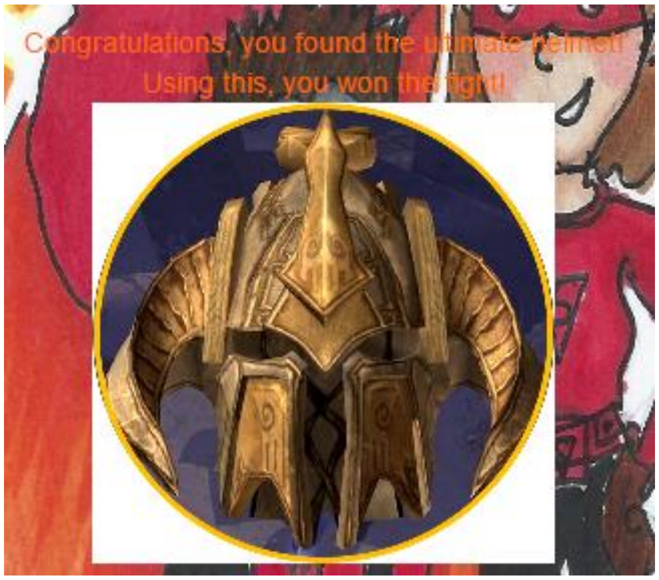


12. Choose your character from the list of four options by clicking on their photo.

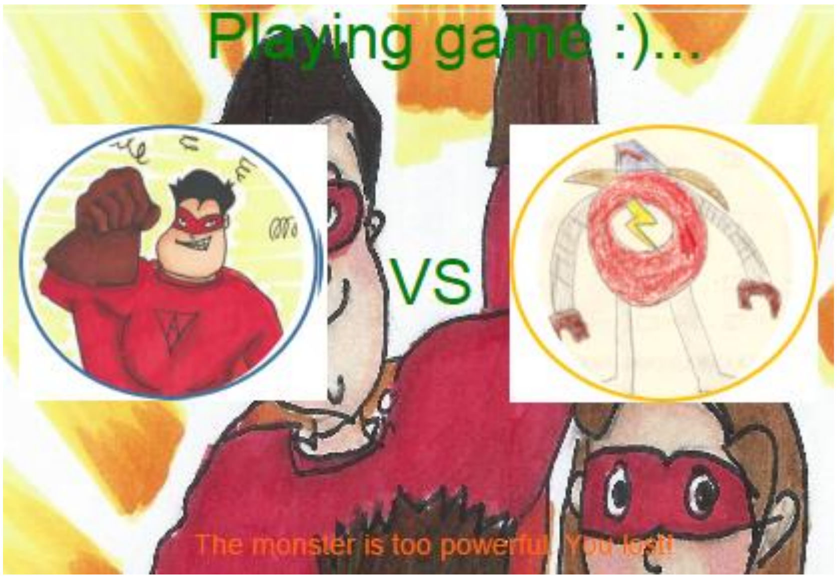


13. At this stage, your web app invokes the Azure ML web service to obtain a churn prediction. If churn is not predicted, you should see a message stating that the monster is too powerful and that you lost. If churn is predicted, you will see a new helmet that helps you win the fight. Note how these results correspond to the behavior that you saw in the code from Activity 4. Remember that the training dataset provided had churn events for about one out of every eight gamers. You may need to play a few rounds before you see a helmet appear.

Helmet provided for Churn intervention, and player wins:


Congratulations, you found the ultimate helmet!
Using this, you won the fight!

Helmet not provided, and player loses:


Playing game :)...

VS

The monster is too powerful. You lost!

Congratulations! You have completed all of the activities.

This glossary contains terms relevant to your experiment in Azure ML.

- **Accuracy:** measures how good a classification model performs as the proportion of true results to total cases
- **AUC:** area under the curve plotted with true positives on the y axis and false positives on the x axis. This metric is useful because it provides a single number that lets you compare models of different types.
- **Decision Tree / Boosted Decision Trees:** a type of classification algorithm
- **Experiment:** the core working environment in Azure Machine Learning
- **False Positive:** a test result that detects a condition when that condition is not present
- **ML:** machine learning
- **Precision:** the proportion of true results over all positive results
- **Recall:** the fraction of all correct results returned by the model
- **ROC:** Receiver Operator Characteristic
- **SMOTE:** Synthetic Minority Oversampling Technique used to enhance small sample sizes
- **Support Vector Machine (SVM):** a type of classification algorithm
- **True Positive:** a test result that detects a condition when that condition is present
- **Web Service:** a standardized way of integrating web applications over the internet

Terms of Use