Resampling Methods for Algorithm Validation Applied to A Differently Transformed Data Set

Mariana Ávalos Arce
Universidad Panamericana
Data Mining, Fall 2021

Abstract

The purpose of this document is to apply five resampling methods that are commonly used to validate Machine Learning Algorithms, in this case, Logistic Regression for a Classification problem. These resampling methods are: 1) Cross Validation, 2) Cross Validation with Repetitions, 3) Division by Percentage, 4) Division by Percentage Repeated and 5) Leave One Out Cross Validation; and these will be applied to three different versions of the same data set, which will be computed from the application of data transformations, leaving us with the three versions being Raw Data, Standardized Data and Standardized plus Yeo-Johnson Transformed Data. By applying the methods to the different versions of the data set, we will arrive at some important conclusions about what transformations seem to improve the quality of the data that will be the input for the ML algorithm, and also the differences in performance and execution of these Resampling Methods.

1 The Data Set

The data set used in the following pages is owned by UCI Machine Learning [1], and was found through Kaggle's data set browser. In simple words, this data set is used for a Supervised Classification Problem in which a Machine Learning algorithm learns to predict whether a female patient's breast tumor is malign or benign, given some attributes from said tumor. The features or attributes in the data set were extracted from digitized images of a breast cancer cell in different tumors, which describe the cell nuclei presented in the images. A brief description of each column or attribute is provided for clarity on further sections:

1. `id`: unique number of the patient.
2. `diagnosis`: M = malign or B = benign.
3. `radius_mean`: mean of distances from center to points on the perimeter.
4. `texture_mean`: standard deviation of gray-scale values found on the tumor's image.
5. `perimeter_mean`: mean size of the core tumor.
6. `area_mean`: mean area of the tumor.
7. `smoothness_mean`: mean of local variation in radius lengths that represent smoothness.
8. `compactness_mean`: mean of perimeter raised to the power of 2 over the area.
9. `concavity_mean`: mean of severity of concave portions of the tumor's contour.
10. `concave points`: mean for number of concave portions of the tumor's contour.
11. `symmetry_mean`: tumor's symmetry measure.
12. `fractal_dimension`: mean for "coastline approximation" - 1.

2 Data Set Cleaning

2.1 Erasing Useless Columns

For the purposes of training a Logistic Regression algorithm, we need to binarize the `diagnosis` column, so that instead of `B` and `M` we have `0` and `1` as the possible values for this attribute, making it our class column. Additionally, the `id` attribute does not represent a numeric column per se, because its values do not intervene in the classification of a tumor. Thus, this column will better be dropped, as well as a column named `Unnamed: 32` that

unexpectedly appears at the end of all the mentioned columns, and is filled with `NaN` entries. In this way, we are making our data set to have **only numeric columns**, and **one binary class column** that determines the classification attribute, which makes our problem **supervised binary classification**. The following code snippet performs this data set operations.

```
import pandas as pd

url = 'db/data.csv'
bcancer = pd.read_csv(url, header=0)
bcancer['diagnosis'] = bcancer['diagnosis'].apply(lambda x: 1 if x == 'M' else 0)
del bcancer['Unnamed: 32']
del bcancer['id']
```

2.2 Dropping Off Based On Correlation

Now, after having our data set as only numeric columns, the next step is to look for **data redundancy**. An attribute (column or feature) of a data set is called **redundant** if it can be derived from any other attribute or set of attributes. This can be perfectly identified whenever a column presents a strong correlation to another attribute, which usually means a correlation close to 1. For a quick visualization of the correlations in the data set, a heat map is plotted using the calculated correlation with the *Pearson* method for all the possible combinations of column pairs, presented in Figure 1.

