|  |  |
| --- | --- |
| Parker J. May | February 10, 2018 |

|  |  |  |
| --- | --- | --- |
| McKesson | Final Project | Deep Azure |

Topic: Azure Machine Learning

**Problem Statement:**

There are few problems facing modern society as pressing as predicting the quality of a bottle of wine. With the rising popularity of wine, prices rise as well. Finding a good bottle at a good price seems more like luck than anything else. This project will combine the Azure Machine Learning service and wine rating data to create a model for predicting wine quality based on several factors. It will use an Azure SQL Database to store the data and Azure API Management Services to make a REST API available to a browser based client.

**Description of Data:**

Data for this project was obtained from Kaggle. The data was originally culled from wine reviews and presents 10 data points for each review. Of those 10 data points, I'm using five in my regression model.

Four of the data points will be the input features used to predict one output attribute.

**Inputs:**

Country: the country that the wine is from

Province: the province or state that the wine is from

Variety: the type of grapes used to make the wine (e.g. Pinot Noir)

Price: the cost for a bottle of the wine

**Output:**

Points: the number of points Wine Enthusiast rated the wine on a scale of 1-100

The data was downloaded from <https://www.kaggle.com/zynicide/wine-reviews>

**Description of Hardware:**

Most of the work was done on my home computer: a Lenovo laptop running Windows 10.

Some work was done on my work computer: a Dell laptop running Windows 7.

**Description of Software:**

**Azure Services:**

Azure Machine Learning Studio. (<https://studio.azureml.net>)

Azure SQL Database. (<https://docs.microsoft.com/en-us/azure/sql-database/>)

Azure API Management Services. (<https://docs.microsoft.com/en-us/azure/api-management/>)

**Utilities to prepare and upload data into the SQL database:**

Download the bcp and sqlcmd utilities from the Microsoft Download Center.

<https://www.microsoft.com/en-us/download/details.aspx?id=36433>

That installation required the Microsoft ODBC Driver 11 for SQL Server:

<https://www.microsoft.com/en-us/download/details.aspx?id=36434>

**Utilities to manipulate the data file:**

Linux shell and utilities from Cygwin:

<https://cygwin.com/install.html>

**Browser based client application:**

I created my client application using the Angular.io framework. This involves first installing node.js and npm:

<https://nodejs.org/en/download/>

And then installing the Angular CLI npm package at the global level:

|  |
| --- |
| npm install -g @angular/cli |

**Example data file:**

Below are the first 10 lines from the data file as downloaded from Kaggle:

<https://www.kaggle.com/zynicide/wine-reviews>

,country,description,designation,points,price,province,region\_1,region\_2,variety,winery

0,US,"This tremendous 100% varietal wine hails from Oakville and was aged over three years in oak. Juicy red-cherry fruit and a compelling hint of caramel greet the palate, framed by elegant, fine tannins and a subtle minty tone in the background. Balanced and rewarding from start to finish, it has years ahead of it to develop further nuance. Enjoy 2022â€“2030.",Martha's Vineyard,96,235.0,California,Napa Valley,Napa,Cabernet Sauvignon,Heitz

1,Spain,"Ripe aromas of fig, blackberry and cassis are softened and sweetened by a slathering of oaky chocolate and vanilla. This is full, layered, intense and cushioned on the palate, with rich flavors of chocolaty black fruits and baking spices. A toasty, everlasting finish is heady but ideally balanced. Drink through 2023.",Carodorum SelecciÃ³n Especial Reserva,96,110.0,Northern Spain,Toro,,Tinta de Toro,Bodega Carmen RodrÃ­guez

2,US,"Mac Watson honors the memory of a wine once made by his mother in this tremendously delicious, balanced and complex botrytised white. Dark gold in color, it layers toasted hazelnut, pear compote and orange peel flavors, reveling in the succulence of its 122 g/L of residual sugar.",Special Selected Late Harvest,96,90.0,California,Knights Valley,Sonoma,Sauvignon Blanc,Macauley

3,US,"This spent 20 months in 30% new French oak, and incorporates fruit from Ponzi's Aurora, Abetina and Madrona vineyards, among others. Aromatic, dense and toasty, it deftly blends aromas and flavors of toast, cigar box, blackberry, black cherry, coffee and graphite. Tannins are polished to a fine sheen, and frame a finish loaded with dark chocolate and espresso. Drink now through 2032.",Reserve,96,65.0,Oregon,Willamette Valley,Willamette Valley,Pinot Noir,Ponzi

4,France,"This is the top wine from La BÃ©gude, named after the highest point in the vineyard at 1200 feet. It has structure, density and considerable acidity that is still calming down. With 18 months in wood, the wine has developing an extra richness and concentration. Produced by the Tari family, formerly of ChÃ¢teau Giscours in Margaux, it is a wine made for aging. Drink from 2020.",La BrÃ»lade,95,66.0,Provence,Bandol,,Provence red blend,Domaine de la BÃ©gude

5,Spain,"Deep, dense and pure from the opening bell, this Toro is a winner. Aromas of dark ripe black fruits are cool and moderately oaked. This feels massive on the palate but sensationally balanced. Flavors of blackberry, coffee, mocha and toasty oak finish spicy, smooth and heady. Drink this exemplary Toro through 2023.",Numanthia,95,73.0,Northern Spain,Toro,,Tinta de Toro,Numanthia

6,Spain,"Slightly gritty black-fruit aromas include a sweet note of pastry along with a hint of prune. Wall-to-wall saturation ensures that all corners of one's mouth are covered. Flavors of blackberry, mocha and chocolate are highly impressive and expressive, while this settles nicely on a long finish. Drink now through 2024.",San RomÃ¡n,95,65.0,Northern Spain,Toro,,Tinta de Toro,Maurodos

7,Spain,"Lush cedary black-fruit aromas are luxe and offer notes of marzipan and vanilla. This bruiser is massive and tannic on the palate, but still lush and friendly. Chocolate is a key flavor, while baked berry and cassis flavors are hardly wallflowers. On the finish, this is tannic and deep as a sea trench. Drink this saturated black-colored Toro through 2023.",Carodorum Ãšnico Crianza,95,110.0,Northern Spain,Toro,,Tinta de Toro,Bodega Carmen RodrÃ­guez

8,US,"This re-named vineyard was formerly bottled as deLancellotti. You'll find striking minerality underscoring chunky black fruits. Accents of citrus and graphite comingle, with exceptional midpalate concentration. This is a wine to cellar, though it is already quite enjoyable. Drink now through 2030.",Silice,95,65.0,Oregon,Chehalem Mountains,Willamette Valley,Pinot Noir,BergstrÃ¶m

9,US,"The producer sources from two blocks of the vineyard for this wineâ€”one at a high elevation, which contributes bright acidity. Crunchy cranberry, pomegranate and orange peel flavors surround silky, succulent layers of texture that present as fleshy fruit. That delicately lush flavor has considerable length.",Gap's Crown Vineyard,95,60.0,California,Sonoma Coast,Sonoma,Pinot Noir,Blue Farm

The entire dataset has 150936 lines.

