CS909 - ASSIGNMENT 2 - REPORT

RICHARD COLLINS

Part A

If the order of the input data is presented randomly the weights and intercept converge to different values.

Part B

1. We can calculate the expected error using the formula:

$$CI = p \pm k\sqrt{\frac{p(1-p)}{n}} \tag{1}$$

where, as stated in the question, $p = 0.0667$ and $n = 145$. Since we want to find CI at the 95% confidence level, $k = 1.96$. Plugging these values into equation 1 gives the result:

$$CI = 0.0667 \pm 0.0406 = [0.0261, 0.1073]$$

2. The bulk of the actual calculations involved in answering this question can be found in the Excel spreadsheet `t-stat-calc.xlsx`. A paired t-test was chosen to determine whether the two algorithms perform significantly differently. The t-score between the two algorithms was calculated using the formula:

$$t = \frac{\frac{\sum \delta_i}{n}}{\sqrt{\frac{\sum \delta_i^2 - \frac{(\sum \delta_i)^2}{n}}{n(n-1)}}} \tag{2}$$

where δ_i is the difference in accuracy between the two algorithms of the ith fold, and where n is the number of folds. Evaluating this equation gives $t = 0.973$. To determine whether this is significant or not we have to compare it to the p-value, which is found by looking it up with respect to the degrees of freedom. In this case, $DOF = n - 1 = 9$. Looking up p-value at a 95% confidence level, we find that p-value $= 2.262$. Therefore, we cannot say that, with a 95% confidence level, algorithm 1 will outperform algorithm 2. It is not until the 60% confidence level (p-value $= 0.883$) that the two algorithms can be said to be significantly difference in terms of performance. Therefore, we can say with 60% confidence that algorithm 1 will outperform algorithm 2.

3. The three clustering algorithms chosen for this question are: k-means, Hierarchical clustering, and DBSCAN. Before applying any clustering algorithms, a new reduced dataset (without the "type" column) called `ai2013papers.reduced` was made.

DBSCAN. The DBSCAN algorithm clusters data by considering the density of data points. Two parameters, ϵ (measure of neighbourhood radius of a particular point) and minPts (the minimum number of points in a neighbourhood for them to be included in a particular cluster). To determine ϵ we first calculate the average distances of every point to its k nearest neighbours (in our case $k = 8$. Next, these k-distances are plotted in ascending order. The "knee point" corresponds to the optimal ϵ. This plot is shown in Fig. 1.

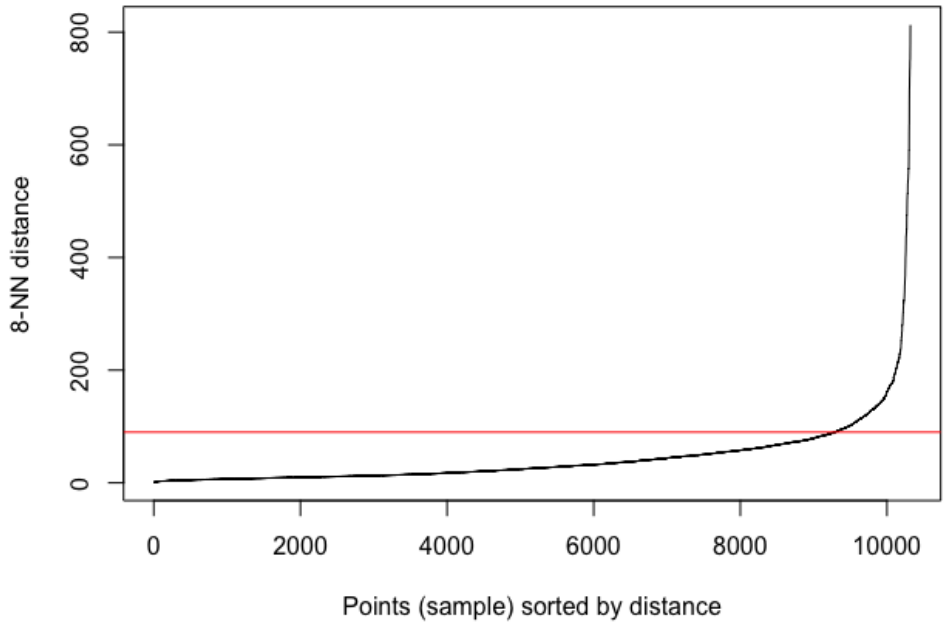

FIGURE 1. Determining the optimal value of ϵ.

