Shahid Beheshti University

Anita Soroush

Dataset 1: [mobile-price-classification]

Getting more familiar with the dataset.

At the beginning the raw dataset contains 21 columns (features) and 2000 rows (records).

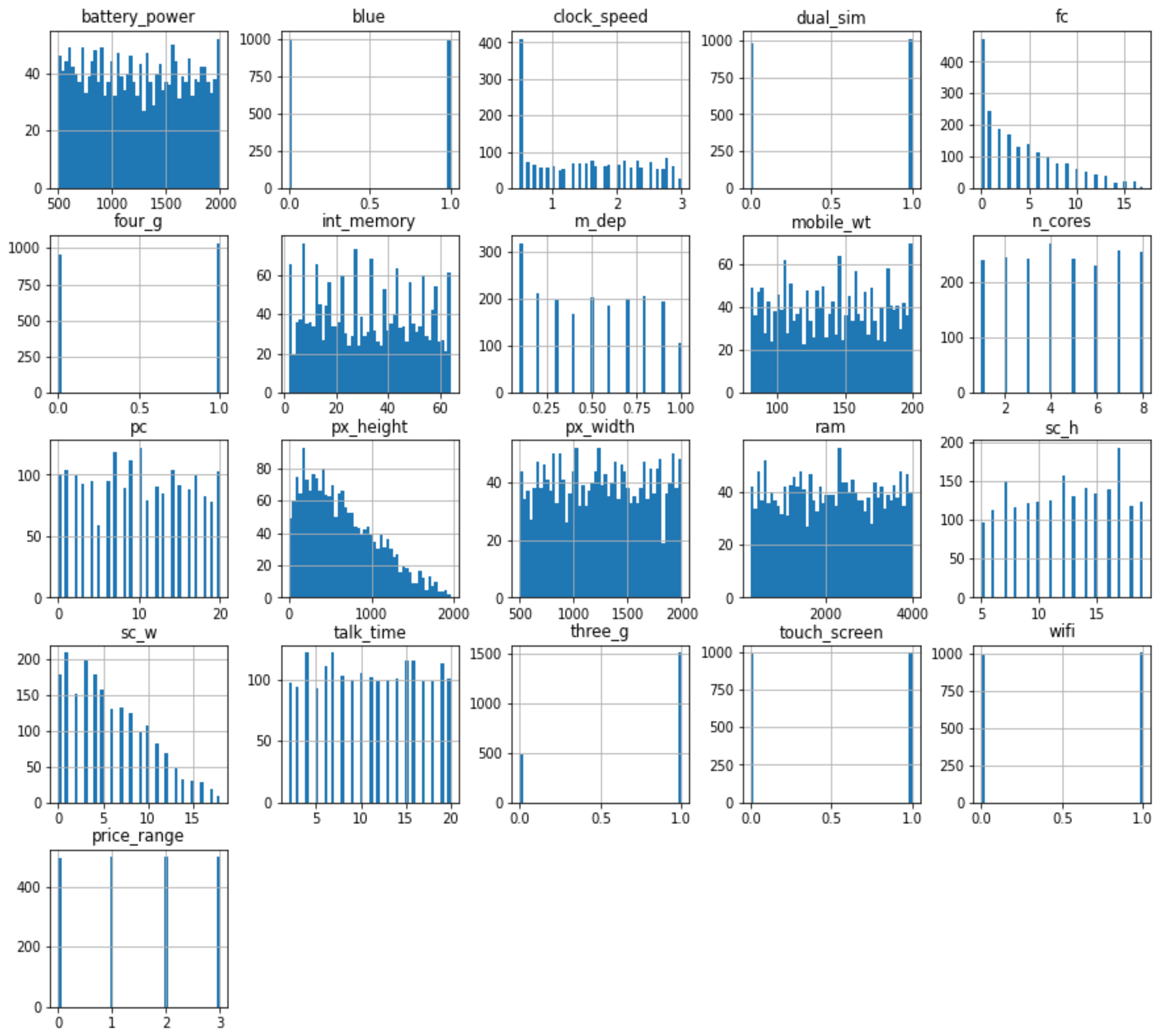
a short description of each feacher:
battery_power: **Total energy a battery can store in one time measured in mAh**
blue: Has bluetooth or not
clock_speed: speed at which microprocessor executes instructions
dual_sim: Has dual sim support or not
fc: Front Camera megapixels
four_g: Has 4G or not
int_memory: Internal Memory in Gigabytes
m_dep: Mobile Depth in cm
mobile_wt: Weight of mobile phone
n_cores: Number of cores of processor
pc: Primary Camera megapixels
px_height: Pixel Resolution Height
px_width: Pixel Resolution Width
ram: Random Access Memory in Megabytes
sc_h: Screen Height of mobile in cm
sc_w: Screen Width of mobile in cm
talk_time: longest time that a single battery charge will last when you are talking
three_g: Has 3G or not
touch_screen: Has touch screen or not
wifi: Has wifi or not

price_range: **This is the target variable with values of 0(low cost), 1(medium cost), 2(high cost) and 3(very high cost).**

The dataset **does not involve** any **null** or **duplicated** records.

But what about the **outliers**? (using standard deviation)
An outlier is nothing but the unusually extreme values present in the dataset.
We will see an upper limit and lower limit using 3 standard deviations. Every data point that lies beyond the upper limit and lower limit will be an outlier.
After using this algorithm the number of rows reduced from 2000 to 1988. **So it seems that the raw dataset had 12 outliers.**

Histogram of each feature.



Some hypothetical tests.

---

1) It seems reasonable if the parameter talk_time has a normal distribution. But does it really have?
To find the answer, I use **Shapiro-Wilk Test** and the result is:

stat=0.947, p=0.000
Probably not Gaussian

2) According to the heatmeap shown in the previous question, it seems that there is a positive relation between RAM capacity and price range. But let's check it:
To check this correlation, I use **Pearson's Correlation Coefficient** and the result is:

stat=0.739, p=0.000
Probably dependent

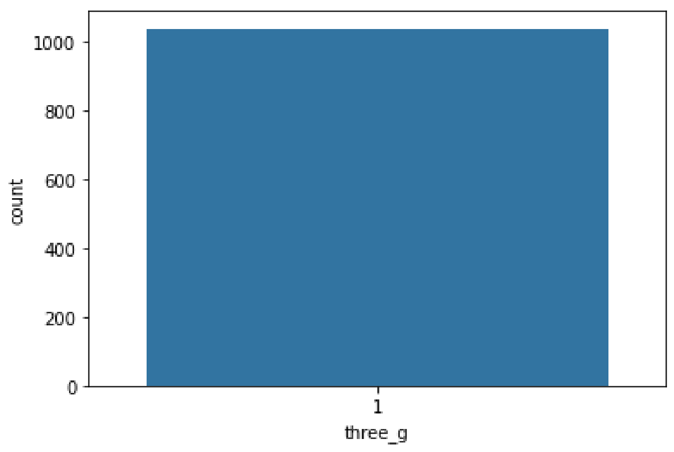
3) Is there a significant difference between the cheapest phones and the others when it comes to the mean value of Primary Camera megapixels? In other words do the cheaper phones have significantly weaker primary cameras?
To find the answer, I use **Student's t-test** and the result is:

stat=-1.446, p=0.148
Probably the difference is not significant

4) Do manufacturers try to focus on either RAM capacity or the quality of the primary camera? In other words, do these two features have a negative correlation?
To check this correlation, I use **Pearson's Correlation Coefficient** and the result is:

stat=-0.236, p=0.000
Probably dependent

5) Before explaining the hypothesis test, I want to mention a point and that is if a device supports 4g Internet, it supports 3g too. The following count plot shows the three_g component of all rows where four_g value is 1:



Moving on to the hypothesis, can we say that the mean price range of phones that support 4g Internet is significantly larger from those that just support 3g? To find the answer, I use **Student's t-test** and the result is:

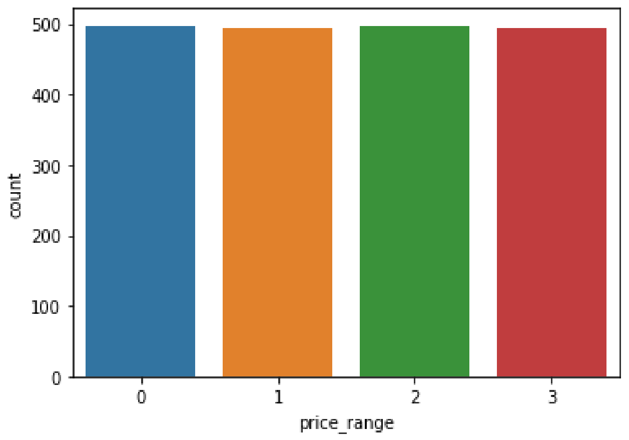
stat=-0.069, p=0.945

Probably the difference is not significant

StandardScaler : It transforms the data in such a manner that it has mean as 0 and standard deviation as 1. In short, it standardizes the data.

