# Data Mining & Machine Learning
# (F20DL)

## Coursework 2

**By**
**Cameron McBride, Chris Walsh, Jordan Walker**

**Date: November 7, 2018**

**Link to all documents including scripts, data Files etc. can be found at:**

https://github.com/cdw2/data_mining_cw2

***will be public after Submission Date.**

***All members of this coursework contributed equally to the final output of this document.***

## Improvements From Coursework One.

Upon receiving feedback for Coursework One we have made modifications to our various applications some of which are highlighted in the implementation of the various tasks. These changes have been undertaken for various reasons including:

- *Gaining better performance and more accurate and realistic results.*
- *More knowledge of the subject allowing us to better understand the use of particular tools.*

## Changes to Filters/ Techniques:

For this coursework, we have also changed from using CorrelAttributeEval as a means to extract the best pixels. We have replaced this by now using CFSSubSetEval to now list the best pixel values of which to run the tests conducted in the research gathering of Coursework Two. This was done by the feedback given on our results for Coursework One.

## Changes to Testing Environments:

Due to multiple testing environments, we decided as a group to multi-thread our implementations to make the best use of time as we can run all instances of testing simultaneously. By integrating threads, we can also guarantee that while we run our tests, there are no race conditions that affect the accuracy of our data.

In our first Coursework, we implemented a technique taught in F20BC (Biological Computing) where we sum and take an average of particular sections of the image initially 48 x 48 pixels and then condense the photos to make 24 x 24 pixelated images. By doing this, we create our own filter to eliminate noise from the picture but still keep the original composition of the image. In this assignment, we refer to "Reduced Dataset" when we use this method for our research. In addition to this, we will also run tests on the full 48 x 48 pixels in task 1 to contrast any differences to the measurements we encounter during the research undertaken.

## *Implementing task 1*

Task One sees us repeating the steps that were made in steps 4 through 6 in the first coursework to compare and contrast our results with using Naive Bayes as a classification tool opposed to using IBK-Nearest Neighbour. For this task, we have chosen to measure the Correctly Classified Instances, Incorrectly Classified Instances, Run Time, Mean Absolute Error and Root Mean Squared Error. In addition to this, we will also be comparing the results for each of these measures on various test options of Nearest Neighbour and Naive Bayes which includes Cross-Validation and Hold-Out where we supply a training set generated by our TestTrain.py program which allows us to automate this process. Furthering from this we created a wrapper in Python that will enable us to use all our dependent scripts to automate the entire testing process as opposed to manually having to do tasks through the Weka GUI.

An example of how we build and cross evaluate the Naive Bayes classifier using the python Weka wrapper is shown below:

```
cls = Classifier(classname="weka.classifiers.bayes.NaiveBayes")
cls.build_classifier(self.training_data)
evl = Evaluation(self.training_data)
evl.crossvalidate_model(cls, self.training_data, 10, Random(1))
```

We encapsulated this code in a function for each of the classifiers that we wanted to run as well as some auxiliary code to print and save the results to file.

## *Results of task 1*

***Comparison of Naïve Bayes Networks Vs IBK Neigherst Neighbour On Reduced Dataset***

| | Correctly Classified Instances | Incorrectly Classified Instances | Run - Time (Seconds) | Mean Absolute Error | Root Mean Squared Error |
|---|---|---|---|---|---|
| *Naïve Bayes Networks - Cross-Validation* | 26.44% | 73.55% | 6.9 | 0.2132 | 0.4209 |
| *Naïve Bayes Networks - Hold Out* | 26.33% | 73.66% | 0.8 | 0.2132 | 0.422 |
| *IBK - Neigherst Neighbour -Cross-Validation* | 35.36% | 64.63% | 285.82 | 0.1901 | 0.3542 |
| *IBK - Neigherst Neighbour -Hold Out* | 34.7% | 65.22% | 52.96 | 0.1924 | 0.3559 |

***Table 1 - Naive Bayes Comparison Against IBK on Reduced Dataset***

***Comparison of Naïve Bayes Networks Vs Neigherst Neighbour On Full Dataset***

| | Correctly Classified Instances | Incorrectly Classified Instances | Run - Time (Seconds) | Mean Absolute Error | Root Mean Squared Error |
|---|---|---|---|---|---|
| *Naïve Bayes Networks - Cross-Validation* | 21.49 | 78.50 | 192.82 | 0.2244 | 0.4715 |
| *Naïve Bayes Networks - Hold Out* | 21.56 | 78.43 | 16.59 | 0.2242 | 0.4714 |
| *IBK - Neigherst Neighbour -Cross-Validation* | 34.87 | 65.12 | 6250.68 | 0.1921 | 0.3564 |
| *IBK - Neigherst Neighbour -Hold Out* | 34.79 | 65.20 | 1145.01 | 0.1935 | 0.3574 |