From this plot we can see that the optimum $\epsilon = 90$. Using this and testing for various values on minPts, a hull plot was made, which can be seen in Fig. 2 along with its confusion matrix in Fig. 3.

We can see that DBSCAN is not an ideal clustering algorithm for these data. DBSCAN works well for data whose shape is not regular (not spherical), but has issues with varying densities and data with high dimensionality (since it relies on distance to determine cluster membership). This poor performance can be seen

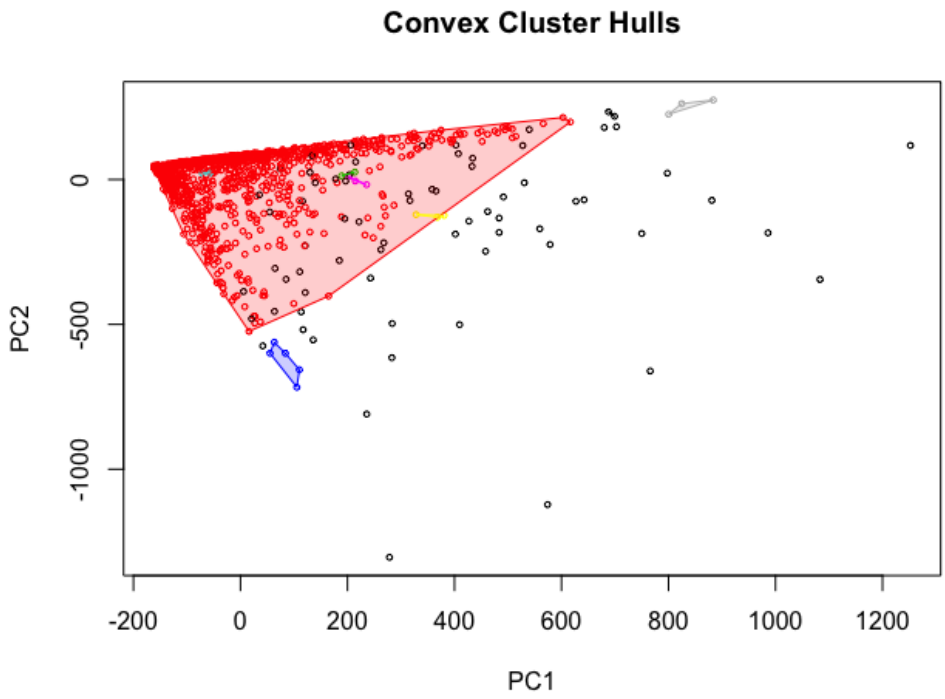

FIGURE 2. 2D plot of DBSCAN clustering algorithm.

```
> table(ai2013papers$type, ai2013.db.fit$cluster)
```

	0	1	2	3	4	5	6	7	8
Case Study	8	101	0	0	0	0	0	0	0
Correspondence	0	100	0	0	0	0	0	0	0
Essay	0	200	0	0	0	0	0	0	0
Opinion	2	91	0	0	0	0	0	0	0
Perspective	0	200	0	0	0	0	0	0	0
Research	21	167	2	4	2	2	2	0	0
Review	37	269	0	1	0	1	1	3	2
Viewpoint	0	74	0	0	0	0	0	0	0

FIGURE 3. Confusion matrix of DBSCAN clustering algorithm.

in the confusion matrix with one cluster containing the vast majority of samples, with only a few in the other 7 clusters.

k-means. A plot of the cluster solution in shown in Fig. 4 and confusion matrix in Fig. 5. We can see that much of the data is densely clustered with many

overlapping clusters. k-means was chosen because of its fast execution time and robustness even when some of its assumptions are broken (having balanced classes or similar cluster densities, for example). k-means is sensitive to outliers, which can skew some clusters unnecessarily. This effect can be seen when one examines Fig. 4 and obseres two outliers in the top left-hand corner of the plot.

