CSE 446: Machine Learning
Winter 2018

Assignment 1

from
Lukas Nies
University of Washington

01/18/18

Contents

0 Policies

0.1 List of Collaborators

My collaborators were Edith Heiter (discussed Problem 1 and Problem 3) and Julia Hesss (discussed Problem 3 and Problem 4). The development of the answers though was completely independent and individually.

0.2 List of Acknowledgments

None.

0.3 Policies

I have read and understood these policies.

1 Problem: Criteria for Choosing a Feature to Split

We build a tree with the dataset D consisting n negative examples (label 0) and p positive examples (label 1).

1.1 Not Splitting

If we are at the bottom of our decision tree and don't have any features left to split, consider the subset D' of data D with n' negative and p' positive examples. The smallest number of mistakes we can make in this subset is given by

$$\text{err}(D') = \min(n', p') = \begin{cases} p' & \text{if } p' < n' \\ n' & \text{if } n' < p' \end{cases} \tag{1}$$

To achieve the minimum the node must be labeled accordingly, with 1 if $n' < p'$ or with 0 if $n' > p'$.

1.2 Splitting

Now we have a new feature Φ which splits the subsection D' according to the contingency table: By splitting we generate two new sub-nodes: (n_0, p_0) and (n_1, p_1). The error for splitting is given by the sum of the errors of both nodes

$$\text{err}(D') = \min(n_0, p_0) + \min(n_1, p_1), \tag{2}$$

where $n' = n_0 + n_1$ and $p' = n_1 + p_1$. The error reduction rate (err_red) is given by the reduction of error if comparing the error of "not splitting" with the error of "splitting", divided by the total number of examples in node D':

$$\text{err\_red}(D') : \frac{\min(n', p') - (\min(n_0, p_0) + \min(n_1, p_1))}{|D'|}. \tag{3}$$

y	Φ	
	0	1
0	n_0	n_1
1	p_0	p_1

Table 1: Contingency table for Φ'

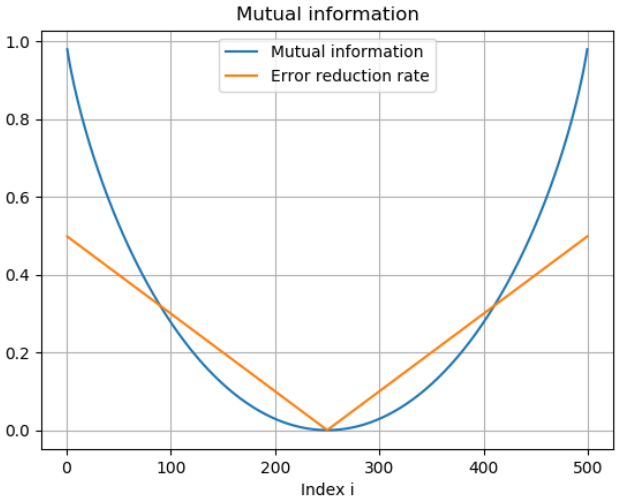


Figure 1: Comparison of mutual information between feature and label and error reduction rate for splitting the dataset.

Consider the maximal possible error (in this case for binary data) when $n' = p' = 0.5|D'|$

$$\mathrm{err}_{\max}(D') = \min(n', p') = 0.5|D'|, \tag{4}$$

then the maximal possible error reduction is given by

$$\mathrm{err\_red}(D') : \frac{\min(0.5|D'|, 0.5|D'|) - (\min(n_0, p_0) + \min(n_1, p_1))}{|D'|} = 0.5, \tag{5}$$

where either $p_0 = 0$ or $n_0 = 0$ and $p_1 = 0$ or $n_1 = 0$ (maximal information gain).

1.3 Mutual Information

The mutual information is a measure of information gain when splitting a dataset. In figure 1 the mutual information and the error reduction rate are plotted for problem 1.3. Both functions are symmetric around index 250 which represents equally distributed examples in form of $n_0 = n_1 = p_0 = p_1 = 250$.

As we showed in the previous part of this problem, the error reduction rate is maximal for the case where the error of a node is 0 after splitting. Corresponding to this the information gain in form of the mutual information is highest.

For increasing indices the information gain and the error reduction rate decrease until a minimum in $i = 250$, afterwards rising again.

2 Decision Stumps and Trees

2.1 Built two decision stumps

In figure 2 the comparison of of error rates and mutual information for different threshold splits and features are plotted.

The top row shows the error as a function threshold. The feature "age" generates two minimums corresponding to two possible splits to minimize the error when splitting. For feature "salary" we get only one minimum but the error reduction is larger.

