CA4010 Data Mining Assignment

Group 38
Sean Quinn 13330146
Jacob O’Keeffe 13356691

Predicting the mathematics grade of Portuguese secondary level students based on lifestyle and family background.

Idea and Dataset Description

We obtained our dataset from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Student+Performance).
It contains information about students in two Portuguese secondary schools: Gabriel Pereira and Mousinho da Silva. The dataset contains 31 attributes for each student which can be broken down into three categories: school life, home life, and social life. See appendix for list of attributes in full.

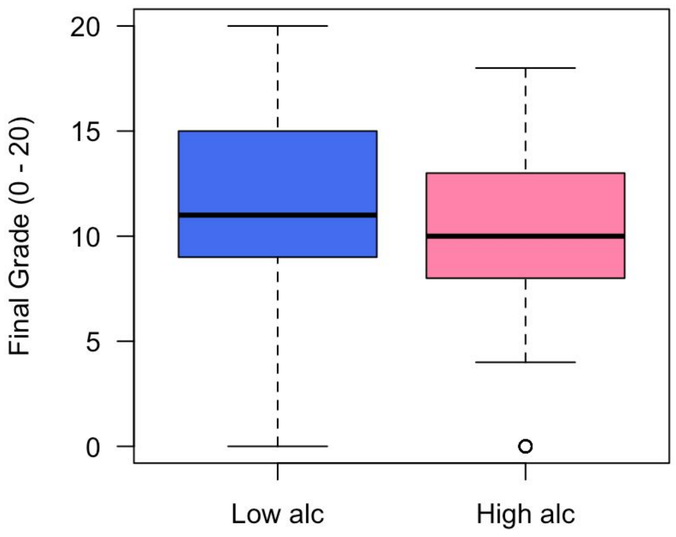
Our goal was to predict a student's academic performance (final grade) using these behavioural and environmental factors. We therefore ignored the student's first and second period grades when attempting to predict their final grade. There were a number of questions we hoped to address; How does alcohol consumption affect academic performance? Does spending more time studying equate to better results? How accurately can we predict the final grade?

If we can answer some of these questions then we can help struggling students and their teachers to identify what the problems are and what needs to be addressed. If we achieve a high prediction accuracy then it could be interpreted to mean that there is a set of criteria that a student must match to attain good academic results. If this is the case, then it should be possible to engineer an algorithmic solution which a student can follow to attain good results.

However, if our prediction accuracy is unexceptional then it could be interpreted to mean that some students possess an innate aptitude for mathematics which cannot be learned. This ties in nicely with the 'nature vs nurture' debate which is an argument regarding whether a person's development is predisposed in their DNA, or a majority of it is influenced by life experiences and their environment. While this is an interesting undercurrent, we are not hoping to give a definitive answer to the argument, it's just something which we felt was worth noting.

Data Preparation

Our workshop on data dispersion helped us to gain an insight into how different attributes affect student performance. We realised that individual attributes had very little effect on student performance in isolation.



For example, we plotted students with higher than the mean weekly alcohol consumption against students with lower than the mean weekly alcohol consumption. As you can see, there is a large overlap of the inter-quartile ranges. We obtained similar results with other attributes. Some attributes appeared to have no impact on the median or the inter-quartile range.

We broke the target attribute, final grade, into three classes; poor, average, and good. A grade of 0-6 is considered poor, a grade of 7-13 is considered average, and a grade of 14-20 is considered good.

A number of the categorical attributes in our dataset were in string format so we replaced these with numerical values. We incorporated a normalisation function into our code so that large values such as students' age would not skew the Euclidian distance formula we intended to use in our prediction. We achieved this using the formula outlined in our lectures: for each attribute *a*, the normalised value is ((*a - min) / (max - min)*).

Algorithm Description

Our project was a classic supervised learning example and we felt a classification approach was much more suitable than clustering.

We implemented a slightly adapted version of the k-nearest neighbours algorithm, using a weighted Euclidian distance as our distance metric. The weightings were assigned using the Pearson product-moment correlation coefficient.

We chose to build our classifier program in java and to implement all of the algorithms ourselves as opposed to using existing machine learning libraries. Using an existing library would have reduced the amount of work required and would have provided an adequate ‘out of the box’ solution. However, we felt that implementing our own predictor from scratch would allow us much greater flexibility and this alone warranted the additional overhead. We eventually found that the best results were achieved when standard techniques were slightly altered to suit our particular problem and dataset, such as our decision to use an altered version of the K-nearest neighbours algorithm.

