Machine Learning Demo Program

John Ferguson, IRIS Engineering, June 2019

Introduction

This document describes the Machine Learning Demo solution, which provides working examples of Machine Learning tasks built using Visual Studio and ML.Net. The objective is to provide basic understand for developers to then embed this logic in their own applications.

The intended audience are developers who are comfortable with Visual Studio & .NET (C#), but who are relative novices when it comes to ML & Data Science.

The document does not go into the theory & in-depth details of individual algorithms; if you're interested in the Maths / Data Science behind the ML algorithms, there are details on the ML.Net website: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet

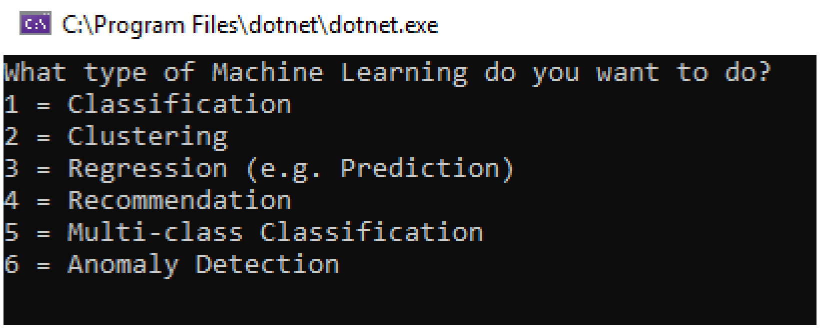
Program Overview

The program is a C# Console Application written in Visual Studio. There are four dependencies installed via NuGet.

- Microsoft.ML – the core ML framework
- Microsoft.ML.FastTree – the algorithm used in the Regression demo
- Microsoft.ML.Recommender – the algorithm used in the Recommendation demo
- Microsoft.ML.TimeSeries – the set of libraries used in the Anomaly detection demo

Running the Program

The application will present you with a choice of one of 6 ML Tasks.



```
What type of Machine Learning do you want to do?
1 = Classification
2 = Clustering
3 = Regression (e.g. Prediction)
4 = Recommendation
5 = Multi-class Classification
6 = Anomaly Detection
```

Once chosen, the program will then ask the user if they want to re-train the model (excluding Anomaly Detection). If a model hasn't already been trained, you should press 'y' otherwise the program will throw an error when it tries to load the existing model. Re-training the model will take some time on some of the tasks, so if you've already built one, and just want to test it, press 'n'.

Machine Learning Tasks

1 - Binary Classification

Classification tasks are used to determine the patterns of features that makes a particular object a certain class of ‘thing’, so that given similar features of a new object in the future, it can predict its class.

Classification can be either Binary or Multi-Class. In this program both are covered, but the option 1 in the program is Binary.

The Data Set

The data set is **twitterdata.csv**. It’s a very large data set containing 1.6 million Tweets, which have been pre-labelled with a sentiment. Either 0 for negative, or 1 for positive. The data set contains a variety of other columns which are not relevant for the ML task.

Binary Classification is supervised, therefore there needs to be both a Train & Test set. The program will ask you what percentage of the data set you want to reserve for Testing. The recommendation is 20%, but you can play around with this value and see how it affects model accuracy.

The Trainers

Binary Classification trainers expect two inputs; a Boolean ‘*Label’* column and a numeric ‘*Features’* column. The Label denotes whether the item is positive or negative, the Features are used to derive a pattern.

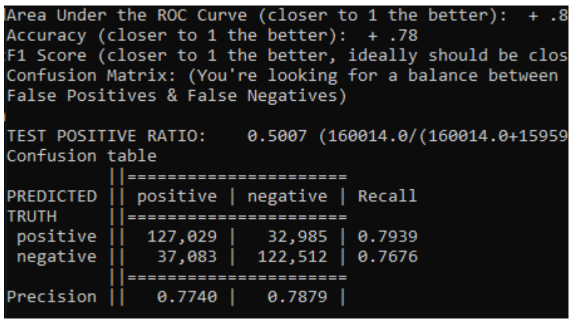
The Process

The Tweet text is ‘Featurised’; i.e. the Text is broken into a numeric chain, breaking the n-grams (groups of words) into hashed values. The result creates the ‘Features’ column, which is then used to train the Model.

Note: given the huge data set in this demonstration, the Model training can take several minutes. The output Model is approximately 30MB in size.

Testing

Model testing produces a set of values and a table called a *Confusion Matrix*. What you’re looking for is balance in the matrix. The Model should have relatively equal sets of Positive and Negative predictions. If one column is highly distorted, something has gone wrong. In terms of the predictions, the Model should be ‘right’ more often than wrong. The four values in the totals row and totals column should be roughly the same. In this example, you can see that the Model is right approximately 75% of the time and there’s a good balance in the columns.