See the cleansing section for detailed information on how the data was manipulated before being loaded.

**Cleansing of data. Sample of final dataset:**

As is described in the next section, I'm uploading my data in an Azure SQL Database and am using the bcp utility to bulk load the csv file into my table.

I'll cover the upload process in detail in the next section, but my first attempts at importing the data failed because some fields are in double quotes because they have commas in the text. I'm sure there's a way to handle this with bcp, but I haven't found a solution that doesn't require all fields to be double quotes.

The primary field that uses the double quotes is the description field and I'm not using that field in my analysis, so I'm using a sed command to remove the data inside the quotes and the quotes themselves.

|  |
| --- |
| sed 's/"[^"]\*"//' winemag-data\_first150k.csv > noquotes.csv |

There are a relative handful of lines that have double quotes in other fields. The easiest thing to do here is just get rid of those lines:

|  |
| --- |
| sed -i '/"/d' noquotes.csv |

This leaves me with 149145 lines. 1791 were removed.

Because of the number of words from other languages, there are a lot of letters with diacritics which means they're part of the extended unicode set. This is likely to be another problem for the utility I'm using to upload the data. Most of the cases here are in fields I don't plan to use so, at the risk of looking like an English-centric xenophobe, I'm using the Linux utility iconv to change the non-ASCII characters into the closest ASCII representation. This has the interesting effect of, for example, making Rosé into Ros'e.

|  |
| --- |
| iconv -f utf8 -t ascii//TRANSLIT//IGNORE noquotes.csv > final-dataset.csv |

As noted above, most of the fields that have the diacritics are fields I'm not using, but the relative handle of entries that will show up with the extra punctuation mark are a bit distracting. This isn't the most elegant solution, but we're on the clock here, so I'm going to remove the punctuation add by the iconv command. Of course, this will remove those characters even if they were previously part of the data, but after eyeballing the data set, I don't think this will negatively impact the fields of interest.

|  |
| --- |
| sed -i "s/[\^~'\"\`]//g" final-dataset.csv |

Now that's a sharp looking data file.

Here are the first 10 rows:

,country,description,designation,points,price,province,region\_1,region\_2,variety,winery

0,US,,Marthas Vineyard,96,235.0,California,Napa Valley,Napa,Cabernet Sauvignon,Heitz

1,Spain,,Carodorum Seleccion Especial Reserva,96,110.0,Northern Spain,Toro,,Tinta de Toro,Bodega Carmen Rodriguez

2,US,,Special Selected Late Harvest,96,90.0,California,Knights Valley,Sonoma,Sauvignon Blanc,Macauley

3,US,,Reserve,96,65.0,Oregon,Willamette Valley,Willamette Valley,Pinot Noir,Ponzi

4,France,,La Brulade,95,66.0,Provence,Bandol,,Provence red blend,Domaine de la Begude

5,Spain,,Numanthia,95,73.0,Northern Spain,Toro,,Tinta de Toro,Numanthia

6,Spain,,San Roman,95,65.0,Northern Spain,Toro,,Tinta de Toro,Maurodos

7,Spain,,Carodorum Unico Crianza,95,110.0,Northern Spain,Toro,,Tinta de Toro,Bodega Carmen Rodriguez

**Upload of data, transformations:**

The Azure Machine Learning Studio has the ability to upload a file directly from your local disk to use as an input to a learning model. However, my final product will be a browser based single page application and the user will need to enter the attribute values they want to use to predict a rating. I don't want to leave those fields as free-form entry fields because the regression model won't be able to make use of unexpected values.

My plan is to upload the dataset into an Azure SQL Database. The Azure Machine Learning Studio can connect to it to get the data for the experiment. I can also expose queries as REST endpoints to get the list of distinct values for the fields for which I want to control input.

First I had to create the database. I did this use Azure CLI.

As is my practice, I set environment variables for the common values.

|  |
| --- |
| set resourcePrefix=pmay-finalproject  set resourceGroupName=%resourcePrefix%-rg  set location=eastus  set adminLogin=pmay-admin  set adminPassword=\*\*\*\*\*\*\*\*\*\*\*\*\*  set clientIpAddress=74.109.194.99 |

Create the resource group.

|  |
| --- |
| C:\final-project>**az group create --name %resourceGroupName% --location %location%**  {  "id": "/subscriptions/a5280073-bceb-4175-9bfd-c51134647c8b/resourceGroups/pmay-finalproject-rg",  "location": "eastus",  "managedBy": null,  "name": "pmay-finalproject-rg",  "properties": {  "provisioningState": "Succeeded"  },  "tags": null  } |

Create the database server.

|  |
| --- |
| C:\final-project>**az sql server create --name %resourcePrefix%-server --resource-group %resourceGroupName% --location %location% --admin-user %adminLogin% --admin-password %adminPassword%**  {| Finished ..  "administratorLogin": "pmay-admin",  "administratorLoginPassword": null,  "fullyQualifiedDomainName": "pmay-finalproject-server.database.windows.net",  "id": "/subscriptions/a5280073-bceb-4175-9bfd-c51134647c8b/resourceGroups/pmay-finalproject-rg/providers/Microsoft.Sql/servers/pmay-finalproject-server",  "identity": null,  "kind": "v12.0",  "location": "eastus",  "name": "pmay-finalproject-server",  "resourceGroup": "pmay-finalproject-rg",  "state": "Ready",  "tags": {  "ccSubOwner": "ebvtly1",  "techOwner": "ek0a6xw"  },  "type": "Microsoft.Sql/servers",  "version": "12.0"  } |

