CO395 Machine Learning CW1

Chang, Yoon Hyuk
01115052
Miller, Tom
01075719
Bagga, Shreyus
01070439
Liaskas, Athanasios
01079253

February 13, 2018

Contents

1 Implementation

1.1 Abstract

The implementation of this coursework task consisted of constructing a class to represent decision trees as well as the corresponding functions needed to build them from the provided training data. The trees that were constructed for each emotion were provided testing data and, through a predicting function, determined the matching emotion. The results of this process were then displayed through a confusion matrix, with evaluation conclusions drawn from measures such as recall rates, precision rates, F1 measure, and finally the classification rate.

1.2 Decision Tree

The structure of the decision tree class is as follows:

- tree.nodeOp (example: AU5)
- tree.nodeKids (example: [treeNode, treeNode])
- tree.isLeaf (example: False)
- tree.leafClass (example: 1)

Relevant class functions:

- tree.setKid(i, node) - sets kid i to node

To build the decision tree from a set of data, the $DTL(examples, attributes, binary\_targets)$ function is used, which takes 3 arguments: a set of feature data arrays, an array of attributes and an array of corresponding binary targets that was constructed for the emotion tree being built.

The implementation of this function is based on the ID3 algorithm which operates by splitting the tree on the attribute with highest information gain into sub trees, and once no attributes are left to split then the mode of the *binary_targets* is returned as a leaf. Other sub-cases such as, the splitting on an attribute returning a branch with no example data, also results in this being returned. Finally, if all elements of *binary_targets* are equal then the value of those elements is returned as a leaf.

1.3 Selecting the best attribute

In implementing the ID3 algorithm the actual selection of the best attribute is done through the function $CBDA(ex, attr, b\_targets)$.

This function operates on the premise of the formulas below:

- If p is the positive and n the negative examples in the training data,
  - $Gain(attribute) = I(p, n) - Remainder(attribute)$
  - $I(p, n) = -\frac{p}{p+n}\log_2(\frac{p}{p+n}) - \frac{n}{p+n}\log_2(\frac{n}{p+n})$
- and if p_0, n_0 is the number of positive and negative examples with the attribute value 0 respectively, and p_1, n_1 conversely,
  - $Remainder(attribute) = \frac{p_0+n_0}{p+n}I(p_0, n_0) + \frac{p_1+n_1}{p+n}I(p_1, n_1)$

It implements the above by first calculating the total entropy of the given data through passing the **number of positives** p (1's) and the **number of negatives** n (0's) in *b_targets* into $calcEntropy(p, n)$. Each attribute then has its gain calculated through working out the remainder of the attribute to be subtracted from the entropy. The attribute with the largest information gain is then returned as the best attribute.

1.4 Cross-validation

The motivation behind the use of cross validation lies in the fact that in training and fitting the models to the entire dataset, there is no data left for out-sample testing. Cross validation therefore separates a dataset into **training** and **testing** sets, most commonly in a **9:1** split.

Our implementation used **10-fold** cross validation, which is the process of rotating through the whole dataset in the 9:1 ratio of training to testing data, and building a confusion matrix from each fold. Combining those matrices then gives us a representation of the whole dataset.

1.4.1 Code

The function performing this is `crossValidation(nFolds, x2, y2)`, which takes 3 arguments as input: the number of folds, a set of feature data arrays and an array of binary targets.

The implementation of this function begins by splicing `x2` and `y2` into arrays of size `(dataset size)/nFolds`, and appending them to xSplit and ySplit. For each fold enumerated through, an element of `xSplit` and `ySplit` is used as test data (`xTest`, `yTest`) whilst the rest is used to train the trees (`xTrain`, `yTrain`). This data is then passed through `genTrees` and `testTrees` before calling `confusionMatrix` and appending it's output to an array. The array of confusion matrices are then summed into a matrix that is representative of the whole dataset.

1.5 Evaluation: Confusion Matrix and average results

In order to evaluate the **quality** of the **classification process** after the test examples are run through the decision trees, several measures are calculated; comprising of the precision rate, recall rate and classification error. The inputs for the evaluation function are:

- array of predicted emotions for each example
- array of actual emotions for each example

The **confusion matrix** was built up by incrementing the relevant index of the **6x6 matrix**, indexed by the actual emotion (a number between 1-6) and the predicted emotion. For cases where the predicted emotion output was correct (equivalent to the actual emotion), the relevant diagonal element of the confusion matrix was incremented. Overall, the diagonal of the matrix represents the True Positive (TP) results.

