How to Calculate the Spearman Rank Order Correlation, Confidence and Signal Shape

Hofbauer and Uhl

This is supplementary material for the paper: Hofbauer16c

Identifying Deficits of Visual Security Metrics for Images,
Heinz Hofbauer, Andreas Uhl
in Signal Processing: Image Communication 46, pp. 60-75, 2016.

1 Introduction

In this Notebook I will show how to calculate the confidence and SROC for image metrics.
We will use the LEG image metric on the LIVE database.
The LIVE database contains mean opinion score values (MOS) which show the impairment of an image, that is, the higher the MOS value the more the image is distorted.
The LEG is a quality metric, that is, the higher the score the higher the quality of the image.
Both of these values are important for the confidence.

```
#order of metric and mos values
LEG_order = "OLQ"
LIVE_order= "OHQ"
#actual values
from leg_live_data import dmos_value
```

2 Rank Order Correlation (Spearman r)

It should be note that if the order of the metric and database do not match then the rank order correlation will be negative. Since this is not of interest to us, usually only the absolute value of the rank order correlation is shown.

The spearman rank order correlation is realtively simple to calculate and there are a number of implementation so it is pointless to repeat them here.

```
import scipy.stats
print("SROC=%5.3f"%( abs(scipy.stats.spearmanr(dmos_value)[0]) ) )
```

```
SROC=0.930
```

3 Confidence

Let us first look at the data via a simple scatterplot.

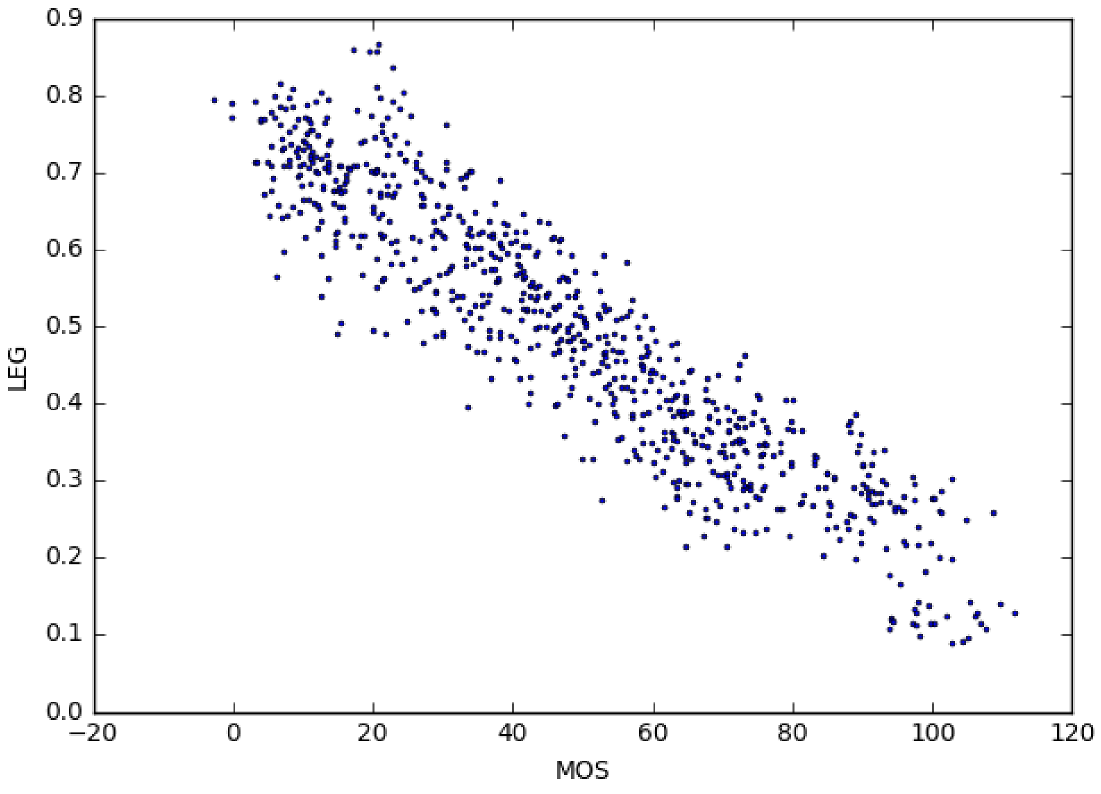
```
#matplotlib with inline graphics
%matplotlib inline
import matplotlib.pyplot as plot

#basic config of image size
plot.rc("figure", figsize=(7.07,5) )
plot.rc("figure", dpi=100 )

x=[dv[0] for dv in dmos_value]
plot.xlabel("MOS")

y=[dv[1] for dv in dmos_value]
plot.ylabel("LEG")

plot.plot(x,y,'o', markersize=2);
```



3.1 Vmin, Vmax and Confidence Value

The databases contain a set of points p representing impaired images with associated values p_v for metric value and p_d for MOS value, both ordered from low to high quality (that is 0HQ in our terms). Based on a target MOS quality score (D) two values can be calculated:

- $V_{min}(D)$ refers to the metric value for which the following holds:
$$p_d > D \implies p_v > V_{min}(D).$$
That is if the metric score is below $V_{min}(D)$ we are sure that the perceived quality is below the MOS quality score (D).

