Machine Learning

Seminar 4 Solution

[Q1 Sample Solution]

- True Positives (TP): These are the cases where the predicted “Yes” actually belonged to class “Yes”.
- True Negatives (TN): These are the cases where the predicted “No” actually belonged to class “No”.
- False Positives (FP): These are the cases where the predicted “Yes” actually belonged to class “No”.
- False Negatives (FN): These are the cases where the predicted “No” actually belonged to class “Yes”.

[Q2 Sample Solution]

a is TP, b is FP, c is FN, d is TN

Sensitivity=TP/(TP+FN)= a/(a+c)

Specificity=TN/(FP+TN)=d/(b+d)

Positive preditive value=TP/(TP+FP)=a/(a+b)

Negative preditive value=TN/(TN+FN)=d/(c+d)

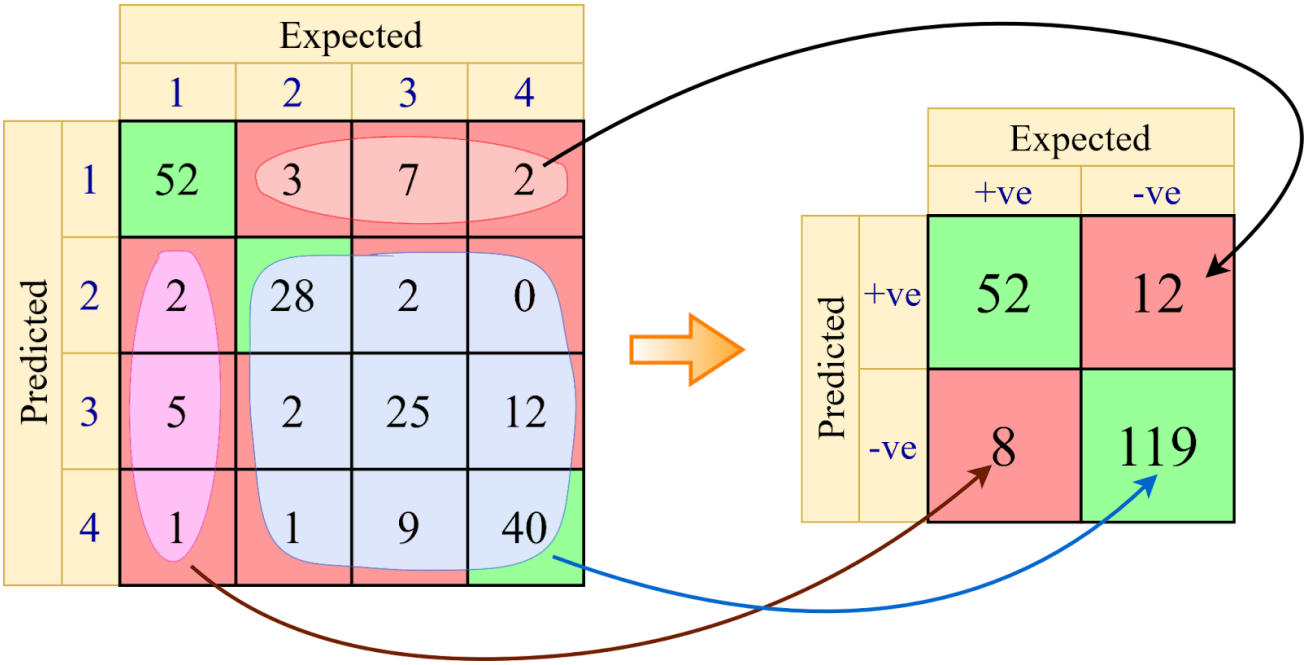
Accuracy=(TP+TN)/Total=(a+d)/(a+b+c+d)

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

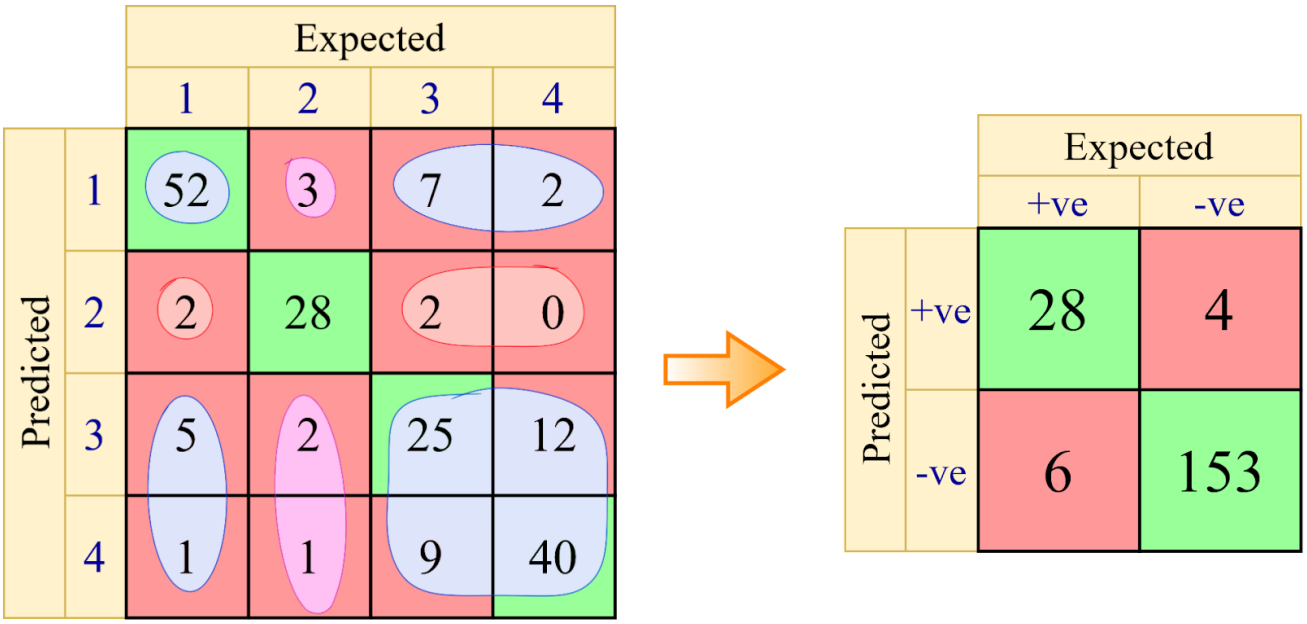
Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	70	20	*Positive Predictive Value*	0.78
	Negative	30	80	*Negative Predictive Value*	0.73
		Sensitivity	*Specificity*	**Accuracy** = 0.75	
		0.70	0.80		