Is the dataset balanced?.

The dataset is balanced, since we have 4 classes and each class contains 500 records which is exactly one fourth of the total. this is shown in the following bar chart:



But what if the dataset was not balanced? these are three solutions to combat this problem:

1) Under-sampling:

In this solution we should delete instances from the over-represented class until the dataset is balanced. This deletion can be done in several ways but the one of the easiest and also commonest ways is to delete randomly.

2) Over-sampling:

In this solution we should add instances to the over-represented class until the dataset is balanced. This can be done just by randomly increasing minority class examples by replicating them.

3) More advanced techniques rely on creating new data-points for the minority class to achieve a balanced distribution of classes. SMOTE (Synthetic Minority Oversampling Technique) is one such method.

Classification.

From now on, the data is splitted into 2 parts of training(80%) and testing(20%) and all the confusion matrices which are attached to the report are showing the performance of the model on the **test data**.

First of all, let me briefly explain the idea behind One-vs-One and One-vs-Rest classification. Say we have a classification problem and there are N distinct classes. In this case, we'll have to train a multiple classifier instead of a binary one.

But we can also force python to train a couple of binary models to solve this classification problem. In Scikit Learn we have two options for this, **one-vs-one** and **one-vs-rest** strategies.

I use 2 models for classification:

1) Logistic Regression:

3 important parameters:
1.1) **tol :** ***float, default=1e-4***
Tolerance for stopping criteria.

1.2) **solver :** ***{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=’lbfgs’***
Algorithm to use in the optimization problem. Default is ‘lbfgs’. To choose a solver, you might want to consider the following aspects:

- For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and ‘saga’ are faster for large ones;
- For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ handle multinomial loss;
- ‘liblinear’ is limited to one-versus-rest schemes.

1.3) **multi_class :** ***{‘auto’, ‘ovr’, ‘multinomial’}, default=’auto’***

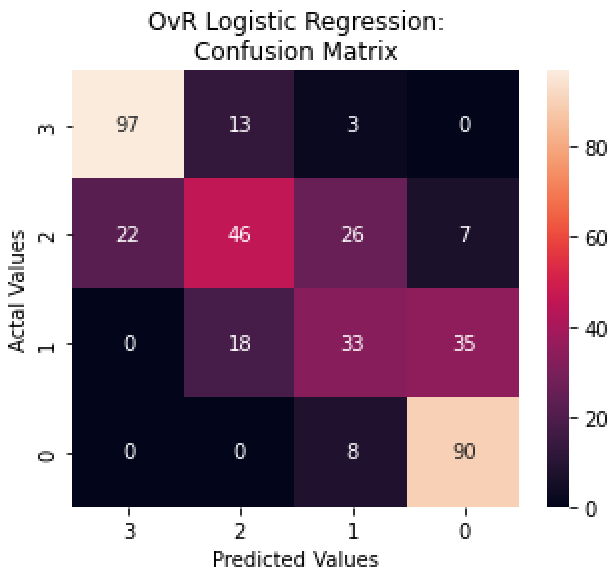
If the option chosen is ‘ovr’, then a binary problem is fit for each label. For ‘multinomial’ the loss minimized is the multinomial loss fit across the entire probability distribution, *even when the data is binary*. ‘multinomial’ is unavailable when solver=’liblinear’. ‘auto’ selects ‘ovr’ if the data is binary, or if solver=’liblinear’, and otherwise selects ‘multinomial’.

OvO or OvR?

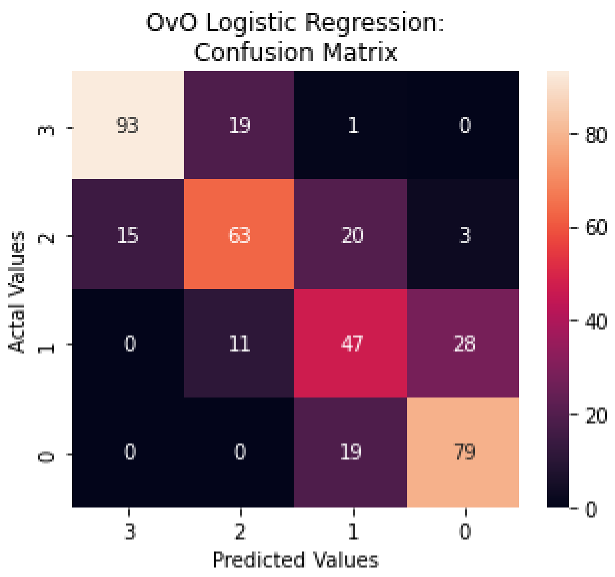
According to the aforementioned **multi_class** parameter, In our case that we have 4 classes of mobile phones price range, in a **default** situation it will go for **multinomial** which is **neither ovo nor ovr**. Instead, the multinomial logistic regression algorithm is an extension to the logistic regression model that involves changing the loss function to cross-entropy loss and predicting probability distribution to a multinomial probability distribution to natively support multi-class classification problems.

After training the model for both strategies, the result is as follows:

Accuracy of OvR Logistic Regression Classifier**:** **<u>0.67</u>**



Accuracy of OvO Logistic Regression Classifier: **<u>0.71</u>**



For this dataset when using Logistic Regression, both strategies worked almost the same, **one-vs-one was slightly better**.

2) SVM:

3 important parameters:

2.1) **kernel: *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'***

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices.

2.2) **degree: *int, default=3***

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

2.3) **decision_function_shape*{'ovo', 'ovr'}, default='ovr'***

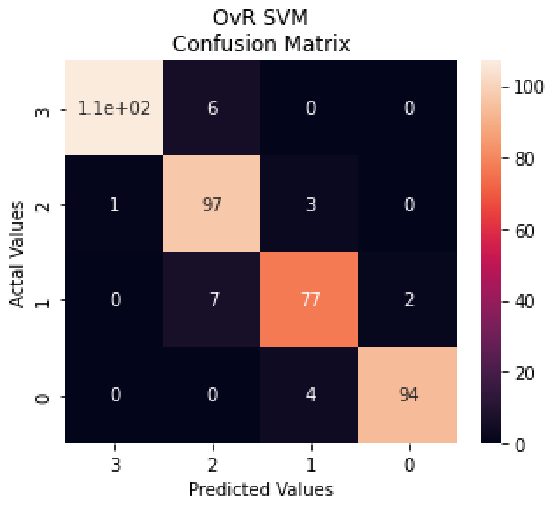
Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, one-vs-one ('ovo') is always used as a multi-class strategy. The parameter is ignored for binary classification.

OvO or OvR?

