Homework 1 - OneR Classifier

Mohammad Amin Samadi

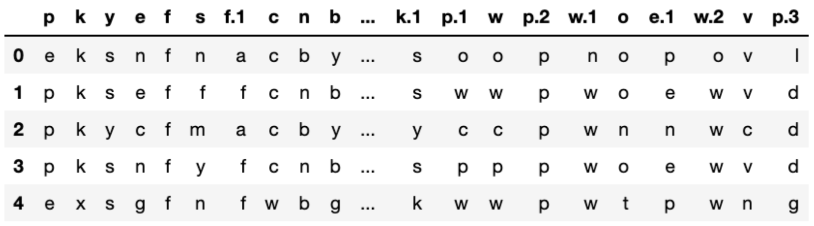
November 21, 2020

OneR Classifier

One Rule (OneR) method is a very simple and computationally light classification method. This algorithm treats different features as possible predictors for a target label. To make this happen, set of rules is extracted from the data based on the co-occurrence of different values of features with different target values. After setting the rules, the feature with the rules that result in highest accuracy is selected and the model makes prediction based on the values of that feature and the set of rules drawn out from the training data.

Data

The data consists of 145 records of mushrooms described by 22 characteristics which have up to 7 possible values. Each mushrooms has a label of ‘p’ or ‘e’, which are short forms of poisonous and edible. A small sample of data is available in figure 1.



	p	k	y	e	f	s	f.1	c	n	b	...	k.1	p.1	w	p.2	w.1	o	e.1	w.2	v	p.3
0	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	o	v	l
1	p	k	s	e	f	f	f	c	n	b	...	s	w	w	p	w	o	e	w	v	d
2	p	k	y	c	f	m	a	c	b	y	...	y	c	c	p	w	n	n	w	c	d
3	p	k	s	n	f	y	f	c	n	b	...	s	p	p	p	w	o	e	w	v	d
4	e	x	s	g	f	n	f	w	b	g	...	k	w	w	p	w	t	p	w	n	g

Figure 1: Small sample of data

The data is balanced enough across these two labels. You can check out figure 2 for a graph showing the distribution of the two labels.

Our aim here is to find the best feature that can act as a predictor for the first column, and evaluate the results of this classification. Due to the small size of training data, k-fold validation is an effective method for that evaluation.

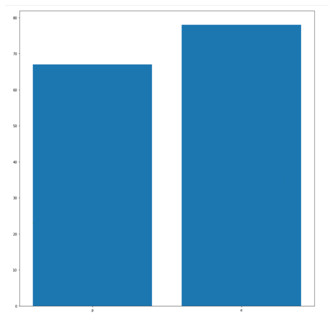


Figure 2: Distribution of data in two labels

My Implementation

Preprocessing

The first step in my implementation after reading the raw data was to preprocess the data for it to be used as a training data for my OneR classifier. I followed the steps below in my preprocessing:

1. shuffling the raw data
2. using the first column as labels and the rest as the input data
3. binary encoding the output as [0, 1], replacing 'p' by 1, and 'e' by 0.

After constracting the training data out of the raw data, I also explored different possible values for each feature for further use in the future steps of implementation.

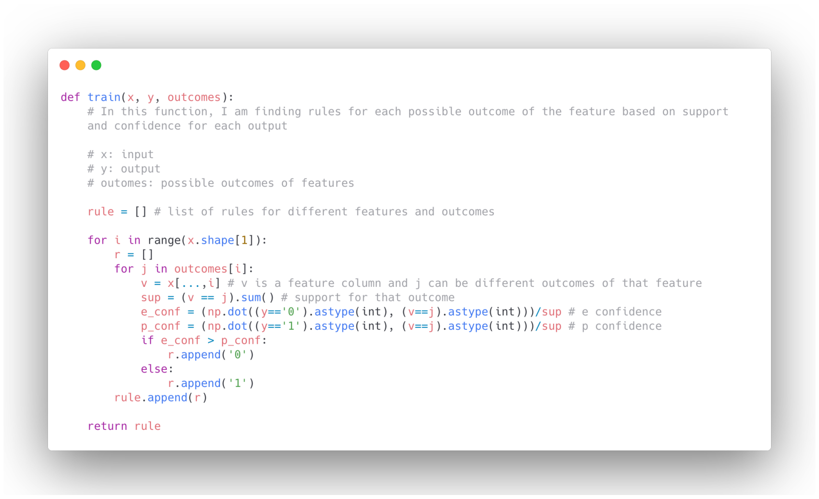
Since I am using k-fold as my validation method, I have to further process the input data to fit the format of training data. In this method, data is splitted into k equal folds. At each step, 1 fold is treated as test set and the rest k-1 folds as training set. Therefore, model will be trained for k times and then evaluated each time on the test set and the final result will be the average of all the test results.

Training

The training procedure is to find appropriate rules for each predictor feature based on the frequency of co-occurrence of each possible value of features with different label values. After determining the support of each value, confidence of (value $\Rightarrow$ output) can also be computed based on that. Therefore, for each value 'v' in every feature, I compute Supp(v), Conf(v $\Rightarrow$ p), and Conf(v $\Rightarrow$ e). In my implementation of this, I am using dot product of vectors to calculate the Supp(v $\cap$ p) and Supp(v $\cap$ e), which I found where handy for this application. Last step in training is to find a rule for every value. The rule is set according to confidence calculated in the previous step. My implementation of training is shown in figure 3.