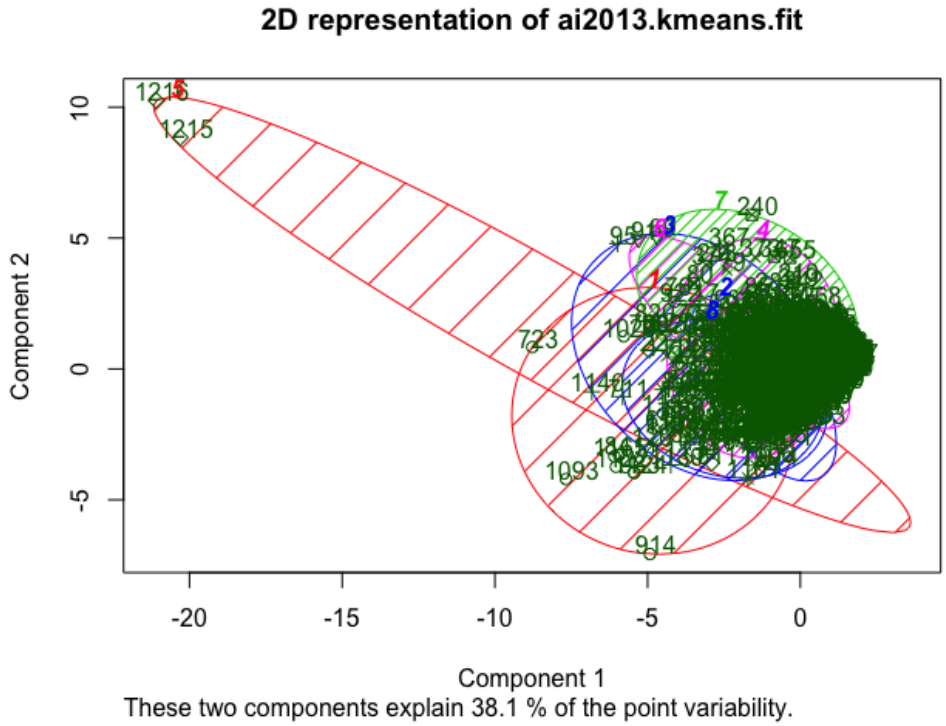

FIGURE 4. 2D plot of k-means clustering algorithm.

Hierarchical clustering. This algorithm was selected due to its flexibility over k-means. For one to apply (successfully) a k-means algorithm to a data set the data must be numeric, and ideally continuous and dense for it to be a good fit. Hierarchical clustering can deal with outliers and varying densities more easily.

PART C

a. The news reports are represented as a CSV file (`reutersCSV.csv`). Each row represents a document, and columns indicate which topics are relevant to a document (in the form `topic.trade`, `topic.corn` etc.). The title of the document and the actual text in the document are also columns of the CSV file. The data is read into a pandas data frame (`reports_df`) using the `read_csv()` function from the pandas library. This is then reduced to include only columns pertaining to

```
> table(ai2013papers$type, ai2013.kmeans.fit$cluster)

                  1   2   3   4   5   6   7   8
Case Study        1  11   4  20   5  24  26  18
Correspondence    0   0   0   0   0   0 100   0
Essay             0   1   7  83   0   7 102   0
Opinion           0   2   0   0   1  41  47   2
Perspective       0   5   0  14   0  22 158   1
Research          8  15  23  45   9  33   4  63
Review           35  19 105 114   8  14   7  12
Viewpoint         0   1   0   1   0  19  49   4
```

FIGURE 5. Confusion matrix of k-means clustering algorithm.

