About Matlab data analytics

Typically when we grab data we either want to give it a target or not,

# Data Analytics

First step normalization, such as for our iris dataset it is important to normalize the data, one way to normalize is using z-score

The equation for using zscores is

Step 1: normalization: we don’t want to compare cm to inchs, or apples to oranges , normalization gives our data the same scale, same units

(Actual\_val – mean(feature column)/Std(feature column)

Mean – data / Std

**Mean:**  Mean is another way of saying average, we want the average of all feature observations(columns) which show us the average value of our dataset; all values added together divied by number of values

**Std:** Standard deviation is the measure of spread, how spread out the data is. For example take two exam scores, the mean for the first is **70 and the second is 80** this seems good for the second exam.. however when taking the standard deviation, we find that the first exams Std is **5 while the second Std is 10**, now we can actually say that although the test score MEAN were lower on the first, the Std of 5 means more of the test were closer to the mean and spread out less, than the second exam which has a larger spread of scores. The first exam had a tighter packed scores than the second which were fluctuant, “how far away from the typical average (mean) also has the same unit as the mean so easy comparison,

For example STD, if we have a STD of 5 and a mean of 70 the 1st STD would be 65 and 75 respectively 2nd would be 60 and 80 from the mean, this is spread.

**Mode:** Is the most frequently observed value in a dataset

**Mean:**

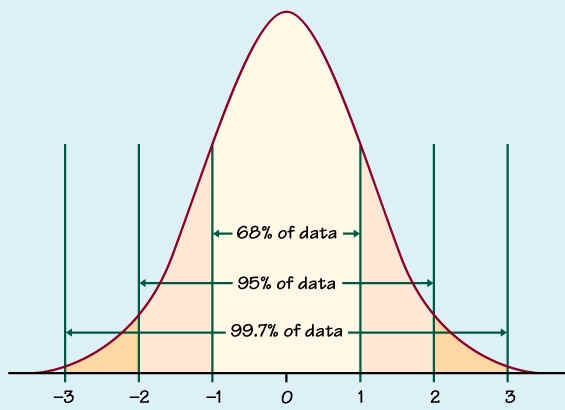
**Median:** is the middle value in a data set order from smallest to largest or vice versa, if a middle is between two values, the median is split

**Range:** Is the largest value minus the smallest value (measures of spread)

**Rules of spread:**  At least 75% if the data will be within two standard deviations of the mean

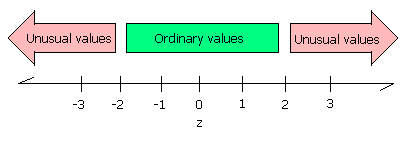
At least 89% of the data will be within three standard deviations of the mean,

Data beyond two standard deviations from the mean are ‘unusual’ 3 STD +



**Z-Score:**  Z-scores tell us how many deviations from the mean this data lies, in the normal distribution curve, also known as ‘relative standing’ (relatively speaking where this data point stands)

--**NOTE:** data that is beyond 2 standard deviations are considered unusual, outliers.



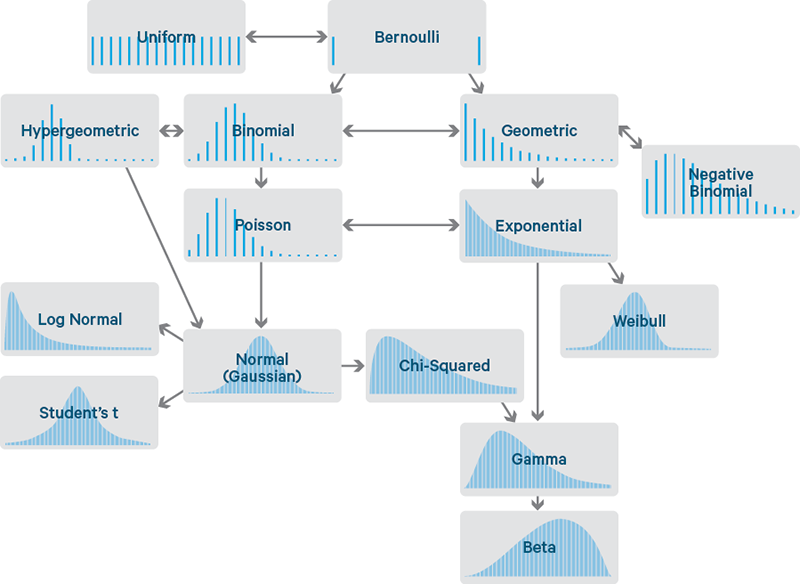
Row score on each measure is converted to a score (‘score’-means core’)/’Std of scores’