Read dataset

In order to perform any analysis on the data in java the file must first be parsed and stored in a suitable object structure. During the data pre-processing stage we used R scripts to convert all the categorical data into numeric format. This was a requirement for us to be able to implement the K-nearest neighbours algorithm, as the Euclidean distance cannot be calculated between categorical data types. The format we chose to store the data in was a set of ArrayLists. The set represented the entire dataset and each ArrayList contained within the set represented an individual record of the dataset. ArrayLists store elements in an ordered index, so the original attribute order is preserved. We also created a data structure to hold the values for each column of the dataset (i.e. all 395 values for each attribute) to allow us calculate weightings and perform normalisation.

```java
private void readDataset(String path) {
    String currentRow = "";
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        // this takes the column headings out, the only non numeric values in file
        String headings = br.readLine();
        while ((currentRow = br.readLine()) != null) {
            // splits string into string [] then into ArrayList<double>
            ArrayList<Double> row = toDoubleArrayList(currentRow.split(","));
            dataset.add(row);
            // add to columns
            for (int i = 0; i < 33; i++) {
                columns.get(i).add(row.get(i));
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Normalise set

Although all attributes are now in numeric format, the nature of our data meant that the ranges of attributes vary wildly. This means that attributes such as age, being in the range 15 to 21, would skew our distance algorithm much more than others. The solution to this problem was to normalise the attributes so that each attributes values were in the range 0-1.

```java
private void normaliseDataset() { // convert all attributes to the range 0-1
    double [] maxValues = new double [30];
    double [] minValues = new double [30];
    for (int i = 0; i < 30; i++) {
        maxValues[i] = Collections.max(columns.get(i));
        minValues[i] = Collections.min(columns.get(i));
    }

    Iterator<ArrayList<Double>> iter = dataset.iterator();
    while (iter.hasNext()) { // loop through rows
        ArrayList<Double> current = iter.next();
        for (int i = 0; i < 30; i++) {
            double oldValue = current.get(i);
            // use the normalisation formula from the lecture notes
            double normValue = Double.parseDouble(
                df.format((oldValue - minValues[i])/(maxValues[i] - minValues[i])));
            current.set(i, normValue);
        }
    }
}
```

Calculate weightings

As part of our implementation of the K-nearest neighbours, we decided to use a weighted Euclidean distance as our distance metric. We decided to use the Pearson product-moment correlation coefficient to calculate the weightings. This is a measure of the linear dependence between two variables X and Y, giving a value between 1 and -1 inclusive, where 1 is total positive linear correlation, 0 is no linear correlation, and -1 is total negative linear correlation. In our case, the positivity or negativity of the correlation was not of importance, we just wanted to know the strength of the relationship, so we took the absolute value of this result and then normalised our weightings so that they summed to 1. The addition of this particular weighting algorithm increased the accuracy of our prediction algorithm from just over 50% to approximately 66%.

```java
private void calculateWeightings() {
    // for each of 30 values calculate correlation with final grade
    for (int j = 0; j < 30; j++) {
        double sx  = 0.0; // sum x
        double sy  = 0.0; // sum y
        double sxx = 0.0; // sum x squared
        double syy = 0.0; // sum y squared
        double sxy = 0.0; // sum x * y

        int n = columns.get(j).size();
        for (int i = 0; i < n; i++) {
            double x = columns.get(j).get(i);  // compares this attribute value
            double y = columns.get(32).get(i); // with its corresponding final grade
            sx  += x;
            sy  += y;
            sxx += (x * x);
            syy += (y * y);
            sxy += (x * y);
        }

        double cov    = sxy / n - sx * sy / n / n;             // covariation
        double sigmax = Math.sqrt(sxx / n - sx * sx / n / n);  // standard error of x
        double sigmay = Math.sqrt(syy / n - sy * sy / n / n);  // standard error of y
        double correl =  (cov / sigmax / sigmay);              // correlation
        euclideanWeighting[j] = correl;
    }

    double totalWeight = 0;
    for (double x: euclideanWeighting) {
        totalWeight += (Math.abs(x)); // we want absolute value
    }