Feature Selection and Prediction

After finding rules for each feature, I had to find the feature that works best as a predictor. To do so, I used the rules to make predictions for each existing feature. I selected the feature with



```python
def train(x, y, outcomes):
    # In this function, I am finding rules for each possible outcome of the feature based on support
    and confidence for each output

    # x: input
    # y: output
    # outomes: possible outcomes of features

    rule = [] # list of rules for different features and outcomes

    for i in range(x.shape[1]):
        r = []
        for j in outcomes[i]:
            v = x[...,i] # v is a feature column and j can be different outcomes of that feature
            sup = (v == j).sum() # support for that outcome
            e_conf = (np.dot((y=='0').astype(int), (v==j).astype(int)))/sup # e confidence
            p_conf = (np.dot((y=='1').astype(int), (v==j).astype(int)))/sup # p confidence
            if e_conf > p_conf:
                r.append('0')
            else:
                r.append('1')
        rule.append(r)

    return rule
```

Figure 3: Implementation of training

highest accuracy on predictions as the main predictor. For example, you can find one of the charts of accuracies for different features in figure 4, with the y-axis being the accuracy and the x-axis being the feature. After finding the best feature, I used the rules of that feature to predict the labels of test sets.

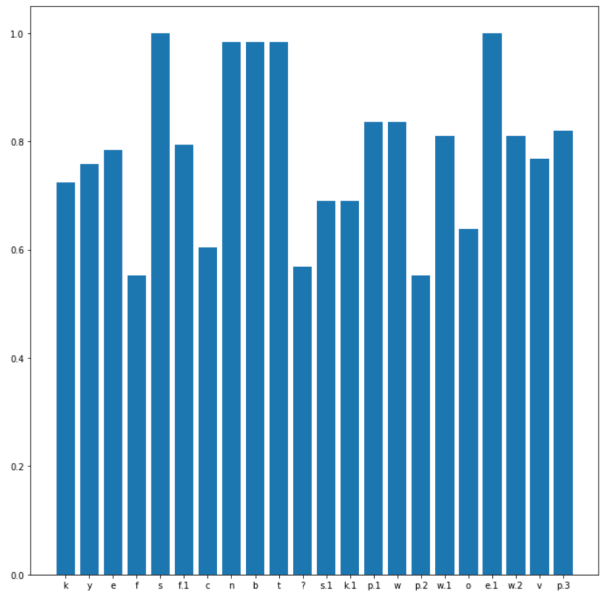


Figure 4: Accuracy chart

Evaluation

To evaluate the predictions made by the model, first I need to construct confusion matrix values. Confusion matrix has 4 different elements can be found in the figure 5.

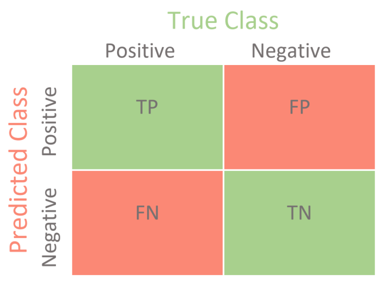


Figure 5: Confusion matrix

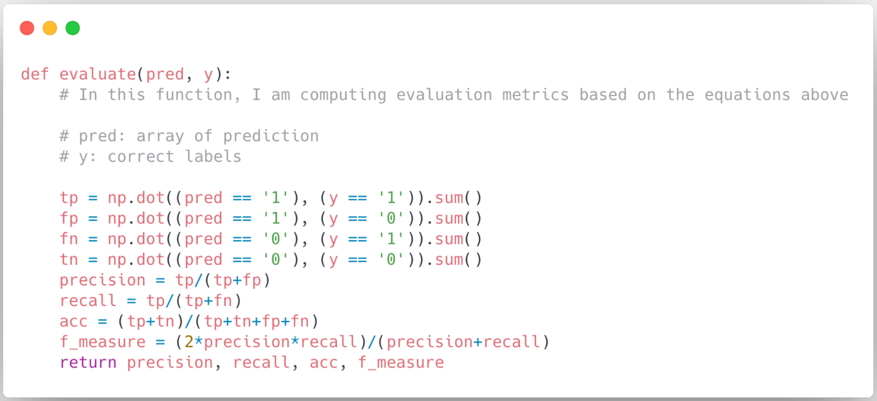
After constructing the confusion matrix, precision, recall, accuracy, and f-measure can be calculated using equations 1, 2, 3, 4 below. My implementation for this section is also available in figure 6. To construct the confusion matrix, once again, I took advantage of dot product of vectors and used the equations to calcute evaluation metrics.

$$Precision = \frac{tp}{tp + fp} \tag{1}$$

$$Recall = \frac{tp}{tp + fn} \tag{2}$$

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{3}$$

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4}$$



```
def evaluate(pred, y):
    # In this function, I am computing evaluation metrics based on the equations above

    # pred: array of prediction
    # y: correct labels

    tp = np.dot((pred == '1'), (y == '1')).sum()
    fp = np.dot((pred == '1'), (y == '0')).sum()
    fn = np.dot((pred == '0'), (y == '1')).sum()
    tn = np.dot((pred == '0'), (y == '0')).sum()
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    acc = (tp+tn)/(tp+tn+fp+fn)
    f_measure = (2*precision*recall)/(precision+recall)
    return precision, recall, acc, f_measure
```

Figure 6: Implementation of evaluation methods

Results

The OneR algorithm although very simple at first, but has proved to be very effective in this task and data resulting in 100% accuracy, 1.0 precision, 1.0 recall, and 1.0 f-measure on numerous ocassions of testing with my implementation.