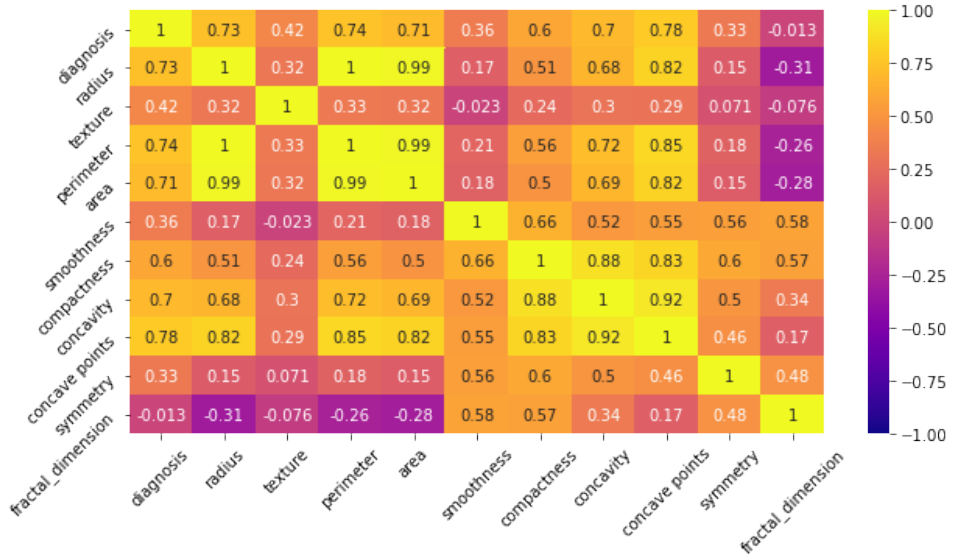

Figure 1: Correlation heat map plot of all the data set attributes.

Indeed there are some strongly correlated attributes, which show values close to 1, such as: `perimeter`, `area` and `radius`. The attributes `perimeter` and `area` present a correlation of 0.99 because of their calculation involving other similar measures, but they are completely different measures of the main properties of the tumor, so they were not dropped from the data set. Thus, the only pair remaining with a strong correlation value is `radius` and `perimeter`. Which one to delete? The column `perimeter` presents a correlation of 0.74 with the `diagnosis` class column, whereas the column `radius` presents a correlation of 0.73 with the `diagnosis`, therefore, `radius` can be considered as redundant, and thus is dropped from the data set. In other words, we delete the radius attribute due to its redundancy and lower relationship with the class. This operation is shown in the snippet below.

```
del df['radius_mean']
df.head()
```

At the end of these operations we can observe some instances that are left in the data set, shown in Table 1.

diagnosis	texture	perimeter	area	smoothness	compactness	concavity	concave points	symmetry	fractal dim
1	10.38	122.80	1001.0	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871
1	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
1	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
1	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
1	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

Table 1: Instances after cleaning operations.

Thus, our data set seems ready to be plugged into the learning models. It is also important to mention that after performing the following,

```
benign = df.groupby('diagnosis').size()[0]
malign = df.groupby('diagnosis').size()[1]
total = malign + benign
benign_p = benign / (total) * 100
malign_p = malign / (total) * 100
print(f"benign: {benign_p:.2f}%")
print(f"malign: {malign_p:.2f}%")
```

we can see that 62.74% of the instances correspond to class 0 and 37.26% belongs to class 1. These percentages, although different, can say the data set is somehow balanced in terms of number of instances per class, to avoid an over-training of the model towards a specific class value, which would arise if the percentages differed much more.

3 Resampling Methods: Cross Validation

3.1 Raw Data

The first Resampling Method is the Cross Validation, in which we split the data into N parts and dedicate 1 to test the model and the N-1 remaining to train it, and iterate N times, each time changing the test part. We apply this method to **raw data**, which means that the **x** varibale containg a matrix of the data set values **without any transformation**, and the class column left out. The Logistic Regression needed in this case around **3500 iterations**. The **y** variable refers to our class column.

```
num_folds = 10
kfold = KFold(n_splits=num_folds, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=3500)
results = cross_val_score(model, x, y, cv=kfold)
acc = results.mean() * 100
stdev = results.std()
print(f"accuracy: {acc:.2f}% std: {stdev:.2f}")
```

As a result we get an **accuracy of 90.51% and a standard deviation of 0.0335** for this method applied with raw data. Requiring 3500 iterations for the ML modeling seems risky, since this is a method that can potentially involve repetitions, and thus cost a lot more time, computationally speaking. With further methods this will show up more clearly.