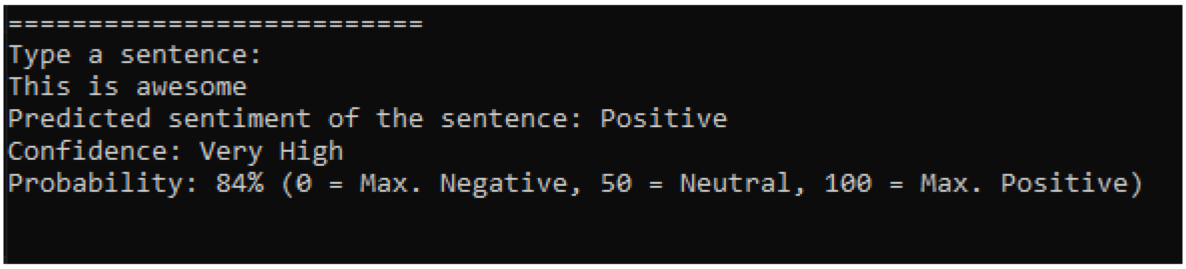


```
Area Under the ROC Curve (closer to 1 the better):  + .8
Accuracy (closer to 1 the better):  + .78
F1 Score (closer to 1 the better, ideally should be clos
Confusion Matrix: (You're looking for a balance between
False Positives & False Negatives)

TEST POSITIVE RATIO:    0.5007 (160014.0/(160014.0+15959
Confusion table
          ||======================
PREDICTED || positive | negative | Recall
TRUTH     ||======================
 positive ||  127,029 |   32,985 | 0.7939
 negative ||   37,083 |  122,512 | 0.7676
          ||======================
Precision ||   0.7740 |   0.7879 |
```

Prediction

The program will offer you the option to pass new values at the model and predict the sentiment. Type some text and observe the output. The program will tell you if the text is positive or negative, but then give you some additional context; a *confidence* and a *probability*. The closer the probability is to 0 the more confident the model is that the text is negative. The closer to 100, the surer it is that its positive. A value close to 50% is a low confidence answer; the Model isn't sure.



```
===========================
Type a sentence:
This is awesome
Predicted sentiment of the sentence: Positive
Confidence: Very High
Probability: 84% (0 = Max. Negative, 50 = Neutral, 100 = Max. Positive)
```

Try a variety of texts to see how the Model responds. You should be able to 'beat' it by mixing together positive and negative words or by trying things like sarcasm.

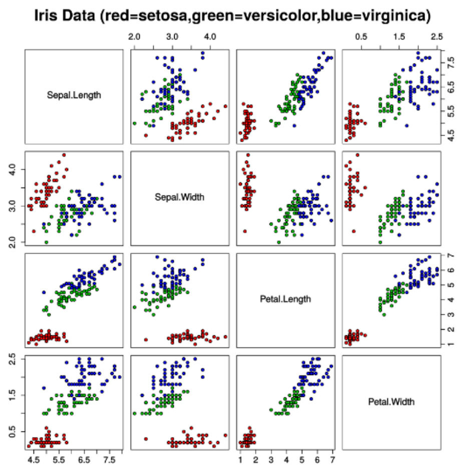
2 – Clustering

Clustering tasks are used to determine the patterns of features in an unlabelled set of data and attempts to group (what it believes are) related data items into groups. It's like Classification, though the big difference is that you're not telling the software, in advance, what the 'correct' answer is. It has to work it out for itself.

The Data Set

The data set is a famous set called Fisher's Iris. Created in 1936 by scientist Ronald Fisher, the data set contains 150 observations of individual flowers, with the widths and lengths of the flower's Petals and Sepals recorded. The flowers belong to one of three species of Iris – Setosa, Veriscolor and Virginca.

The permutations of the four size values is used to create a series of scatter charts. In these charts, you can see the clusters of the three species (represented by colour). Setosa typically stands out in the data, which Veriscolor & Virginca are harder to distinguish.



The data set is in the Data folder and is called **iris.csv**. The image of the scatter plots is also included in the Data folder for reference purposes.

The Trainers

The Clustering trainer uses an algorithm called K-Means to generate a model. K-Means expects a single column input; a numerical column called '*Features*'. You also need to tell the model how many clusters you're expecting in the output.

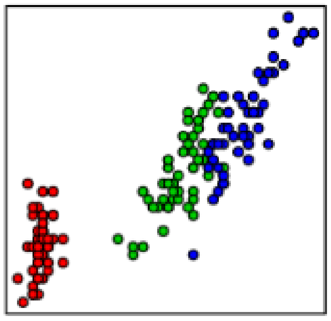
The Process

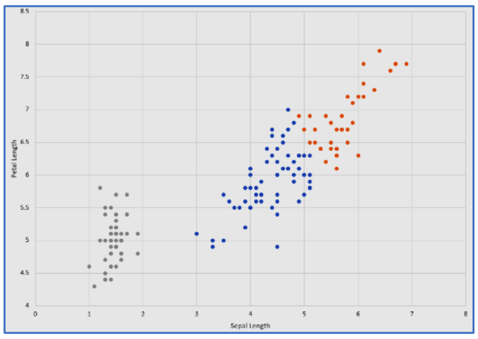
The program concatenates the four flower size values into a single Features column. As the values are already numeric, there's no need to transform them. Note, the fifth column (the species of flower) is ignored.