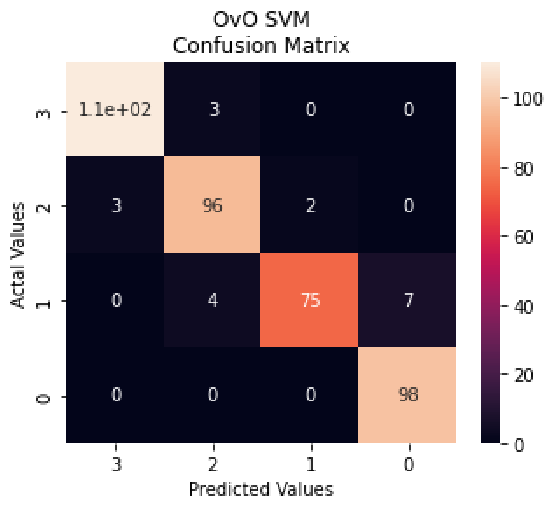
Based on the parameter **decision_function_shape** which was explained beforehand, SVM in scikit-learn package uses one-vs-rest strategy by default.

After training the model for both strategies, the result is as follows:

Accuracy of OvR Classifier: **0.94**



Accuracy of OvO Classifier: **0.95**



SVM came out to be **much more accurate** compared with Logistic Regression. **ovo worked better** than ovr but the difference is negligible.In SVM and Logistic Regression alike, in both strategies of ovo and ovr the first class of price_range (class 0) was detected better than other classes. The reason might be the nature of Logistic Regression which is fundamentally **a binary classifier**; but we should pay attention that the dataset was so small (2000 record) so we **can't** make a general rule that Logistic Regression shows a higher accuracy for the first class in every dataset.

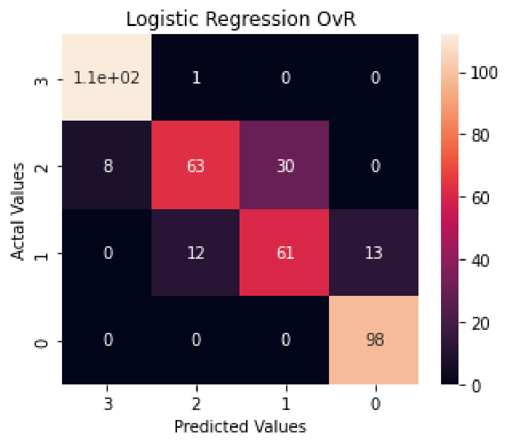
The Impact of different scaling methods.

In all the following scaling methods, first of all I fitted the scalar on training data and used it to scale both training and test data. After these steps I used the scaled training data to make a Logistic Regression Classifier, and I compared the predictions and the real test data.

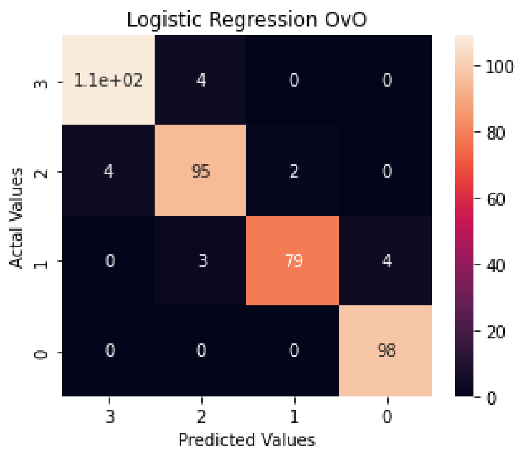
1) Standard Scaler:

It will transform the data such that its distribution will have a mean value 0 and standard deviation of 1.

Accuracy: <u>**0.84**</u>



Accuracy: <u>**0.96**</u>



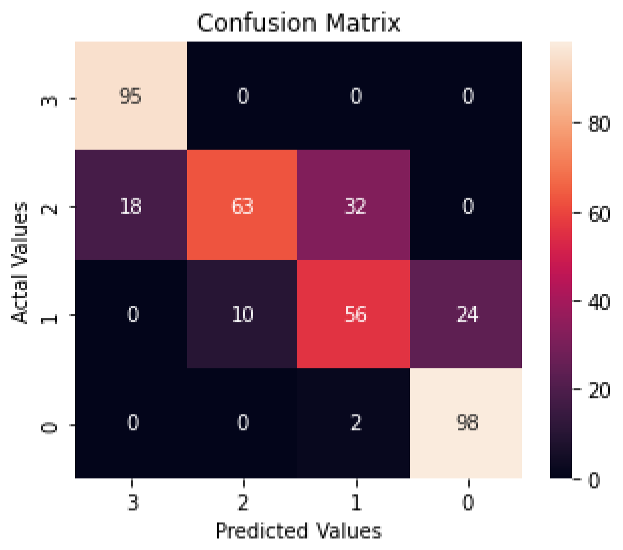
It can be clearly seen that using standard scalar before learning with Logistic Regression in either strategy, **has a very good effect** on the accuracy of the model.

2) Min-Max scalar:

For each value in a feature, MinMaxScaler subtracts the minimum value in the feature and then divides by the range.