3.2 Standardized Data

Now we will transform the **x** data matrix using a transformation called Standardization. This operation on **x** involves that, for each column x_i the following is applied to get every instance:

$$x_{scaled} = \frac{x_i - min(x_i)}{max(x_i) - min(x_i)} \tag{1}$$

This transformation moves the attribute distribution towards a mean of zero and a standard deviation of 1. If we plot the new standardized data columns, we can see such results as in Figure 2. This transformation only translates the distribution to a more centered range, in simple words; this means that the distribution of the attributes is not modified towards a complete Gaussian bell. This standardized data set will be called now **rescaled x**.

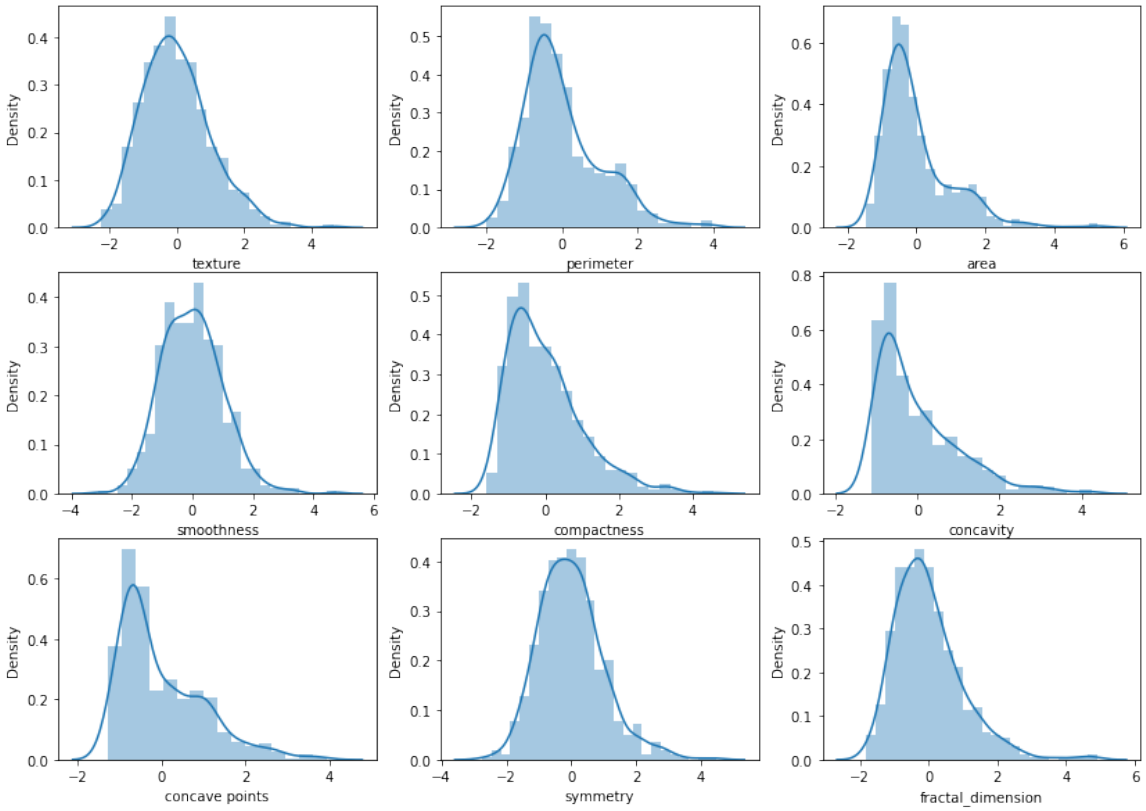

Figure 2: Rescaled x: Standardized data attributes.

```
num_folds = 10
kfold = KFold(n_splits=num_folds, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=100)
results = cross_val_score(model, rescaledX, y, cv=kfold)
acc = results.mean() * 100
stdev = results.std()
print(f"accuracy: {acc:.2f}% std: {stdev:.2f}")
```

The code snippet above outputs an **accuracy of 94.19% and a standard deviation of 0.024** for the method applied to standardized data. Compared to the method in raw data, the accuracy in the prediction of the class by the model increased from 90.51% to 94.19%, but more importantly, the iterations needed were reduced so that the model can converge in just 100 iteration, **reducing the computation time and increasing the accuracy**.