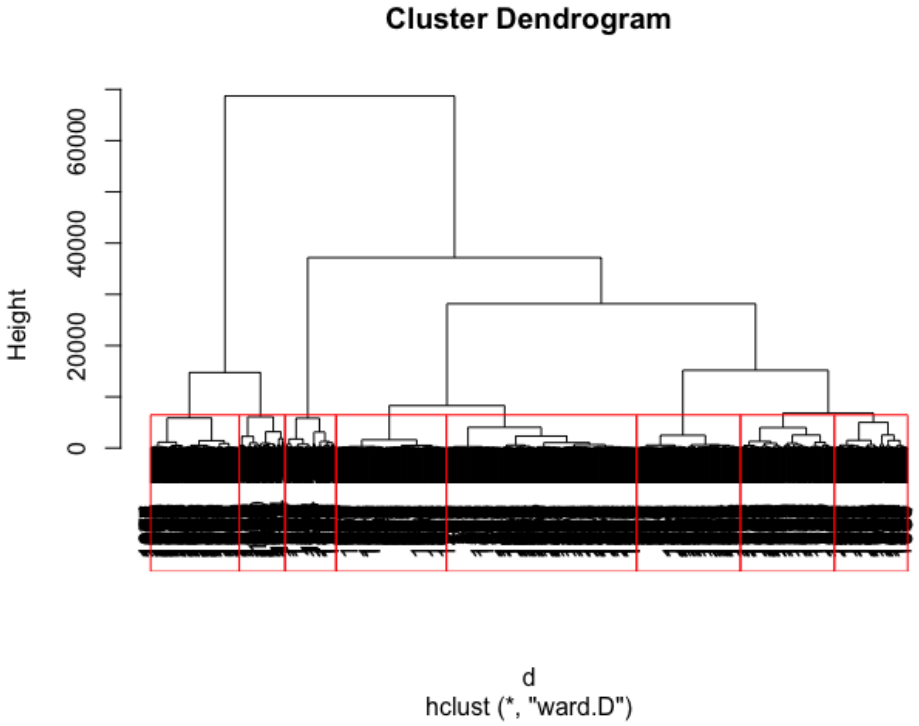

FIGURE 6. Dendrogram of hierarchical clustering algorithm.

the target topics as described in the assignment brief. This is done by defining a `columns_of_interest` list containing column names of relevant topics, "pid", "fileName", "purpose", "doc.title", and "doc.text". Rows that contained `NaN` in the doc.text column are also removed, leaving a data frame with 19779 rows and

```
> table(ai2013papers$type, groups)
                groups
                  1   2   3   4   5   6   7   8
  Case Study     21  17  26  17  18   4   5   1
  Correspondence  0   0   0   0   2  98   0   0
  Essay           5   0   5  84  74  15  17   0
  Opinion        19   3   0   1   1  69   0   0
  Perspective    14   2   2  18  86  78   0   0
  Research       32  47  90   1   1   1   5  23
  Review         24  15  35  56   3   3 124  54
  Viewpoint      10   3   2   0   3  56   0   0
```

FIGURE 7. Confusion matrix of hierarchical clustering algorithm.

15 columns. The number of rows is also reduced to include only reports with topic labels of those listed in the problem, which further reduces the number of rows to 8599.

Processing the text is achieved by using regular expressions to identify string patterns to either replace them, or remove them. All regular expressions and functions to process text can be found in the code from line XX to line XX. Inspecting the text of the reports revealed that not all HTML code rendered correctly when printed to the terminal. An easily identifiable example of this is the string "<", which encodes a "<" character. A regular expression is used to replace this string.

The news reports contain many numeric strings in various different forms. The form of the numeric was identified as potentially containing important information about the report topic (for example, many instances of percentages in the form XX.X could indicate changes in the stoke market, therefore appear more in reports of the "topic.money.fx" class). This line of processing was abandoned however due to the presence of other indicators in the text as to the type of number (e.g. "pct" for a percentage) also acting as an indicator. Instead, any stand-alone numbers in the text are replaced with the placeholder " numeric " (making sure to add white space either side so it does not concatenate with adjacent words). All non-alphanumeric characters are also removed using a regular expression.