***Table 2 - Naive Bayes Comparison Against IBK on Full Dataset***

**Comparison of Naïve Bayes Networks Vs Neigherst Neighbour With Best Pixel Attributes**

| | No. Relevant Pixels | Correctly Classified Instances | Incorrectly Classified Instances | Run - Time (Seconds) | Mean Absolute Error | Root Mean Squared Error |
|---|---|---|---|---|---|---|
| Naïve Bayes Networks - Cross-Validation | 14 | 27.07% | 72.92% | 1.94 | 0.22 | 0.35 |
| | 35 | 25.71% | 74.28% | 3.61 | 0.21 | 0.40 |
| | 70 | 24.76% | 75.23% | 6.34 | 0.21 | 0.42 |
| Naïve Bayes Networks - Hold Out | 14 | 27.15% | 72.84% | 0.49 | 0.22 | 0.35 |
| | 35 | 25.81% | 74.18% | 0.56 | 0.21 | 0.40 |
| | 70 | 24.64% | 75.35% | 0.82 | 0.21 | 0.42 |
| IBK - Neigherst Neighbour -Cross-Validation | 14 | 29.72% | 70.27% | 109.58 | 0.20 | 0.36 |
| | 35 | 31.75% | 68.24% | 182.74 | 0.198 | 0.36 |
| | 70 | 33.03% | 66.96% | 274.36 | 0.19 | 0.36 |
| IBK - Neigherst Neighbour -Hold Out | 14 | 28.54% | 71.45% | 17.54 | 0.20 | 0.37 |
| | 35 | 31.30% | 68.69% | 27.92 | 0.20 | 0.36 |
| | 70 | 32.89% | 67.10% | 52.85 | 0.19 | 0.36 |

*Table 3 - Naive Bayes Comparison Against IBK on Reduced Dataset Most Relevant Pixels*

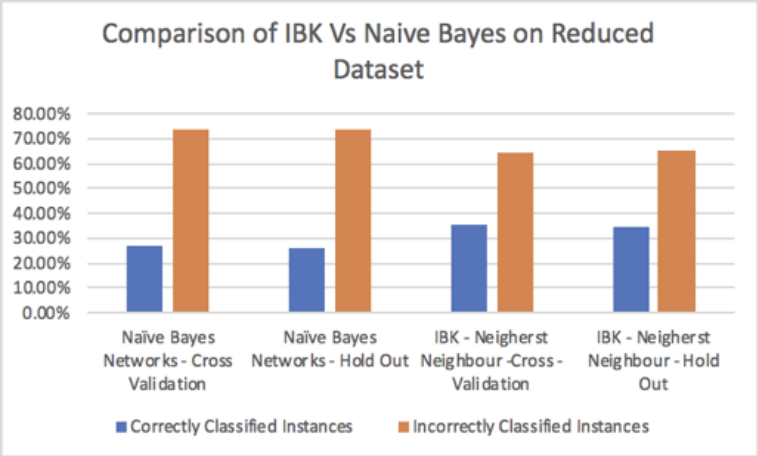


*Figure 1 - Comparison of Correctly Vs Incorrectly Classified instances between IBK and Naive Bayes*

Figure 1. Shows that in comparison Naive Bayes performs worse in both correctly and incorrectly classified instances. Against IBK using Cross Validation, we can see an 8.92% decrease incorrectly classified instances and an increase in incorrectly classified instances. From observation in this scenario, we can see IBK outperforms Naive Bayes in Correctly Classifying the elements of the dataset.

Figure 2. Concluded our assumption the run time of the two classification methods varies in regards to the correctly/ incorrectly classified instances. However, it was not expected that for an 8.92% increase incorrectly classified instances that the runtime would equate to a runtime near 42 times greater using IBK compared to Naive Bayes. Compared to IBK however, Naive Bayes algorithm is not as complicated where it mainly relies on two operations 1. Counting then 2. Dividing. This is the reason we have determined for the vast increase in run-time.

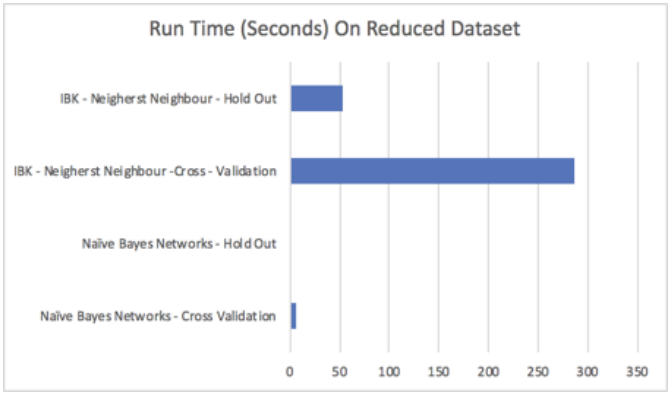


*Figure 2 - Comparison of the Run Time of IBK Vs Naive Bayes on Reduced Dataset*

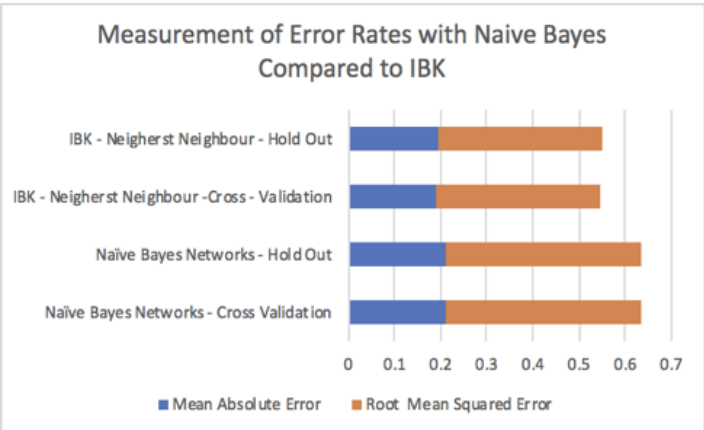


*Figure 3 - Comparison of Error Rates between Naive Bayes and IBK*

In observation of Graph 3., we can see that the error rate for both IBK and Naive Bayes with both methods of Validation hold similar Mean Absolute Error with Naive Bayes having a slightly higher error rate. Comparing the Root Mean Squared Error, however, we can observe a more dramatic increase in error. This is because Root Mean Squared Error penalizes bigger errors due to being squared before being averaged.