3.3 Standardization and Yeo-Johnson Transformed Data

The **rescaled x** matrix of instances, as seen in Figure 2, is centered in a mean of zero and has a unit standard deviation, but it can suffer further transformations so that the column instances have a more Gaussian-like distribution. This transformation can be achieved by using either the *Box-cox method* or the *Yeo-Johnson method*, but in this case the *Yeo-Johnson method* is the only one of the two that accepts negative data values, such as the ones present in our data set. The method tries to fit the best parameters to achieve a Gaussian bell with each of the **rescaled x** attributes, producing an x matrix we will refer as **transformed x**, plotted in Figure 3.

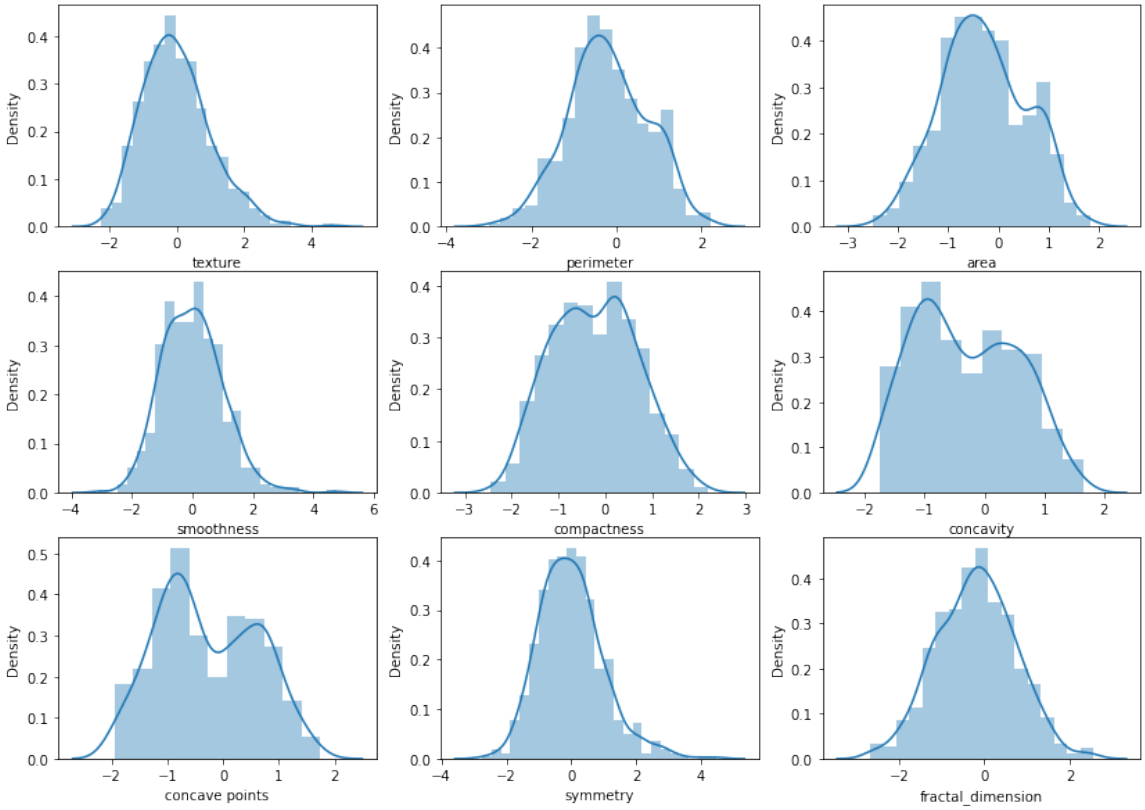

Figure 3: Transformed x: Standardized and Yeo-Johnson transformed data attributes.

