

Irish Senthilkumar 16342613 4BCT ML Assignment 3

The ML Software Package

Weka was chosen as the ML software package for this assignment and the previous assignments, due to a number of reasons:

- It has an intuitive GUI for operation and data visualization. The pre-process window has many features, including metrics that help to visualize the dataset such as the mean, count, min and max. Graphs are also included to show the distribution of the dataset, as shown in Figure 1.

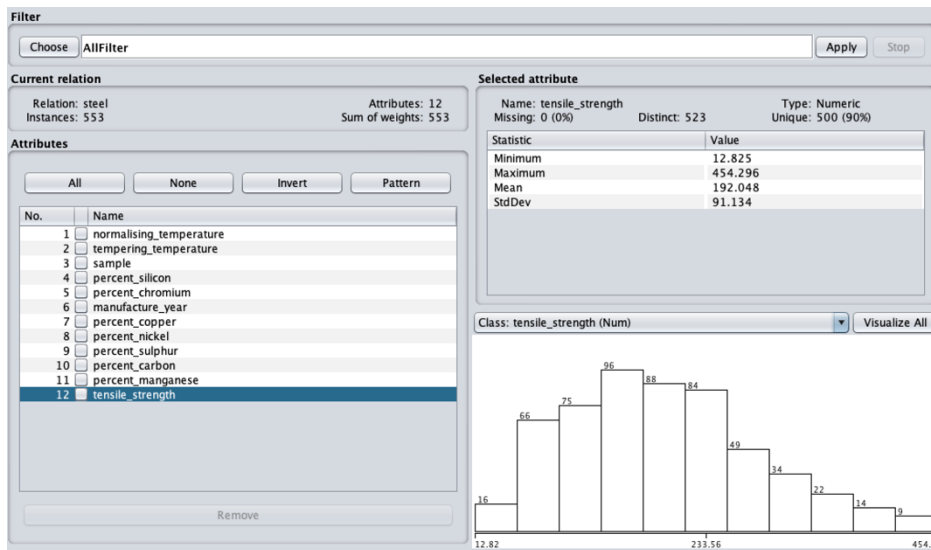


Figure 1: The pre-process window in Weka.

- Weka is platform independent and can therefore run on many different operating systems since it is a Java SE application. I frequently work on both MacOS and Windows machines interchangeably, and this feature is important to me.
- Weka has numerous tools for data preparation and classification. Many classification algorithms are available pre-installed, but even more can be downloaded through the Weka package manager. The classifier output provides useful information such as the computation time, error rates, confusion matrices (for discrete dependent variables) and correlation coefficients (for continuous dependent variables), as shown in Figure 2.

```
=== Classifier model (full training set) ===  
  
IB1 instance-based classifier  
using 2 nearest neighbour(s) for classification  
  
Time taken to build model: 0 seconds  
  
=== Cross-validation ===  
=== Summary ===  
  
Correlation coefficient          0.8807  
Mean absolute error             33.8301  
Root mean squared error         43.5669  
Relative absolute error         46.0916 %  
Root relative squared error     47.7815 %  
Total Number of Instances      553
```

Figure 2: The classifier output for KNN.

Data Preparation

Weka can accept many different file formats for the dataset, but I selected ARFF as the input format due to the straightforward conversion of a table into ARFF. I developed a C# script that would read a text file and populate an ARFF file appropriately. The C# script is quite simple and is included in the appendix of this report.

Firstly, the parser object is created with the column names and their data type, the relation name, and the input file name inside the Assets folder of the project. Setting up the reading of the file is just a simple process where the input file path is computed and the column names for the table is stored. Now, the file can be read, with the rows being separated by new lines, and the columns being separated by the tab escape sequence. Using this separation logic, we can easily isolate each attribute value from the text file and store it inside a DataTable. Datatables are C# objects that can provide a database table representation. Once the DataTable is stored, we can normalise it using range normalisation. Afterwards, we can simply write the contents of the DataTable into another text file using the PopulateOutputFile method, and save this new text file as an ARFF file. The leading lines in the ARFF are shown in Figure 3. Once this output ARFF file has been created, we can reference it in Weka.

```
@RELATION      steel
@ATTRIBUTE     normalising_temperature REAL
@ATTRIBUTE     tempering_temperature  NUMERIC
@ATTRIBUTE     sample                 NUMERIC
@ATTRIBUTE     percent_silicon        REAL
@ATTRIBUTE     percent_chromium       REAL
@ATTRIBUTE     manufacture_year       NUMERIC
@ATTRIBUTE     percent_copper         REAL
@ATTRIBUTE     percent_nickel         REAL
@ATTRIBUTE     percent_sulphur        REAL
@ATTRIBUTE     percent_carbon         REAL
@ATTRIBUTE     percent_manganese      REAL
@ATTRIBUTE     tensile_strength       REAL

@DATA
0,0.0689655172413793,0.633624878522838,0.425709515859766,0.214416399587165,0.3809523809523
81,0.991452991452991,0.265432098765432,0.496336996336996,0,0.0278221580812412
```