ASSUME as X is out data and I is the column of each of our data points

What we want to do is first get the respective means and standard deviations of each feature, meaning (columns)

**Probability Density Functions:**

Probability density functions describe what we think the probability of each outcome is.



PDF: the horizontal axis shows the possible numeric outcomes, the vertical axis describes the probability of outcomes for that specific possible outcome EX flipping a coin assuming there exist only 2 possibilities heads or tails, we can represent that as a *Bernoulli***distribution function**  where the probability is evenly split between heads and tails .5

NOTE: PDF percentages will ALWAYS add to 1, they come in many shapes, but only one size: **probabilities in distributions always add up to 1.**

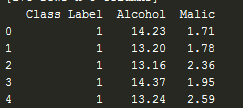
* Makes sense why normal distribution the highest probabilities lie in the middle, if we plot the PDF onto of a normal distribution, then we can see that the **MEAN** or average observed data point, which is to say the most seen data, is the **HIGHEST** probability we will see that outcome out of all outcomes since it’s the average/mean of our dataset (meaning the already most average seen outcome).

**Discrete variables:** Discrete variables are values like a ‘class’ label they can only have 1 type or there is already a finite KNOWN label for these values say in a feature we expect to see either 0,1,5,7 are discrete values. Where the feature is a discrete variable.

# Python Findings

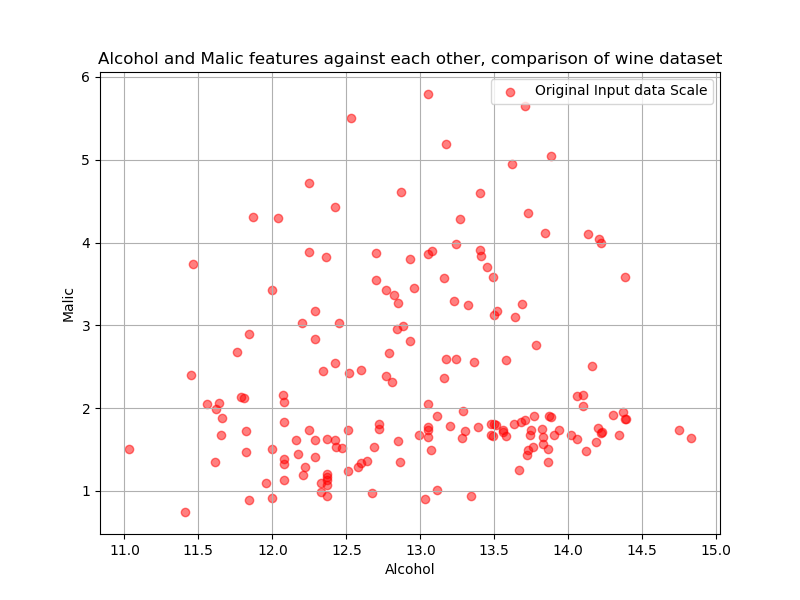
## Step 0: understanding the dataset

## Assume using the Windata.csv



This is the top 5 data points in our observations, in this dataset we know that the alcohol is measured in Percentages, and the Malic amount is measured in lug/u which tells us we will need to normalize, think of about comparing temperature co. Vs. gravity which is 1000 + these scales are completely different and will not mean the same thing.

Furthermore to be more concert and visual, below is a scatter plot of the two feature vectors on a 2 dimensional subspace, you can see that for alcohol the axis of measurement is far beyond that of the Malice measure.