The final stage of text processing is to lemmatise the words in each document. This is achieved by importing the `WordNetLemmatizer` from the `nltk.stem.wordnet` library. This is done to reduce the vocabulary size (therefore reduce overall running time). Before getting unigrams of the reports, each report is standardised by separating each word by a single whitespace, carried out by the `remove_extra_whitespace(doc)` function.

b. The `get_feature_space(df, training = True)` function not only carries out the text processing described above, but also generates unigram features from the corpus. The `getUnigrams(vec, training corpus = None, testing corpus = None)`function takes in a count vectoriser and a corpus of text (a list of strings). The output is a numpy array of size $D \times V$ where D is the number of documents and V is the vocabulary size. `TfidfVectorizer(stop words = ‘‘english’’)` is used for the vectoriser instead of `CountVectorizer()` because it penalises words that occur frequently in the corpus, therefore giving more importance to rarely occurring words.

`getUnigrams` contains vestiges of code written for the case of splitting the data set into training and testing sets. Since 10-fold cross validation is applied, splitting in this manner is unnecessary. However, keeping the code as it is doesn't affect the functionality of the model. In practice only the case `‘‘training_corpus = True’’` is used where all 8599 reports are considered. After generating the feature space from these reports, the shape of the numpy array `X` is (8599, 24779), reflecting the large vocabulary of the corpus. The matrix is sparse, being populated with mostly zeros and fractions indicating the presence and relative frequency of a word in each report. Each column represents a word in the feature space, which can be looked up using the `mapping` dictionary, which maps word to column number.

The column vector `y` list the topic labels of each report, which are represented as ones and zeros in a 10-item list (reflecting the 10 topics of interest). A one in the second position of a list indicates a `‘topic.acq’` label, in the third position `‘topic.money.fx’`, etc.

c. One decision tree and two random forest algorithms were used to classify the reports. Decision trees are easily interpreted and performs well with both numerical and categorical data. The largest drawback is the tendency to overfit when the feature space is large and sparse (as is the case in this example). A random forest is an ensemble classifier made up of a number of decision trees built from subsets of the training data. Averaging results over all the decision trees increases classification accuracy and avoids overfitting.

The effects of overfitting can be elevated further by performing cross validation with k folds. Standard practice dictates that 10 folds is an optimum choice. This is a compromise between having enough examples in the training set, to lowering the variance of classification on a small test set. 10-folds mean that 90% of data is held for training, and then the remaining 10% used for testing. This process is repeated 10 times to allow each example to be used in training and testing, reducing the risk of over fitting. Further tactics to prevent overfitting include adjusting the settings of each classifier, details of which are included in the next section.

d. Three parameters were adjusted to improve accuracy. These were max_depth (the number of layers in a decision tree), min_samples_leaf (minimum number of

samples in each leaf), and, for the random forests, n_estimators (the number of decision trees over which the results were averaged). Evaluation was carried out by assessing the accuracy, as well as precision and recall. From these results, the F1-score was also calculated by:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3}$$

where

$$\text{recall} = \frac{TP}{TP + FN} \qquad \text{and} \qquad \text{precision} = \frac{TP}{TP + FP} \tag{4}$$

It was decided that micro-averaging would be used instead of macro-averaging. This is because micro-averaging takes into account unbalanced classes. Fig. ref-fig:classCount shows this imbalance.

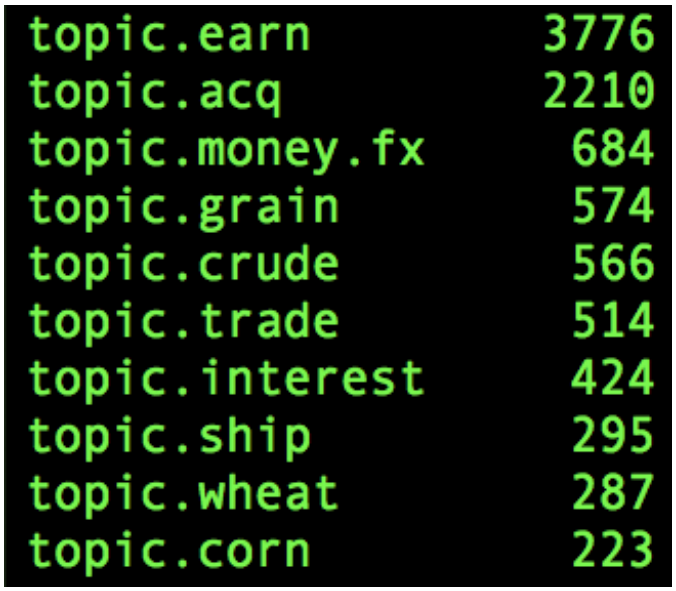

FIGURE 8. Count of each topic label.