    for (int i = 0; i < euclideanWeighting.length; i++) { // ensure all weights sum to 1
        euclideanWeighting[i] = Math.abs(euclideanWeighting[i] / totalWeight);
    }
}
```

Classify & Partition set

The set was classified by iterating through the set and comparing the final result grade to our classification criteria and assigning it a numeric class value; 1 : (0-6), 2: (7-13), 3: (14-20). We partitioned the set into a 200 record training set and a 195 record validation set.

Prediction and Validation

This involved using the weighted distance algorithm we previously defined to calculate the distance from the current row to all rows in the training set and then select the six nearest neighbours. Due to us having three classes there is a potential that in our six nearest neighbours we may end up with a 2-2-2 or 3-3-0 split. Here we decided to make an alteration to the standard KNN algorithm. In the event of a tie, our algorithm will take in a seventh neighbour into consideration so as to establish a single most probable class This means that we always have one dominant class. This alteration increased the performance of our algorithm by 6%.

We validated our predictions by comparing the predicted class of a row with it’s actual value.

```
private void predictClassKNN() { // train on first 200, run on second 195
    HashSet<ArrayList<Double>> trainingSet = new HashSet<>();
    HashSet<ArrayList<Double>> validationSet = new HashSet<>();
    Iterator<ArrayList<Double>> iter5 = dataset.iterator(); // splits into two sets
    int rnd;
    while (iter5.hasNext()) { // loop through rows
        rnd = ThreadLocalRandom.current().nextInt(1, 11);
        if (rnd <= 5)
            trainingSet.add(iter5.next()); // 200 rows
        else
            validationSet.add(iter5.next()); // 195
    }

    // this will hold 195 (row, predictedClass) prediction pairs
    HashMap<ArrayList<Double>,Double> resultsMap = new HashMap<>();

    // classify validation set one by one using knn and add to resultsMap
    Iterator<ArrayList<Double>> iter6 = validationSet.iterator();
    while (iter6.hasNext()) {
        // this will hold 200 (dist, class) pairs. TreeMap keep them in sorted order -
can use pollFirstEntry
```

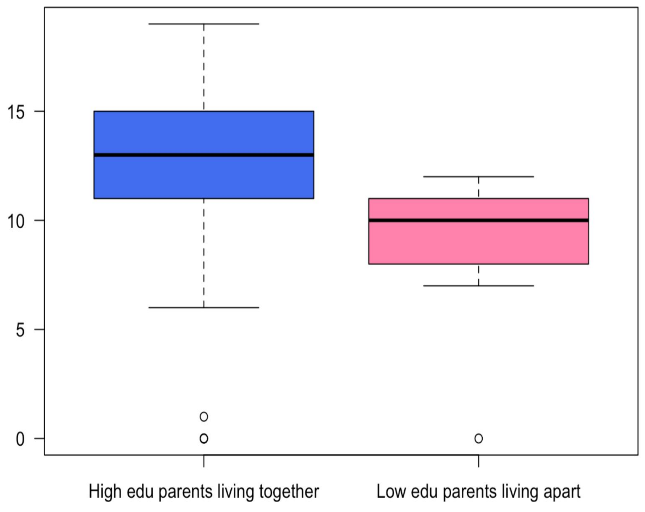
```
        TreeMap<Double,Double> distanceMap = new TreeMap<Double,Double>();
        ArrayList<Double> current = iter6.next(); // current validation row

        // calculate distance from this to each row in training set
        Iterator<ArrayList<Double>> iter7 = trainingSet.iterator();
        while (iter7.hasNext()) {
            ArrayList<Double> trainRow = iter7.next();
            double dist = weightedEuclideanDistance(current, trainRow);
            distanceMap.put(dist, trainRow.get(33)); //  index 33 holds clasification
value
        }
        int [] classCount = new int[4]; // we wont use index 0.
        for (int k = 0; k < 6; k++) { // 6 nearest neighbours
            // return and remove smallest key entry in treemap - closest neighbour
            Map.Entry<Double,Double> a = distanceMap.pollFirstEntry();
            // converts double to int and increments index 1,2 or 3 of classCount
            classCount[Integer.valueOf(a.getValue().intValue())]++;
        }

        // in the event of a tie (2-2-2) (3-3-0) - increase to 7 nearest neighbours
        if (!isOneMax(classCount)) {
            Map.Entry<Double,Double> a = distanceMap.pollFirstEntry();
            classCount[Integer.valueOf(a.getValue().intValue())]++;
        } // there is now definitely only one max

        if ((classCount[1] > classCount[2]) && (classCount[1] > classCount[3]))
            resultsMap.put(current, 1.0);
        else if ((classCount[2] > classCount[1]) && (classCount[2] > classCount[3]))
            resultsMap.put(current, 2.0);
        else
            resultsMap.put(current, 3.0);
    }