Data balance: in our data set vheicles.csv, we have a very nice scatter of the data however when we look at the scatter plot or any of the plots, we can see an overabundance of each class of points, further more

Show the number of each class label:

|  |  |
| --- | --- |
| Count | Count |
| 0 | 3392 |
| 1 | 15924 |
| 2 | 2944 |
| This abundance of plots is due to there being so much of an UNBALANCED dataset, to balance we can apply various techniques |  |

Up sampling, means when we pick random samples of the minority class and copy them to replace the count to match our majority class, the steps to proceed are to split the classes into their groups

Example purpose and findings out dataset was a premised, which I built to mind

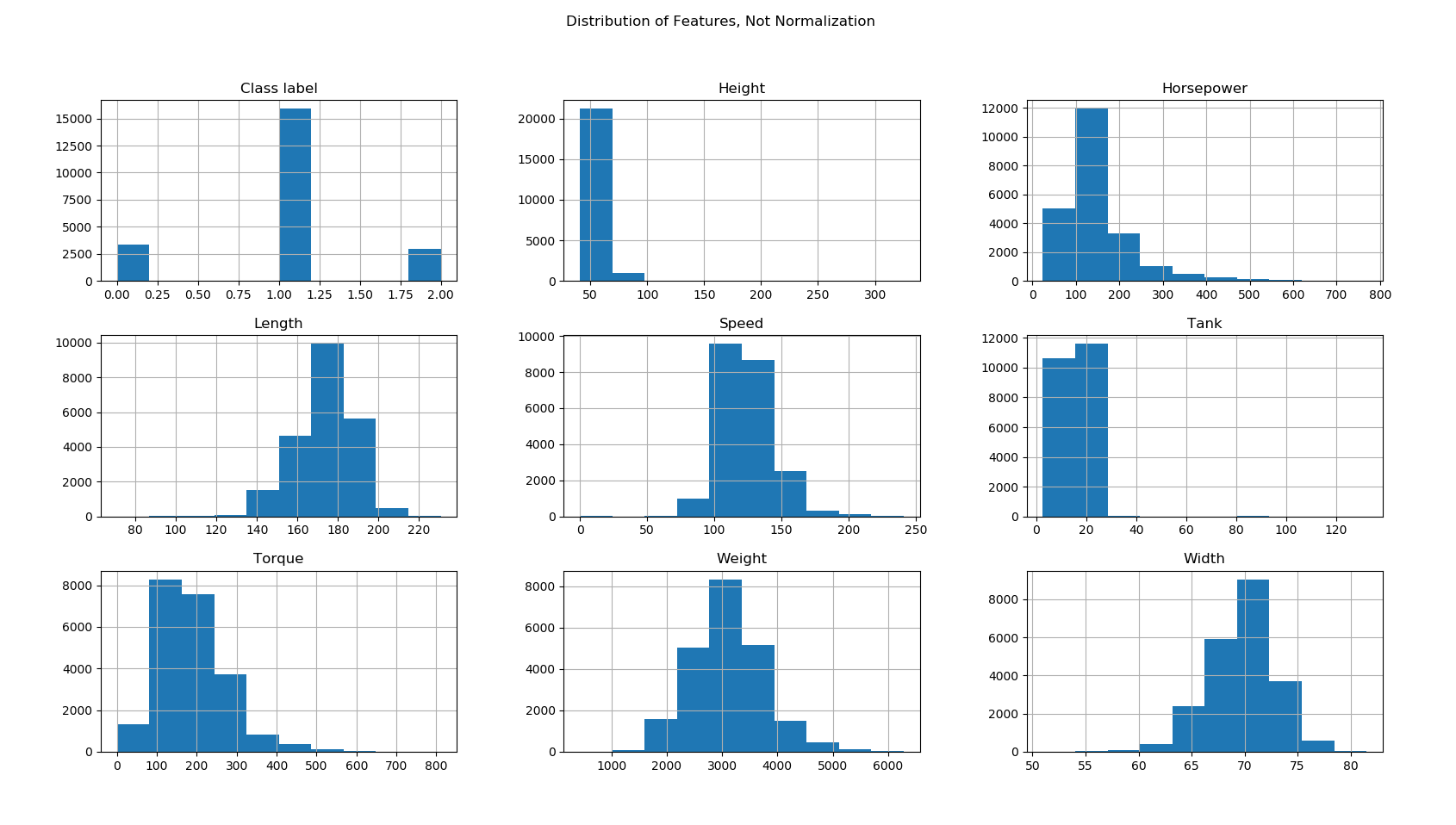
There are total of 22,260 X9 data observations here, when we first look at the data we want to make sure we know that the measurements taken here are