From analysing the information gathered we can see that IBK outperforms Naive Bayes in all areas which follow the trend shown in Graph 1. The interesting note here, however, is that the correctly classified instances rise for IBK when more attributes are used as opposed to Naive Bayes which correctly classified instances increase with a lower amount of attributes as seen in Table 3.

## ***The conclusion of Results of Task 1***

By using Naive Bayes we can further understand the data set by learning the mean probability value associated with each pixel and how it affects the overall classification when running analysis on the dataset using this method.

Overall there were no improvements in reflection and observation undertaken during analysis we can see that the accuracy of the classification went down dramatically and that the error rate went up however we observed a better run-time compared to IBK.

Upon analysing the confusion matrix associated with running our experiments in task 1 we discovered that fear, angry, disgust, happy and neutral were the hardest emotions to recognise using Naive Bayes and were most easily confused this differs from our initial experiment using IBK. The differ is the inclusion of fear and disgust that was more easily distinguishable In the results of Coursework 1.

We can conclude then that by using Naive Bayes i.e. conditional probability that we cannot aim to produce a notable classification method for dealing with the *FER2018* dataset and other methods must be explored.

## ***Implementing task 3***

For Task 3 we move from the simplicity of Naive Bayes to a more complex structure, the Bayesian Network. Both models technically are versions of Bayesian Classifications. However, Naive Bayes is a more simplistic model, which assumes a conditional independence i.e. $P(X|Y, Z)=P(X|Z)$ between attributes which in larger data sets can be quite aggressive on the accuracy of the classification. In contrast, a Bayesian Network offers more complexity as the user can choose the attributes which are factually conditionally independent which in turn one would assume better classification results when compared.

To record our results for task three we have again used the same measurements to allow us to show the comparisons of the previously run tests against the newly implemented Bayes Networks.

In the tests we only record the TAN algorithm once as TAN does not have max parent selection, instead TAN uses its own algorithm to work out the parents to best create the network. Hill Climbing works similarly however, max parents can be set pre-running the tests from our automated script. Likewise, we can use this script to also dictate all initial parameters for the different Bayes Networks.

In order to efficiently run multiple tests at once we decided to multi-thread our test functions. We built own thread class to accept and instantiate the functions to run our algorithm tests. This is shown below:

```
class myThread (threading.Thread):
  def __init__(self, threadID, name, function, args=None):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
    self.function = function
    self.args = args
  def run(self):
    print ("Starting " + self.name)
    self.function(self.args)
```

This meant that we could create multiple threads to run several tests with different parameters simultaneously such as varying the maximum number of parents as shown below.

```
thread1 = myThread(1, "run_bayes_split", run_bayes_split, (1))
thread2 = myThread(2, "run_bayes_split", run_bayes_split, (3))
thread3 = myThread(3, "run_bayes_split", run_bayes_split, (5))
```

We also had a method to save results from each test to a folder with a filename representing the parameters of the algorithm.

## ***Results of task 3***

**Implementation of Bayes Network using K2, TAN and Hill Climbing Methods With Best Pixel Attributes**

| | No. Relevant Pixels | Correctly Classified Instances | Incorrectly Classified Instances | Run - Time (Seconds) | Mean Absolute Error | Root Mean Squared Error |
|---|---|---|---|---|---|---|
| Baye Network One -K2 Algorithm No. Parents 1 | 14 | 27.03 | 72.96 | 0.81 | 0.222 | 0.354 |
| | 35 | 26.75 | 73.24 | 1.52 | 0.216 | 0.3975 |
| | 70 | 25.44 | 74.55 | 2.42 | 0.215 | 0.4231 |
| Baye Network One -K2 Algorithm No. Parents 3 | 14 | 30.63 | 69.36 | 0.91 | 0.2244 | 0.3376 |
| | 35 | 31.25 | 68.74 | 2.18 | 0.22 | 0.3395 |
| | 70 | 30.69 | 69.30 | 6.49 | 0.2143 | 0.3473 |
| Baye Network One -K2 Algorithm No. Parents 5 | 14 | 30.69 | 69.30 | 0.88 | 0.2226 | 0.3376 |
| | 35 | 32.11 | 67.88 | 2.74 | 0.2173 | 0.3385 |
| | 70 | 32.89 | 67.10 | 12.3 | 0.2101 | 0.3426 |
| Baye Network Two -TAN Algorithm | 14 | 31.04 | 68.95 | 0.38 | 0.2234 | 0.3365 |
| | 35 | 31.26 | 68.73 | 2.7 | 0.2194 | 0.339 |
| | 70 | 31.76 | 68.23 | 7.41 | 0.213 | 0.3442 |
| Baye Network Three -Hill Climbing Algorithm No. Parents 1 | 14 | 27.03 | 72.96 | 0.59 | 0.222 | 0.354 |
| | 35 | 26.75 | 73.24 | 2.57 | 0.216 | 0.3975 |
| | 70 | 25.44 | 74.55 | 11.17 | 0.215 | 0.4231 |
| Baye Network Three -Hill Climbing Algorithm No. Parents 3 | 14 | 29.03 | 70.96 | 0.87 | 0.2267 | 0.3383 |
| | 35 | 31.68 | 68.31 | 5.17 | 0.2247 | 0.337 |
| | 70 | 30.31 | 69.68 | 28.82 | 0.2252 | 0.3387 |
| Baye Network Three -Hill Climbing Algorithm No. Parents 5 | 14 | 29.17 | 70.82 | 1.08 | 0.2296 | 0.339 |
| | 35 | 30.05 | 69.94 | 7.58 | 0.2286 | 0.3385 |
| | **70** | **29.13** | **70.86** | **38.96** | **0.2294** | **0.339** |