[Q3 Sample Solution]

Converting the matrix to a one-vs-all matrix for class-1 of the data looks like as shown below. Here, the positive class refers to class-1, and the negative class refers to “NOT class-1”. Now, the formulae for the binary-class confusion matrices can be used for calculating the class-wise metrics.



Similarly, for class-2, the converted one-vs-all confusion matrix will look like the following:



[Q4 Sample Solution]

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

TP = number of true positives

FP = number of false positives

FN = number of false negatives

Class	Precision(%)	Recall(%)	F1-Score(%)
1	81.25	86.67	83.87
2	87.50	82.35	84.85
3	56.82	58.14	57.47
4	78.43	74.07	76.19

[Q5 Sample Solution]

hold-out has a major disadvantage.

For example, a dataset that is not completely even distribution-wise. If so we may end up in a rough spot after the split. For example, the training set will not represent the test set. Both training and test sets may differ a lot, one of them might be easier or harder.

Moreover, the fact that we test our model only once might be a bottleneck for this method. Due to the reasons mentioned before, the result obtained by the hold-out technique may be considered inaccurate.

Alternative, we use k-fold cross validation:

k-Fold cross-validation is a technique that minimizes the disadvantages of the hold-out method. k-Fold introduces a new way of splitting the dataset which helps to overcome the “test only once bottleneck”.

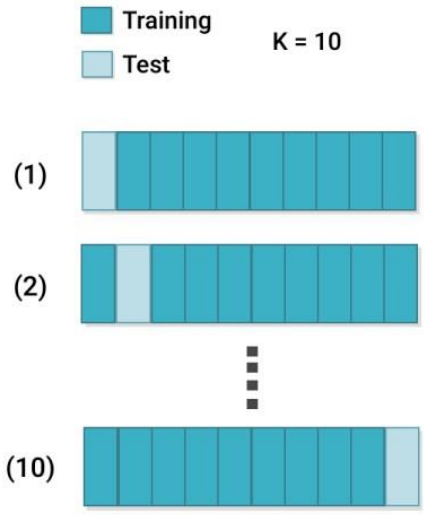
The algorithm of the k-Fold technique:

1. Pick a number of folds – k. Usually, k is 5 or 10 but you can choose any number which is less than the dataset’s length.
2. Split the dataset into k equal (if possible) parts (they are called folds)

3. Choose k – 1 folds as the training set. The remaining fold will be the test set
4. Train the model on the training set. On each iteration of cross-validation, you must train a new model independently of the model trained on the previous iteration
5. Validate on the test set
6. Save the result of the validation
7. Repeat steps 3 – 6 k times. Each time use the remaining fold as the test set. In the end, you should have validated the model on every fold that you have.
8. To get the final score average the results that you got on step 6.

In general, it is always better to use k-Fold technique instead of hold-out. In a head to head, comparison k-Fold gives a more stable and trustworthy result since training and testing is performed on several different parts of the dataset. We can make the overall score even more robust if we increase the number of folds to test the model on many different sub-datasets.

Still, k-Fold method has a disadvantage. Increasing k results in training more models and the training process might be really expensive and time-consuming.



Leave-one-out cross-validation (LOOCV) is an extreme case of k-Fold CV. Imagine if k is equal to n where n is the number of samples in the dataset. Such k-Fold case is equivalent to Leave-one-out technique.

The algorithm of LOOCV technique:

1. Choose one sample from the dataset which will be the test set
2. The remaining n – 1 samples will be the training set
3. Train the model on the training set. On each iteration, a new model must be trained
4. Validate on the test set
5. Save the result of the validation
6. Repeat steps 1 – 5 n times as for n samples we have n different training and test sets
7. To get the final score average the results that you got on step 5.

The greatest advantage of Leave-one-out cross-validation is that it doesn't waste much data. We use only one sample from the whole dataset as a test set, whereas the rest is the training set. But when compared with k-Fold CV, LOOCV requires building n models instead of k models, when we know that n which stands for the number of samples in the dataset is much higher than k. It means LOOCV is more computationally expensive than k-Fold, it may take plenty of time to cross-validate the model using LOOCV.

Thus, the Data Science community has a general rule based on empirical evidence and different researches, which suggests that 5- or 10-fold cross-validation should be preferred over LOOCV.