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Horsepower | Torque | Tank capacity | Weight | Top Speed | Length | Width | Height |
| Ft ill/min | Ft/ill | Gallons | lbs. | Mph | Inches | Inches | inches |

As we can inspect each one of these features are continuous, as well as they are of different unit measures, which tell us we must normalize this data to make it useful



Next we should take a graphical look at each feature and its respective distribution, to get a better understanding of our dataset. Here we can see that at first glass our target distribution is very UN even, this also should be pointed out that because the target class is a discrete variable it will resemble the Bernoulli distribution where we have a distributed concrete separation of bars.

* The height of our data however suggest that there exist a large amount of our data that exist around vehicles being between 50 inches tall, almost all of our data suggest this,
* Taking an overall look I can suggest that Horsepower and Fuel tank capacity as well as height are pretty solid observations of what the average overall measurements are
* It is extremely to also pay attention to the scaling of each feature, not to the distribution which more counts of something in a specific bin squishes the distribution to an normal one, but to the measurements of the scale, we can see that length and height as well as horsepower and torque are of the only sets that have the same or similar scaling, this suggest that if we want to use all our features a normalization/standardization must be used.
* NOTES: we must normality especially if we are planning on using a distance based method such as KNN, where large UN scaled feature will certainly dominate the feature space. And give us a less accurate prediction, such that we won’t get much information from those large scaled features
* This suggest **Feature Scaling**
* 

Getting our hands dirty if your dataset does not come with a separated test/train datasets and is a complete raw data all mushed together like mines, then it’s a good time to either split the data now or always split the datasets into a comfortable 80/20 or 70/30 train and test/validation dataset, this is important it’s up to you to remember, suggestively though normalizing and feature transposition and all that then performing the split is much more recommended since you will not need to worry about performing the same normalizations twice also a non-split early means more data for your feature selection and engineering to see.

## Splitting data

“*Learning the parameters of a prediction function and testing it on the same data is a methodological mistake”*—scikit-learn.org

When testing a model with the training data that it just saw, is a big mistake, it would just repeat the labels of the samples that were just seen making a perfect classification performance, this is also known as **Overfitting, To avoid overfitting** its common practice in supervised learning to hold off a portion of the overall data as a test set for unseen data, which is test and testy