If we compare Figure 2 and Figure 3, we can see that not every column was transformed by Yeo-Johnson's method, because not all attributes seemed to have a significant enough skewness to be transformed, such is the case of `texture`, `smoothness` and `symmetry` columns. This was conlcuded after analysing which box plots presented a strong skewness towards some side of the other, such the top plot in Figure 4. As a result, the bottom plot in Figure 4 shows also how the **number of atypical instances was reduced by Yeo-Johnson's method**, which most of the times reduces the solution search process in a Machine Learning algorithm.

With **transformed x** matrix ready, we compute now the Cross Validation to this data in the code snippet below.

```
num_folds = 10
kfold = KFold(n_splits=num_folds, shuffle=True)
model = LogisticRegression(solver='lbfgs', max_iter=100)
results = cross_val_score(model, transformedX, y, cv=kfold)
acc = results.mean() * 100 # percentage of accuracy
stdev = results.std() * 100
print(f"accuracy: {acc:.2f}% std: {stdev:.2f}%")
```

Unfortunately, the **accuracy did not increase from last method, since now we get 91.56% with a stdev of 0.0271**, but it does increase with respect to the method applied in raw data. The algorithm also converges in around 100 iterations, so this method also involves a reduction from computation time when compared to raw data.

4 Resampling Methods: Cross Validation with Repetitions

4.1 Raw Data

Next up, the Cross Validation with Repetitions consists in, as its name implies, the performance of repeated Cross Validation methods. But, why would it help to repeat this method over and over? Because Cross Validation itself

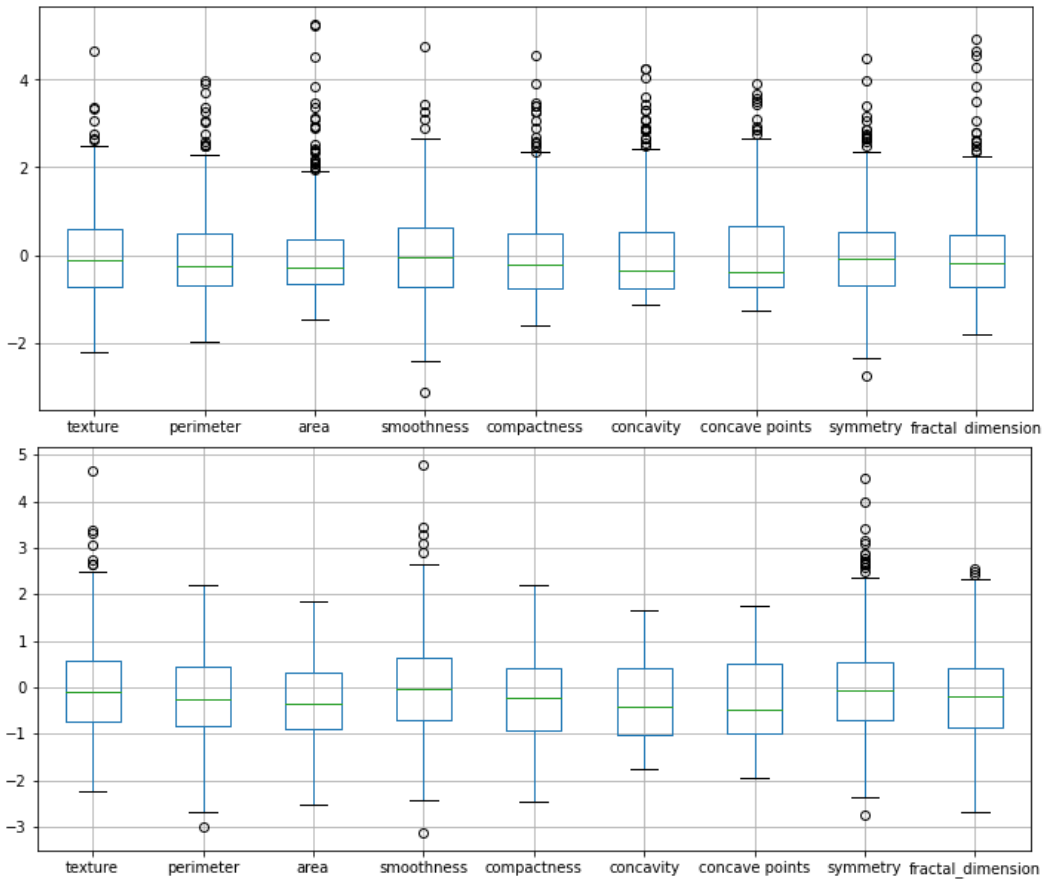

Figure 4: Rescaled x (top) Transformed x (bottom): box plots.