Configure a firewall rule to allow my client to connect.

|  |
| --- |
| C:\final-project>**az sql server firewall-rule create --name %resourcePrefix%-fr --resource-group %resourceGroupName% --server %resourcePrefix%-server --start-ip-address %clientIpAddress% --end-ip-address %clientIpAddress%**  {  "endIpAddress": "74.109.194.99",  "id": "/subscriptions/a5280073-bceb-4175-9bfd-c51134647c8b/resourceGroups/pmay-finalproject-rg/providers/Microsoft.Sql/servers/pmay-finalproject-server/firewallRules/pmay-finalproject-fr",  "kind": "v12.0",  "location": "East US",  "name": "pmay-finalproject-fr",  "resourceGroup": "pmay-finalproject-rg",  "startIpAddress": "74.109.194.99",  "type": "Microsoft.Sql/servers/firewallRules"  } |

Create the database with a Basic service level.

|  |
| --- |
| C:\final-project>**az sql db create --resource-group %resourceGroupName% --server %resourcePrefix%-server --name %resourcePrefix%-db --service-objective Basic**  {| Finished ..  "collation": "SQL\_Latin1\_General\_CP1\_CI\_AS",  "containmentState": 2,  "createMode": null,  "creationDate": "2018-02-06T23:38:59.340000+00:00",  "currentServiceObjectiveId": "dd6d99bb-f193-4ec1-86f2-43d3bccbc49c",  "databaseId": "3d4be361-54f1-462f-9415-3765eacfa971",  "defaultSecondaryLocation": "West US",  "earliestRestoreDate": "2018-02-07T00:09:33.513000+00:00",  "edition": "Basic",  "elasticPoolName": null,  "failoverGroupId": null,  "id": "/subscriptions/a5280073-bceb-4175-9bfd-c51134647c8b/resourceGroups/pmay-finalproject-rg/providers/Microsoft.Sql/servers/pmay-finalproject-server/databases/pmay-finalproject-db",  "kind": "v12.0,user",  "location": "East US",  "maxSizeBytes": "2147483648",  "name": "pmay-finalproject-db",  "readScale": "Disabled",  "recommendedIndex": null,  "recoveryServicesRecoveryPointResourceId": null,  "requestedServiceObjectiveId": "dd6d99bb-f193-4ec1-86f2-43d3bccbc49c",  "requestedServiceObjectiveName": "Basic",  "resourceGroup": "pmay-finalproject-rg",  "restorePointInTime": null,  "sampleName": null,  "serviceLevelObjective": "Basic",  "serviceTierAdvisors": null,  "sourceDatabaseDeletionDate": null,  "sourceDatabaseId": null,  "status": "Online",  "tags": {  "ccSubOwner": "ebvtly1",  "techOwner": "ek0a6xw"  },  "transparentDataEncryption": null,  "type": "Microsoft.Sql/servers/databases"  } |

As noted in the section about software, I'm using a utility called bcp to bulk upload my data. Packaged with that utility is sqlcmd which allows me to create my wine\_rating table from the command line. I had to run this in PowerShell because of the multiple lines in the command.

|  |
| --- |
| sqlcmd.exe -S pmay-finalproject-server.database.windows.net -d pmay-finalproject-db -U pmay-admin -P \*\*\*\*\*\*\*\* -I -Q "  CREATE TABLE wine\_rating  (  country VARCHAR(100),  points INT,  price MONEY,  province VARCHAR(100),  variety VARCHAR(100),  )  ;  " |

If my table exactly matched my input file, I could just run bcp. However, I only want select columns. This requires a format file to specify how to load my data file. The best way to create the initial format file is to run the format option from the bcp utility.

|  |
| --- |
| bcp wine\_rating format nul -c -f wine-rating.fmt -t, -S pmay-finalproject-server.database.windows.net -d pmay-finalproject-db -U pmay-admin -P \*\*\*\*\*\*\*\* |

This creates a format file called wine-rating.fmt. I need to edit the format file to exclude the fields from my source file that I'm not interested in. Here's the modified version of the file.

|  |
| --- |
| 14.0  11  1 SQLCHAR 0 100 "," 0 rowid SQL\_Latin1\_General\_CP1\_CI\_AS  2 SQLCHAR 0 100 "," 1 country SQL\_Latin1\_General\_CP1\_CI\_AS  3 SQLCHAR 0 1000 "," 0 description SQL\_Latin1\_General\_CP1\_CI\_AS  4 SQLCHAR 0 100 "," 0 designation SQL\_Latin1\_General\_CP1\_CI\_AS  5 SQLCHAR 0 12 "," 2 points ""  6 SQLCHAR 0 30 "," 3 price ""  7 SQLCHAR 0 100 "," 4 province SQL\_Latin1\_General\_CP1\_CI\_AS  8 SQLCHAR 0 100 "," 0 region\_1 SQL\_Latin1\_General\_CP1\_CI\_AS  9 SQLCHAR 0 100 "," 0 region\_2 SQL\_Latin1\_General\_CP1\_CI\_AS  10 SQLCHAR 0 100 "," 5 variety SQL\_Latin1\_General\_CP1\_CI\_AS  11 SQLCHAR 0 100 "\n" 0 winery SQL\_Latin1\_General\_CP1\_CI\_AS |

Finally I can load the modified dataset using my modified format file. The actual output of this command listed many lines of "1000 rows sent". I removed most of those line here, but you get the idea.

|  |
| --- |
| C:\final-project>**bcp wine\_rating in final-dataset.csv -f wine-rating.fmt -F 2 -S pmay-finalproject-server.database.windows.net -d pmay-finalpro**  **ject-db -U pmay-admin -P \*\*\*\*\*\*\*\*\*\***  Starting copy...  1000 rows sent to SQL Server. Total sent: 1000  1000 rows sent to SQL Server. Total sent: 2000  1000 rows sent to SQL Server. Total sent: 3000  1000 rows sent to SQL Server. Total sent: 4000  .  .  .  1000 rows sent to SQL Server. Total sent: 147000  1000 rows sent to SQL Server. Total sent: 148000  1000 rows sent to SQL Server. Total sent: 149000  149144 rows copied.  Network packet size (bytes): 4096  Clock Time (ms.) Total : 48704 Average : (3062.25 rows per sec.) |

I connected to the database through Azure Portal and ran a few queries to confirm the data looked good. I won't include a lot here, but the most basic query was:

|  |
| --- |
| select count(\*) from wine\_rating;  Results  149144 |

**Create the Azure Machine Learning experiment and export as a web service.**

**Create a new experiment**

Using Microsoft's Azure Machine Learning Studio (<https://studio.azureml.net>), I create a new experiment called "Wine Rating"

**Import data from the database**

I pulled in an Import Data module. At least for now, I put the connection information directly into the Import Data module. This can be configured to get the information when deployed as a Web Service and that is clearly the right way to do a real application, but in the interest of time, I'm putting the connection information right in my module..