Testing

With an unsupervised learning task, we don't perform proper statistical testing. But we can observe the results. The program loops back through all 150 flowers and uses the Model to predict which cluster it belongs to. It then writes this out into a file called **clusterresult.csv**.

Open the file and copy the values, then open the Excel file **irisflower_dataviewer.xlsx**. Paste the values from the result file into the green section of the viewer. The graph on the right-hand side shows the relationship between Sepal Length and Petal Length, with the three colours representing the predicted clusters. You can then compare this graph to the actual scatterplot from Fisher's data set.



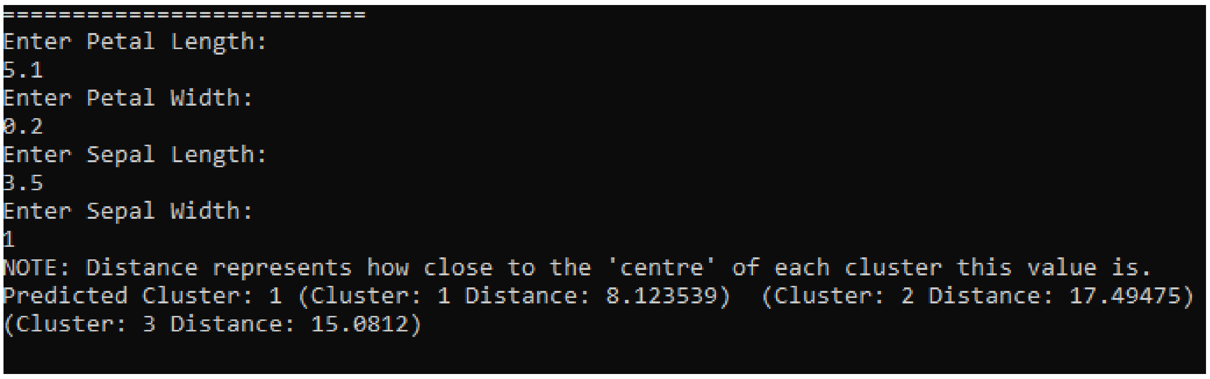


You'll notice that the Model has broadly predicted the correct clusters. The Setosa cluster is identified easily. The Model also fairly accurately delineates the boundary between the Veriscolor & Virginca clusters, though there are some mistakes with Virginca flowers that have relatively short petal lengths.

You can rejig the data-viewer to display the results for different combinations of columns, and you should see similar rates of success.

Prediction

The program gives you the option to introduce a new flower to the Model and predict which cluster it belongs to. You'll be asked to provide values for each of the four variables. Ideally, look at the graph in the data-viewer and enter an x and y value for Petal & Sepal length that will give you a good idea of what the 'right' answer should be.



```
==========================
Enter Petal Length:
5.1
Enter Petal Width:
0.2
Enter Sepal Length:
3.5
Enter Sepal Width:
1
NOTE: Distance represents how close to the 'centre' of each cluster this value is.
Predicted Cluster: 1 (Cluster: 1 Distance: 8.123539)  (Cluster: 2 Distance: 17.49475)
(Cluster: 3 Distance: 15.0812)
```

The result will tell you the predicted cluster, as well as the 'distance' from the centre of each cluster. What you're looking for here is that one of the distances is noticeable shorter than the other two. Go back to the data-viewer, plot the new value, and confirm if the correct cluster has been predicted.

3 – Regression

Regression tasks are used to predict a future value of a 'thing' by determining patterns in the variables of historical instances of the thing. There are numerous types of regression; the option 3 in the program is an example of *Linear Regression*.

The Data Set

The file **taxi.csv** contains over a million records of journeys from a taxi / ride sharing company. The values included are:

- The code of the vendor
- The code of the Rate being used for the journey
- The number of passengers
- The length of the journey is seconds
- The distance of the journey in miles
- How the customer paid (e.g. cash, card, etc.)
- The cost of the fare

The value we're interested in predicting is the fare. Are there patterns in the other attributes that would allow us to predict a future fare?

The program will ask you what percentage of the data set you want to reserve for Testing. The recommendation is 20%, but you can play around with this value and see how it affects model accuracy.

The Trainers

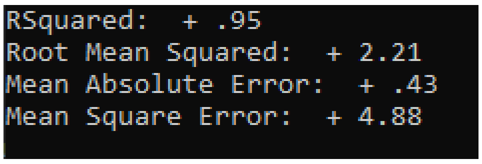
The Regression trainers expect two columns; a numeric '*Label*' column and a numeric '*Features*' column. The Label is the value you're trying to predict (in this case, the Fare), the Features column is used to derive patterns.

The Process

The textual columns (e.g. the vendor and pay type) are encoded into numbers, and then all columns other than the fare are concatenated together into a single numerical Features column. The algorithm (FastTree) is then used to build a Model.

Testing

The Test set you allocated earlier is used to confirm the model accuracy. This includes some of the most common metrics used to validate a regression model.