Scikitlearn Model\_selection import train\_test\_split

Scikitlearn Cross\_validation iomport train\_test\_split

These are two ways to split your data into different parts

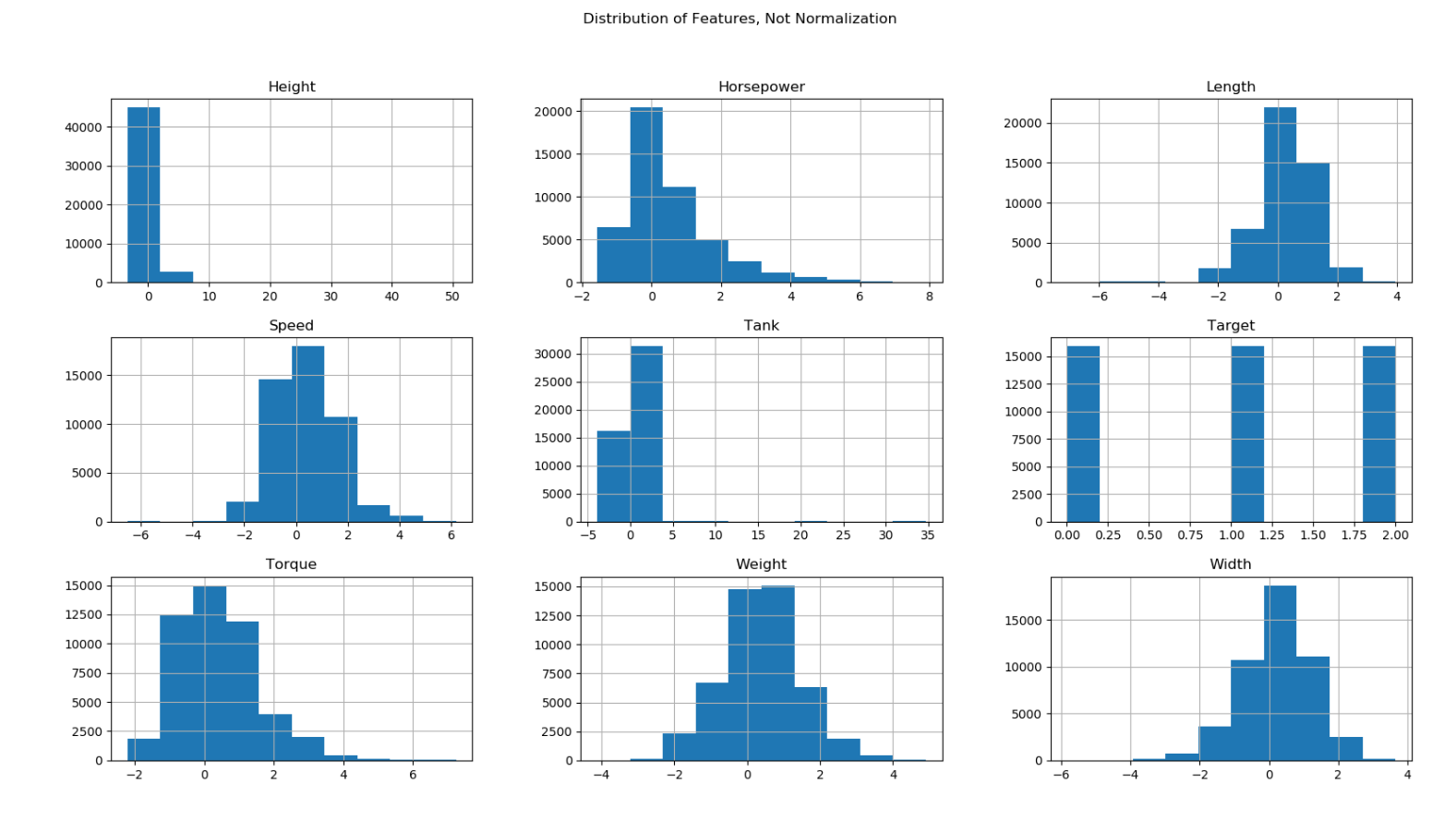
xTrain,xTest,yTrain,yTest = train\_test\_split( data, targets, test\_size=.30, random\_state=5)  
print("xTrain size", xTrain.shape)  
print("xTest Size ", xTest.shape)  
  
  
print(xTrain.head())  
  
print(yTrain.value\_counts())

Where the data is a panda’s dataframe that does not contrain the target column or class column train\_test\_spit does the following:

Splits the data given the percent able test\_size: as 30% is to the training set

Splits the class variables approximately evenly

Even with the header in the dataframe it does not interfere moreover it copies the headers to the new split dataframe



First we plot the distributions for the respective features, remember this is unstandarilized and normalized display of the normal distribution, what we can take from this is,

Yes as we can see the targets plot show a billion distribution that is because this is a discrete variable, therefore it will show that way, however all of our features are continuous from the height we can see that the

Snippet, last step 0 would be to just test our raw data in a model, a good model for classification is the KNN model k nearest neighbor here:

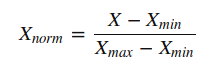
*def.* trainingOnRawData(*data*, *targets*):  
 *from* sklearn.neighbors *import* KNeighborsClassifier  
 *from* sklearn.model\_selection *import* train\_test\_split  
 *from* sklearn.metrics *import* accuracy\_score  
 #from sklearn.cross\_validation import train\_test\_split  
 *from* sklearn *import* svm  
 #########################################################  
 #initalize the training model  
 knn = KNeighborsClassifier(n\_neighbors=5)#choose a neightbor of 5 units  
  
