Python For Applied Machine Learning - Practical 5

Michael Halstead

May 2, 2024

1 Introduction

Today we will be touching on three topics:

1. Outlier detection and how to remove them from data;
2. We will build our own precision recall curve class; and
3. We will use our GMM class to classify pixel colours and output the precision recall curve use sklearn metrics.

2 Outlier Detection

Outliers can be a frustrating part of any data, they can cause your data to be unreadable (NaN, infinite) or skew your data away from the desired outcome (large values in a small distribution), think about our housing data where we had the cap at 500000. There are a number of ways to deal with these outliers, we've already done it once with the housing data where we removed the NaN's from the data.

There is pickle file here called *data/outlier_mvg.pkl* you will need to load it yourself, but the primary distribution of the data is:

$\mu =$ [0.0, 0.0] $\Sigma =$ [[1000.0, 800.0]
[800.0, 1000.0]]

This distribution is what we are aiming to get to, we will need to do a couple of things to get there. The removal of NaN (np.nan) and infinite (np.inf) values can be started by using *np.isnan()* and *np.isinf()* to identify if any value in a matrix is one of these. Once we have the boolean response we can use *np.where()* to convert these into locations, we are really only interested in the unique values in the first array of the tuple output. Then we can delete them using this unique output using *np.delete(array, unique_location, 0)*, you can get the unique location by using *np.unique(array)*.

Once you have removed the NaNs and infinite values we need to actually perform some kind of outlier detection. There are a number of functions for this but we will use the *z-score* and the *modified z-score* to remove our outliers.

The *Z-score* is a basic measure that we have actually seen before. It starts by normalising the data to be mean of 0 and std of 1, then based on the assumption that the std=1 we can use a simple threshold to exclude points external to this data.

$$Z_i = \frac{X_i - \mu}{std} \tag{1}$$

Then we exclude values in Z based on a threshold, in practice this is somewhere around 2.5. To use this threshold you will compare the *z-score* to make sure it is within -2.5 and 2.5 of the standard deviation. If it falls within that range you will return a new vector with only the true values associated with your boolean command (i.e z ¡ threshold and z ¿ -threshold). To help you here you may wish to use *np.logical_and(condition 1, condition 2)* for each of the columns.

$$\begin{aligned} True =& np.logical\_and(True, True) \\ False =& np.logical\_and(False, True) \\ False =& np.logical\_and(True, False) \\ False =& np.logical\_and(False, False) \end{aligned} \tag{2}$$

The second metric is the modified $Z - score$ based on the median rather than the mean and standard deviation[1]. I will call this the $Z - median$:

$$\begin{aligned} D =& median(|x_i - m|), \\ Zm_i =& \frac{0.6745(x_i - m)}{D} \end{aligned} \tag{3}$$

where m represents the median of the full data X, and 0.6745 is a precomputed constant from the previously mentioned paper. In the case of $Z - median$ the expected threshold would be around 3.5. You will do the same thing for sorting out which values to return based on if the *modified z-score* falls within your thresholds ($\pm$).

Once you have output from the two outlier detection routines you will scatter plot the original data, the *z-score* data, and the *modified z-score* data. You will use 3 subplots and ensure the x and y axis are shared between the plots.

3 Precision Recall - Class

Now we will build a precision recall class that we can use to understand how this metric is calculated and what it means. This class will not be very efficient

[1] Boris Iglewicz and David Hoaglin (1993), "Volume 16: How to Detect and Handle Outliers", The ASQC Basic References in Quality Control: Statistical Techniques, Edward F. Mykytka, Ph.D., Editor.

and for your exam I would recommend using the function described in the next section. However, to understand the concept we will code up a simple version.

First, we need to create some basic univariate data that you will plot using *hist* from *matplotlib.pyplot*. Your histogram should have 50 bins and have a range $0. \rightarrow 1.$

```
samples = 1000
X0 = np.random.normal( 0.7, 0.3, size=(samples, 1) )
X0 = np.clip( X0, 0., 1. )
X1 = np.random.normal( 0.3, 0.1, size=(samples, 1) )
X1 = np.clip( X1, 0., 1. )
L0 = np.ones( (samples, 1) )
L1 = np.zeros( (samples, 1) )
X = np.vstack( (X0, X1) )
L = np.vstack( (L0, L1) ).astype( bool )
# Now plot this as a histogram to see what it looks like.
# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.
    html
```

Now we will code up the first part of the precision recall curve class in a new *metrics.py* library file. The pseudo-code is below.

```
class pr_metric():
  def __init__( self, epsilon=0.00000001 ):
    store epsilon as a member of the class, we will use it later to
     ensure we don't divide by 0

  def calculate_statistics( self, labels, predictions ):
    calculate the true positives (TP)
    calculate the false positives (FP)
    calculate the missed detections (MD)

    calculate and return the precision, recall, and the f1 score.
```

The true positives (TP) are the sum of where the labels are True and the predictions are True. The false positives (FP) are where the labels are false and the predictions are True. And the missed detections (MD) are when the labels are True and the predictions are False.

To calculate the precision, recall and f1 score we use the following three equations,

$$\begin{aligned} p &= \frac{TP}{TP + FP + epsilon} \\ r &= \frac{TP}{TP + MD + epsilon} \\ f1 &= \frac{2 * p * r}{p + r + epsilon} \end{aligned} \tag{4}$$

Now based on your visualisation of the data we will select three thresholds to check using this method. You will print out the precision, recall, and f1-score for each of your thresholds. Keep in mind that the $predictions = X > threshold$.

So this is time consuming and you would have to manually enter the values until you got the best score. We will now use the dunder call method to iterate

over a range of scores to find precisely the best f1 value we can from our data. To do this you will add the following code to your *pr_metric* class.

```
def __call__( self, labels, scores )
  create a new variable with the unique ascending ordered scores.
  create P, R, F1 which are vectors of the same size as your new
    sorted and unique scores.

  iterate (for loop with s) over your unique and sorted scores.
    use the calculate_statistics method with the labels and scores>
    s and store the result in P, R, F1.

  calculate the best f1-score

  plot the precision recall curve with a mark where the best f1
    score is located along the curve.

  print out the best precision, recall, f1-score and threshold.
```

4 Precision Recall of Colour GMM - Sklearn

We will now use the GMM class we created last week and use an inbuilt version of the precison recall curve from *sklearn.metrics*. The key thing with the precision recall curve is that it can only be used on two variables (black, and white), if you have more than two (black, white, and red) you have to use something else like a confusion matrix to evaluate performance. So for this we will only use two colours from the colour snippet dataset each time, you can change which ones you want to use.

This means we have to slightly modify our code from last week. Primarily, we only want to load in two colour sets (black and white) so modify your code to only do this. The other thing you will need to be careful with is the labels, you only want 0's or 1's, so modify your code to do this. You will also need to ensure that your *MultiGMM.predict* method now returns the classification and the raw scores that you use to classify. You will then code up the following pseudo-code:

```
create your gmm object
train your gmm object
get the scores from gmm predict of the test set
use column 1 (not 0) from your scores and reshape to -1, 1

use from sklearn.metrics import precision_recall_curve to predict
    the p, r, t
calculate the f1-score

get the best f1-score
plot your precision recall curve and mark where the best f1-score
    is.
```

Why do you think we used an inbuilt precision recall curve instead of our class we coded up? Maybe try it and see what happens. Why did we use the first column and not the 0th column?