I entered a query to the get the columns that I want to use in my regression experiment. I found out the hard way that the Import Data module doesn't support decimal type data columns like my money type columns. The workaround is to cast the data as float.

select country, province, variety, cast("price" as float) as price, points from wine\_rating;

I run that module and eyeball the output. The individual data points look good and the overall stats look right.

rows: 149144

columns: 5

**Set up the regression experiment**

I add a Clean Missing Data module to remove the handful of rows that are missing any column data.

Running this module shows I'm down to 135775 rows. I lost more than I thought, but still plenty for a good experiment I think.

I don't need a Select Columns in Dataset module because I'm only selecting what I need in the Import Data module.

I add a Split Data module to use 75 percent of my input data for training my model and the remaining 25 percent to evaluate my model.

Select Machine Learning -> Initialize Model -> Regression -> Linear Regression and drag it next to Split Data. The split data and the regression algorithm will both feed into the training.

Select Machine Learning -> Train -> Train Model and drag it under Split Data and Linear Regression and connect them both as inputs to Train Model. By making these connections, I’ve said that I want to train the model with 75% of my input data using a simple linear regression. In the properties for Train Model, I select "points" as this is the value I want the model to estimate.

To see how well the model is doing, I need to score the model.

Select Machine Learning -> Score -> Score Model and drag it under Train Model and Split Data, linking each as an input into Score Model.

Save and Run, right mouse on the output circle and select Visualize. This shows the actual points (rating) and the points determined by the model for each of the 25% of input rows that were used to score the model. Some look very close, some not so much.

To evaluate the model, select Machine Learning -> Evaluate -> Evaluate Model, drag it under Score Model and connect them.

**Save and Run and Visualize the output:**

|  |  |
| --- | --- |
| Mean Absolute Error | 2.12662 |
| Root Mean Squared Error | 2.679113 |
| Relative Absolute Error | 0.814243 |
| Relative Squared Error | 0.688569 |
| Coefficient of Determination | 0.311431 |

Well, I'm not going to lie. Those results are little underwhelming.

I want to compare outputs of multiple models, so I’m using an R script to add the algorithm name to each of the outputs.

Select R Language Modules -> Execute R Script, drag it under Evaluate Model and connect them.

I add the following code to the R Script module:

|  |
| --- |
| #Get the metrics from the Evaluate model map the input data to port 1  dataset <- maml.mapInputPort(1)    # Add algorithm name as a column to data frame  data <- data.frame(Algorithm='Simple Linear Regression')  data <- cbind(data, dataset[1:5])    #output  maml.mapOutputPort("data"); |

I save and run and confirm I’m getting the same metrics from Evaluate Model with the extra column for the algorithm name.

Now I add a new algorithm – Boosted Decision Tree Regression algorithm. To set this up, I select and drag over the Boosted Decision Tree Regression module along with a new Train Model, Score Model, Evaluate Model and Execute R Script. Wire them up and have the Split module input into my new Train Model and Score Model in addition to still inputting into the existing Linear Regression related modules. The only change I need to make is the name of the algorithm in the R Script.

I save and select the new modules and run them.

To combine the output of these two algorithms, select Data Transformation -> Manipulation -> Add Rows. Connect the output of the two Execute R Script modules to the Add Rows.

Save and run.

Now I see the output metrics of both algorithms as two rows in the Add Rows output.

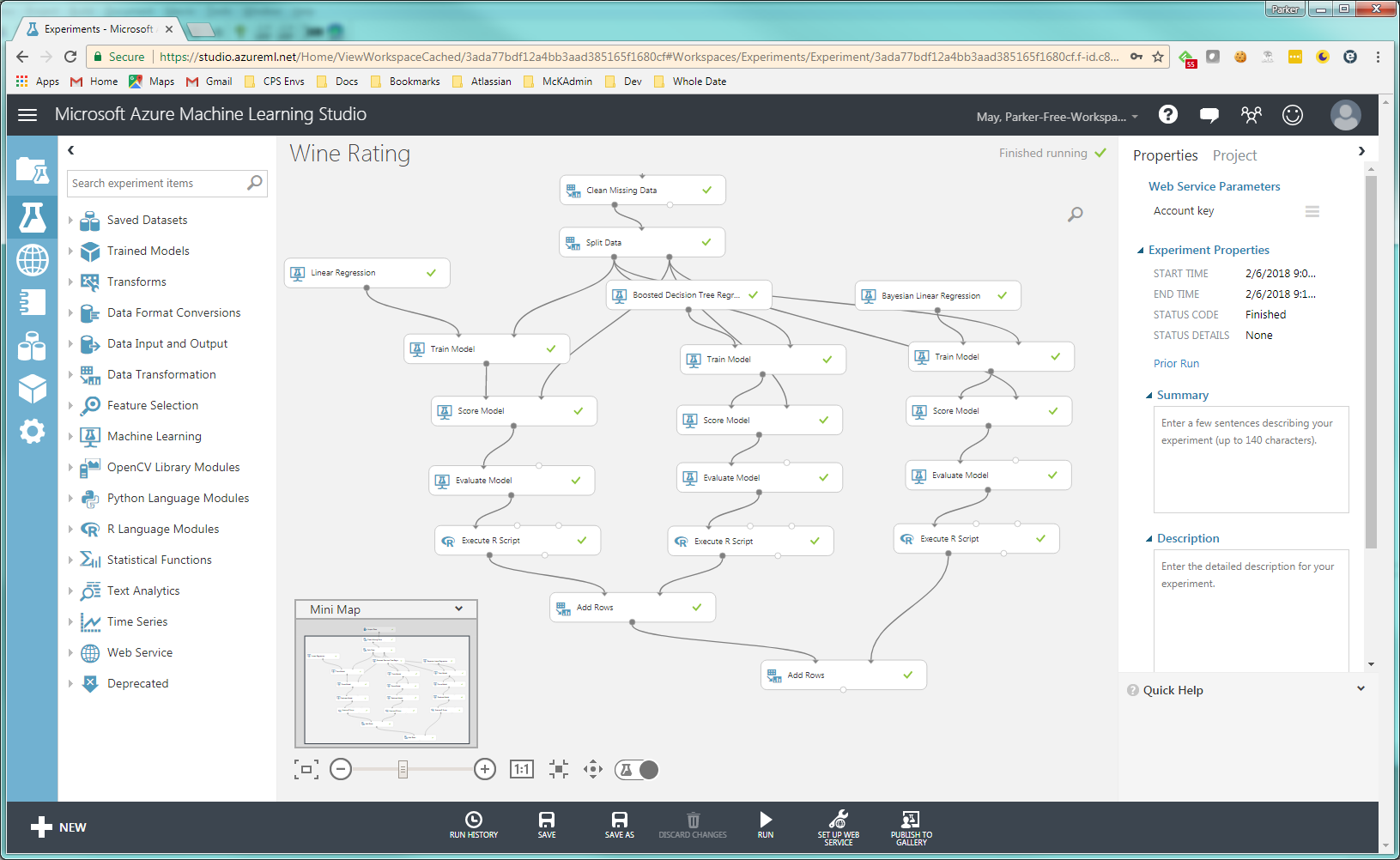
Finally, I add another algorithm – Bayesian Linear Regression. The steps are very similar to what I did when adding the Boosted Decision Tree Regression algorithm, with a couple of exceptions.