  
 #split our data into train and test sections  
 xTrain,xTest,yTrain,yTest = train\_test\_split( *data*, *targets*, test\_size=.30, random\_state=5)  
 print("xTrain size", xTrain.shape)  
 print("xTest Size ", xTest.shape)  
  
 #now train with the training set  
 #fir this data 'train it'  
 knn.fit(xTrain[['Horsepower','Torque','Tank','Weight','Speed','Length','Width','Height']],yTrain)  
 #now predict the unseen data  
 knn\_y\_res = knn.predict(xTest[['Horsepower','Torque','Tank','Weight','Speed','Length','Width','Height']])  
 print("Performance of the knn on raw data %.2f" % accuracy\_score(yTest,knn\_y\_res))#see our results  
 #######################################################################  
 #print(xTrain.head())  
 #lets manual see the performance  
 #we end up with a 82% for perdiction that;s not bad  
 print(yTrain.shape)  
 print(yTest.shape)  
  
 print(yTrain.value\_counts())  
 #manual calculation means overall target class counts/ total number of samples  
 #meaning we want to know the probability of each class :  
 #we can do this such that X/total y/total and z/total this gives each  
 #probaabilty with respect to the total sum think pizza  
 #each person gets a slize that equalt he sum of the pizza  
 print(yTrain.value\_counts() / yTrain.count())  
 #OMG THERE IS A 70% chance that it will be in class 1 and only 15 and 13 percent it will be in class  
 #2 that sucks, we need to fix this  
 #lets see this in our test set if its the same case  
 print(yTest.value\_counts()/ yTest.count())  
 #its the exact same thing

we make the neighbors of overlapping 5, and split our data into test train testy Trainee, then apply the machine learning KNN model to it, what this does it the KNN use the given features to create a fitted function of f that will fit our data points given the parameters of n, next we use the predict method to predict on the unseen test data, then compare the results using the accuracy\_score(),

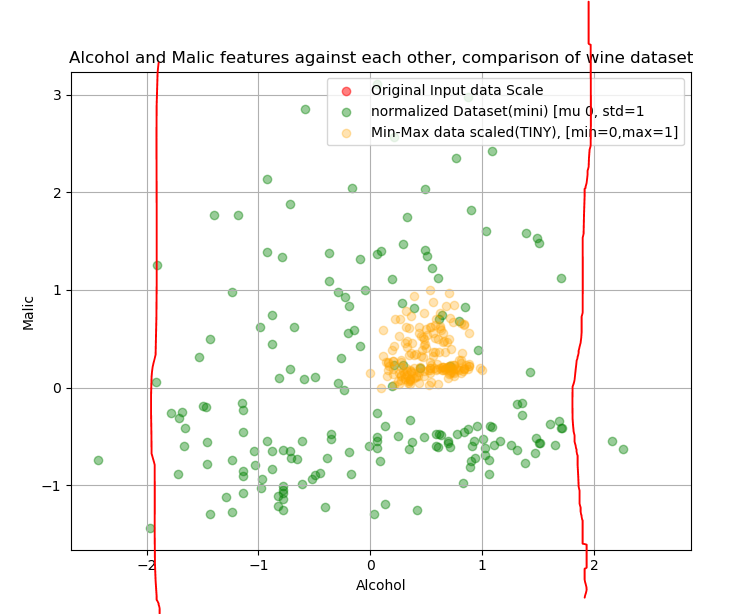
* Findings in the above I see that with the raw data, which keep in mind is unnormalized, and unscaled, as well as very unbalanced with a bias towards class 1, we still get an 80% performance from the KNN,
  + My perdiction could be that by looking at the training set, target vector, we have a whooping 15K of observations that belong to the class 1 variable, what this means is that even whenb we train it with three classes the overly dominating class will highly likely be seen more therefore it will assume predicting the correct point very much often, even when errors occur the over dominating class 1 which is 3 to 4 times the size of the other two classes will be seen more and classified correctly more often than the other two even with the combined errors of class 0 and 2 which combine to 7k , the errors already equate to a 50% chance of classification since the overwhelming class 1 dominates the performance. This is why we need to balance the data
* Without the KNN results we can do a manual run of the sample data, think of the normal probabilities such chances, what is the chance that out of a bag of marbles , assuming the marbles contain 10 marbles 4 blue, 5 red, and 1 green, we can say that the probability of getting a blue marble in the bag is 4/10 approx 40% , red marble 5/10 approx 50% and green 10 % which totals to 100%, lets apply the same itea of “what’s the chance of us grabing a new observation and it classifying into the specific classes
  + So we perform the data.Targets.value\_counts() / data.target.count()
  + This means the total count of classes divided by the total sum of all targets
  + Name: Class label, dtype: float64
  + 1 0.713387
  + 0 0.154388
  + 2 0.132225
* We get the astonishing above results which are a little lower than our classifier but still we can see the **distribution of probabilities…** where the probabilities of drawing a class 1 is WAY higher than the other two classe as mentioned above taking a count of all classes we see that there exist a 15k of class 1 where as the other two classe only have observed 3k and 2k