    // next step is check success rate, compare prediction with actual
    double rightCount = 0;
    int g = 0;

    Iterator it = resultsMap.entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry prediction = (Map.Entry)it.next();
        double actual = ((ArrayList<Double>) prediction.getKey()).get(33);
        if ((actual).equals(prediction.getValue()))
            rightCount++;

        g++;
    }

    // percentage success rounded to 4 places
    double success = Double.parseDouble(df.format((rightCount/195)*100));
    System.out.println(rightCount + " predictions out of 195 correct");
    System.out.println(success + "% of predictions correct");
}
```

Results and Analysis

We achieved an accuracy of 65.641%. This was the mean accuracy from twenty iterations of our program. Each time our program is run, the training dataset is randomly generated. This ensures that our accuracy is not inflated and it also displays the robustness of our solution. We feel that this a decent accuracy score considering how few rows we had in our dataset (395). We understood the disadvantages of choosing such a small dataset, however, we felt that the number of attributes made available to us, helped to combat these disadvantages.

It is certainly worth noting that a number of the attributes (quality of family relationship, free time after school, how often they go out with friends, weekly alcohol consumption, and current health) were generated by asking the students to assign each a value between 1 and 5. There are a couple of issues with this approach; students may not be entirely truthful, and a value of say 3 might be interpreted completely differently by two students. The weightings assigned to these attributes must be taken with a pinch of salt by anyone examining our results.

It was quite interesting to examine the weightings which were generated using the correlation coefficients; they were quite out of line with what we had expected. The most heavily weighted attributes (in descending order) were: past class failures, wants to pursue higher education, parents level of education, student's age, romantic relationship status, how often they go out with friends, and their travel time to and from school. Past class failures is unique in the fact that it is directly affected by a student's past academic performance. It's weighting reflects this direct correlation; it is weighted nearly twice as heavily as the next heaviest, "wants to pursue higher education".



When we plotted parents with little or no education against highly educated parents we saw that the entire box plot (minus any outliers) of parents with poor education was entirely contained within the first quartile of the highly educated parents box plot. We therefore expected to see the parents education attribute weighted quite heavily.

The least heavily weighted attributes (in ascending order) are as follows: reason for choosing school, amount of free time after school, whether or not the student partakes in extracurricular activities, who their legal guardian is, whether or not they receive educational support from family members, number of school absences, and the school they attended. It is somewhat surprising to see that free time after school is weighted so weakly, but it is line with the point made earlier regarding the legitimacy of the value assigned to this attribute by students. It is understandable that a student's reason for choosing a school is weighted weakly as it usually not the student's choice in the first place.

It was interesting to compare the weightings generated using the correlation coefficients and the weightings which we manually came up with using the knowledge we had gained from our workshop preparation. Despite all we had learned from our investigations the fact that the algorithms weightings were roughly 10% more effective that our own estimations illustrates the power of data mining and machine learning algorithms. Harnessing these algorithms allows analysts to identify patterns which may not be evident to humans restrained by their perceptions.

It is somewhat disheartening to discover that many of the attributes which are indicative of a good student can not be easily adopted by struggling students. A student cannot change their parent's level of education, nor how long they have to travel to and from school. The lack of desire to pursue higher education is a major indicator of a poor student, however, these students do not have any major incentive to perform better academically so that is not something which needs to be addressed.

The fact that our prediction algorithm was inaccurate ⅓ of the time indicates that there are other factors at play. It is possible that a student who fits our model of a good student may have panicked or simply not performed on the day of the exam and received a poor/average mark. This could form the basis of an argument for 100% continuous assessment. Similarly, a student who fit our model of a poor student may have been able to cram in the days leading up to the final exam. It is also possible that some students were simply lucky with the content that appeared on the final exam. We also must not rule out the possibility that some students may have cheated in the final exam. The is simply too much scope in a number of the key attributes in our dataset.

We believe that we performed as well as possible given the context. We outperformed the SciKit Learn (http://scikit-learn.org/stable/) machine learning algorithms (decision tree: 55.412%, K-nearest neighbours: 60.44%, neural network: 59.889%) using our slightly adpated Java implementation of the K-nearest neighbours algorithm. In doing so, we displayed our deep understanding of the algorithm and the nature of our dataset. It also gave us a benchmark to with which we could compete and validate against.