The output of this algorithm has an extra metric. Since the other algorithms lack this metric, I skip it in the R Script. Notice the dataset index in the cbind command below:

|  |
| --- |
| #Get the metrics from the Evaluate model map the input data to port 1  dataset <- maml.mapInputPort(1)    # Add algorithm name as a column to data frame  data <- data.frame(Algorithm='Bayesian Linear Regression')  data <- cbind(data, dataset[2:6])    #output  maml.mapOutputPort("data"); |

The other change is that because Add Rows only takes two inputs, I wire the output of the previous Add Rows in my new Add Rows as one input and the output of the Bayesian R Script as the second input.

This is a screenshot of my experiment:



**Pick the best model**

I did a Save and Run and list the results below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Mean Absolute Error** | **Root Mean Squared Error** | **Relative Absolute Error** | **Relative Squared Error** | **Coefficient of Determination** |
| **Simple Linear** | 2.12662 | 2.679113 | 0.814243 | 0.688569 | 0.311431 |
| **Boosted Decision Tree** | 1.882871 | 2.391604 | 0.720916 | 0.548711 | 0.451289 |
| **Bayesian Linear** | 2.144943 | 2.71422 | 0.821258 | 0.706733 | 0.293267 |

Boosted Decision Tree is the best out of the bunch, through a Coefficient of Determination of 0.45 is not inspiring.

I'm going to try some variations. I wonder if Country is really having an impact at all. Province should really be a more specific version of that information.

I modify the Import Data module's SQL:

|  |
| --- |
| select province, variety, cast("price" as float) as price, points from wine\_rating; |

The output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Mean Absolute Error** | **Root Mean Squared Error** | **Relative Absolute Error** | **Relative Squared Error** | **Coefficient of Determination** |
| **Simple Linear** | 2.126512 | 2.679013 | 0.814201 | 0.688517 | 0.311483 |
| **Boosted Decision Tree** | 1.888928 | 2.398972 | 0.723235 | 0.552097 | 0.447903 |
| **Bayesian Linear** | 2.362853 | 3.558066 | 0.904692 | 1.214488 | -0.214488 |

Simple Linear and Boosted Decision Tree are pretty similar, but Bayesian Linear went negative.

Clearly I have a lot to learn about how these regression algorithms work. In way, it's reassuring that I can't just waltz in here and plug some numbers into a GUI-built model and extract amazing insights. I find these statistical models and algorithms fascinating and I intend to spend more time after this class getting a better understand of how this all works. But for now, I need to push on.

I tried removing country and price, leaving only province and variety as inputs.

|  |
| --- |
| select province, variety, points from wine\_rating; |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Mean Absolute Error** | **Root Mean Squared Error** | **Relative Absolute Error** | **Relative Squared Error** | **Coefficient of Determination** |
| **Simple Linear** | 2.338276 | 2.924716 | 0.895313 | 0.818477 | 0.181523 |
| **Boosted Decision Tree** | 2.320162 | 2.896257 | 0.888377 | 0.802627 | 0.197373 |
| **Bayesian Linear** | 2.565917 | 3.666825 | 0.982476 | 1.286529 | -0.286529 |

Well, that was awful.

How about just variety and price?

|  |
| --- |
| select variety, cast("price" as float) as price, points from wine\_rating; |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Mean Absolute Error** | **Root Mean Squared Error** | **Relative Absolute Error** | **Relative Squared Error** | **Coefficient of Determination** |
| **Simple Linear** | 2.215889 | 2.778283 | 0.844237 | 0.734345 | 0.265655 |
| **Boosted Decision Tree** | 1.966863 | 2.480503 | 0.74936 | 0.585365 | 0.414635 |
| **Bayesian Linear** | 2.563105 | 4.781691 | 0.976524 | 2.175256 | -1.175256 |

This is in the ballpark of province, variety and price.

And finally just province and price.

|  |
| --- |
| select province, cast("price" as float) as price, points from wine\_rating; |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Mean Absolute Error** | **Root Mean Squared Error** | **Relative Absolute Error** | **Relative Squared Error** | **Coefficient of Determination** |
| **Simple Linear** | 2.146554 | 2.803265 | 0.820636 | 0.75305 | 0.24695 |
| **Boosted Decision Tree** | 1.920588 | 2.435924 | 0.734248 | 0.568621 | 0.431379 |
| **Bayesian Linear** | 2.355075 | 3.890668 | 0.900354 | 1.450588 | -0.450588 |

I was hoping for a little better correlation, but I'm going to continue forward using the Boosted Decision Tree regression algorithm with province, variety and price. I'm dropping country because it didn't make much difference.