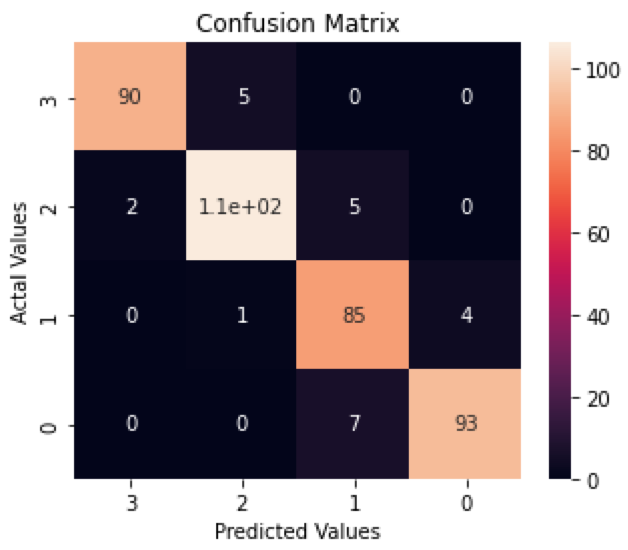
Accuracy: **0.78**

Logistic regression OvR



Accuracy: **0.94**

Logistic Regression OvO

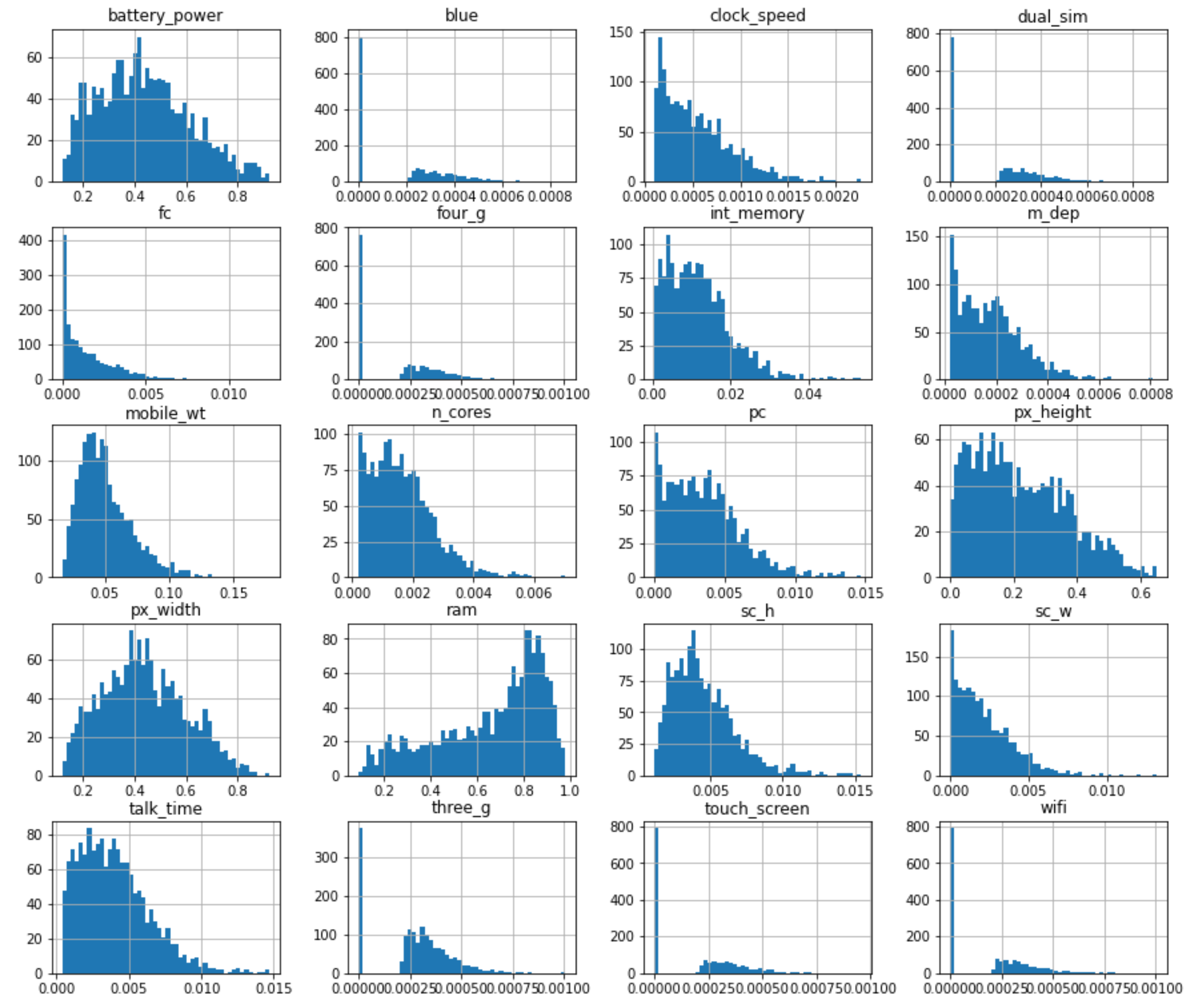


It can be seen that both outcomes have positively changed but **how one-vs-one strategy is impressed is so noteworthy.**

3) Normalizer:

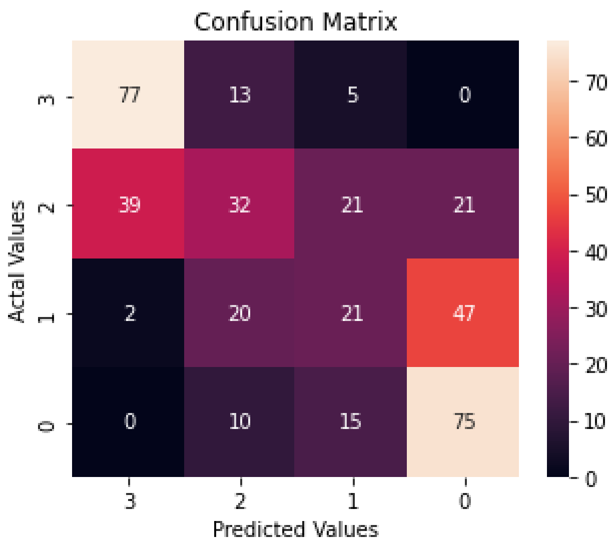
Normalizing samples individually to unit norm. Each sample (i.e. each row of the data matrix) with at least one non zero component is rescaled independently of other samples so that its norm (l1, l2 or inf) equals one.

The effect of normalizing is shown in the following plots:



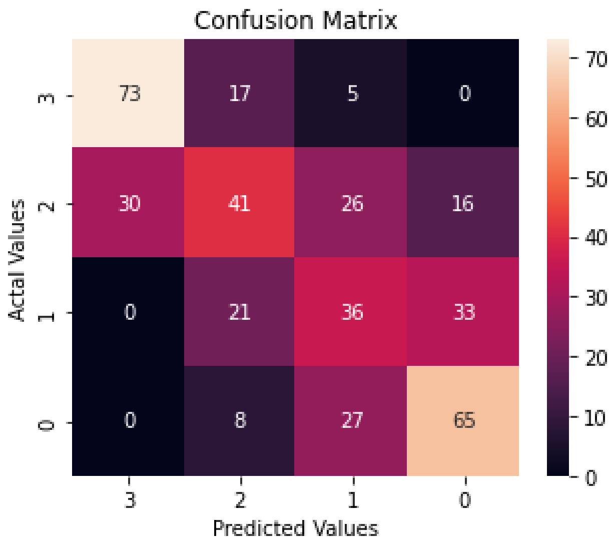
But how does normalizing affect the learning process in Logistic Regression? Check out the matrices below:

Accuracy of OvR Classifier: <u>**0.52**</u>



Actal \ Predicted	3	2	1	0
3	77	13	5	0
2	39	32	21	21
1	2	20	21	47
0	0	10	15	75

Accuracy of OvO Classifier: <u>**0.54**</u>

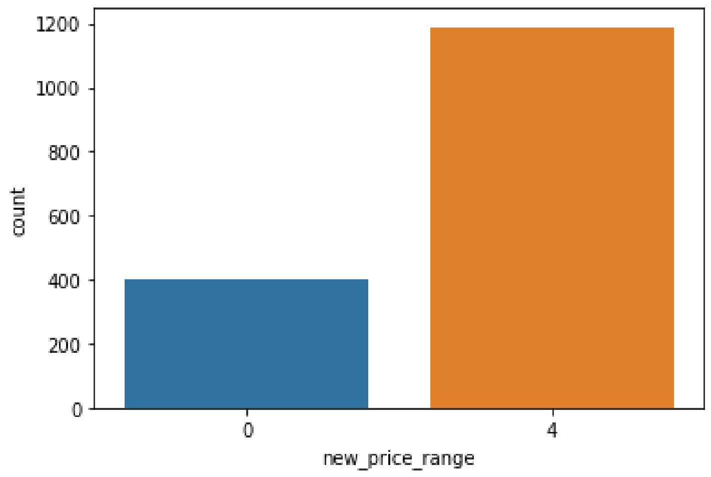


Actal \ Predicted	3	2	1	0
3	73	17	5	0
2	30	41	26	16
1	0	21	36	33
0	0	8	27	65

Apparently **it reduces** the accuracy of the model.

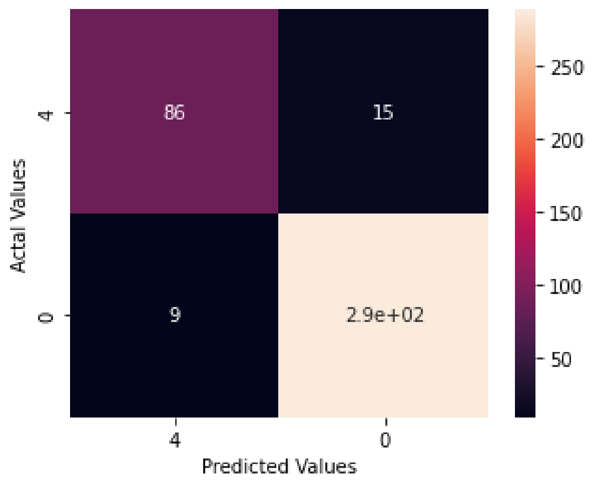
What if the data was imbalanced?