	Predicted					
Actual	**Anger**	**Disgust**	**Fear**	**Happiness**	**Sadness**	**Surprise**
Anger	TP	FN	FN	FN	FN	FN
Disgust	FP	TN	TN	TN	TN	TN
Fear	FP	TN	TN	TN	TN	TN
Happiness	FP	TN	TN	TN	TN	TN
Sadness	FP	TN	TN	TN	TN	TN
Surprise	FP	TN	TN	TN	TN	TN

Figure 1: Confusion matrix with its elements represented as positive for the emotion Anger, and negative for every other emotion on the rows/columns. This is the structure used when obtaining the **TP**, **TN**, **FP**, **FN** values in the evaluation function, for each emotion (in this case, anger).

The two main measures used to evaluate how well the classifier performs on a set of test data are the **recall rate** (completeness of the information retrieval) and the **precision rate**

(accuracy of the information retrieval). These two individual measures are combined in the F_1 measure. The three measures are calculated as follows:

$$precisionrate = TP/(TP + FP)$$
$$recallrate = TP/(TP + FN)$$
$$F_1 = 2\frac{precision * recall}{precision + recall}$$

1.5.1 Averaging Results

An average of the results are obtained after K-fold cross validation to have a final confusion matrix and evaluation that is more representative of the overall performance on unseen data compared to the evaluation on only one fold. The confusion matrices for each of the 10 folds are summed together in the function $crossValidation(10, x2, y2)$ and fed into $printStats(confusion\_matrix, 1)$ which uses the resultant summed confusion matrix to calculate the **precision rate recall rate** and F_1 measure for each class as well as the overall **classification rate**.

2 Results

2.1 Evaluation: Clean Data

	Anger	*Disgust*	*Fear*	*Happiness*	*Sadness*	*Surprise*
Anger	74	13	8	7	27	2
Disgust	22	121	10	16	17	12
Fear	10	10	72	5	4	17
Happiness	11	18	5	168	7	6
Sadness	14	16	10	16	58	18
Surprise	13	11	28	20	15	119

Figure 2: Confusion matrix - 10 fold Cross Validation Summation

	Anger	*Disgust*	*Fear*	*Happiness*	*Sadness*	*Surprise*
Recall (%)	56.5	61.1	61.0	78.1	43.9	57.8
Precision (%)	51.4	64.0	54.1	72.4	45.3	68.4
*F*1 (%)	53.8	62.5	57.4	75.2	44.6	62.6

Figure 3: Statistics Measurements - 10 fold Cross Validation Summation

–Classification Rate: 61.2%

–TP: 74 121 72 168 58 119
–FP: 70 68 61 64 70 55
–TN: 799 734 821 721 798 739
–FN: 57 77 46 47 74 87

When running the 10-fold cross validation on clean data, a **classification rate** of **61.2%** is achieved. Observing the **recall rates** for each of the 6 emotions, the **strongest** completeness and accuracy of information retrieval was that of **happiness** (highest recall, precision rates and F1 value), while the **weakest** at completeness and accuracy was **sadness**.

Looking at the AUs of each emotion, **happiness** has a very simple AU definition: $\{12\}$ and $\{6, 12\}$. Furthermore, since AU12 only appears in the happiness emotion, the resulting decision tree for happiness has a root node of AU12 and the tree is arguably more effective at classifying

its emotion than the other trees. Another reason for this is that happiness had the largest total amount of input sample data when training the trees (215 samples), improving the tree's classification performance on average for each fold. In contrast, **sadness** which has a much longer definition, was often confused by disgust, happiness and surprise (as can be seen by the **false negatives** in the sadness row of the confusion matrix) and had only 132 samples of data in total to train and test on during cross validation. Although it should be noted that the emotion **fear** had even fewer samples at 118, but is defined with many more AUs than sadness.

2.2 Evaluation: Noisy Data

	Anger	*Disgust*	*Fear*	*Happiness*	*Sadness*	*Surprise*
Anger	21	12	19	11	16	9
Disgust	15	117	13	12	21	9
Fear	13	25	89	20	20	20
Happiness	11	25	15	135	11	11
Sadness	17	8	13	9	50	13
Surprise	13	26	22	26	21	112

Figure 4: Confusion matrix - 10 fold Cross Validation Summation

	Anger	*Disgust*	*Fear*	*Happiness*	*Sadness*	*Surprise*
Recall (%)	23.9	62.6	47.6	64.9	45.5	50.9
Precision (%)	23.3	54.9	52.0	63.4	36.0	64.4
*F*1 (%)	23.6	58.5	49.7	64.1	40.2	56.9

Figure 5: Statistics Measurements - 10 fold Cross Validation Summation

–Classification Rate: 52.4%

–TP– 21 117 89 135 50 112 –TN– 843 717 731 714 801 718
–FP– 69 96 82 78 89 62 –FN– 67 70 98 73 60 108

As expected, the classification rate for **noisy data (52.4%)** was lower than for **clean data (61.2%)**. One reason for this observation is that the clean data was obtained by human experts which is assumed to be correct for every example, whereas the noisy data was collated by an automated system, hence the "actual" outputs recorded for each sample may be inaccurate. Thus, the lower classification rate suggests that the hypothesis that noisy data was less accurate is correct.