**Deploy as web service**

The next step is to deploy the model as a web service so I can access it from my client application. I select the Train Model for Boosted Decision Tree, click Set Up Web Service and select Predictive Web Service

This creates a new Predictive Experiment. I click Run and, when complete, Deploy Web Service.

I copy the API key and click Test in the REQUEST/RESPONSE row.

This gives me a cool little form to enter my data on.

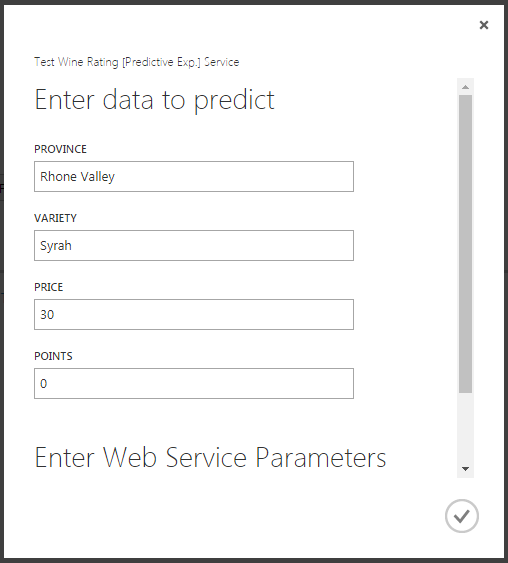
I enter sample values:

PROVINCE: Rhone Valley

VARIETY: Syrah

PRICE: 30

and my API key.



I click the Check mark and get back the results:

'Wine Rating [Predictive Exp.]' test returned ["Rhone Valley","Syrah","30","0","88.7319107055664"]...

It also provides the full, unformatted JSON response from the web service:

Result: {"Results":{"output1":{"type":"table","value":{"ColumnNames":["province","variety","price","points","Scored Labels"],"ColumnTypes":["String","String","Double","Int32","Double"],"Values":[["Rhone Valley","Syrah","30","0","88.7319107055664"]]}}}}

The predicted point rating is 88.7.

Awesome.

Clicking the REQUEST/RESPONSE link under API HELP PAGE, I get all of the details to run my own REST call.

Using the Sample Request, I create a file called test.json with the following contents to be the body of my POST request:

|  |
| --- |
| {  "Inputs": {  "input1": {  "ColumnNames": [  "province",  "variety",  "price",  "points"  ],  "Values": [  [  "Rhone Valley",  "Syrah",  "30",  "0"  ]  ]  }  },  "GlobalParameters": {  "Account key": "xxxx"  } } |

Then I run curl with the appropriate headers:

|  |
| --- |
| [A4KQM2H2][/c/final-project]$ **curl -d "@test.json" -H "Authorization:Bearer xxxx" -H "Content-Type:application/json" -H "Accept: application/json" -X POST https://ussouthcentral.services.azureml.net/workspaces/3ada77bdf12a4bb3aad385165f1680cf/services/6135fc1463eb4ff9b25f28556b25fe16/execute?api-version=2.0&details=true**  % Total % Received % Xferd Average Speed Time Time Time Current  Dload Upload Total Spent Left Speed  100 460 100 114 100 346 192 584 --:--:-- --:--:-- --:--:-- 584{"Results":{"output1":{"type":"table","value":{"Values":[["Rhone Valley","Syrah","30","0","88.7319107055664"]]}}}} |

I run the output through an online JSON formatter (<https://jsonformatter.curiousconcept.com/>) for clarity:

|  |
| --- |
| {  "Results":{  "output1":{  "type":"table",  "value":{  "Values":[  [  "Rhone Valley",  "Syrah",  "30",  "0",  "88.7319107055664"  ]  ]  }  }  }  } |

Same results as when using the Test provided by ML Studio. Looking pretty good so far.

**Set up an Azure API Management service to make the web service accessible:**

The next thing I did was to create a client application to use my web service. However, when I first tried to a POST call through my browser-based application, it failed with an error:

Failed to load https://ussouthcentral.services.azureml.net/workspaces/3ada77bdf12a4bb3aad385165f1680cf/services/ab8f67e1e16e4fc496b512ae3af31dc8/execute?api-version=2.0&details=true: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:4200' is therefore not allowed access. The response had HTTP status code 400.

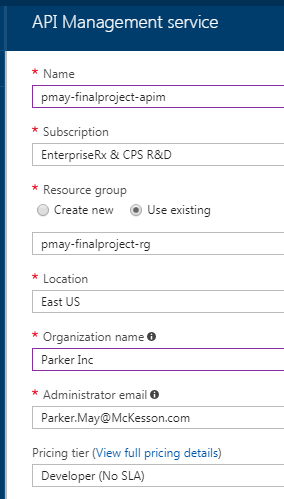
It turns out the web deployment of Azure Machine Learning Studio does not support CORS so while accessing via a standalone program will work, coming through a browser based application will not. See [this](https://social.msdn.microsoft.com/Forums/en-US/b6ddeb77-30e1-45b2-b7c1-eb4492142c0a/azure-ml-published-web-services-cross-origin-requests?forum=MachineLearning) article for more information.

It seems the only way to make the service available to my application is to use Azure API Management to provide a front-end URL that will handle traffic and route to my ML service.

OK, I did not anticipate this but, because I'm a glass-is-half-full kind of guy (sometimes), I'll embrace this as an opportunity to feature a complementary Azure technology in my project. Unfortunately, the time spent on this comes at the cost of some other features I had planned for my project.

Using [this link](https://docs.microsoft.com/en-us/azure/api-management/get-started-create-service-instance) as a guide, I created a new Azure API Management service instance.

Through Azure Portal, I selected New -> API Management.



Click Create.

Wow, that takes a *really* long time to deploy. I wonder what's happening behind the scenes there.