Figure 3: The first lines of the ARFF file.

Algorithms

Many different regression algorithms were available, but I chose linear regression and k-nearest neighbours to develop a model.

Linear Regression:

Linear regression is a ‘model-fitting’ algorithm which can be used to create an equation that best describes the target in terms of the attributes. Linear regression can be carried out in two ways – the closed form solution, and the gradient descent approach. Gradient descent is the widely used approach, since the closed form solution is computationally expensive [1]. Given the set of attributes $[x_1, x_2, x_3 \dots]$, we need to produce a set of attribute coefficients $[\theta_1, \theta_2, \theta_3 \dots]$, such that $\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots$ results in an accurate estimation of the target. We need to use a loss function, the sum of the squared residuals, to formalise this:

$$SquaredResidual(\theta) = (ActualValue(\theta) - PredictedValue(\theta))^2$$

We need the coefficients whose values give the smallest cost i.e. the smallest sum of the squared residuals. We firstly choose a reasonable random value for the coefficients. Then we calculate the derivative of the loss function with respect to each of the coefficients separately, e.g. for θ_1 :

$$SlopeOfSumOfSquaredResiduals_{\theta_1} = \frac{d}{d\theta_1} SumOfSquaredResiduals(\theta_1, \theta_2, \theta_3, \dots)$$

The closer we get to the optimal value for θ_x , the closer the slope of the sum of the squared residuals gets to zero. In order to determine what value for θ_x to use next, we need to find out the step size:

$$StepSize_{\theta_x} = (SlopeOfSumOfSquaredResiduals_{\theta_x})(LearningRate)$$

The learning rate is a predetermined number which directly influences the increment/decrement in the current value of θ_x . We need to manually set this to a reasonable value to avoid overshooting the minimum point of slope of the sum of squared residuals. After adding the step size to θ_x , we can once again calculate the derivative of the loss function with respect to each of the coefficients. The step size should get smaller and smaller as we approach the minimum point. Once the step size is below a certain threshold or the number of step size calculations grows too high, we can stop the calculations and use the latest values for the coefficients $[\theta_1, \theta_2, \theta_3 \dots]$. This gives an accurate hypothesis function.

K-Nearest Neighbours:

K-Nearest neighbours (KNN) divides the feature space into a Voronoi tessellation, which create local models (neighbourhoods) defined by the training set. Firstly, we populate the feature space with the training data. Then we need to compute the distance metric (usually Euclidean) from the query case to the stored instances.

$$EuclideanDistance(a, b) = \sqrt{\sum_{i=1}^m (a[i] - b[i])^2}$$

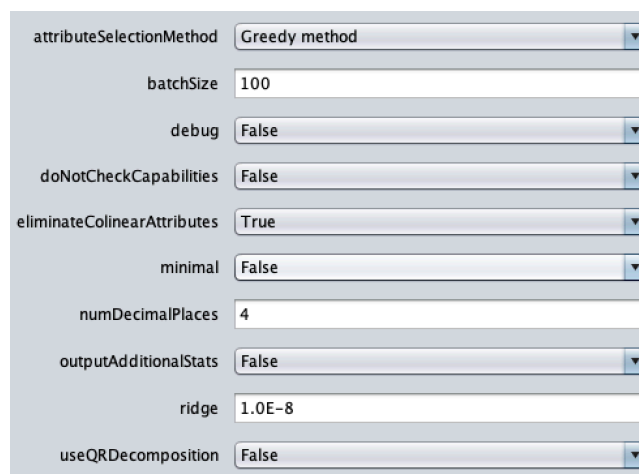
Now we sort the distances and choose the K instances with the smallest distances for the next step. These K instances 'vote' on the classification of the query case. We can use distance-weighted KNN for this step, since this limits underfitting by limiting the 'voting power' of stored instances that are further away from the query case than closer ones. Now we can accurately classify the query case.