After changing 1's, 2's and 3's into a new label (4), the new dataset is imbalanced, the countplot below clarifies this:

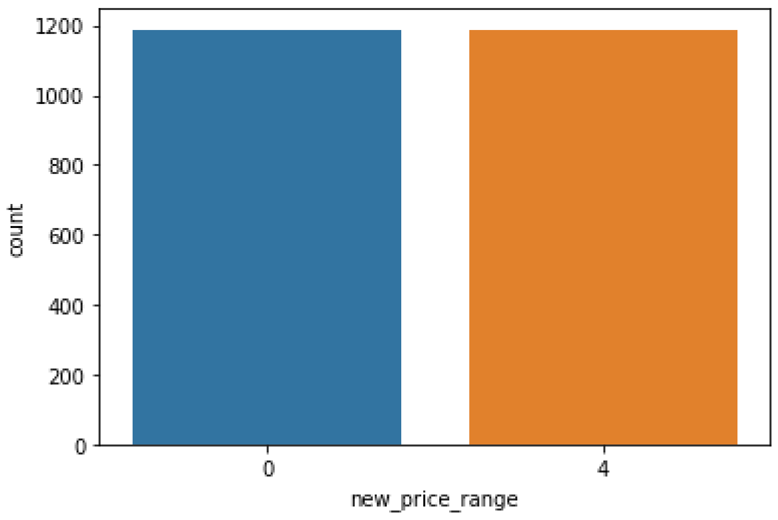


After training the ovo Logistic regression model using this imbalanced data, the result is as follows:

Accuracy: **0.94**

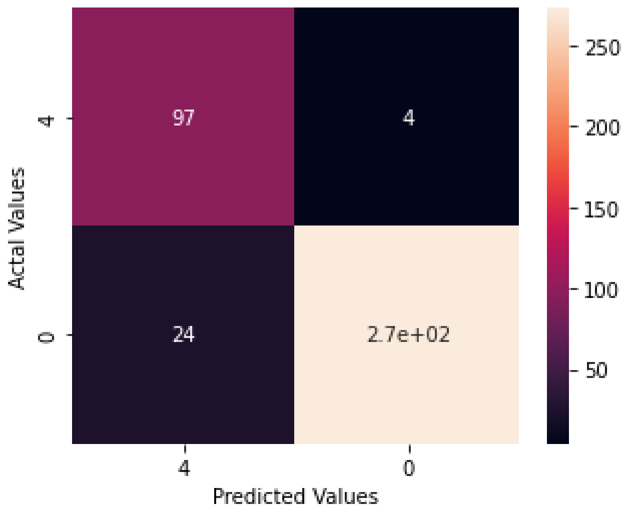


After **oversampling** the **training data** randomly the training part is balanced:



and the training accuracy is shown in the following confusion matrix:

Accuracy: **0.93**



oversampling did not make any specific difference here. maybe because the original dataset is too small (2000 records)

The impact of dimension reduction.

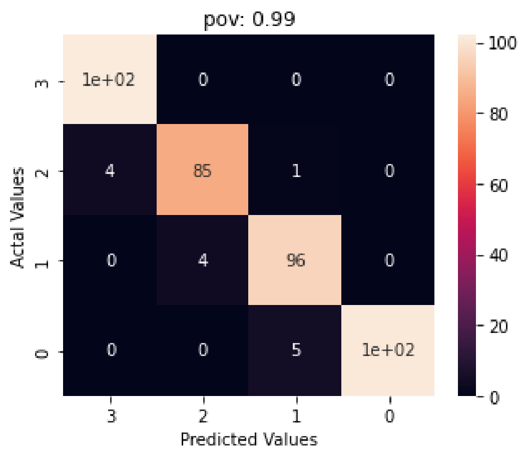
The variance ratio is **the percentage of variance that is attributed by each of the selected components**. Ideally, you would choose the number of components to include in your model by adding the explained variance ratio of each component until you reach a total of around 0.8 or 80% to **avoid overfitting.**

But how to set the amount of pov when using pca?

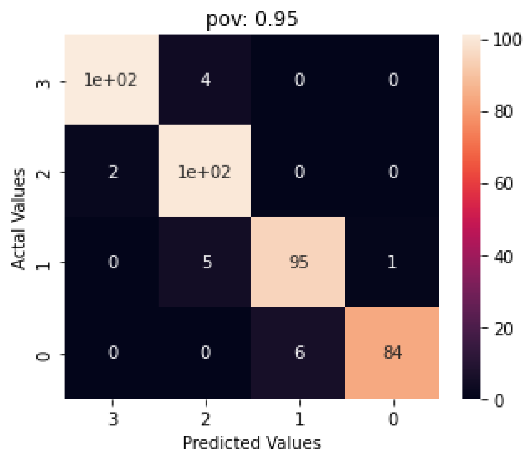
n_components: ***int, float or 'mle', default=None***
If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

svd_solver : ***{'auto', 'full', 'arpack', 'randomized'}, default='auto'***

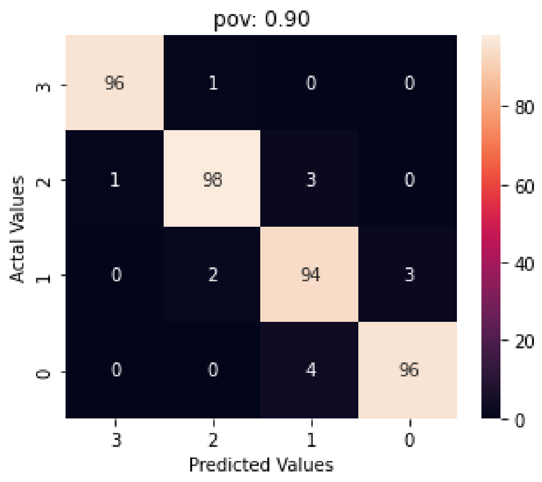
Accuracy of LR OvO Classifier: 0.96
number of principal components: 4



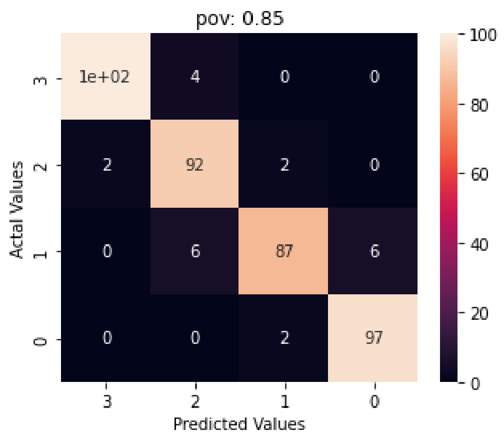
Accuracy of LR OvO Classifier: 0.95
number of principal components: 4



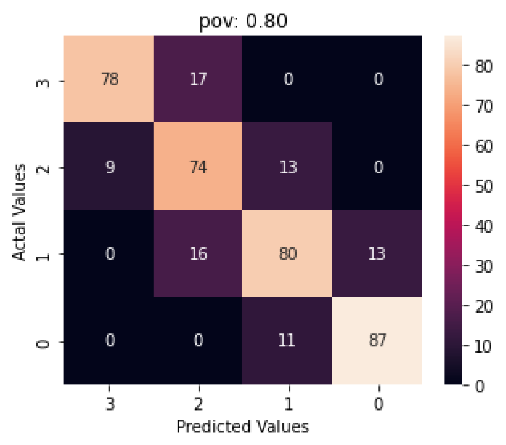
Accuracy of LR OvO Classifier: 0.96
number of principal components: 3



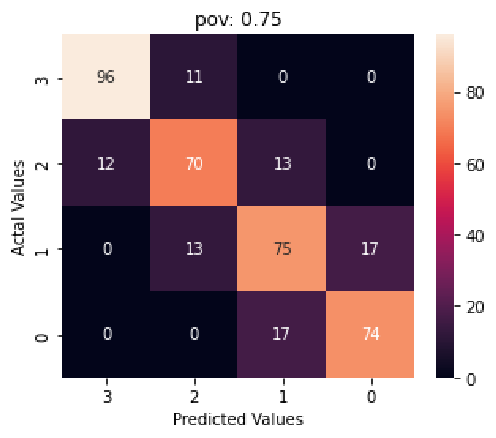
Accuracy of LR OvO Classifier: 0.94
number of principal components: 3



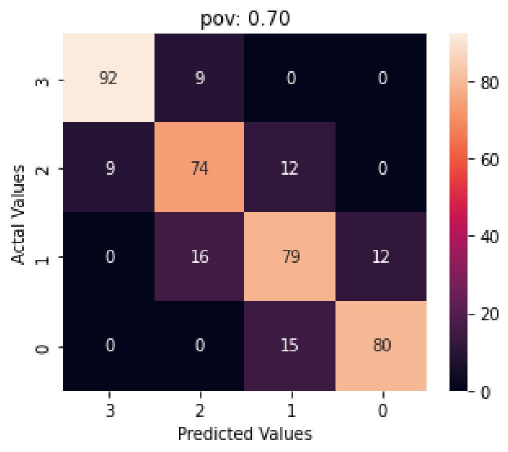
Accuracy of LR OvO Classifier: 0.80
number of principal components: 2