- $V_{max}(D)$ refers to the metric value for which the following holds:
$$p_v > V_{max}(D) \implies p_d > D.$$
That is if the metric score is above $V_{max}(D)$ we are sure that the perceived quality is above the MOS quality score (D).

If either or both of metric and MOS ordering is different we have to adjust the respective equation.

In python this results in the following function, which takes dmos_value pairs with ordering and returns the vmin and vmax values.

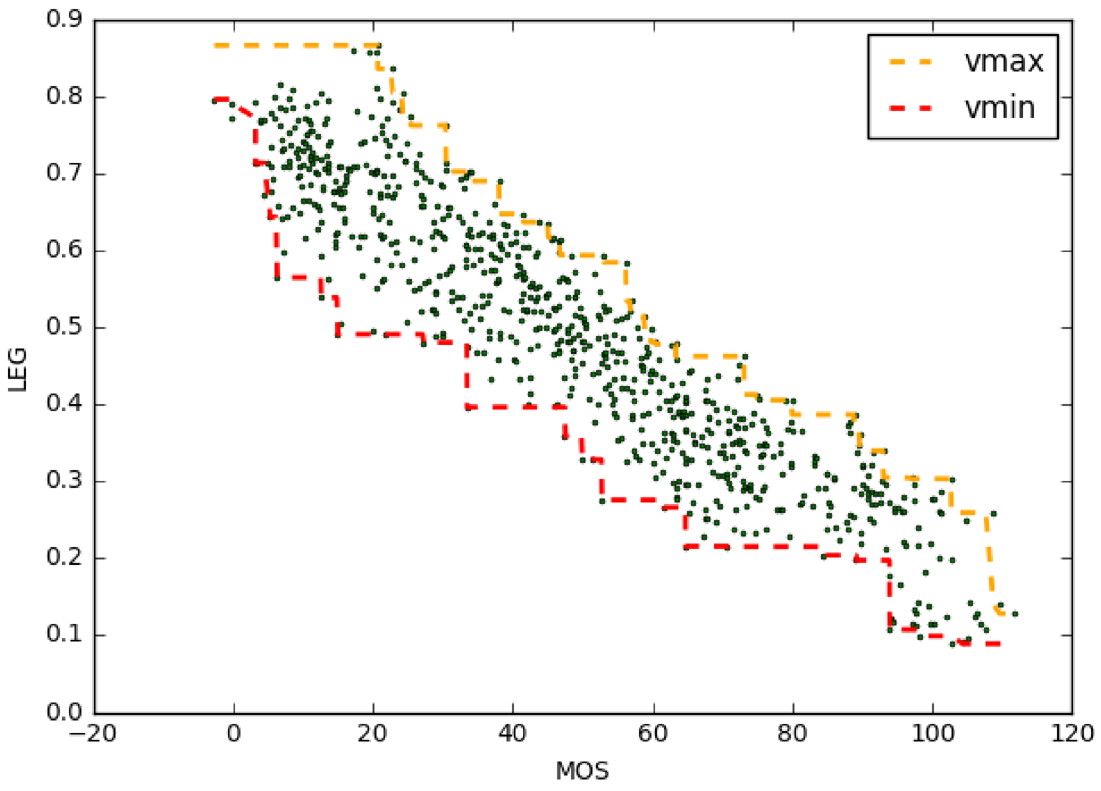
```python
def generate_vmin_vmax(dmos_value, dmos_order="OHQ", value_order="OHQ"):
        return_value=[]
        for idx in xrange(len(dmos_value)):
            dmos,val = dmos_value[idx]
            vmin=val
            vmax=val
            if dmos_order=="OHQ":
                dhigh = [i[1] for i in dmos_value if i[0] < dmos] # higher dmos qual
                dlow = [i[1] for i in dmos_value if i[0] > dmos] # lower dmos qual
            else:
                dhigh = [i[1] for i in dmos_value if i[0] > dmos] # higher dmos qual
                dlow = [i[1] for i in dmos_value if i[0] < dmos] # lower dmos qual

            if value_order=="OHQ":
                if len(dhigh) > 0:vmax = max(dhigh)
                if len(dlow) > 0: vmin = min(dlow)
            else:
                if len(dhigh) > 0: vmin = min(dhigh)
                if len(dlow)> 0: vmax = max(dlow)

            return_value.append([vmin,vmax])
        return return_value
```

Now we can generate (and plot) the minimum and maximum values for the dmos value pairs we have.

```python
vminmax = generate_vmin_vmax( dmos_value, LIVE_order, LEG_order)
vmin = [v[0] for v in vminmax]
vmax = [v[1] for v in vminmax]
plot.plot(x,y,'o',markersize=2,color="green")
plot.xlabel("MOS")
plot.ylabel("LEG")
plot.plot(x,vmax,'--',color="orange",linewidth=2,label="vmax")
plot.plot(x,vmin,'--',color="red",linewidth=2,label="vmin")
plot.legend();
```