*Table 4 – Results of using more Complex Bayes Network Algorithms*

From analysing the results, we achieved using Bayes Networks compared to using Naive Bayes for classification we can observe that the accuracy increased, and the root squared error rate dropped quite significantly. We can also observe that by using the correct Bayes Network with the right parameters that we can achieve higher results based on the initial conditions and the right data input. For example using the Hill Climbing algorithm with number of parents initialised to 3 and the best 35 attributes we can collect much better classification results to when we use other parameters such as increasing the number of parents or increasing and decreasing the best pixel attribute size.

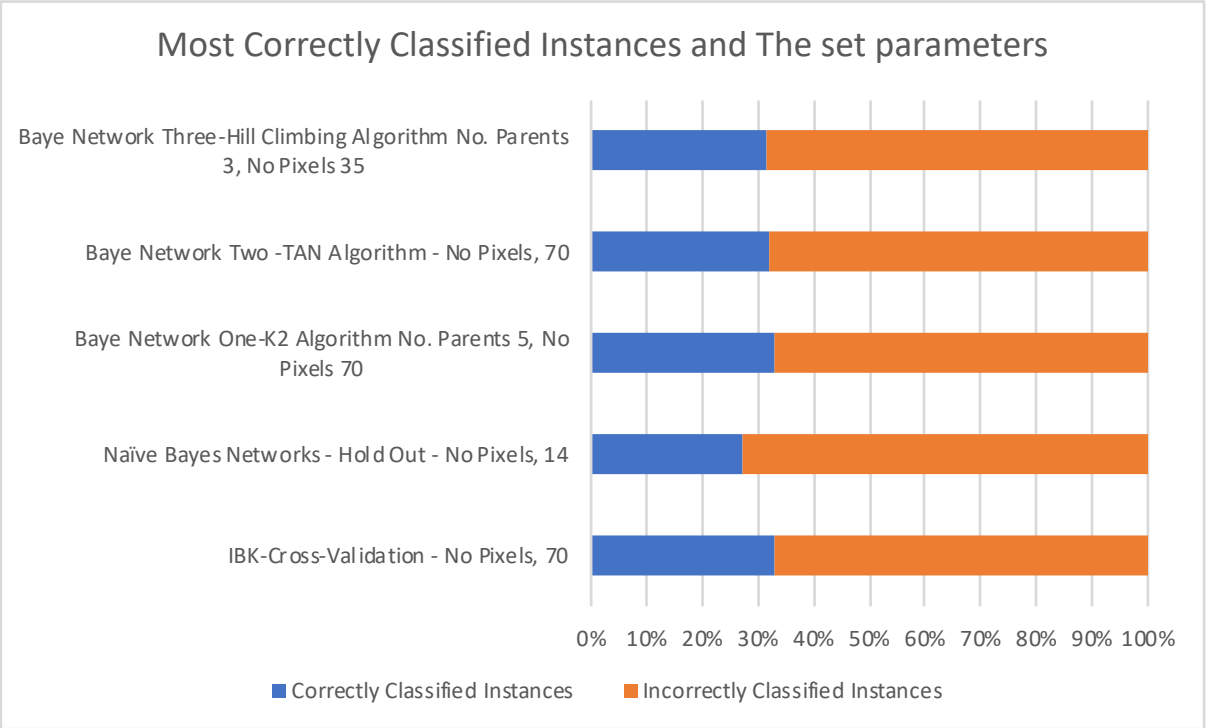


*Figure 4 - Best Performance of Correctly Classified Instances In Each Implementation*

Figure 4 shows the all implementations we have implemented so far during both coursework 1 and 2. It is clearly shown in this graph that Naïve Bayes is easily out-performed by all the other implementations. What is surprising however is when comparing the correctly classified instances of the more complex Bayes Networks with the right parameters we can achieve similar results in a shorter space of time as shown in Figure 5.