chooses the part that will be test **randomly**, and thus, by repeating it several times, we will reach further testing of our data with different combinations of N sets. We go back to applying this method to the raw data matrix **x**, as in the code snippet below.

```
num_folds = 10
num_repeated = 5
repeatedkfold = RepeatedKFold(n_splits=num_folds, n_repeats=num_repeated)
model = LogisticRegression(solver='lbfgs', max_iter=3500)
results = cross_val_score(model, x, y, cv=repeatedkfold)
acc = results.mean() * 100 # percentage of accuracy
stdev = results.std()
print(f"accuracy: {acc:.2f}% std: {stdev:.2f}")
```

Did it help? Well, the **accuracy in prediction was raised from 90.51% to 90.65% and the standard deviation from 0.0335 to 0.0370 with respect to Cross Validation with raw data**, which is somehow a growth in accuracy when raw data is used, but it is so small that it is likely to be insignificant and variate because of the randomness of the method. The fact that the standard deviation grew also a small bit is probably due to the fact that we simply have now more combinations of sets. Thus, at least with this data set, the repeated Cross Validation seems to be redundant and not significantly helpful when it comes to using raw data.

4.2 Standardized Data

Taking back the **rescaled x** data matrix, which is plotted in Figure 2, now it is time to see if the small increase in accuracy with raw data becomes a more significant increase in accuracy when applied to the standardized **x** matrix. We perform this Cross Validation with Repetitions below.

```
num_folds = 10
num_repeated = 5
repeatedkfold = RepeatedKFold(n_splits=num_folds, n_repeats=num_repeated)
model = LogisticRegression(solver='lbfgs', max_iter=100)
results = cross_val_score(model, rescaledX, y, cv=repeatedkfold)
acc = results.mean() * 100
stdev = results.std()
print(f"accuracy: {acc:.2f}% std: {stdev:.2f}")
```

The actual result was **the decrease to 93.85% prediction accuracy and 0.0326 standard deviation decrease as well**, and this decrease may mean the model without the repetitions was a bit over-trained and now with more sets of testing and training, we reduced this 94% accuracy to a 93% due to the bigger amount of combinations.

4.3 Standardization and Yeo-Johnson Transformed Data

Now that we know what and why use Yeo-Johnson method, we can go back to the **transformed x** matrix, which includes standardization and Yeo-Johnson transform the the raw **x**.

```
num_folds = 10
num_repeated = 5
repeatedkfold = RepeatedKFold(n_splits=num_folds, n_repeats=num_repeated)
model = LogisticRegression(solver='lbfgs', max_iter=100)
results = cross_val_score(model, transformedX, y, cv=repeatedkfold)
acc = results.mean() * 100
stdev = results.std()
print(f"accuracy: {acc:.2f}% std: {stdev:.2f}")
```

Interestingly, the same thing occurred when we compared the Cross Validation with raw data to Cross Validation with Standard and Yeo's method, **we go from 90% to 91.60% accuracy**, while the highest (93.85%) accuracy was reached with only the Standard Transformation. The fact that the same behaviour from the simple Cross Validation and Cross Validation with Repetitions happens tells us that maybe **the iterative process in still redundant and gives the same results with or without the repetitions**.

5 Resampling Methods: Division by Percentage

5.1 Raw Data

5.2 Standardized Data

5.3 Standardization and Yeo-Johnson Transformed Data

References

[1] UCI Machine Learning. *Breast Cancer Wisconsin (Diagnostic) Data Set*. URL: `https://www.kaggle.com/uciml/breast-cancer-wisconsin-data`.