The next step is to import my ML service REST endpoint and publish a public API. I mostly used [this link](https://docs.microsoft.com/en-us/azure/api-management/import-and-publish) for instruction, but also relied heavily on the "Echo API" example that was included with my API Management deployment.

In my newly created API Management service, I click on APIs. This lists the current APIs - namely the example one automatically created called "Echo API".

I click Add API -> Blank API. Filled out the screen as follows:



In the new API, I click Add operation.

Select POST, enter /estimate for the URL and give it a Display of Estimate.

Under Request, I add a representation for "application/json". I also add the request body I used earlier to the SAMPLE field for easy testing from inside the Portal.

Under Response, I add "200 OK".

Save.

Now my new POST operation appears in my list of operations. I click on All operations.

This is where I'll add support for CORS.

In the Inbound Processing section, click the pencil to edit.

Click code view

I place my cursor after policies -> inbound -> base

Click +CORS

The default settings are a pretty liberal CORS policy. Remove the origins tag that has URI. Everything is wide open.

Save.

To test my new REST endpoint, I select the POST Estimate operation and click Test. I add a header to put my API key in:

Authorization:Bearer \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

And would you look at that. I got the response:

|  |
| --- |
| HTTP/1.1 **200 OK**  date: Wed, 07 Feb 2018 21:47:33 GMT etag: "37dd6ec8e476453b801f5fd78fe63f2f" vary: Origin ocp-apim-trace-location: https://apimgmtstvdy7iarbikf56hq.blob.core.windows.net/apiinspectorcontainer/FIxvooftEBj3vrYVB4C01A2-2?sv=2015-07-08&sr=b&sig=%2FZEaCLvMR39gKSh725jfvdt%2F19WM7VCdlXamNLbGsZg%3D&se=2018-02-08T21%3A47%3A33Z&sp=r&traceId=e7fd2b3e90124fe19dbd15e2807fe116 content-type: application/json; charset=utf-8 x-ms-request-id: d4fcb9db-8d45-44be-ae8d-fc6a6e147230 content-length: 114 x-ms-request-duration: 00:00:00.0211251  {  "Results": {  "output1": {  "type": "table",  "value": {  "Values": [  ["Rhone Valley", "Syrah", "30", "0", "88.7319107055664"]  ]  }  }  } } |

Now the truth is that took a little more trial-and-error than what I capture here. OK, a lot of trial-and-error. But these steps are what finally got me a successful REST call against my ML web service.

The real test, of course, will be accessing it through my application.

**Source code:**

My client application is going to be a browser based application using the Angular.io framework.

I described installing nodejs / npm and installing the Angular CLI in the Software section of this document.

To create a new angular application, I ran the ng new command.

|  |
| --- |
| ng new final-project --style=scss  cd final-project |

I upgraded the angular CLI version:

|  |
| --- |
| npm install @angular/cli@latest --save |

I added support for Bootstrap for a slightly nicer default look.

|  |
| --- |
| npm install jquery --save  npm install bootstrap@3.3.7 --save  npm install bootstrap-sass --save |

And had to make these changes to implement Bootstrap:

|  |
| --- |
| # Add the following to the auto generated styles.scss:  $icon-font-path: "../node\_modules/bootstrap-sass/assets/fonts/bootstrap/";  @import '../node\_modules/bootstrap-sass/assets/stylesheets/\_bootstrap';  # In the "scripts" array in .angular-cli.json, tell Angular to include these two libraries:  "../node\_modules/jquery/dist/jquery.min.js",  "../node\_modules/bootstrap/dist/js/bootstrap.min.js" |

**Source Files**

The Angular framework automatically creates quite a few files to support my project. Here are the main files that I modified/created to make my application work.

**src/app/app.module.ts**

This file defines the root module that tells Angular how to assemble the application.

|  |
| --- |
| import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';  import { HttpClientModule } from '@angular/common/http';  import { AppComponent } from './app.component';  import { EstimateService } from './services/estimate.service';  @NgModule({  declarations: [  AppComponent  ],  imports: [  BrowserModule,  FormsModule,  HttpClientModule  ],  providers: [EstimateService],  bootstrap: [AppComponent]  })  export **class** AppModule { } |

**/src/app/app.component.ts**

This is the main typescript for controlling the prediction page.

Part of my original motivation to use the SQL Database was to look up the values in the dataset for province and variety and populate dropdown lists in my UI with those values. It wouldn't be very useful if those inputs were freeform text fields.

Unfortunately with the time spent in other areas, I couldn't expose those query results as REST endpoints like I wanted. So, I connected to the database through Portal as the admin user and simply ran a couple of queries to get a list of values and am putting them in my source code as hard coded values. The limiting by a minimum count of 10 was pretty arbitrary.

|  |
| --- |
| select province, count(\*)  from wine\_rating  group by province  having count(\*) >= 10  order by province;  select variety, count(\*)  from wine\_rating  group by variety  having count(\*) >= 10  order by variety; |

The hard coded arrays are in this source file. There are over 500 lines, so I'm leaving the bulk of those out of this source file listing.

The main function in this file is requestEstimate() which passes the input values entered by the user to the estimate service to make the REST call.

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import { EstimateService } from './services/estimate.service';  @Component({  selector: 'app-root',  templateUrl: './app.component.html',  styleUrls: ['./app.component.css']  })  export **class** AppComponent **implements** OnInit {  province: string;  variety: string;  price: number;  estWineRating: string = '0';  **constructor**(**private** estimateService: EstimateService) { }  ngOnInit() {  }  requestEstimate() {  this.estimateService.calculateEstimate(this.province, this.variety, this.price)  .subscribe(data **=>** {  this.estWineRating = data.Results.output1.value.Values[0][4];  })  }  provinces = [  'Aconcagua Valley',  'Aegean',  .  .  .  'Wurttemberg',  'Zupa'  ];  varieties = [  'Agiorgitiko',  'Aglianico',  .  .  .  'Zinfandel',  'Zweigelt'  ];  } |

**/src/app/app.component.html**

This is in the main (really, only) screen for the application. Input the input values, view the output value.