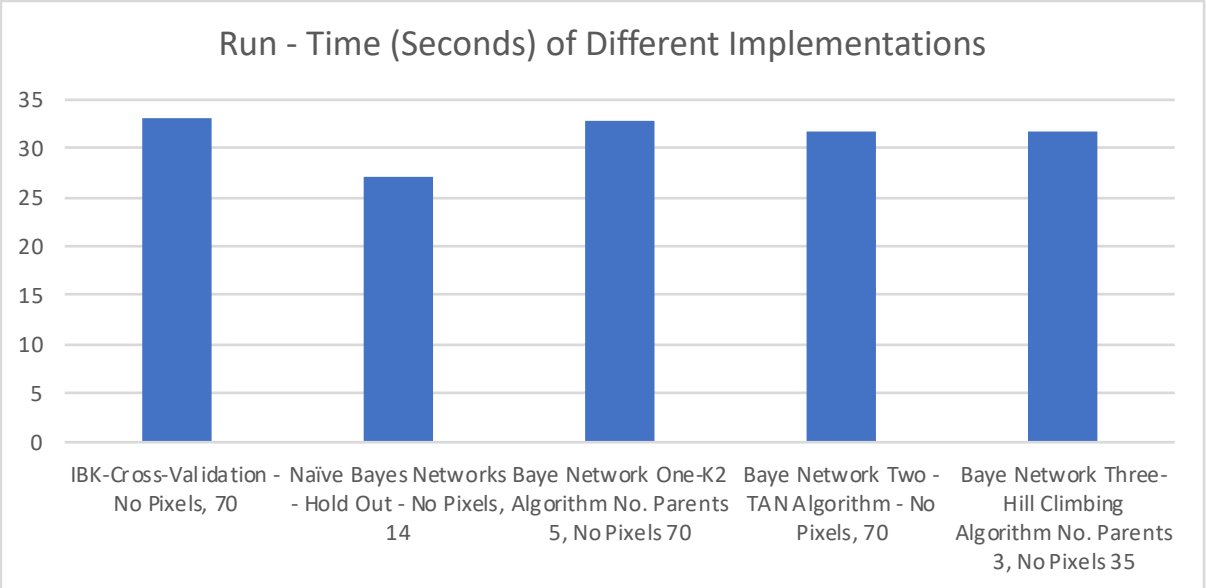


*Figure 5- Run Time of the Implementations that Achieved the Best Results*

When visualised the graph of the Bayes network with a maximum of 3 parents for a reduced version of the fer2018.arff dataset is shown below. We can use this graph to see which attributes are dependant or independent on each other by examining the links between nodes. By looking at the probability distribution table for a node we can also pick out the maximum A posteriori probability to see which attribute is most likely to predict a certain emotion given the values of the prior attributes.

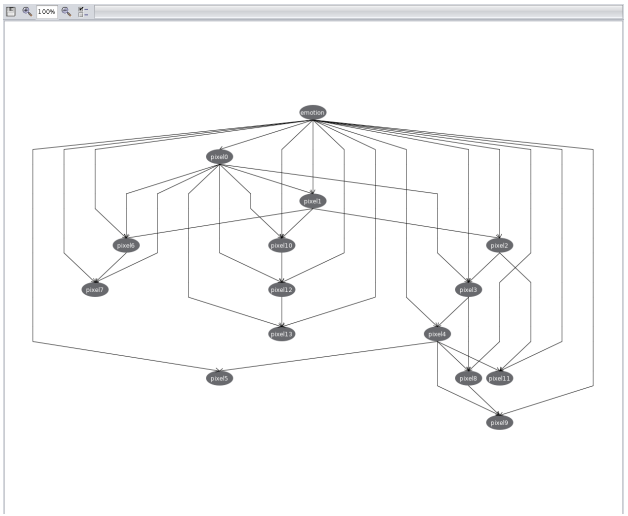


*Figure 6 Visualization of Bayes Network*

Bayes Network Maximum 3 Parents "fer2018.arff"

## ***The conclusion of Results of Task 3***

We can conclude after undertaking task 3 that using the more sophisticated Bayes Network we can gain an overall increase in our correct classification accuracy. When comparing the results achieved here we can see that some of the algorithms associated with Bayes Networks can rival the use of IBK as a classification tool with almost similar accuracy and in less run time showing an overall performance increase when using these two measurements. Also by using Bayes Networks we can obtain more information about the used dataset. What is meant by this is that we can select the most relevant attributes that are needed for classification.

## ***Implementing task 5***

In this task, we have been asked to cluster the data sets of fer2017.arff and all the individual emotion .arff files using the K-Means algorithm. It is tasked that we perform this in two ways firstly we remove the class attribute i.e. in our dataset the "emotion" of the .arff file to emulate when learning is achieved in an unsupervised manner. Secondly, we cluster with the class attribute and upon completion record the results and then make comparisons.
We experimented with different seed numbers when executing K-Means on the reduced dataset. The percentage of incorrectly classified instances ranged between 81.1% to 81.3% across different seed numbers between 1 and 20. We also found that the percentage of instances classified to each cluster had a difference of up to 3% depending on the seed number. This showed us that while altering the starting seed number did slightly affect the proportion of instances assigned to each cluster, the overall difference this made to classification was negligible.

To carry out the clustering we used our automation script and the python Weka wrapper to run the simple k means algorithm with N=2 on the individual emotion files to reflect the named emotion and the "other" classes. For the full fer2018.arff file we clustered with N=7 to reflect all of the possible classes. We run the simple k means algorithm 2 times both with and without the class attribute in the data, the second time performing a classes to clusters evaluation.