Model Development

The data was divided using 10-fold cross validation. This means that the full dataset was split into 10 stratified sections. We keep 9 random sections as the training set, and the last section as the testing set. This is repeated 10 times, using a different section as the testing set for each time, and the final result is averaged.

Linear Regression:

Linear regression was executed using Weka's *LinearRegression* function. This function had 10 tuneable parameters as shown in Figure 4, although only a few as explained below would make any noticeable difference [3].



The image shows a screenshot of the 'LinearRegression' parameter settings window in Weka. The window has a light blue background and contains ten parameters, each with a label and a control element (either a text box or a dropdown menu). The parameters and their values are: 'attributeSelectionMethod' (Greedy method), 'batchSize' (100), 'debug' (False), 'doNotCheckCapabilities' (False), 'eliminateColinearAttributes' (True), 'minimal' (False), 'numDecimalPlaces' (4), 'outputAdditionalStats' (False), 'ridge' (1.0E-8), and 'useQRDecomposition' (False).

Parameter	Value
attributeSelectionMethod	Greedy method
batchSize	100
debug	False
doNotCheckCapabilities	False
eliminateColinearAttributes	True
minimal	False
numDecimalPlaces	4
outputAdditionalStats	False
ridge	1.0E-8
useQRDecomposition	False

Figure 4: The parameters for Weka's linear regression.

The *attributeSelectionMethod* parameter is a value that instructs Weka to perform feature selection. This was set to *Greedy Local Search*, although this only resulted in 0.15% more correlation and a similar reduction in RMSE over disabling feature selection, and an even lower increase in correlation over using the M5 method for feature selection.

The *eliminateColinearAttributes* parameter is a value that allows Weka to detect and remove highly correlated attributes. Changing this parameter had no effect on correlation, because the correlating attributes were already removed during feature selection.

The *ridge* parameter is used to tune the ridge regression, a technique which detects if the data is suffering from multicollinearity, when the variances of the least squares is unbiased but far from the true value [4]. I left this parameter as the default, since changing it had no effect on the correlation. After tuning the parameters as best as I could, the final classifier output was 90.76% correlation with RMSE of 0.0866 (due to the range normalisation), as shown in Figure 5. Even if underfitting and overfitting was present in the model, there were not enough tuneable parameters to rectify the issue.

```
Linear Regression Model

tensile_strength =

    0.5843 * normalising_temperature +
    0.34   * tempering_temperature +
   -0.1612 * sample +
    0.3681 * percent_silicon +
   -0.3152 * percent_carbon +
    0.1139 * percent_manganese +
    0.0475

Time taken to build model: 0.01 seconds

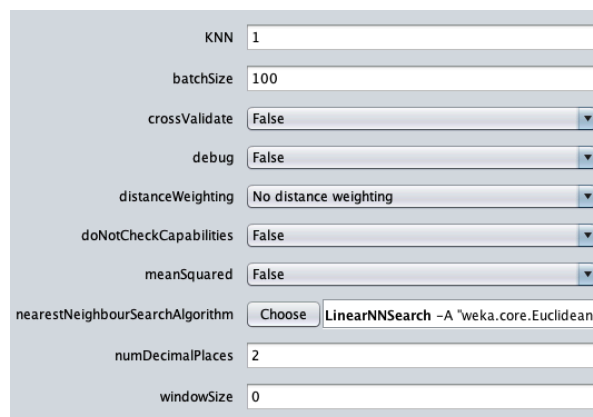
=== Cross-validation ===
=== Summary ===

Correlation coefficient      0.9076
Mean absolute error         0.0701
Root mean squared error    0.0866
Relative absolute error     42.1567 %
Root relative squared error 41.9211 %
Total Number of Instances   553
```

Figure 5: Linear Regression Output.

K-Nearest Neighbours:

KNN was executed using Weka's *IBk* function. This function also has 10 tuneable parameters as shown in Figure 6, and once again only a few would make any noticeable difference to classifier accuracy [5].



KNN	1
batchSize	100
crossValidate	False
debug	False
distanceWeighting	No distance weighting
doNotCheckCapabilities	False
meanSquared	False
nearestNeighbourSearchAlgorithm	Choose LinearNNSearch -A "weka.core.Euclidean"
numDecimalPlaces	2
windowSize	0

Figure 6: The parameters for Weka's KNN.