|  |
| --- |
| <div class='app-main'>  <form>  <h1>Predict Wine Rating</h1>  <div class="form-group">  <label for="provinceInput">Choose a Province</label>  <select id="provinceInput" class="form-control" [(ngModel)]="province" name="province">  <option \*ngFor="let provinceOpt of provinces" [value]="provinceOpt">{{provinceOpt}}</option>  </select>  </div>  <div class="form-group">  <label for="varietyInput">Choose a Variety</label>  <select id="varietyInput" class="form-control" [(ngModel)]="variety" name="variety">  <option \*ngFor="let varietyOpt of varieties" [value]="varietyOpt">{{varietyOpt}}</option>  </select>  </div>  <div class="form-group">  <label for="priceInput">Enter a Price</label>  <input type="text" id="priceInput" class="form-control" [(ngModel)]="price" name="price">  </div>  <button class="btn btn-primary" (click)="requestEstimate()">Run Prediction</button>  <div class="form-group" \*ngIf="estWineRating > 0">  <h3>Predicted Wine Rating: {{estWineRating | number:'1.1-1'}}</h3>  </div>  </form>  </div> |

**/src/app/services/estimate.service.ts**

This service makes the call to the REST endpoint I've created through Azure API Management to ultimately call the ML web service. It returns the estimated rating.

The keys have been x'ed out. Ideally these would be read from a configuration service or to use some kind of trusted access.

|  |
| --- |
| import { Injectable } from '@angular/core';  import { HttpClient, HttpHeaders } from '@angular/common/http';  import { Observable } from 'rxjs/Observable';  import { of } from 'rxjs/observable/of';  import { Estimate } from '../models/estimate.model';  @Injectable()  export **class** EstimateService {  *// azureMlUrl = 'xxxx';*  apiMgmtUrl = 'xxxx';  apiMachineLearningKey = 'xxxx';  apiMgmtKey = 'xxxx';  **constructor**(**private** http: HttpClient) { }  calculateEstimate(province: string, variety: string, price: number): Observable<Estimate> {  **const** httpOptions = {  headers: new HttpHeaders({  'Authorization': 'Bearer ' + this.apiMachineLearningKey,  'Content-Type': 'application/json',  'Cache-Control': 'no-cache',  'Ocp-Apim-Trace': 'true',  'Ocp-Apim-Subscription-Key': this.apiMgmtKey  })  };  **let** body = {  "Inputs": {  "input1": {  "ColumnNames": [  "province",  "variety",  "price",  "points"  ],  "Values": [  [  province,  variety,  price,  0  ]  ]  }  },  "GlobalParameters": {  "Account key": this.apiMachineLearningKey  }  };  return this.http.post<Estimate>(this.apiMgmtUrl, body, httpOptions);  }  } |

**/src/app/models/estimate.model.ts**

Models the return object from the ML web service so that I can extract the rating.

|  |
| --- |
| export **interface** Estimate {  Results: {  output1: {  value: {  Values: Array<Array<string>>  }  }  }  } |

All the source is available in GitHub at <https://github.com/parkermay/final-project>

**Compile and Run code:**

If you're downloading the source from GitHub, you need to run npm install before doing anything else to obtain all of the supporting libraries.

|  |
| --- |
| npm install |

To run the Angular server:

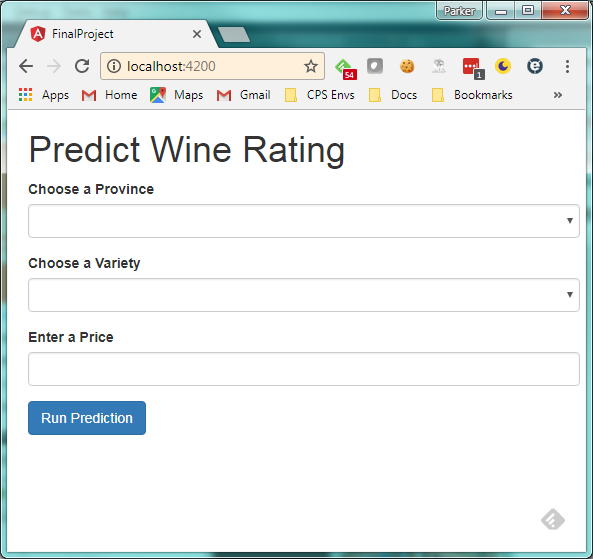
|  |
| --- |
| ng serve |

Then, in the browser, navigate to:

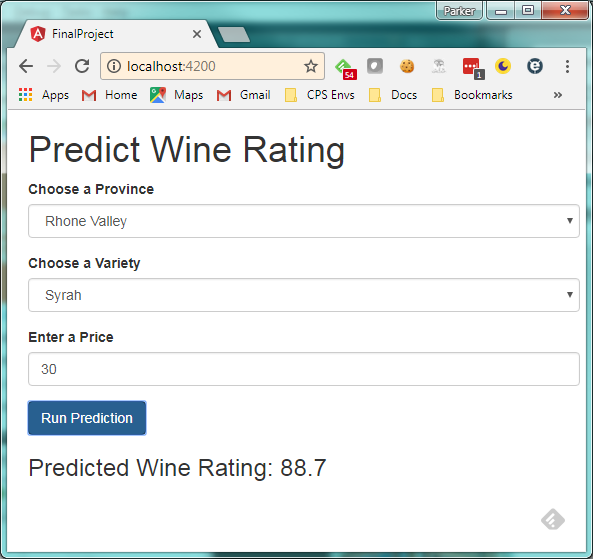
|  |
| --- |
| http://localhost:4200/ |

**Show results:**

The application as it starts up:



And after performing an estimation run:



Yep, that's it. It's simplicity belies the absurd amount of effort that went into making this project work.

**Lessons Learned:**

The biggest takeaway I had with this exercise is that it takes a lot more understanding and hard work to create useful machine learning models that I realized. I find the inner workings of these statistical algorithms very interesting and intend on spending more time digging in after this class is over.

I had originally planned to pull data to seed the options in the UI from the database. I'd like to learn how to expose REST endpoints that query the database.

I would like to have had time to do an anomaly detection version of this where you give all the information including the rating and it tells you if that's an unusual rating based on the other factors.

**URLs:**

* Two minute (short): <https://youtu.be/Fi_lH-p_n4M>
* 15 minutes (long): <https://youtu.be/Y4gIIUNv0a0>
* GitHub: <https://github.com/parkermay/final-project>