Shown below is a sample of the code we wrote to build and evaluate a clustering algorithm.

```
clusterer = Clusterer(classname=clusterer_name, options=["-N", ""+str(num_clusters)])
        clusterer.build_clusterer(data)

        evl = ClusterEvaluation()
        evl.set_model(clusterer)
        evl.test_model(self.data)
```

We encapsulated this code within a method so that we could build and run multiple clustering's using multithreading as we did for task 3.

***Results of task 5***

| Emotion | Correct Count | Clustered Instances | % Difference |
|---|---|---|---|
| **Angry** | 4952 | 5473 | +10% |
| **Disgust** | 547 | 4861 | +788% |
| **Fear** | 5121 | 4758 | -7% |
| **Happy** | 8989 | 5589 | -37% |
| **Neutral** | 6198 | 5425 | -12% |
| **Sad** | 6077 | 4639 | - 23% |
| **Surprise** | 400 | 5142 | +1185% |

*Table 5 – Results of Clustering with No Class and Comparison to How accurate it was by difference*

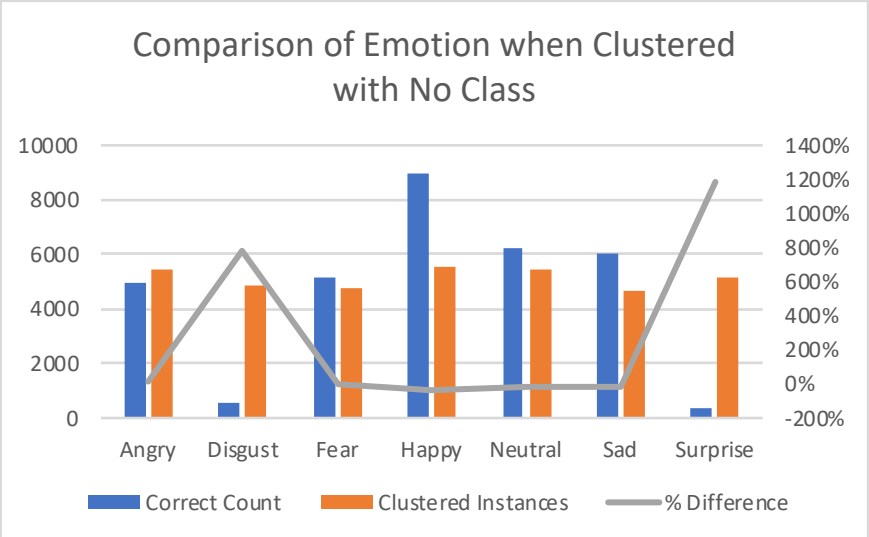


*Figure 7 - Graph Depicting the Accuracy of Clustering with No Class*

Upon observing the results given by clustering with no class attribute what we can observe is a mild accuracy amongst specific emotions such as Angry, Fear and Neutral. However, what is interesting to see is the difference when observing Disgust and Surprise. By comparing each clustered instance, we can see that the model is trying to evenly distribute each case of emotion thus we gain huge increases in difference for these emotions.