The *KNN* attribute sets the number of nearby neighbours to consider. This is the most important attribute to tune, as this has a significant effect on the classification accuracy. Weka can automatically set a value for *k* based on analysis if the *crossValidate* parameter is set to true, but I set this to false for experimentation and to have more control over the model. As shown in Figure 7, *K* = 4 gave the highest correlation and lowest RMSE. Having a very high value for *K* gives rise to underfitting, as too many neighbours are considered. A value for *K* that is too low can also give rise to overfitting if the training set is noisy.

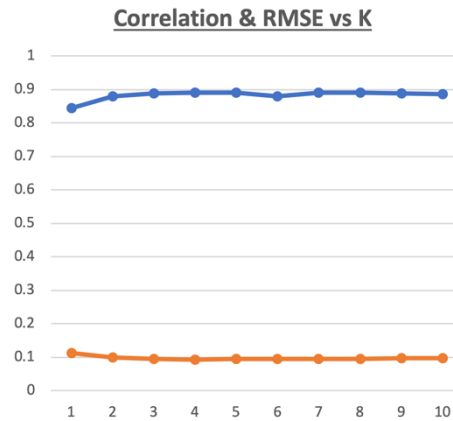


Figure 7: Graphing correlation and RMSE versus the size of K.

The *distanceWeighting* parameter is a value that instructs Weka to perform distance-weighted KNN on the training data. This was set to true to prevent stored instances far away from the query case from skewing the classification, and therefore reducing underfitting.

The *nearestNeighbourSearchAlgorithm* parameter decides the distance metric used in the model. Using Manhattan distance resulted in 1.7% higher correlation and similarly lower RMSE than using Euclidean distance. Using Minkowski distance with order greater than 2 significantly degraded correlation and increased RMSE, so Manhattan distance was the chosen distance metric. After tuning the parameters as best as I could, the final classifier output was 91.24% correlation with RMSE of 0.0846 (due to the range normalisation), as shown in Figure 8.

```
IB1 instance-based classifier
using 4 inverse-distance-weighted nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient          0.9124
Mean absolute error             0.0651
Root mean squared error         0.0846
Relative absolute error         39.1502 %
Root relative squared error     40.9672 %
Total Number of Instances       553
```

Figure 8: KNN output.

Model Comparison

The linear regression model had 90.76% correlation and 0.0866 RMSE, while the KNN model had 91.24% correlation with 0.0846 RMSE, implying that KNN was slightly more accurate than linear regression. I believe that linear regression was slightly worse than KNN here because the data may not have had a linear relationship, but linear regression assumes that the data has a linear relationship. Polynomial regression may have been a better choice here. KNN on the other hand does not make the assumption that the data has a linear relationship, and was therefore more accurate.

References

- [1] Aksakal, *Why use gradient descent for linear regression, when a closed-form math solution is available?*, viewed 20 November 2019
<<https://stats.stackexchange.com/questions/278755/why-use-gradient-descent-for-linear-regression-when-a-closed-form-math-solution>>
- [2] StatQuest with Josh Starmer, *Gradient Descent, Step-by-Step*, viewed 20 November 2019
<<https://www.youtube.com/watch?v=sDv4f4s2SB8>>
- [3] Jason Brownlee, *Linear Regression*, viewed 25 November 2019

<<https://machinelearningmastery.com/use-regression-machine-learning-algorithms-weka/>>

- [4] NCSS.com, *Multicollinearity*, viewed 25 November 2019
<https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf>
- [5] Jason Brownlee, *K-Nearest Neighbours*, viewed 27 November 2019
<<https://machinelearningmastery.com/use-regression-machine-learning-algorithms-weka/>>