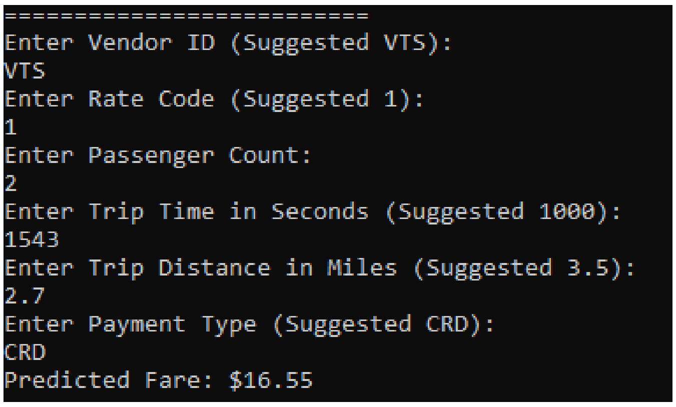


```
RSquared:  + .95
Root Mean Squared:  + 2.21
Mean Absolute Error:  + .43
Mean Square Error:  + 4.88
```

With RSquared, the closer to 1 the better. With the Mean Absolute Error (MAE), the lower the better.

Prediction

The program gives you the option to predict a future value given a set of input parameters. Enter values for a theoretical taxi journey (some suggested values are given) and then look at the predicted fare.



```
==========================
Enter Vendor ID (Suggested VTS):
VTS
Enter Rate Code (Suggested 1):
1
Enter Passenger Count:
2
Enter Trip Time in Seconds (Suggested 1000):
1543
Enter Trip Distance in Miles (Suggested 3.5):
2.7
Enter Payment Type (Suggested CRD):
CRD
Predicted Fare: $16.55
```

Try modifying the values and see how it impacts the prediction. Which values are relevant, and which aren't?

4 – Recommendation

Recommendation tasks are simply Regression tasks; they make predictions of future values based on the patterns in associated variables of historical instances. The difference is based on how they derive the pattern; Recommendation trainers use a matrix of x & y values and look for relationships between the values.

The Data Set

The file **movieratings.csv** contains a list of around 100,000 movie ratings; users (represented as an ID number) rating a movie (represented as an ID number) out of 5. A separate file, **movieref.csv** provides the name of each movie for a given ID. An Access database **movievalidation.accdb** provides a combination of the two files; while not used as part of the Machine Learning task, it can make it easier to validate the output.

The program will ask you what percentage of the data set you want to reserve for Testing. The recommendation is 20%, but you can play around with this value and see how it affects model accuracy.

The Trainers

The Recommendation trainer requires three inputs; a *Matrix Column*, a *Matrix Row*, and a *Label*. The column and row must be in the 'Key' type (which, for example, prevents the same movie appearing in multiple columns). In this context, the Column is user ID, the row is Movie ID and the Label is the Rating.

The Process

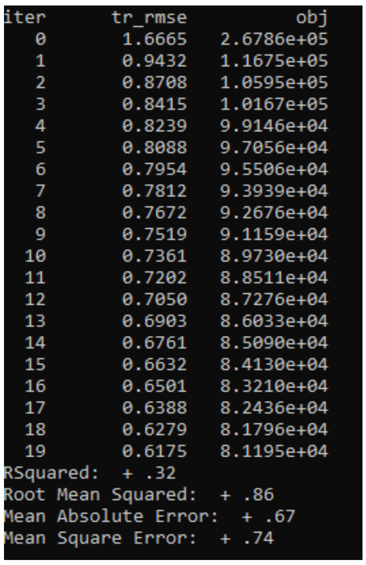
The User ID and Movie ID values are converted into Key types, the Rating value is made the Label column and the Matrix Factorisation algorithm is run. With this algorithm, we perform multiple iterations to refine the model. The number of iterations is a parameter which you will be prompted to enter in the console. The recommendation is 20. There are diminishing returns after a certain number of iterations and running too many iterations can – conversely – negatively affect the model. You can explore this in more detail through trial and error.

In the background, the movie reference list is uploaded into a memory to serve as a lookup for when we want to provide recommendations.

Testing

The Test set you allocated earlier is used to confirm the model accuracy. You will see console output describing the result of each iteration. The tr_mse (Training Mean Square Error) should reduce after each iteration, though its likely you will see the improvements begin to slow down after a handful of iterations. If you attempt to run several hundred iterations, you'll eventually see the value level out.

The final metrics mirror those of the Regression exercise. With RSquared, the closer to 1, the better. With Mean Absolute Error (MAE), the closer to 0 the better.