The mutual information is plotted in the bottom row. Since the mutual information gives the information gain for a split we have to search for maximums. Both features give one maximal values which we can choose as a threshold to split the node.

In figure 3 the decision stumps are plotted for the best thresholds found in figure 2. The error rate depends on choosing a feature to split and on the method to calculate the best threshold. The best errors after splitting only once were achieved by choosing mutual information as a criteria (both features) or choosing "salary" when counting errors (again, see figure 3).

2.2 Build Decision Tree

In figure 4 the greedily built decision trees are shown. We can see that the lower tree (built by using mutual information) only needs six nodes to reach an overall error of zero on the training set while the upper tree (built by counting errors) needs ten nodes.

2.3 Multivariate Decision Tree

To built the multivariate decision tree we can utilize some of the basic geometric foundations used in the Perceptron algorithm. We have to learn the function

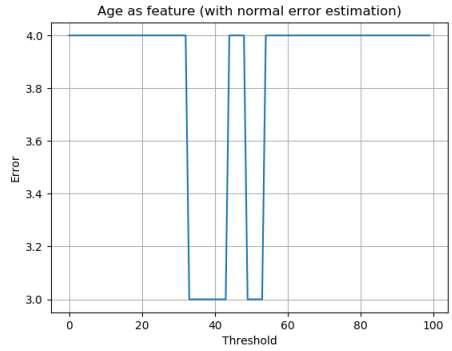
$$f(\vec{x}) = \left[\sum_{d=1}^{2} w_d x_d\right] - 1 = w_1 x_{\text{age}} + w_2 x_{\text{salary}} - 1. \tag{6}$$

Since we work in two dimensions it's most convenient to just "guess" the hyperplane of the data which is in this case of dimension one, a line. Let $\vec{h}$ be the vector of this hyperplane. By drawing a straight line to separate negative and positive examples we can calculate the slope. This yields in

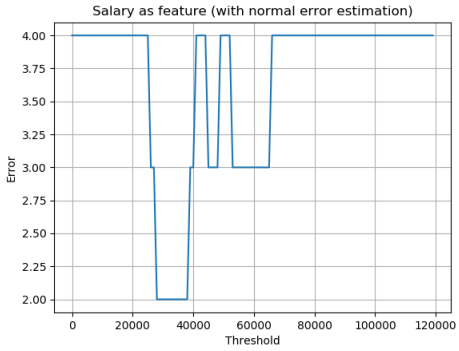
$$\vec{h} = \begin{pmatrix} 35 \\ 40000 \end{pmatrix}$$

where the first component corresponds to the age and the second component corresponds to the salary. According the the Perceptron the weight vector $\vec{w}$ is perpendicular to the hyperplane. To get $\vec{w}$ we can perform a rotation of $\vec{h}$ by applying a 2D rotation matrix:

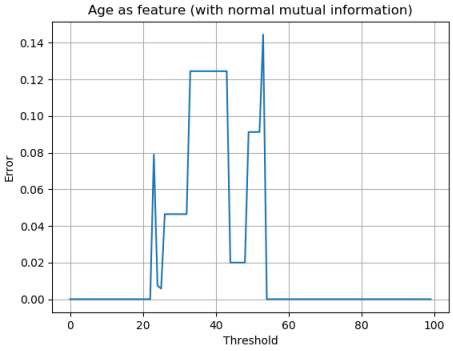
$$\vec{w} = \text{rot}(90°) \cdot \vec{h} = \begin{pmatrix} \cos(90°) & \sin(90°) \\ -\sin(90°) & \cos(90°) \end{pmatrix} \cdot \begin{pmatrix} 35 \\ 40000 \end{pmatrix} = \begin{pmatrix} 40000 \\ -35 \end{pmatrix}$$



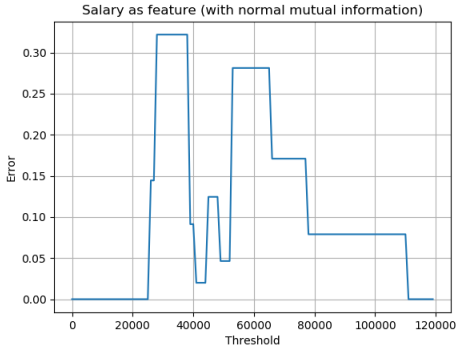
(a) Error for the feature "age"



(b) Error for the feature "salary"



(c) Mutual information for the feature "age"



(d) Mutual information for the feature "salary"

Figure 2: Comparison of error rates and mutual information for different threshold splits and features. For generating this data a script has been written to scan through different thresholds with a fine reasonable step width. The error respectively mutual information was calculated as shown in problem 1.