Appendix

```
#region Information
// Name : Irish Senthilkumar
// ID : 16342613
// Module : Machine Learning
// Module Code : CT4101
// Module Name : Machine Learning
// Date : 20/11/19
#endregion

using System;
using System.IO;
using System.Data;
using System.Collections.Generic;
using System.Linq;

namespace ML_ExcelParser_Assignment_1
{
    public class TextToARFFParser
    {
        private string filePath;
        private string textInFile;
        private string relationName;
        private DataTable data = new DataTable();
        private Dictionary<string, string> columnMappings;
        private const string outputFileName = "Output_File.arff";

        /// <summary>
        /// The constructor
        /// </summary>
        /// <param name="filePath"></param>
        /// <param name="columnNames"></param>
        public TextToARFFParser(string filePath, string relationName, Dictionary<string, string> columnMappings)
        {
            this.relationName = relationName;

            SetUp(filePath, columnMappings);
        }

        /// <summary>
        /// Sets up reading the text file
        /// </summary>
        /// <param name="filePath"></param>
        /// <param name="columnNames"></param>
        public void SetUp(string filePath, Dictionary<string, string> columnMappings)
        {
            this.filePath = filePath;
```

```

this.columnMappings = columnMappings;

Directory.SetCurrentDirectory("../Assets/");

foreach (string columnName in this.columnMappings.Keys)
{
    data.Columns.Add(columnName);
}
}

/// <summary>
/// Read the file and convert it to a Datatable
/// </summary>
public void ReadFile()
{
    textInFile = System.IO.File.ReadAllText(filePath);
    textInFile = textInFile.Replace("\r", "");

    string[] attributesList = textInFile.Split(Environment.NewLine.ToCharArray());

    // Loop through the rows
    for (int i = 0; i < attributesList.Length; i++)
    {
        string[] attributes = attributesList[i].Split('\t');

        if (attributes.Length != columnMappings.Count)
        {
            continue;
        }

        data.Rows.Add(data.NewRow());
        object[] rowItemArray = new object[attributes.Length];

        // Loop through the columns
        for (int j = 0; j < attributes.Length; j++)
        {
            rowItemArray[j] = attributes[j];
        }

        data.Rows[i].ItemArray = rowItemArray;
    }
}

/// <summary>
/// Normalise the data
/// </summary>
public void Normalise()
{
    for (int i = 0; i < data.Columns.Count; i++)
    {
        if (columnMappings[data.Columns[i].ColumnName] != "REAL" && columnMappings[data.Columns[i].ColumnName] != "NUMERIC")
        {
            continue;
        }

        double maxValue = Convert.ToDouble(data.Rows[0].ItemArray[i]);
        double minValue = Convert.ToDouble(data.Rows[0].ItemArray[i]);
    }
}

```

```

for (int j = 1; j < data.Rows.Count; j++)
{
    double currentValue = Convert.ToDouble(data.Rows[j].ItemArray[i]);

    if (currentValue > maxValue)
    {
        maxValue = currentValue;
    }

    if (currentValue < minValue)
    {
        minValue = currentValue;
    }
}

for (int j = 0; j < data.Rows.Count; j++)
{
    double currentValue = Convert.ToDouble(data.Rows[j].ItemArray[i]);

    double normalisedValue = (currentValue - minValue) / (maxValue - minValue);

    data.Rows[j][i] = normalisedValue.ToString();
}
}
}

/// <summary>
/// Populate the ARFF file
/// </summary>
public void PopulateOutputFile()
{
    using (System.IO.StreamWriter writer = new System.IO.StreamWriter(outputFileName))
    {
        writer.WriteLine("@RELATION\t" + relationName);
        writer.WriteLine("\n");

        foreach (KeyValuePair<string, string> attribute in columnMappings)
        {
            writer.WriteLine("@ATTRIBUTE\t" + attribute.Key + "\t" + attribute.Value);
        }

        writer.WriteLine("\n");
        writer.WriteLine("@DATA");

        foreach (DataRow row in data.Rows)
        {
            for (int i = 0; i < row.ItemArray.Length; i++)
            {
                writer.Write(row.ItemArray[i]);

                if (i != (row.ItemArray.Length - 1))
                {
                    writer.Write(",");
                }
            }
        }
    }
}

```



```
        writer.Write("\n");
    }

    writer.Close();
}

public static void Main(string[] args)
{
    TextToARFFParser program = new TextToARFFParser("steel.txt",
        "steel",
        new Dictionary<string, string>()
        {
            { "normalising_temperature", "REAL"},
            { "tempering_temperature", "NUMERIC"},
            { "sample", "NUMERIC"},
            { "percent_silicon", "REAL"},
            { "percent_chromium", "REAL"},
            { "manufacture_year", "NUMERIC"},
            { "percent_copper", "REAL"},
            { "percent_nickel", "REAL"},
            { "percent_sulphur", "REAL"},
            { "percent_carbon", "REAL"},
            { "percent_manganese", "REAL"},
            { "tensile_strength", "REAL"},
        });

    program.ReadFile();
    program.Normalise();
    program.PopulateOutputFile();
}
}
```