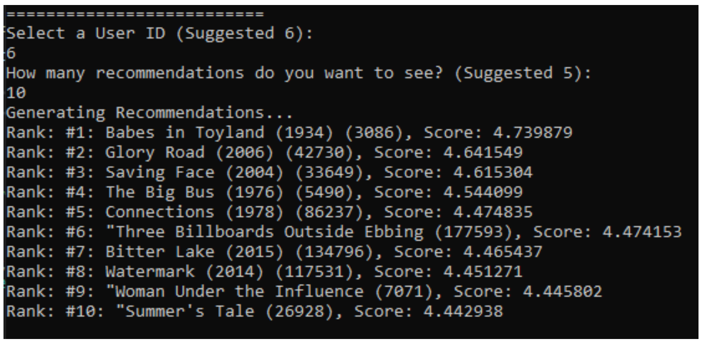


```
iter        tr_rmse            obj
   0         1.6665    2.6786e+05
   1         0.9432    1.1675e+05
   2         0.8708    1.0595e+05
   3         0.8415    1.0167e+05
   4         0.8239    9.9146e+04
   5         0.8088    9.7056e+04
   6         0.7954    9.5506e+04
   7         0.7812    9.3939e+04
   8         0.7672    9.2676e+04
   9         0.7519    9.1159e+04
  10         0.7361    8.9730e+04
  11         0.7202    8.8511e+04
  12         0.7050    8.7276e+04
  13         0.6903    8.6033e+04
  14         0.6761    8.5090e+04
  15         0.6632    8.4130e+04
  16         0.6501    8.3210e+04
  17         0.6388    8.2436e+04
  18         0.6279    8.1796e+04
  19         0.6175    8.1195e+04
RSquared:  + .32
Root Mean Squared:  + .86
Mean Absolute Error:  + .67
Mean Square Error:  + .74
```

Prediction

The solution actually predicts the rating a user would give a movie. To convert this into a usable set of recommendations, we need to run this model against *all* movies, though we can exclude movies the user has already seen. This results in a list of movies and their predicted ratings. We can then order this list in descending order to produce a recommendation list.

In the console, the program asks which user we're recommending for, and how many recommendations we want to see. The program then outputs the top *x* predicted ratings to the screen, looking up the actual name of the movie from the reference list to make it more readable.



```
=========================
Select a User ID (Suggested 6):
6
How many recommendations do you want to see? (Suggested 5):
10
Generating Recommendations...
Rank: #1: Babes in Toyland (1934) (3086), Score: 4.739879
Rank: #2: Glory Road (2006) (42730), Score: 4.641549
Rank: #3: Saving Face (2004) (33649), Score: 4.615304
Rank: #4: The Big Bus (1976) (5490), Score: 4.544099
Rank: #5: Connections (1978) (86237), Score: 4.474835
Rank: #6: "Three Billboards Outside Ebbing (177593), Score: 4.474153
Rank: #7: Bitter Lake (2015) (134796), Score: 4.465437
Rank: #8: Watermark (2014) (117531), Score: 4.451271
Rank: #9: "Woman Under the Influence (7071), Score: 4.445802
Rank: #10: "Summer's Tale (26928), Score: 4.442938
```

Try using different iteration counts and different test data sizes and see how it impacts the recommendations. You can use the Access database in the Data folder to see which movies the user actually saw (and rated) which should give you some idea as to whether these recommendations are reasonable.

5 – Multi-Class Classification

Multi-Class Classification is effectively the same as a Binary Classification except rather than being limited to True / False categories, the items can be categorised against *n.* labels.

The Data Set

The file **githubissues.tsv** contains thousands of technical issues recorded against a project in GitHub. The file contains the following fields:

- Issue ID
- Issue Area (Category)
- Title
- Description

The field we're interested in predicting is Area. Given the title and description of a new issue, can we predict the issue area that it belongs to?

The program will ask you what percentage of the data set you want to reserve for Testing. The recommendation is 20%, but you can play around with this value and see how it affects model accuracy.

The Trainers

The MultiClassification trainers expect two inputs; a '*Label'* column which is a Key and a numeric '*Features'* column. The Label describe the area, the Features are used to derive a pattern.

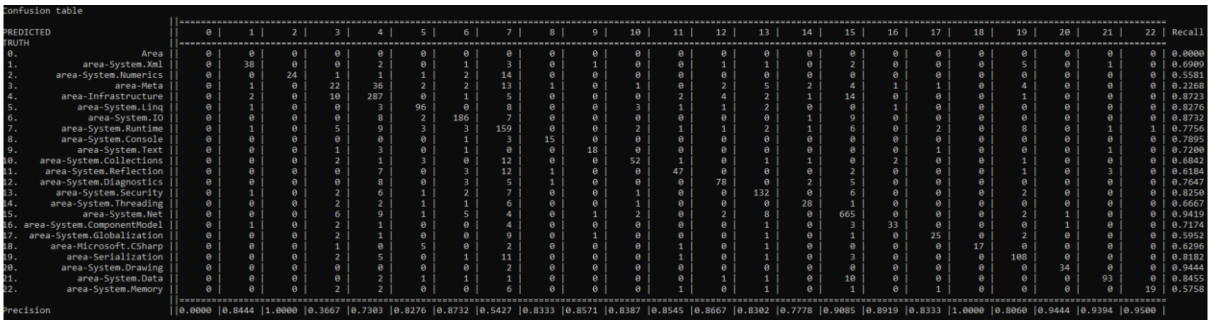
The Process

Both the Title and Description fields are 'Featurised'; i.e. the text is broken into a numeric chain, breaking the n-grams (groups of words) into hashed values. The two newly encoded columns are then concatenated into a single Features column. The Area column is converted to a Key and renamed as a Label. The trainer is executed, and then the Label key is reconverted back into a value (PredictedLabel) so that it can be read.