The evaluation measures have some similarities with clean data, such as **happiness** having the highest recall and precision rates and highest F1 value as a result. This can be attributed to a higher sample size and more concise AU definition, as described previously for clean data. **Anger** was by far the worst in terms of accuracy and completeness of information retrieval, as seen by the recall and precision rates and F1 value (23.6%). This could be attributed to the very low sample size for the emotion (88 samples) compared to the rest of the emotions. Therefore, the decision tree for anger was on average not as robust as for other emotions per fold due to a smaller training sample size for that emotion. The main incorrect classifications of anger were **fear** and **sadness**.

Anger, fear and sadness all have an overlap in AUs. For example, fear and anger share $\{4, 5, 25 \parallel 26\}$ which makes up half of some of the attribute combinations of anger. Since the data is noisy, some of the recorded AU attributes for anger may have been misclassified as those

that belong to fear or sadness. The noisy data was obtained by an autonomous system and the evaluation results suggest that the three emotions can look similar, especially to an autonomous emotion recognizer.

3 Questions

3.1 Noisy-Clean Datasets

There is a **small but noticeable** difference in performance between the clean and noisy datasets. In general, the performance for noisy dataset was slightly worse than the performance against the clean dataset, which is expected since the noisy data set could contain inaccurate or missing attributes due to the noise component. Should any of these attributes play a key role in classifying a particular emotion, it could impact how well it can identify that emotion.

There was about a **9% reduction** in the classification rate for noisy data versus clean. Since the classification rate is influenced by the quality of data and the type of algorithm used, where in this case the algorithm is fixed, it can be deduced that the noisy data has a lower quality of data than the clean dataset.

Comparing the **F1 values** across the two datasets gives us a measure of both precision and recall rate for each emotion; **Anger** saw the highest reduction in F1 value, from $\approx 53.8\% \rightarrow 23.6\%$. This could perhaps be due to the amount of noise that has been applied onto the noisy dataset versus the clean dataset for that emotion and also the low training sample size of anger. All other F1 values saw a slight decrease of $\approx 10\%$.

Given that the clean data was hand-picked whereas it was machine generated for the noisy case, it is expected that there is an overall decrease in performance when training on the noisy data versus the clean dataset. The auto-generated data may have made mistakes in capturing the attributes correctly, leading to errors in generalization during the learning process.

3.2 Ambiguity

The ambiguity in classifying input data into an emotion comes from the fact that multiple trees can give a positive match, or alternatively, no trees could match at all. In addressing this problem, and the strategies to implement, an increase in the classification rate was the key indicator of success.

The ambiguity strategy is only a partial factor in achieving a good classification rate. To judge the range of this factor, it was important to calculate the maximum rate that could be achieved from the ambiguity strategy alone. This maximum rate value was obtained by comparing all tree match predictions to the actual data outputs. If any of the potential predicted emotions for an example matched the actual correct emotion, it was counted as a positive match for that example. After doing this through 10-fold cross validation and then averaging over 3 runs, it was found that the current implementation of the decision trees **limited** a perfect ambiguity strategy to approximately **69.2%**. Knowing this limit allows a better judgment of the exact impact of any ambiguity strategy implemented.

Each ambiguity strategy must take care of two scenarios: multiple emote matches, and no emote matches. The following two strategies were implemented:

Random Selection

- Multiple Matches: Select a random emotes from those that match
- No Matches: Select a random emote

Depth Selection

- Multiple Matches: Select the emote whose tree had the minimum depth from those that match

- No Matches: Select a random emote

The advantages of **Random Selection** focus on its ease of implementation, as it simply takes a list of emote matches and randomly selects an element to be the classification for the tested data. Its disadvantages are that it assumes each emote is equally as likely to be the correct match, taking no note of of the performance of each tree or any details corresponding to tree structures. These disadvantages were the basis for the second implemented strategy.

The advantages of **Depth Selection** stem from the fact that the details of the tree traversal from which an emote is predicted is taken into account. Through comparing the depths at which the matches were returned, the emote that matched at the minimum depth was chosen to be the classification for the tested data. The disadvantage of this is that deeper, more accurate trees may be unfairly disadvantaged.