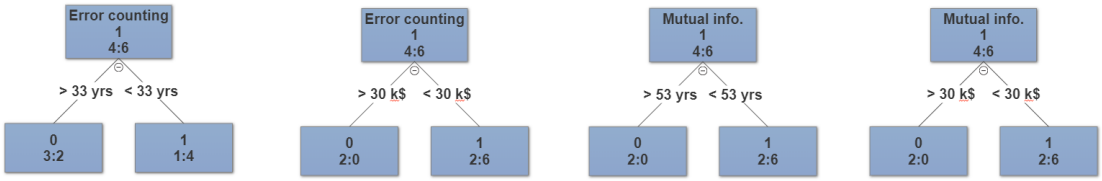


Figure 3: Decision stumps built after searching for the best threshold (Fig 2).

We can norm this vector to get some smaller values (not necessary but for the sake of readability):

$$\vec{w}_n = \frac{1}{\sqrt{(40000)^2 + (-35)^2}} \begin{pmatrix} 40000 \\ -35 \end{pmatrix} = \begin{pmatrix} 0.999999617 \\ -8.75 \times 10^{-4} \end{pmatrix}$$

This weight vector (which we "learned" by manually looking at the data) gives us now the hypothesis f to test on the trainings set. The decision now is

$$f(\vec{x}) = \text{sign}\left[0.99999961 \cdot x_{\text{age}} - 8.75 \times 10^{-4} \cdot x_{\text{salary}} - 1\right]. \tag{7}$$

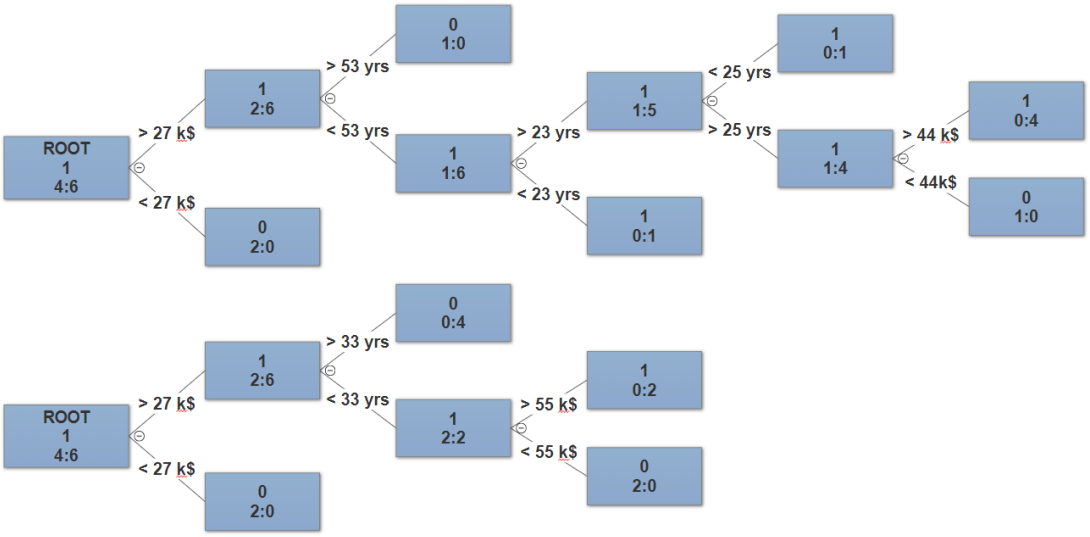


Figure 4: Greedily built decision trees. The splits for the upper tree were decided by searching for the best threshold to lower the total error. The splits for the lower tree were motivated by the best threshold corresponding to the largest information gain (mutual information). The label in the graphic for each node depicts the given label (upper row) and the abundance of negative examples (lower row, left value) and positive examples (lower row, right value). The errors for both trees are vanishing for deep nodes.

In figure 5 the multivariate approach is shown. We can see that it only takes one split to get an error of zero (note: this might not be true for real life data, in this case it is just coincidence of the constructed data). Because of this, the information gain is 1.

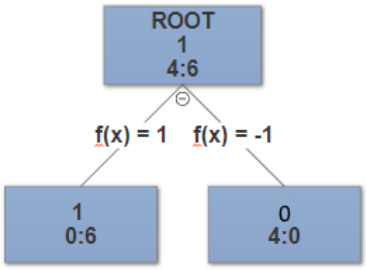


Figure 5: Decision tree built by using a multivariate approach.

2.4 Discussion

Some thoughts on univariate and multivariate decision trees: considering 2D data (like in our example above) we can clearly see that the multivariate approach in this cases works very efficiently. This is because the data points are easily separable by one straight line. If the data appears to be clustered then this algorithm works very efficient.