***Clustering results with K-Means Using 14 Pixels With Class Attribute***

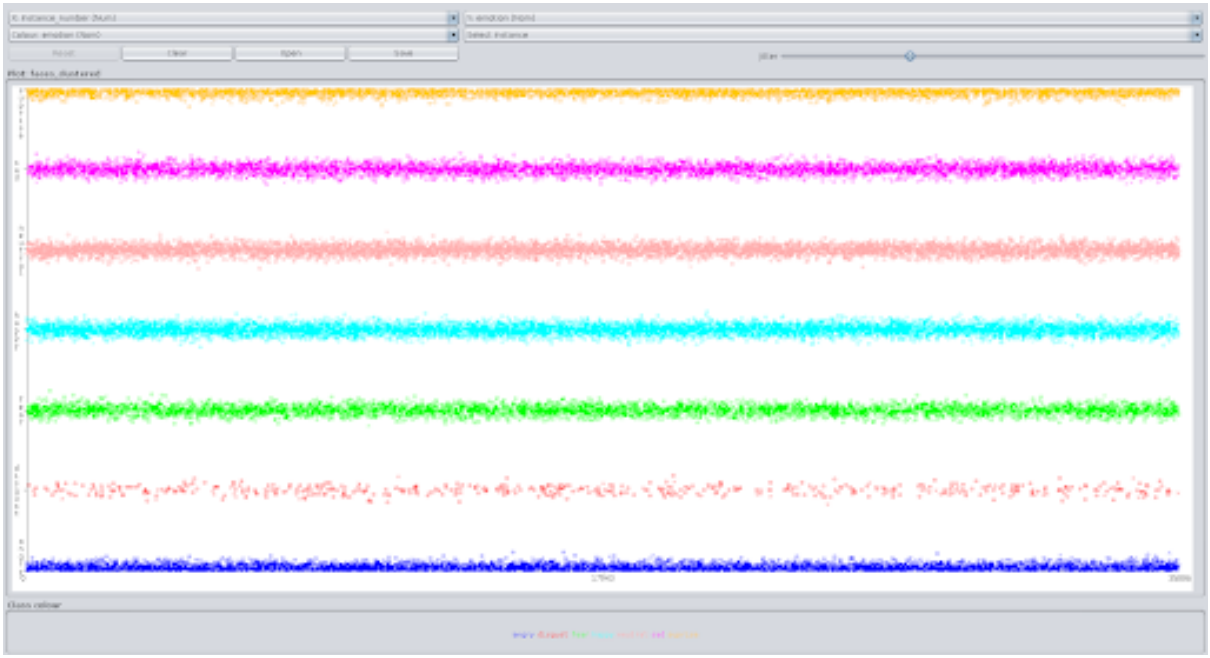

*Figure 8 - Visualization of Clustering with Class Attribute using 14 Pixels*

***Comparison of K-Means Implementation (Reduced Dataset) With Class Attribute***

| | Correctly Classified Instances | Incorrectly Classified Instances | Run - Time (Seconds) | Iterations |
|---|---|---|---|---|
| Fer2018 | 18.73 | 81.27 | 36.17 | 66 |
| Fer2018happy | 50.73 | 49.27 | 6.52 | 24 |
| Fer2018angry | 50.54 | 49.46 | 6.16 | 24 |
| Fer2018sad | 52.21 | 47.79 | 6.14 | 24 |
| Fer2018suprise | 55.54 | 44.46 | 6.16 | 24 |
| Fer2018disgust | 51.49 | 48.51 | 6.61 | 24 |
| Fer2018neutral | 51.42 | 48.58 | 6.26 | 24 |
| Fer2018fear | 53.85 | 46.15 | 6.49 | 24 |

*Table 6 – Results of Clustering when using Class Attribute*

To conclude task 5 you can see that the overall error rate on the fer2018.arff file with all the possible classes is quite high at 81.27% when using simple k means compared to the 64% and 73% of the IBK and Naive Bayes algorithms. However, the error rate when only having to cluster the instance to the specified emotion or the "other" class, is much lower reaching an error rate of only 44.46% for surprise. This suggests that a hierarchical clustering might work quite well were it can recursively split the data into one class and an "other" at each level until all emotions have been clustered.

## ***Implementing task 7***

Weka offers numerous clustering algorithms each holding their own advantages and disadvantages. A paper published in the International Journal of Emerging Technology and Advanced Engineering describes each of the clustering algorithms individual advantages and disadvantages [2]. This paper ultimately gave our group an understanding of each algorithm and what assumptions we can make initially when using our own dataset.

Due to the nature of the EM algorithm we were unable to apply this algorithm to the reduced dataset even with the best 14 pixels selected from that when setting the maximum number of clusters to -1, where the algorithm can find the optimal number of clusters on its own [3]. However, when we applied our max number of clusters we found that the model took too long to compute with our resources. Thus we decided it was best to move forward by testing on a varied range of max clusters to find the optimal.

During testing, we did not run tests on FilteredClusterer and MakeDensityBasedCluster as they essentially allow you to run filters for the clustering algorithms. Instead we wrote an option apply filters to the data directly, if required, when loading the dataset using our python wrapper.
An example of the code to perform runs with different cluster sizes of the em algorithm is shown below.

```
def run_clusters_auto1(num_clusters):
  global filename, testName
  jvm_helper = classify.cw2_helper()

  simplek_full = classify.cw2_classifier()
  simplek_full.load_data(filename)

  clusterList = [2,4,7,10,15]
  for clusterNum in clusterList:
     simplek_full.run_clustering_task7_manual("results/"+str(testName),"weka.clusterers.EM",clusterNum)
```

This allowed us to run this test in one thread while running the Canopy, Coweb or Farthest First tests in another.

## ***Results of task 7***

| Algorithm | | | Error Rate | | |
|---|---|---|---|---|---|
| Canopy | 18 Clusters | | 80.85 | | |
| Coweb | 1 Cluster | | 74.95% | | |
| | **N=2** | **N=4** | **N=7** | **N=10** | **N=15** |
| Simple K | 78.01% | 78.19% | 81.27% | 85.27% | 89.16% |
| EM | 77.46% | 78.04% | 81.22% | 85.30% | 88.68% |
| Farthest First | **75.82%** | 75.83% | 76.26% | 77.30% | **80.03%** |

The best improvement to the clustering results from task 7 is seen when using the farthest first algorithm at N=15 nodes. This resulted in nearly a 9% reduction in error from simple k means alone. We think this is due to the fact that the farthest first algorithm assigns each point to the farthest

centroid first. This results in a higher error during the "pval" calculation and therefore possibly reduces the sensitivity to the noise that will be present in numeric data like this dataset. N=15 was double the number of actual clusters in the dataset giving the algorithm more granularity in choice when assigning a data point to a cluster. The pros and cons of running each algorithm on the dataset is given below.