## Step1: Normalization

Scklearn provises a min-max scaler

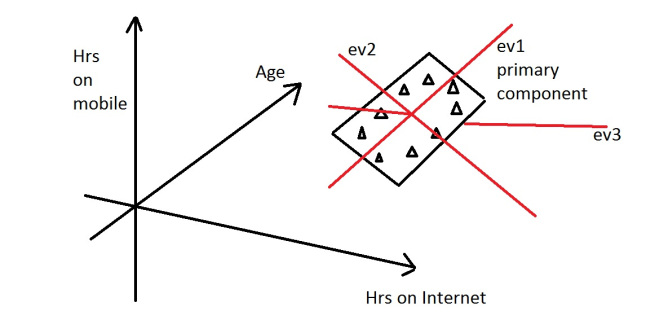
Because





Zooming in we can see that typically , anything greater than 2 standard deviatons are unusaually as we can see using standard scaling is between the bounds of [2, 2] compared to the orginal data which is largely , looking back we can see the information scaled way lower.

In the above we can see an example where our data is far out of the scale, in which the p1 and p2 vector spaces are uneven one side meaures a 2 digit whle the other measures a single digit, looking at the second graph we can see that by using a normalized data set we have scaled our data to be within the bounds of 2 and 2 , if we take a look at the entire plot we can see the information has not been lost or changed, it is a mear replicated image of the original data set on a smaller scale.

PCA 

PCA:

Ev1 is the primary first eigen vector, which tells us the most variance in our dataset the above assumes we have a 3D dataset ,(3 features). The second eigenvector is ev2 which is found orthrogonal to ev1, always , now, keeping in mind, for each dimension of the dataset exist a pair of eigine values and eigen vectors, where an eigen vector is the direction in which there is most variance, the eign value tells us how much variance there is in the direcxtion of the eigen vector, think about it, we want the most variance, if ev1 and ev2 values are above 40 and ev2 is 0.1 we can omit ev3 because it doesn’t tell us much information alternatively also, if all eigine values are near zero, that means our dat is pretty much in ‘good’ standing we cannot remove features they all are pretty variant. Or of similar magnatuide, or same length

Steps for PCA:

Remove the target label/class label

2: compute the d-dimensional mean vector- the means for every dimension of the who dataset

3: compute the scatter matrix (or covariance matrix) of the whole dataset

$: compute the eigenvectors (e1,e2,…ed) and corresponding eigenvalues(lamda1,lambda2,..lamdad)

5: sort the eigen vectors by descreasing eigenvalues and choose k eigenvectors with the largest eigenvalues to form a dxK diemnsal matrix W where every column represents an eignevector

Sources

**Normalizations**

<http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/visualization-with-matlab/>

<http://howto.commetrics.com/methodology/statistics/normalization/>

<https://en.wikipedia.org/wiki/Normalization_(statistics)>

**Statistics**

<http://www.comfsm.fm/~dleeling/statistics/s63/zscore.html> :z-scores

<https://blog.cloudera.com/blog/2015/12/common-probability-distributions-the-data-scientists-crib-sheet/> : PDF