Given the complexity and size of the data, the Model may take a few minutes to generate.

Testing

The Test set you allocated earlier is used to confirm the model accuracy. Like the Binary Classification, a key test output is a *Confusion Matrix*, although the Multi-Class version is far bigger. In this case, we can see that there were twenty-two areas described in the data, which results in a 22x22 matrix.



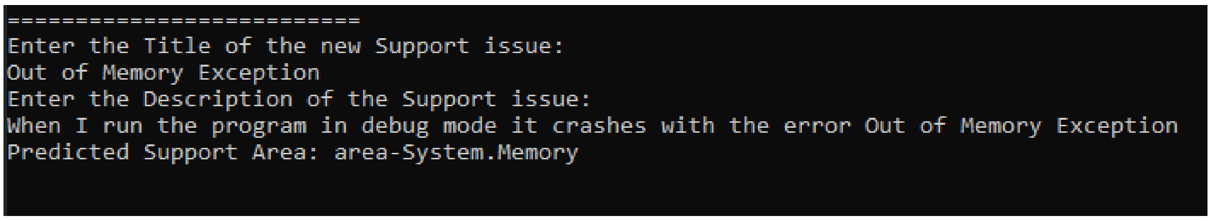
```
Confusion table
                                 ||=========================================================================================================================================================
PREDICTED                        ||     0 |     1 |     2 |     3 |     4 |     5 |     6 |     7 |     8 |     9 |    10 |    11 |    12 |    13 |    14 |    15 |    16 |    17 |    18 |    19 |    20 |    21 |    22 | Recall
TRUTH                            ||=========================================================================================================================================================
 0.                         Area ||     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.0000
 1.              area-System.Xml ||     0 |    38 |     0 |     0 |     2 |     0 |     1 |     3 |     0 |     1 |     0 |     0 |     1 |     1 |     0 |     2 |     0 |     0 |     0 |     5 |     0 |     1 |     0 | 0.6909
 2.         area-System.Numerics ||     0 |     0 |    24 |     1 |     1 |     1 |     2 |    14 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.5581
 3.                    area-Meta ||     0 |     1 |     0 |    22 |    36 |     2 |     2 |    13 |     1 |     0 |     1 |     0 |     2 |     5 |     2 |     4 |     1 |     1 |     0 |     4 |     0 |     0 |     0 | 0.2268
 4.          area-Infrastructure ||     0 |     2 |     0 |    10 |   287 |     0 |     1 |     5 |     0 |     0 |     0 |     2 |     4 |     2 |     1 |    14 |     0 |     0 |     0 |     1 |     0 |     0 |     0 | 0.8723
 5.             area-System.Linq ||     0 |     1 |     0 |     0 |     3 |    96 |     0 |     8 |     0 |     0 |     3 |     1 |     1 |     2 |     0 |     0 |     1 |     0 |     0 |     0 |     0 |     0 |     0 | 0.8276
 6.               area-System.IO ||     0 |     0 |     0 |     0 |     8 |     2 |   186 |     7 |     0 |     0 |     0 |     0 |     0 |     0 |     1 |     9 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.8732
 7.          area-System.Runtime ||     0 |     1 |     0 |     5 |     9 |     3 |     3 |   159 |     0 |     0 |     2 |     1 |     1 |     2 |     1 |     6 |     0 |     2 |     0 |     8 |     0 |     1 |     1 | 0.7756
 8.          area-System.Console ||     0 |     0 |     0 |     0 |     0 |     0 |     1 |     3 |    15 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.7895
 9.             area-System.Text ||     0 |     0 |     0 |     1 |     3 |     0 |     1 |     0 |     0 |    18 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     1 |     0 |     0 |     0 |     1 |     0 | 0.7200
10.      area-System.Collections ||     0 |     0 |     0 |     2 |     1 |     3 |     0 |    12 |     0 |     0 |    52 |     1 |     0 |     1 |     1 |     0 |     2 |     0 |     0 |     1 |     0 |     0 |     0 | 0.6842
11.       area-System.Reflection ||     0 |     0 |     0 |     0 |     7 |     0 |     3 |    12 |     1 |     0 |     0 |    47 |     0 |     0 |     0 |     2 |     0 |     0 |     0 |     1 |     0 |     3 |     0 | 0.6184
12.      area-System.Diagnostics ||     0 |     0 |     0 |     0 |     8 |     0 |     3 |     5 |     1 |     0 |     0 |     0 |    78 |     0 |     2 |     5 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.7647
13.         area-System.Security ||     0 |     1 |     0 |     2 |     6 |     1 |     2 |     7 |     0 |     0 |     1 |     0 |     0 |   132 |     0 |     6 |     0 |     0 |     0 |     2 |     0 |     0 |     0 | 0.8250
14.        area-System.Threading ||     0 |     0 |     0 |     2 |     2 |     1 |     1 |     6 |     0 |     0 |     1 |     0 |     0 |     0 |    28 |     1 |     0 |     0 |     0 |     0 |     0 |     0 |     0 | 0.6667
15.              area-System.Net ||     0 |     0 |     0 |     6 |     9 |     1 |     5 |     4 |     0 |     1 |     2 |     0 |     2 |     8 |     0 |   665 |     0 |     0 |     0 |     2 |     1 |     0 |     0 | 0.9419
16.   area-System.ComponentModel ||     0 |     1 |     0 |     2 |     1 |     0 |     0 |     4 |     0 |     0 |     0 |     0 |     0 |     1 |     0 |     3 |    33 |     0 |     0 |     0 |     1 |     0 |     0 | 0.7174
17.     area-System.Globalization ||     0 |     0 |     0 |     2 |     1 |     0 |     0 |     9 |     0 |     1 |     0 |     0 |     0 |     1 |     0 |     1 |     0 |    25 |     0 |     2 |     0 |     0 |     0 | 0.5952
18.        area-Microsoft.CSharp ||     0 |     0 |     0 |     1 |     0 |     5 |     0 |     2 |     0 |     0 |     0 |     1 |     0 |     1 |     0 |     0 |     0 |     0 |    17 |     0 |     0 |     0 |     0 | 0.6296
19.           area-Serialization ||     0 |     0 |     0 |     2 |     5 |     0 |     1 |    11 |     0 |     0 |     0 |     1 |     0 |     1 |     0 |     3 |     0 |     0 |     0 |   108 |     0 |     0 |     0 | 0.8182
20.          area-System.Drawing ||     0 |     0 |     0 |     0 |     0 |     0 |     0 |     2 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |     0 |    34 |     0 |     0 | 0.9444
21.             area-System.Data ||     0 |     0 |     0 |     0 |     2 |     1 |     1 |     1 |     0 |     0 |     0 |     0 |     1 |     1 |     0 |    10 |     0 |     0 |     0 |     0 |     0 |    93 |     0 | 0.8455
22.           area-System.Memory ||     0 |     0 |     0 |     2 |     2 |     0 |     0 |     6 |     0 |     0 |     0 |     1 |     0 |     1 |     0 |     1 |     0 |     1 |     0 |     0 |     0 |     0 |    19 | 0.5758
                                 ||=========================================================================================================================================================
Precision                        ||0.0000 |0.8444 |1.0000 |0.3667 |0.7303 |0.8276 |0.8732 |0.5427 |0.8333 |0.8571 |0.8387 |0.8545 |0.8667 |0.8302 |0.7778 |0.9085 |0.8919 |0.8333 |1.0000 |0.8060 |0.9444 |0.9394 |0.9500 |
```