| Algorithm | Pro's | Cons |
|---|---|---|
| ***Simple K-Means*** | • Simple and Efficient to Run O(n) time complexity.<br>• Good when cluster shape is roughly circular or "hyper spherical" in more than 2 dimensions. | • Sensitive to Noise<br>• Not accurate when clusters are not circular.<br>• Not very repeatable since K-Means starts with a random choice of cluster centres.<br>• Requires prior knowledge or a guess at number of clusters within data.<br>• Hard assigns a point to a cluster, no information on the relationship between that point and the other clusters. |
| ***Hierarchical Clustering*** | • Does not require knowledge of prior number of clusters within data.<br>• Good for data analysis, provides more information than non-hierarchical clustering methods.<br>• Useful when k-means can't be used on nominal data sets. | • Not very efficient O(n^2) time complexity. |
| ***Expectation Maximisation (EM)*** | • Does not depend on the cluster shape being spherical.<br>• Can be used to determine the optimum number of clusters using cross validation on the log likelihood.<br>• Soft assigns a point to clusters, so you get a probability of a point being assigned to all clusters. | • Inefficient to run on some datasets if the optimal number of clusters. |
| ***Coweb*** | • Not as sensitive to noise as K-means clustering is. | • Cannot handle correlations between attributes very well.<br>• Can be computationally complex. |
| ***Farthest First Algorithm*** | • Reduces number of iterations compared to kmeans since placing each data point at the farthest cluster centroid first, reduces overall resignment.<br>• Requires prior knowledge of number of clusters. | • More complex than k-means, time complexity of O(nk) where n is number of data points and k is number of expected clusters. |
| ***Canopy*** | • Does not require prior knowledge of number of clusters.<br>• Is very efficient and quick to run.<br>• Can use canopy clustering to get the number of clusters then refine the clustering using that number in the k-means algorithm. | • Not very accurate when clustering. |

## Comparison to Bayesian Classification

The Bayes Network using the TAN algorithm had an error rate of only 68.95%. The best run of the clustering algorithm resulted in an error rate of 75.82%. This shows that clustering can group a good majority of instances to their correct classes and recover their class information. But the clustering algorithms will not perform quite as well as a Bayesian Network at classifying new data. Therefore it would be good to use clustering in an unsupervised learning scenario where you have no class information. But better to use Bayesian Networks when classifying data where class information is available.

## Overall Conclusion

As stated many times during the lectures and in the coursework specification the FER2018 dataset is not an easy item to work on. However, by use of various techniques and implementations, we can achieve higher results in regards to correctly classified instances, runtime and error rates. Compared with the results obtained in Coursework One we can see that by using techniques such as various Bayes Networks and Clustering we can achieve higher or similar results with a less overall runtime compared to our implementation of IBK. What has truly been learnt here is there is no "best" way of working with this dataset, or we have not yet yielded an implementation that allows us to classify with higher accuracy, but altogether we have seen there is different approaches to exploring and exploiting the data.

## References

[1]Gandhimathi, S. and Kalaivani, S. (2018). *An Empirical Analysis of Different Classification Algorithms for the Ecoli Protein Dataset*. [online] Ijircce.com. Available at: http://www.ijircce.com/upload/2015/october/36_An%20Empirical.pdf [Accessed 31 Oct. 2018].

[2]Sharma, N., Bajpai, A. and Litoriya, R. (2018). *Comparison the various clustering algorithms of weka tools*. [online] semanticscholar.org. Available at: https://pdfs.semanticscholar.org/eca2/5eb78be04ffe09b029dd1d36f5ba66749f29.pdf [Accessed 31 Oct. 2018].

[3]Frank, E., Hall, M. and Witten, I. (2016). *The WEKA workbench*. 4th ed. Morgan Kaufmann, pp.43-45.

## Appendices

## Further Reading Task One

In a study by S.Kalaivani, S.Gandhimathi at the PGP Arts and Science College, Namakkal, Tamilnadu, India conducted in 2015 research was undertaken to compare various forms of classification which included Naive Bayes and IBK - Nearest Neighbour. The study was carried out on an E-coli Dataset which contains 8 attributes and 336 instances which were then classified by various classification techniques in Weka. The goal was to gain insight into which classification method yielded the best accuracy alongside this the performance and error rate was also recorded and results produced for. In the results, Naive Bayes as a classification technique retained a much stronger performance and accuracy rating and also held a lower error rating. In the results, a performance of 0.047% and accuracy of 4.7619% increase was seen compared to IBK and an error rate decrease of 0.0101% was observed [1]. This differs from our own running of Naive Bayes Vs. IBK. where we experienced a loss of accuracy. However, as stated, in the E-coli dataset there are only 8 attributes to our own minimum of 14. This combined with our larger dataset is what we deem the reasoning for not being able to experience the same findings of S. Kalaivani and S.Gandhimathi.

Shown below is the code we used to filter our data using the CFSSubsetEval functionality of weka. We used this to pick out the most relevant attributes to prediction of the class and also remove redundant pixels that were not contributing. This reduced the dimensionality of our data and helped us both improve classification and reduce runtime.

```
def filter_data(self,data):
    print("Filtering Data..\n")
    filter = Filter(classname="weka.filters.supervised.attribute.AttributeSelection")
    aseval = ASEvaluation(classname="weka.attributeSelection.CfsSubsetEval",options=["-P", "1", "-E", "1"])
    assearch = ASSearch(classname="weka.attributeSelection.BestFirst", options=["-D","1", "-N", "5"])
    filter.set_property("evaluator", aseval.jobject)
    filter.set_property("search", assearch.jobject)
    filter.inputformat(data)
    filtered = filter.filter(data)
    return filtered
```