Overall when testing the two strategies above and applying 10-fold cross validation to clean and noisy datasets, the data in figure 6 and figure 7 was retrieved, in which choosing Depth Selection gave an approximately **4%** and **2%** increase in classification rate respectively. This is consistent with the above reasoning that taking into account tree traversal details is a useful measure in resolving ambiguity.

Random Selection

$$\begin{bmatrix} 70 & 9 & 8 & 8 & 27 & 9 \\ 22 & 125 & 7 & 10 & 23 & 11 \\ 16 & 9 & 57 & 5 & 6 & 25 \\ 11 & 19 & 8 & 155 & 10 & 12 \\ 17 & 16 & 14 & 13 & 50 & 22 \\ 14 & 14 & 27 & 14 & 11 & 126 \end{bmatrix}$$

Classification Rate: 0.583

(a) Clean Data

$$\begin{bmatrix} 19 & 11 & 16 & 15 & 14 & 13 \\ 17 & 108 & 17 & 18 & 12 & 15 \\ 16 & 21 & 86 & 19 & 20 & 25 \\ 11 & 22 & 24 & 127 & 11 & 13 \\ 19 & 12 & 17 & 8 & 41 & 13 \\ 12 & 19 & 20 & 20 & 17 & 132 \end{bmatrix}$$

Classification Rate: 0.513

(b) Noisy Data

Figure 6: Confusion Matrices for Random Selection

Depth Selection

$$\begin{bmatrix} 77 & 11 & 10 & 6 & 23 & 4 \\ 21 & 131 & 6 & 12 & 15 & 13 \\ 13 & 7 & 64 & 3 & 8 & 23 \\ 10 & 16 & 3 & 167 & 7 & 12 \\ 12 & 12 & 12 & 19 & 52 & 25 \\ 10 & 16 & 28 & 13 & 10 & 129 \end{bmatrix}$$

Classification Rate: 0.62

(a) Clean Data

$$\begin{bmatrix} 21 & 14 & 21 & 13 & 12 & 7 \\ 18 & 110 & 19 & 13 & 19 & 8 \\ 20 & 21 & 84 & 27 & 14 & 21 \\ 17 & 18 & 15 & 139 & 9 & 10 \\ 16 & 18 & 11 & 7 & 41 & 17 \\ 9 & 18 & 16 & 21 & 19 & 137 \end{bmatrix}$$

Classification Rate: 0.532

(b) Noisy Data

Figure 7: Confusion Matrices for Depth Selection

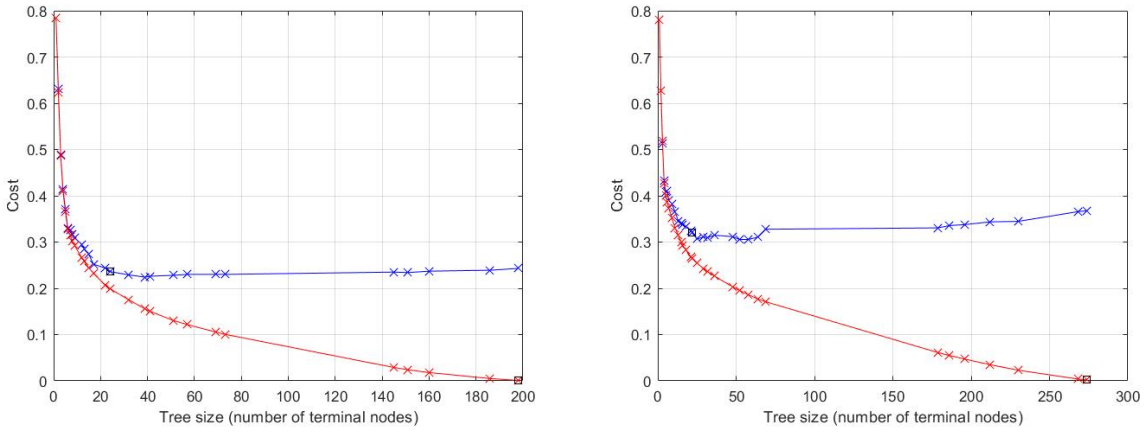
3.3 Pruning Example

The `pruning_example` function addresses one of the **disadvantages** of the **ID3 algorithm**. The disadvantage lies in the fact that the decision tree may over-fit the data as the algorithm

grows the tree to perfectly classify the training examples without any backtracking. Therefore, the learning algorithm may converge to a locally optimal solution, rather than a global solution.