However, if the data is mixed and no distinct clustering can be found then this approach might be very inefficient. In this case univariate decision trees might be a better choice.

3 A "Baby" no Free Lunch Theorem

Considering a function

$$f : \{0,1\}^m \to \{0,1\} \tag{8}$$

mapping a binary string of length m to a label, either 0 or 1.

3.1 The Number of Inputs

The number of possible input strings is given by

$$N_{\text{input}} = 2^m \tag{9}$$

and for the length $m = 5$ we have 32 different input strings.

3.2 The Number of Function

Each bit in input string $\{0,1\}^m$ can be mapped to two possible labels, either 0 or 1. The input bit itself can be either 0 or 1, giving now two times two possibilities of mapping. Since we have m input bits we get

$$N_{\text{functions}} = (2 \cdot 2)^m = (2^2)^m = 2^{2m} \tag{10}$$

mappings functions. For $m = 5$ we have 1024 different functions.

3.3 Obtaining low expected loss

If we consider Alice to give Bob only uniformly and randomly distributed distinct inputs from the set of inputs $\{0,1\}^m$ then Bob can learn for each new distinct input, which he has never seen before, a new mapping from this input to the true label. So far, so good. If Bob wants to learn Alice' function with an expected misclassification rate of 50% or better, then he hast to see at least 50% of all possible distinct inputs out of the input set $\{0,1\}^m$. This corresponds to $\mathcal{O}(\frac{1}{2} \cdot 2^m)$ distinct inputs. For $m = 5$ this will be around 16 distinct inputs. But as always: more is better!

This is only true if the test set which tests Bob's knowledge is equally generated as the uniformly distributed trainings set. If Alice gives Bob only inputs for testing which he has never seen before than his error rate will never be better then simply guessing.

3.4 Discussion

The only possible way Bob has learned Alice' function with less inputs then discussed in previous section is that he knows the fundamentally distribution after how Alice generates the randomly distributed distinct inputs. By this knowledge Bob is able to predict the distinct inputs with correct labels with a misclassification rate better than guessing. He can convince Janet by correctly predicting further input samples provided by Alice without ever having seen them before.

4 Implementation: *K*-Means

The script loops over $K \in \{1, 2, \ldots, 10\}$, randomly initializes five times for each K and returns the result with the lowest within-group sum of squares.

The upper panel of figure 6 shows the distortion as a function of K. For an increasing number of centroids the distortion decreases. This is expected since we need at least four centroids to get a good prediction for the four possible labels. We can assume that the distortion converges for higher numbers of K around 150000 to 200000.

The lower panel shows the mistake rate as a function of K. Again, the error rate decreases with increasing numbers of centroids. This also is due to the number of possible labels. The rate seems to converge around 20%.

Both graphs have the same shape: the performance gets better for larger numbers of K. This is as expected.

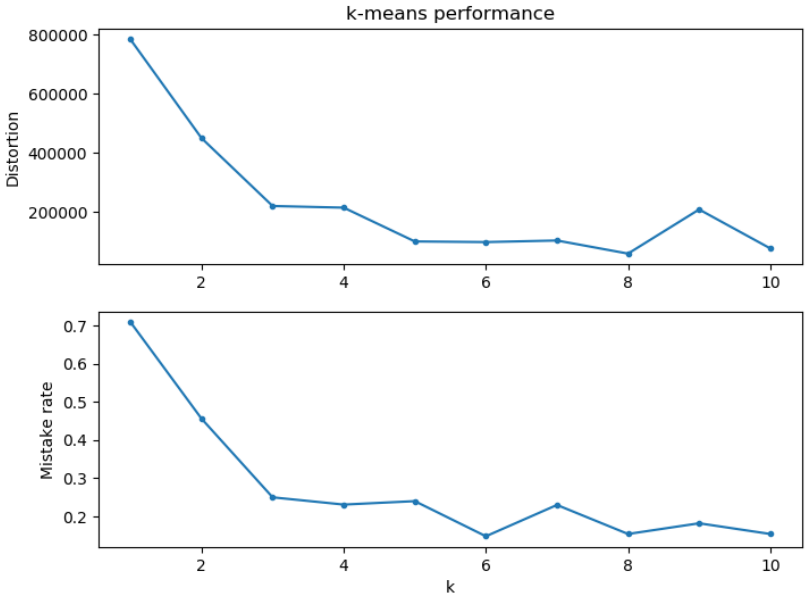


Figure 6: Result of the K-means routine. Top panel: within-group-sum-of-squares as a function ok K. Bottom panel: mistake rate as a function of K.