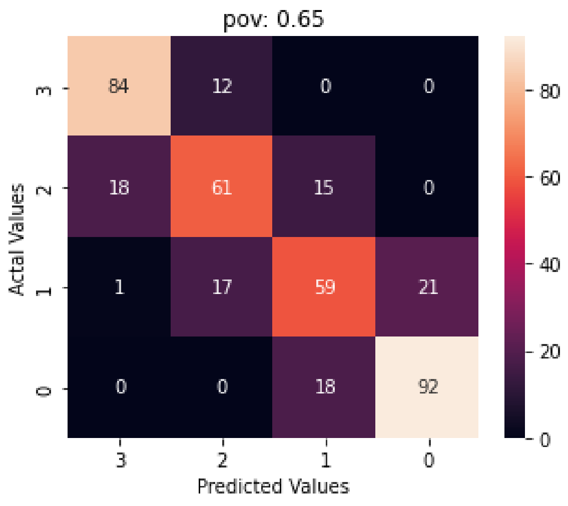
Accuracy of LR OvO Classifier: 0.79
number of principal components: 2



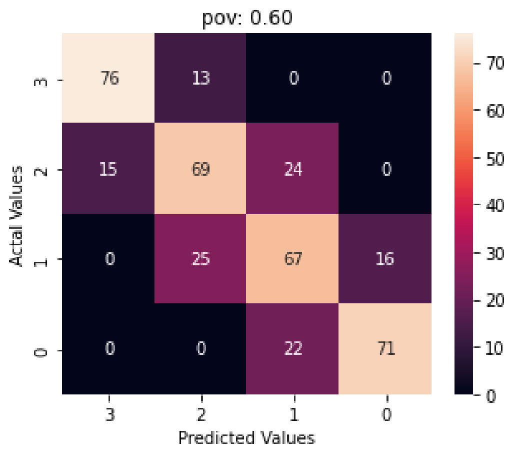
Accuracy of LR OvO Classifier: 0.82
number of principal components: 2



Accuracy of LR OvO Classifier: 0.74
number of principal components: 1



Accuracy of LR OvO Classifier: 0.71
number of principal components: 1



In this dataset, with the reduction of percentage of variance (pov) the accuracy of the model reduced.
Despite making the result even worse, we sometimes need to reduce the number of features (usually using PCA) so that we can **visualize** the dataset in 2 or 3 dimensions. It can help us with choosing more probable methods and more suitable models.

Forward selection

Before running the selection algorithm, I want to merge the 2 lower price classes into one class and the two higher ones into another class so the problem will be converted into a **binary classification** one and the processing will be easier. So I simply relabeled the target column:

$0, 1 \Rightarrow 0$

$2, 3 \Rightarrow 1$

From now on, I will use this new dataset

Now moving on to the forward feature selection, the algorithm that I implemented selected the 5 following features in the same order in which they are reported. the related AUC is also printed alongside the selected features:

1.	ram	0.8941290191290191
2.	battery_power	0.9204568579568579
3.	px_height	0.966880341880342
4.	px_width	0.9801841676841676
5.	mobile_wt	0.9907407407407407

Logistic regression model using the 5 chosen features.

Now let's make a logistic regression model using the 5 features chosen by the forward selector [ram, battery_power, px_height, px_width, mobile_wt].
The performance of this model can be measured using different metrics like:

- The **precision** is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0.
- The **recall** is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.

- The **F1 score** can be interpreted as a harmonic mean of precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:
  F1 = 2 * (precision * recall) / (precision + recall)

These values for the aforementioned model are as follows:

precision:	1.0
recall:	0.9814814814814815
F1_score:	0.9906542056074767

What if we use PCA?

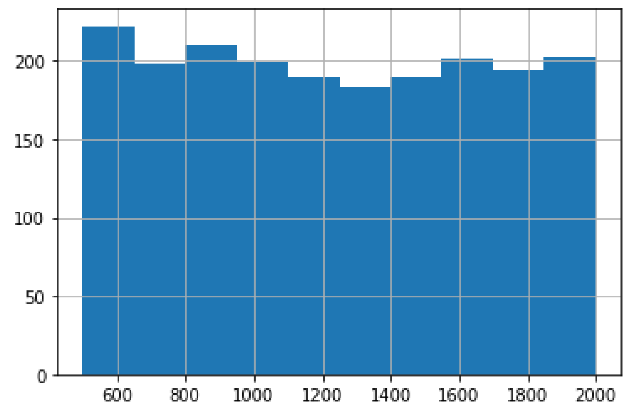
Now let's see the result of another logistic regression model in which the dataset is changed using a **PCA** algorithm with 5 components.

precision:	0.9844559585492227
recall:	1.0
F1_score:	0.9921671018276762

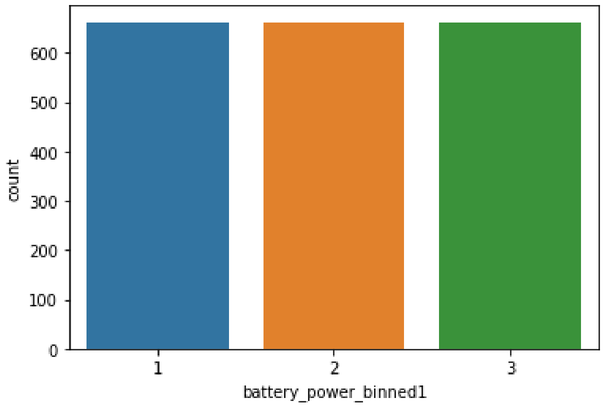
Feature Engineering.

a. binning battery power:

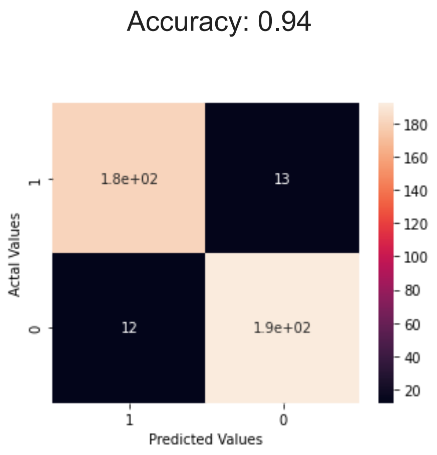
Let's cut the battery power into 3 **unequal-sized** groups. The histogram below illustrates that this feature has a uniform distribution.



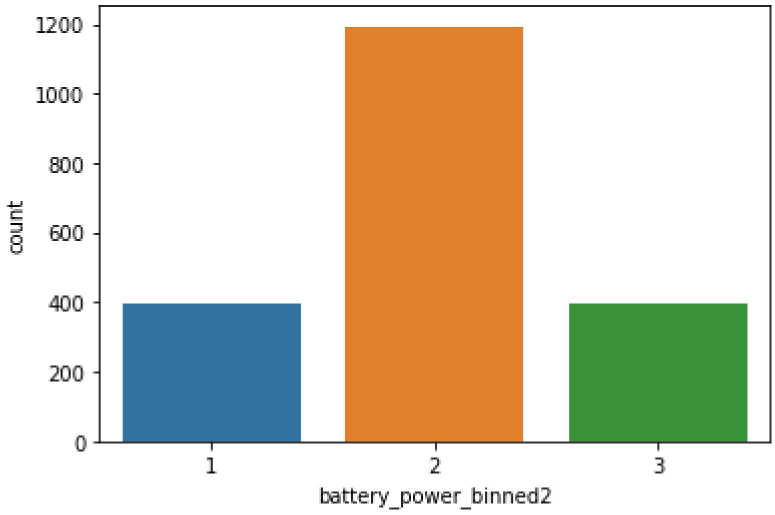
So if we want to have 3 **equal-sized** bins, we simply have to divide the whole interval into 3 equal-sized subintervals.



And how does SVM work on this?