What we're looking for is notable increase in values running diagonally from top left to bottom right. This represents a Model which is accurately predicting the right category more times than not. For example, we can see that of the 45 times the Model predicted the category was 1 (area-system.XML), it was right 38 times.

The Precision values and Recall values (in both cases, 1 is best) give an overview. These values appear on the bottom row (precision) and far right column (recall). We're looking for a good balance. In this case, we can see that most categories are scoring in the 80% range for both, though category 3 (area-system.meta) is scoring poorly for both. This suggests the Model is struggling to determine a pattern in the data for issues designated in the "Meta" area. This actually makes sense; Meta is used as a general "catch-all" for issues which don't fit into one of the other areas.

Prediction

The program will offer you the option to create a new issue and predict the area it belongs to. You enter the title and description and the program displays the predicted area.



```
==========================
Enter the Title of the new Support issue:
Out of Memory Exception
Enter the Description of the Support issue:
When I run the program in debug mode it crashes with the error Out of Memory Exception
Predicted Support Area: area-System.Memory
```

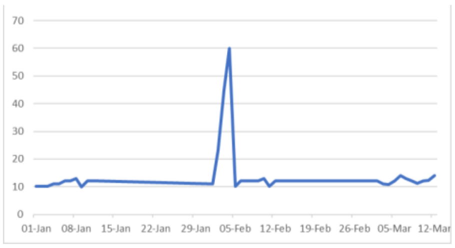
6 – Anomaly Detection

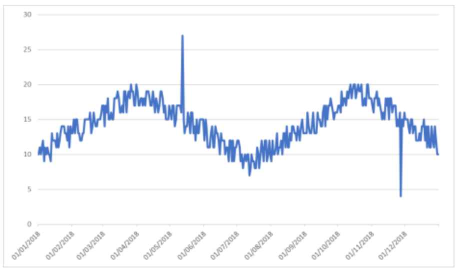
Anomaly detection is a special class of trainers that work on *Time Series* data. In the case of DetectSpike (which is the task used in this program), the Model looks for deviations in the values of a sequential chain of values and generates an alert.

The Data Set

There are two data set files; **anomalydetection.csv** & **anomalydetection2.csv**. You're given the choice on the Console as to whether to pick the small or large data set.