<https://georgemdallas.wordpress.com/2013/10/30/principal-component-analysis-4-dummies-eigenvectors-eigenvalues-and-dimension-reduction/> :PCA(AMAZING)

# Python

## Matplot Library

**1: import** **matplotlib.pyplot** **as** **plt**

2: plt.plot([1,2,3,4], [1,4,9,16], 'ro')

3: plt.axis([0, 6, 0, 20])

4: plt.show()

basic usage:

Line 1: imports the basic plotting library, which is most used

line2: plot() function is very versatile because given a single(vector/array) it will assume a single parameter of y vectors, and find a scaling x corresponding value, in the event we explicitly pass 2 arguments what are [x],[y] pairs of the same dimension, it will plot x versus y, the third option is style of the line, which you can see below links

line3: describes an array of [xMin, xMax, yMin, yMax](list) for the plots x and y axis respectively to show on the graph.

Line3: Displays the graph, there are also functions such as plot.xLabel(string), and plot.yLabel(string).

Note: plot is useless if all it took was arrays/list, that’s why matplotlib actually converts all list into numpy arrays internally before displaying the plots on the figure.

**Working with multiple plots and figures:**

In matplotlib we can **import** **matplotlib.pyplot** **as** **plt**

plt.figure(1) *# the first figure*

plt.subplot(211) *# the first subplot in the first figure*

plt.plot([1, 2, 3])

plt.subplot(212) *# the second subplot in the first figure*

plt.plot([4, 5, 6])

plt.figure(2) *# a second figure*

plt.plot([4, 5, 6]) *# creates a subplot(111) by default*

plt.figure(1) *# figure 1 current; subplot(212) still current*

plt.subplot(211) *# make subplot(211) in figure1 current*

plt.title('Easy as 1, 2, 3') *# subplot 211 title*

we can use plt.figure(1) however this is optional it just like matlab tells us which figure this plots belong to on the next code segment(following code segment), the important parametrers to remembera re the plot.subplot(numRow,numCol,fogNum) where:

numRow specifies how many rows we want for our subplot

numCols specifies how many columns we want for our subplot

figNum maen which number is this figure, where figNum ranges from 1 to numRow \* numCols.

If you want to subplot figure a comma are unnesscary if numrows\* umCols < 10

Ex subplot(211) same as subplot(2, 1, 1)

Sources

<https://matplotlib.org/users/pyplot_tutorial.html> : getting started tutorial

<https://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot> : plotting symbols and styles

<https://plot.ly/matplotlib/histograms/> : histograms

# Python

## numPy Library

NumPy was created in the early 2000’s it’s a library that is used to represent a matrix, or list of list.

In Numpy a rank is known as the number of dimensions, each dimension is called an axis. so the rows are the first axis and the columns are the second axis

To create a matrix of all zeros, just in case you want to allocate the array but don’t have values for it yet, (which is common) np.zeros( (numRows, numCols)) will do that

**Reading from files:** NumPy can read directly from files and load them into your system,

Random numbers numpy

Np.random.rand(numRows,NumCols)

Np.arange(start,stop(exclusive),step) #returns an list,array of the various items

Slicing arrays, if we insteads want to select first three itme sfrom the fourth column, we can do this using the colon( : ) a Colon insdications that we want to select all elements from the starting index up to but not including the exding index.

->select the first 4 elements from the 4th column

Data[0:3, 3] , 012 will be selected from the 4th column

Limitations: one of the limitations of numPy is that all the datatypes want to be read must be the same type.

My Observations Steps

1. look at data, understand the data, its units
   1. Look at its normal distribution of items, see its units in play, scatter plot matrix
   2. Notice continue and discrete variables
2. Plot the raw data, look at the results, see the prediction to get a baseline of where to start and how to improve, if we can’t improve baseline there is no problem or point
   1. Look at the values, the perdiction, choose 3 top models to use to predict the problem and why,
   2. Remember \* before training to split the data using the test\_train\_split() function to split the data evenly
3. After training raw data, take a closer look at the data itself
   1. The number of oberservations, the number of classes in training and testing dtat set,
      1. Are there extreme unbalanced number of classes in either test or training set
      2. Perform normal probability/chance to explain why the performance posibiliy