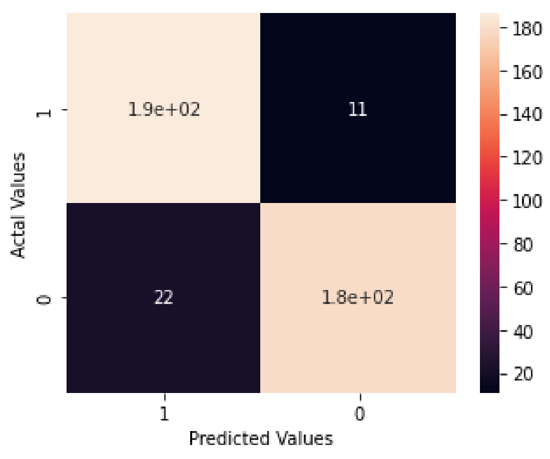


If we determine the sizes of the subintervals in an **unequal** way like:



Then the accuracy will decrease:

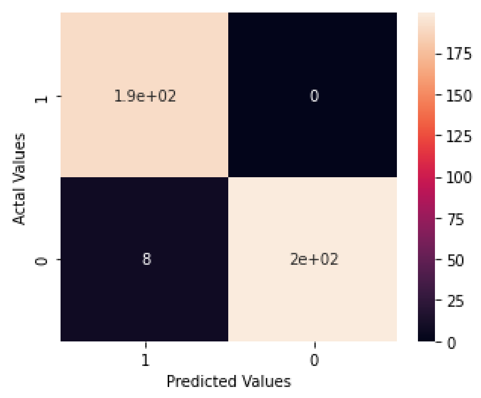
Accuracy: 0.92



b. One hot encoding:

While cleaning the dataset, I dropped some of the features from the raw data. In the current dataset, The Only categorical feature **which is not binary** is n_cores. After using one hot encoder, the SVM result is as follows:

Accuracy: 0.98

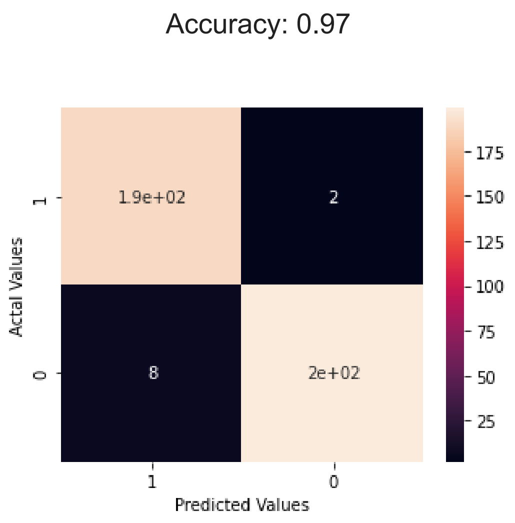


Some general reasons why we use one hot encoding are:

- Some classifiers can only use numerical values. So without one hot encoding we cannot use some categorical non-numeric features.
- One hot encoding makes our training data more useful and expressive, and it can be rescaled easily.
- By using numeric values, we more easily determine a probability for our values.

c. Making new features like Area and Volume of each mobile:

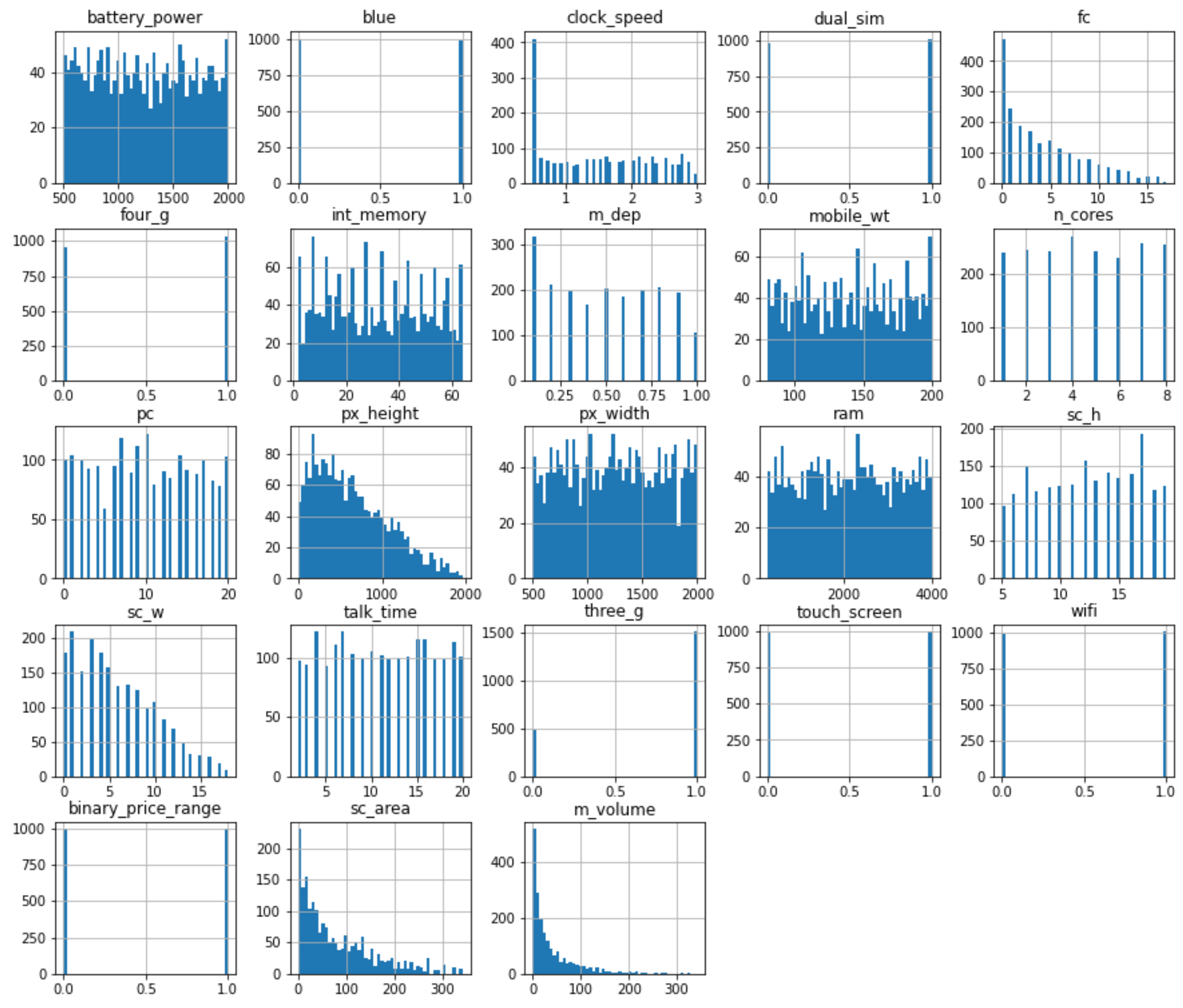
The SVM results:



d. Transformations:

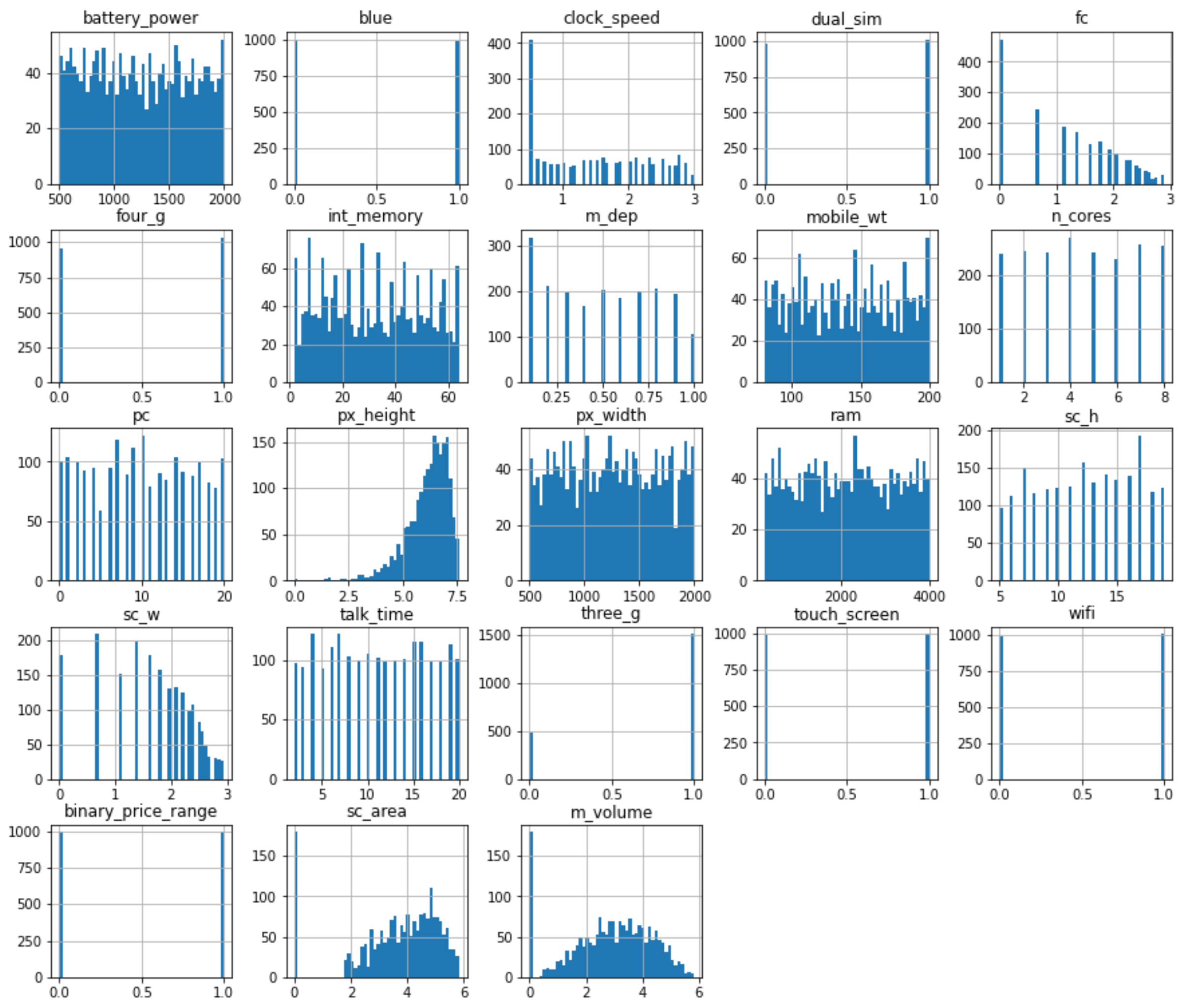
The Log Transform is one of the most popular Transformation techniques. It is primarily used to convert a skewed distribution to a normal distribution/less-skewed distribution. In this transform, we take the log of the values in a column and use these values as the column instead. But the point is that Log Transform can only be helpful when the data has a right-skewed distribution. A log transformation in a left-skewed distribution will tend to make it even more left skew, for the same reason it often makes a right skew one more symmetric. In a left-skewed distribution an exponential transformation will be helpful. Apart from exponential and log transformation, there are also other techniques like square or cube roots transformations.

Moving on to our own dataset, the features histograms are as follows:

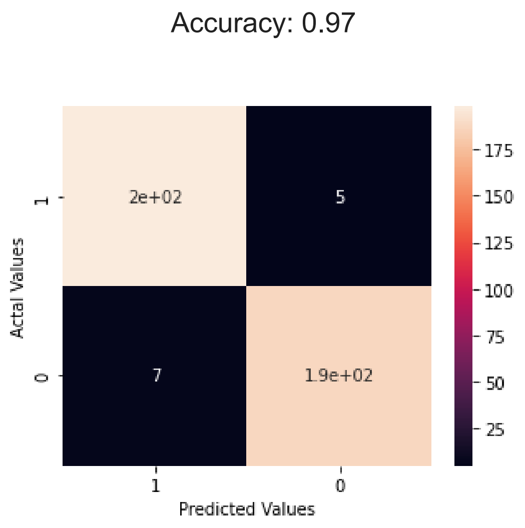


It is clearly seen that none of the features is left-skewed. so there is no need to use exponential transformation but log transformation can be helpful for **fc, px-height, sc_w, sc_area, m_volume.**

After applying log transformation for the aforementioned features the histograms have a more normal distributions:



And the SVC outcome is as follows:



Some General Information:

1) Cross validation vs Bootstrapping.

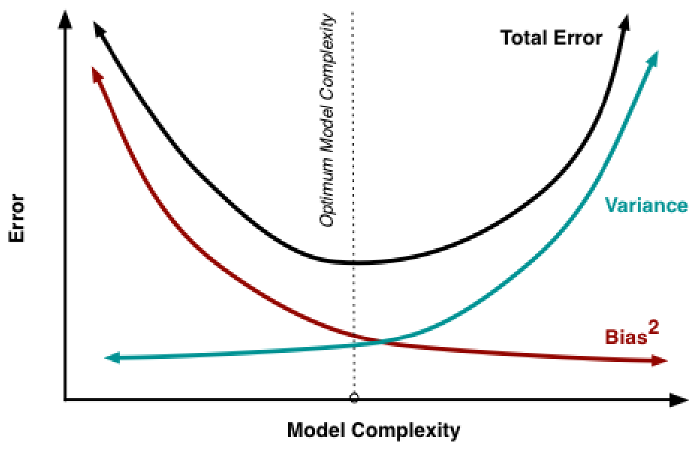
bootstrapping is a **resampling** technique that involves repeatedly drawing samples from our source data with replacement, often to **estimate a population parameter.** Cross validation splits the available dataset to create multiple datasets, and Bootstrapping method uses the original dataset to create multiple datasets after resampling with replacement. Bootstrapping Validation is a way to predict the fit of a model to a hypothetical testing set **when an explicit testing set is not available.**

2) More in Cross-validation

As I mentioned in the previous part, Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure **has a single parameter called k** that refers to the **number of groups that a given data sample is to be split into.** For example In **2-fold cross-validation**, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d_0 and validate on d_1, followed by training on d_1 and validating on d_0. The expression **5*2 cross validation** refers to **5 iterations of 2-fold cross-validation.** Repeated k-fold cross-validation provides a way to **reduce the error** in the estimate of mean model performance.

3) Elbow method for bias-variance tradeoff.

In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and **picking the elbow of the curve as the number of clusters to use.** The problem with clustering is that the more clusters we add, the more variance we explain because if we push the exercise to the extreme, we get as many clusters as points and the variance is fully explained and the bias is almost 0. We are then faced with a case of **overfitting.** To avoid overfitting, we generally use the so-called "elbow method", which looks for a bend in the curve plotting the explained variance versus the number of clusters. However, **the elbow method doesn't always work well;** especially if the data is not very clustered and we see a fairly smooth curve, and it's unclear what is the best value of k to choose. In other words the elbow chart for the dataset **does not have a clear elbow.**



References:

- https://analyticsindiamag.com/outlier-detection-using-z-score-a-complete-guide-with-python-codes/
- https://www.analyticsvidhya.com/
- https://scikit-learn.org/
- https://stats.stackexchange.com/questions/31908/what-is-percentage-of-variance-in-pca
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://machinelearningmastery.com/statistical-hypothesis-tests-in-python-cheat-sheet/
- https://towardsdatascience.com/hypothesis-testing-in-machine-learning-using-python-a0dc89e169ce
- https://michael-fuchs-python.netlify.app/2019/11/13/ovo-and-ovr-classifier/
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- https://www.analyticsvidhya.com/blog/2021/04/forward-feature-selection-and-its-implementation/
- https://www.kdnuggets.com/2018/06/step-forward-feature-selection-python.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
- https://discuss.analyticsvidhya.com/t/transformations-to-convert-left-and-right-skewed-distributions-into-normal/1463