**ACS 341 – Machine Learning Technical Report**

**Ruairi McElhatton**

Table of Contents Page No.

Task 1 – Domain Analysis 2

Task 2 – Data cleaning, Pre-processing, and feature engineering 3

Task 3 – Regression model for predicting Torque from RPM 4

Task 4 – Using learning curves to discuss effectiveness of regression model 6

Task 5 – Cross validation technique discussion 8

Task 6 – Applying logistic regression to predict when a machine will break 8

Task 7 – Applying Decision trees to predict when a machine will break 10

Task 8 – Comparing Task 6 and 7 results with two new machine learning algorithms 12

Conclusion 13

Bibliography 14

**Task 1 - Domain analysis**

1. Introduction

The domain I am creating a machine learning pipeline for is predicting machine failure. I have been provided with a data set which includes eight features, and I am to use this to help a company understand why their machines fail and under what conditions. In precision machining processes surface quality of the manufactured part can be related to the conditions of the cutting tools, therefore, many companies in this industry are interested in “in-process tool conditioning monitoring (TCM) systems”.

1. Glossary

* Product ID: This is the machine ID.
* Type: This feature relates to the type of machine.
* Air Temperature [K]: This is the air temperature around the machine measured in Kelvin (K).
* Process temperature [K]: This is the temperature, measured in Kelvin (K), that is generated by the machine during operation.
* Rotational speed [rpm]: Measured in Revolutions per minute, this is the speed the machine’s spindle runs at.
* Torque [Nm]: Is the measure of force used to generate angular momentum during rotations by the machine.
* Tool wear [min]: This measures the gradual failure of parts of the machine due to regular operation
* Machine failure: This indicates if the machine has failed or not.

1. General knowledge about the domain

Tool wear is an important factor which affects the machined surface characteristics. However, it is impossible to directly measure tool wear without interrupting the machining process. CNC operators can measure it via visual inspection of cutting edges or by online measures. More modern unmanned CNC machines may use TCM as an approach to indirectly detect the tool wear level in machining. This kind of tool wear is different to the tool wear listed in the glossary. This may be from measuring other factors using different sensors such as a thermometer, an accelerometer, and others.   
In my research of the domain, I found that torque and rpm are related by moment of inertia, this is something that I will use in the feature engineering stage of my pipeline.

1. Competing Software

There are already models which have been made for the purpose of tracking tool wear to make product quality more consistent. For example, an artificial neural network model has been made using MATLABs neural network toolbox. The analysis of this model once parameters had been altered showed it to be reliable in predicting the tool wear.

1. Similarities across domains and organizations:

This domain could be easily recreated by other companies who wish to use the software as the parameters/features used in my model aren’t too hard to measure using various sensors.

**Task 2**

One of the things I did in the pre-processing stage was to encode the columns specifying product type and machine failure. This is so the specified data in these columns could be used in machine learning algorithms. I did this encoding in excel as I found it easier on that software compared to MATLAB or orange. The key to this encoding is in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Product type code:** | |  | **Failure code:** | |
| Low(L) | 0 |  | Failure | 0 |
| Medium(M) | 1 |  | No Failure | 1 |
| High(H) | 2 |  | NA | 2 |

After this encoding, I imported the new data file into MATLAB. It is attached along with the MATLAB code and the orange pipeline. It is called ‘EncodedData.csv’.

In the data cleaning part of the operation, I decided to remove the product ID column as this was irrelevant to what we were trying to achieve. More specifically, it gave additional unnecessary information, the product type will be useful but the code after the letter will not be when trying to find relationships between feature classes.

Another aspect which required cleaning was removing the rows of data which had missing information, for this I used the ‘rmmissing’ function in MATLAB. This is known as imputing, I decided to do this to remove any chance of bias.

In the next part of my data cleaning process, I looked for outliers in the data this was to avoid them affecting the results of my model in a bad way. However, I decided against following this course of action as there were no obvious outliers when I plotted all the variables in one figure using a function made for us in one of the labs. This is shown below. Additionally, when I did try to use the ‘filloutliers’ function in MATLAB for task 3, it would skew the data and create new obvious outliers. This also happened when I tried to use Orange for the same purpose. This data, supposedly without outliers, led to less accurate models being built.

Text

Description automatically generated  
**All data from original features plotted**

Another key part of the pre-processing stage in the machine learning pipeline was the standardisation of the data. I chose to standardise between 0 and 1 as it made the most sense given how I had encoded machine failure, therefore it was easy to draw a direct link from the contributing features. This was needed because a lot of different machine learning methodologies require standardised data. Also, when I performed task 3 with and without the data being standardised, it performed much better when standardised.

Another pre-processing technique I used was to discretise the data. I compared the accuracy of my model with the discretised data and without it to see if it was worth the extra processing and decided that it was well worth it due to a large increase in the accuracy scores of all my models.