DecisionTreeClassifier.

- max_depth = 11
- min_samples_leaf = 1

For `DecisionTreeClassifier`, all the parameters take their default values except for max_depth, which is set to 11. This has two functions, the first is to increase the number of examples in each leaf, therefore increasing the accuracy with which a classification can be made. The second, more practical function is to reduce running time. While testing the DecisionTreeClassifier it took on average

45s per fold to fit, which became impractical when running multiple tests. Interestingly, limiting the depth of the tree did not reduce the accuracy of the model significantly. The exact figures quoted vary from test to test. A screenshot of one such test is shown in Fig. 9.

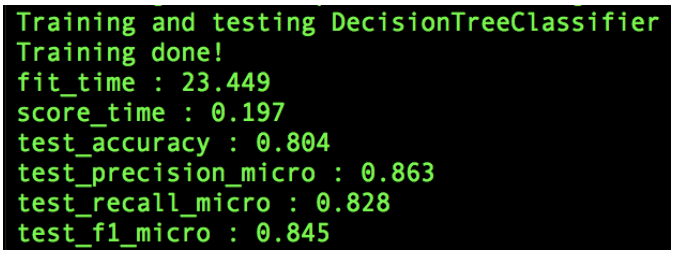

FIGURE 9. Example results of `DecisionTreeClassifier`.

All figures in the figures are mean values from the 10 folds of the cross validation. The parameter `fit_time` gives the average time to fit the data in seconds. The accuracy is fairly high for the decision tree classifier at 80.4%, however, this could be due to overfitting (despite using cross validation).

RandomForestCLassifier1.

- n_estimators = 3
- max_depth = None
- min_samples_leaf = 1

Two key changes were made for `RandomForestCLassifier1`. The first and most obvious is the increase in number of decision trees used to classify reports to three instead of the one used in the `DecisionTreeClassifier`. The second chage was to have no limit on the depth of the trees. This was shown, in preliminary testing, that using max_depth = 11 had little to no effect on the `DecisionTreeClassifier` accuracy. A screenshot of the output from this classifier is shown in Fig. 10.

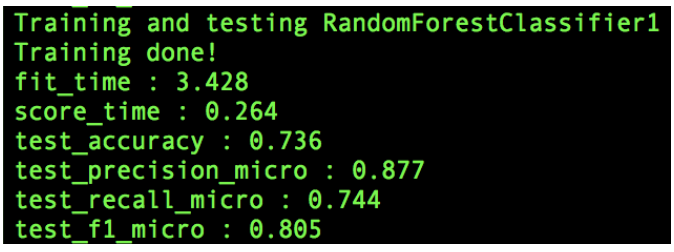

FIGURE 10. Example results of `RandomForestCLassifier1`.

Noticeably, the accuracy has decreased, owing in part to the averaging effect over three trees. Interestingly, the average fit_time for each fold is much lower than that of the `DecisionTreeClassifier`, even though there are three tree to construct for each fold instead of one.

RandomForestCLassifier2.

- n_estimators = 30
- max_depth = None
- min_samples_leaf = 2

Two further changes were made to the final classifier. These were to increase the number of trees to 30, and to require at least 2 samples per leaf. The effect of this is to increase the number of samples over which averages can be calculated, and substantiating the classification of each leaf (further reducing the risk of overfitting). Results from this classifier can be found in Fig. 11

```
Training and testing RandomForestClassifier2
Training done!
fit_time : 13.973
score_time : 0.383
test_accuracy : 0.721
test_precision_micro : 0.974
test_recall_micro : 0.708
test_f1_micro : 0.819
```

FIGURE 11. Example results of `RandomForestCLassifier2`.