The `pruning_example` function essentially addresses this problem by the following steps:

1. Create a classification tree using `classregtree`
2. Test the tree against two testing methods: cross validation and substitution, returning a vector of costs and the best level or depth.
3. Prune these trees on their best levels
4. Return plot of cost versus tree size(depth) based on the two different testing methods



(a) Clean Data (b) Noisy Data

Figure 8: Plots for clean, noisy data

The plots show a clear difference in the cost of pruning versus tree length for the two different testing methods. For **cross-validation** testing, the lowest cost point is around 40 terminal nodes for clean data and 52 or 35 terminal nodes for noisy data. Whereas for the **re-substitution** method, the lowest cost point was seen to be around 200 and 275 terminal nodes respectively.

4 Tree Diagrams

The **decision trees for each emotion** on the clean dataset are shown below, limited to `depth=5`. The trees are presented in left to right direction (rotated left). The full tree can be seen by running `print(decision_tree)`, and its implementation can be seen under the `class Tree.__str__()` method. The parameter `printLimit` can be tuned to delimit the maximum depth of the tree printed (`printLimit=-1` for the entire tree)

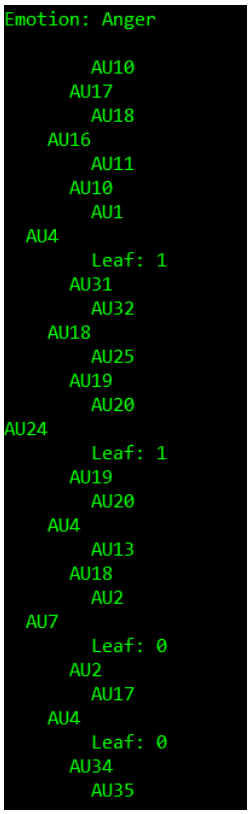


Figure 9: Decision tree for anger

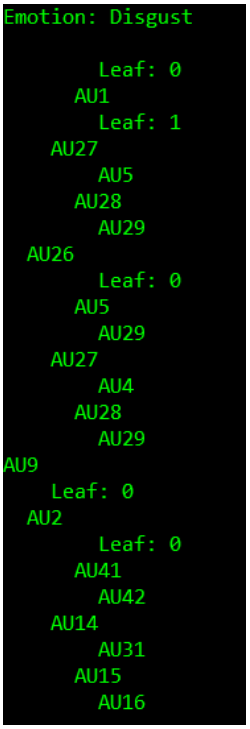


Figure 10: Decision tree for disgust

Figure 11: Decision tree for fear

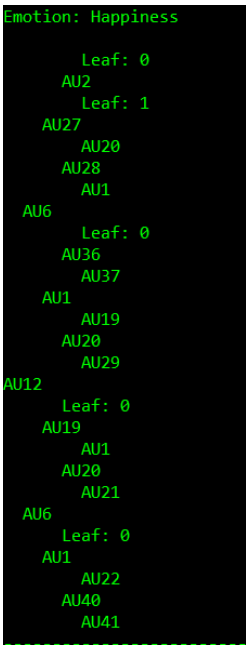


Figure 12: Decision tree for happiness

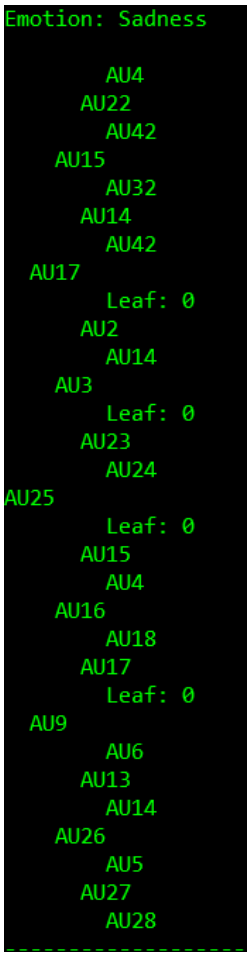


Figure 13: Decision tree for sadness

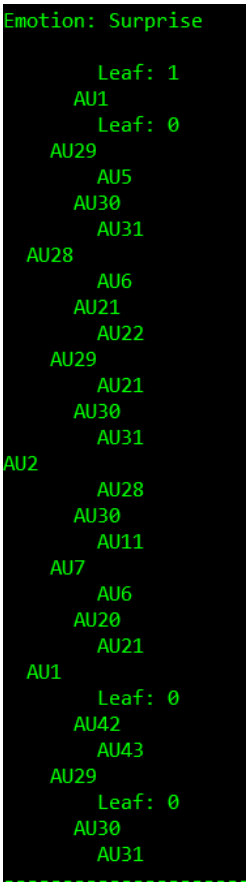


Figure 14: Decision tree for surprise