Another step I did for this task was some feature engineering. Using my understanding of the domain, I was able to combine torque and RPM to create a single feature, moment of inertia. These features are connected by the equation Torque/RPM = moment of inertia (MOI). I calculated this in excel using the formula feature.

Finally, when sampling the data, I saw that the large product type was underrepresented and therefore I decided to ‘upsample’ it. This would mean that the classes would be more balanced and therefore the model would be better prepared to deal with the minority class.

**Task 3**

For this task I took the MATLAB script given to us in lab 4. I then adapted the script to suit my needs. I made RPM x, the dependent variable. And Torque y as I was trying to build a regression model to predict it.

Having the degree of the polynomial at 1 and 2 was visibly inaccurate so I went against this. The errors were also higher in the learning curve. The below plot shows the predictive capability of my hypothesis function.

Graphical user interface, chart, scatter chart

Description automatically generated  
**Predictions using 3rd degree polynomial model (optimal)**

As you can see above, the predictions follow the data set quite well and this is a good sign. The above data is the standardised data.

Below, I have shown what the predictions look like first for a linear model and then for a model with a high polynomial degree of 30.

Chart, scatter chart

Description automatically generated  
**Linear regression model**

Figure above showing linear regression model predictions. We can see a prime case of underfitting here. The linear model does not have the capabilities to follow the curve of the relationship between RPM and torque. This model would not be able to generalise to new data.

Graphical user interface, chart, scatter chart

Description automatically generated  
**30th degree polynomial regression model**

Figure above showing polynomial regression model with polynomial degree of 30. Some of the predictions are majorly skewed and this could be due to overfitting. This will lead to the model not being able to recognise new data instances even though the data is part of the domain.

**Task 4**

As you can see below, both errors converge to a small value as the training sample size increases. The degree of polynomial I ended up opting for was 3 because if I went any higher, even though both errors would end at an even lower point, several errors would occur in MATLAB saying the polynomial was badly conditioned. The troubleshoot would tell me to reduce the degree of the polynomial to correct this. As seen above with how the model became overfitted, this error message was correct.

Application

Description automatically generated with low confidence  
**Figure showing the learning curve for a polynomial function with degree 3 before standardisation of the data**

Below is the standardised learning curve:

Graphical user interface, application, table

Description automatically generated  
**Standardised data learning curve**

As you can see there is a significant decrease in both training and test error after data has been standardised.

After I cross validated the model using leave one out cross validation, the learning curve looked like the image below:

Graphical user interface, application, Word

Description automatically generated  
**Standardised data learning curve from leave one out cross validation**

I then zoomed in on the y-axis to get a closer view as there is a very high start for the test rmse. By zooming in you can get a better view of how the polynomial is a good fit.

Graphical user interface, application, Word

Description automatically generated  
**Zoomed in version of standardised data learning curve from leave one out cross validation**

The almost identical finishing position of the two curves shows that the model is a great fit. This shows a balanced model, where both errors converge to small values.

**Task 5**

The cross-validation technique I used above to give me the second learning curve was leave one out. It is a more computationally demanding version of K-fold, but this was not a large enough deterrent from using it on this model as the data set wasn’t too large. If the data set had been massive it may have been more relatively beneficial to use K-fold cross validation.

I chose this technique as I thought it suited validating a regression model the best as it would be the most rigorous, least biased and therefore the most accurate.

**Task 6**

Predicting machine failure using logistic regression. I decided to create my machine pipeline in Orange for this task as there were some steps which seemed much easier here than in MATLAB. I initially made a model which involved giving the various features a weighting based on their correlation coefficient the with machine failure outcome. However, the accuracy of this model was not great. It was about 0.6 and I decided this was not too useful. Therefore, I changed my model significantly.

The input file below is not the original dataset we were given. It is my final dataset which I made in MATLAB in Task 2. I have attached it along with the orange pipeline and the MATLAB code. It is called ‘WeightedData.xlsx’.

Diagram

Description automatically generated  
**Machine learning pipeline in orange for logistic regression**

Above is the pipeline shown in Orange, but it is not the whole pipeline as I performed some steps before here. These steps are outlined in Task 2. I found the steps of the pipeline not visible here easier to perform in MATLAB.

I used the feature statistics widget to determine whether I needed to perform log transform by checking if my data was skewed. It wasn’t significantly skewed so I decided against using the log transform.

The next operation I did in this pipeline was to check the correlation coefficients of all the features with reference to the machine failure feature. I used this to create a machine learning model using weightings from the correlations, but it wasn’t very accurate as discussed in task 2 section.

Beyond this I used the pre-process widget to do the following operations.Graphical user interface, text, application

Description automatically generated

First I normalised the features between 0 and 1 because I had already encoded machine failure and no failure to be 0 and 1 respectively. Therefore, I thought it made sense to get all the other variables to the same magnitude range.

Secondly I Discretised all the continuous variables with 5 intervals each. Too many intervals would almost defeat the point of discretising but not enough would make the model less accurate. Discretising the variables led to a huge increase in all the performance characteristic and therefore it seemed foolish to leave this step out.