Both files contain two columns of data; a date column (*Day*) and a decimal column (*NumSales*). There are 40 records in the small sent and 365 in the large set, but you're encouraged to enhance this record sets yourself; add additional records and add anomalies.





Note that the shape of the data sets is different; each has at least one anomaly we're trying to detect, but the smaller data set is relatively linear without much noise. In contrast, the large data set has seasonal (cyclical) data pattern, with a lot of day-to-day random noise.

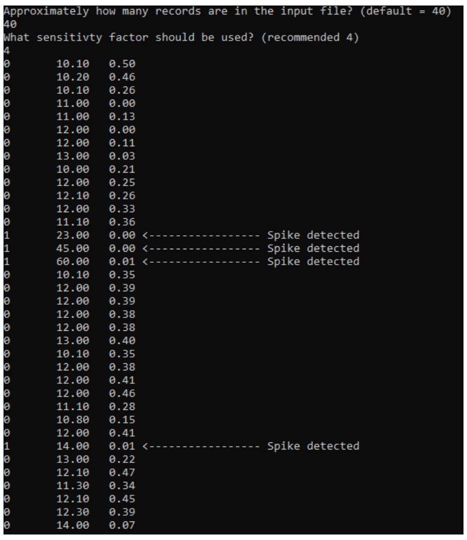
The Trainers

The Anomaly Detection algorithm used is called DetectIidSpike (I.i.d = Independent, identically distributed). It expects one input column which is the numerical value (in this case NumSales). It also expects a *Confidence* value and a *P-Value History Length*. The confidence value is minimum percentage the P-Value (probability value) should be at before indicating a spike has occurred. The P-Value History Length is a 'window size' within the data set used to calculate the P-Value. The recommendation is to set this value at ¼ of the total data set size, but changing this value alters the sensitivity of the detection. The data set size, and the multiplication factor are both offered as user-defined values from the console.

Execution

With this Anomaly Detection demo, we don't test or predict the outcomes; we retrospectively review a data set to look for anomalies. The program writes out the data set to the Console, indicating where it has flagged the detection of a spike. Decreasing the Confidence, setting the Sensitivity higher or understating the data set size will produce more alerts. Increasing the Confidence, reducing the Sensitivity or over-stating the data set size will produce fewer.

In the output to the right, we set the Confidence to 95% on the smaller data set. You can see that the Model has correctly identified the major spike in the data set, but has also identified a second, smaller spike later in the data set. Increasing the Confidence to 99% removes this spike.



```
Approximately how many records are in the input file? (default = 40)
40
What sensitivty factor should be used? (recommended 4)
4
0       10.10   0.50
0       10.20   0.46
0       10.10   0.26
0       11.00   0.00
0       11.00   0.13
0       12.00   0.00
0       12.00   0.11
0       13.00   0.03
0       10.00   0.21
0       12.00   0.25
0       12.10   0.26
0       12.00   0.33
0       11.10   0.36
1       23.00   0.00 <------------------ Spike detected
1       45.00   0.00 <------------------ Spike detected
1       60.00   0.01 <------------------ Spike detected
0       10.10   0.35
0       12.00   0.39
0       12.00   0.39
0       12.00   0.38
0       12.00   0.38
0       13.00   0.40
0       10.10   0.35
0       12.00   0.38
0       12.00   0.41
0       12.00   0.46
0       11.10   0.28
0       10.80   0.15
0       12.00   0.41
1       14.00   0.01 <------------------ Spike detected
0       13.00   0.22
0       12.10   0.47
0       11.30   0.34
0       12.10   0.45
0       12.30   0.39
0       14.00   0.07
```

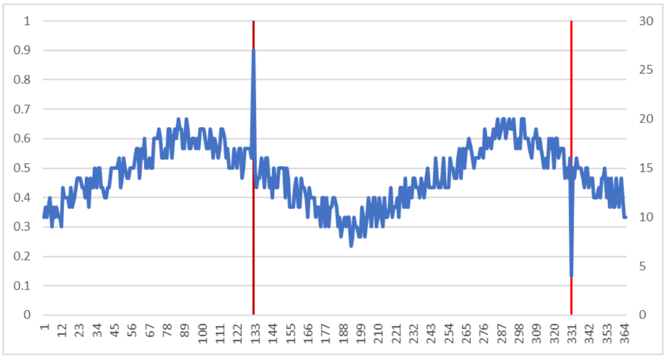
The program also writes the results out to a file in the Data folder called **anomalydetectionresults.csv**. There are three columns in this data set:

- Alert (1 if spike, 0 if everything is fine)
- Score
- P-Value (calculated probability)

Note the Time Series column itself is not output in this data set.

If we run the large data set through the Model at 99% confidence and then output the results, we can plot the combination of the Alert & Score columns in a graph (see image to the right).

The red lines represent the alerts, and you'll notice they align with the two anomalies we'd expect to see.



1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
30
25
20
15
10
5
0
1
12
23
34
45
56
67
78
89
100
111
122
133
144
155
166
177
188
199
210
221
232
243
254
265
276
287
298
309
320
331
342
353
364