The test and score widget below shows high accuracy and precision and recall:

Graphical user interface, application, table

Description automatically generated with medium confidence  
**Scores for Logistic regression**

The high AUC (area under curve) value tells us that the model can predict the correct outcome almost all of the time. High precision means that an algorithm returns more relevant results than irrelevant ones, and high recall means that an algorithm returns most of the relevant results (whether or not irrelevant ones are also returned).

Lastly, here is the confusion matrix:

A picture containing chart

Description automatically generated  
**Confusion matrix for the logistic regression model**

**Task 7**

I then had to use decision trees to provide a transparent explanation of my machine learning pipeline. The only thing I changed from the logistic regression pipeline is the model widget. The new pipeline is shown below.

Diagram

Description automatically generated  
**Pipeline in orange for decision tree**

The tree viewer showed the following:

Diagram

Description automatically generated  
**The decision tree depicted**

As we can see, a high air temperature would mean a certain machine failure. A low MOI would then result in failure too. That seems to split the data effectively and predict the outcome very well.   
The predictions can be seen below with the confusion matrix:

Table

Description automatically generated  
**Confusion matrix for decision tree**

The scores for the logistic regression model and the tree are compared below:

Graphical user interface, text, application, table

Description automatically generated with medium confidence  
**Compared scores of Decision tree and logistic regression**

As we can see, both are very good, but the Logistic regression model is marginally better all round.

**Task 8**

**Diagram

Description automatically generated with medium confidence  
Pipeline with all 4 machine learning algorithms**

Depicted above is my machine learning pipeline in orange with two other algorithms we haven’t covered in class. These two algorithms are neural network and K-Nearest-Neighbour (kNN). Below is the table showing how each algorithm compares.

Text, table

Description automatically generated  
**Scores from all 4 machine learning algorithms**

As we can see, the neural network algorithm has the highest scores across the board and the kNN model has the second highest scores.   
Neural networks (NNs) can use up a lot of computational power, but because the required level of power is becoming increasingly available today, this was no problem for me. NNs are often useful because they use a combined logical approach. One part of this is that they will accept that its prediction will not always be correct, that the prediction is just a best guess. Mathematically, a NN prediction is based off a weighted average of all factors. The hardest part of this is assigning the weightings [2]. A NN will alter weightings to make sure that the answer at the output is always correct. The larger the dataset, the more useful and accurate the predictions are. It would seem, since the NN has the highest scores across the board, that the data set is large enough for it to be effective.   
The quality of the outcome of a NN depends greatly on the quality of the data used in the learning phase [2]. The fact that the scores from the neural network is high points to my data cleaning, pre-processing and feature engineering being very good.

kNN models are relatively simple and easy to implement. The difficulty level of implementation is irrelevant when using orange as all of this is done for you. kNN is a non-parametric algorithm which means there are no assumptions to be met to implement kNN. This makes life easier. kNN models also constantly evolving and therefore by the time the kNN algorithm has processed all the data it is at its most accurate in terms of classification.  
There could be several reasons why the kNN model is not as accurate as the NN model. One of these reasons could be that I haven’t balanced the data well enough. kNN models are more sensitive to unbalanced data than NN models. As stated above in task 2, I had tried to balance the data as the machine types feature was quite severely imbalanced. However, I may not have done enough ‘upsampling’ and therefore the kNN algorithm may have favoured the classes with more representation which has led to inaccuracies. kNN algorithms are also very sensitive to outliers. I have tried to address this but have not really found any significant outliers in the data and therefore I don’t think this is the reason for the minor shortcomings of kNN as a machine learning technique when compared to NNs.

A lot of my work done before actually implementing the machine learning method led to these two new methods being far more effective than they otherwise might have been as both methods have weaknesses dealing with raw data.

**Conclusion**

After using four different machine learning algorithms I saw first hand how important the early steps of ML pipelines are. By carrying out domain analysis, cleaning the data, pre-processing the data and feature engineering, I was able to set up the ML algorithms to perform optimally.

One part of this pre-processing stage that I may have been able to do better was to find and remove outliers but I was aware of this issue throughout and stand by my decision not to remove any as none were obvious. Another part that I wanted to include was to perform log transform on skewed data. Although some of the data was a bit skewed, using log transform was not helpful.

Overall, I thought my models worked well and were very accurate. I have said above what may have been improved but I don’t think there was much that I missed in terms of creating a good, well-rounded ML pipeline.

**Bibliography**

[1] Bulent Kaya, Cuneyt Oysu, Huseyin M.Ertunc, Force-torque based on-line tool wear estimation system for CNC milling of Inconel 718 using neural networks. <https://www.sciencedirect.com/science/article/pii/S0965997810001663#f0010>

[2] Passioned. <https://www.passionned.com/bi/predictive-analytics/neural-network/>

[3] Genesis. (September 25, 2018). *Pros and Cons of K-Nearest Neighbors* [online]. Available. URL: <https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/>