This also means that for a given MOS value D any metric quality score p_g should be in the intervall

$$V_{min}(D) \geq p_g \geq V_{max}(D).$$

Thus we can define the confidence $\mathcal{C}_D$ for a metric score based on a given perceived quality D as

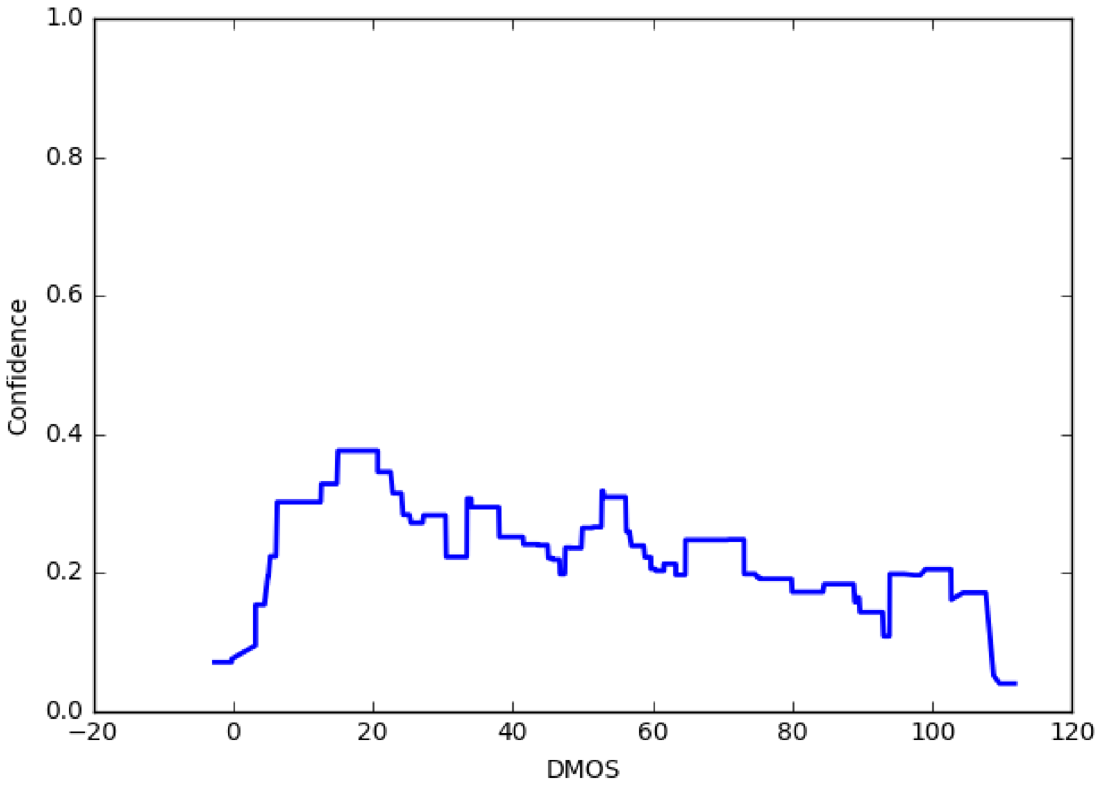
$$\mathcal{C}_D := |V_{max}(D) - V_{min}(D)|.$$

A confidence score over the full perceived quality range can be given as

$$\mathcal{C} = \frac{1}{\#S} \sum_{D \in S} \mathcal{C}_D,$$

where (S) is the set of distinct MOS samples from the database.

```
D=[ vmax-vmin for vmin,vmax in vminmax]
plot.ylim(0,1)
plot.xlabel("DMOS")
plot.ylabel("Confidence")
plot.plot(x,D, linewidth=2);
```



3.1.1 Normalizing the Maximum metric value

Now before we continue consider the case of unbound metrics, like for example PSNR. In order to make the values comparable we normalize to the ranage of $[0, \ldots, 1]$. The LEG the values are normalized already, but we still hold all metrics to the same procedure. As such we will adjust the LEG such that the actual values are normalized such that the highest value is 1.

If values are not strictly above zero or go into negtive unbound values (LSS for example) then it gets slightly more complicated. The following function can be used to get the norm factor for generic cases.

```
def getNormfactor(metric_values):
    minv = min(metric_values)
    maxv = max(metric_values)
    if minv > 0:
        minv=0
    if maxv < 0:
        maxv=0
    return  abs(maxv)+abs(minv)

Dnorm=[ d/getNormfactor(y) for d in D]
```

3.2 Average and Standard Deviation of the Confidence Value

Also note that we can interpret $\mathcal{C}$ as the average over $\mathcal{C}_D$,

$$\mu_{D\in S}(\mathcal{C}_D),$$

and consequently we can also calculate the standard deviation,

$$\sigma_{D\in S}(\mathcal{C}_D).$$

```
import numpy
Dmu = numpy.average(Dnorm)
Dsigma = numpy.std(Dnorm)
print "mu = %5.3f\nsigma = %5.3f"%( Dmu, Dsigma)
```

```
mu = 0.291
sigma = 0.070
```

3.3 Signal Shape

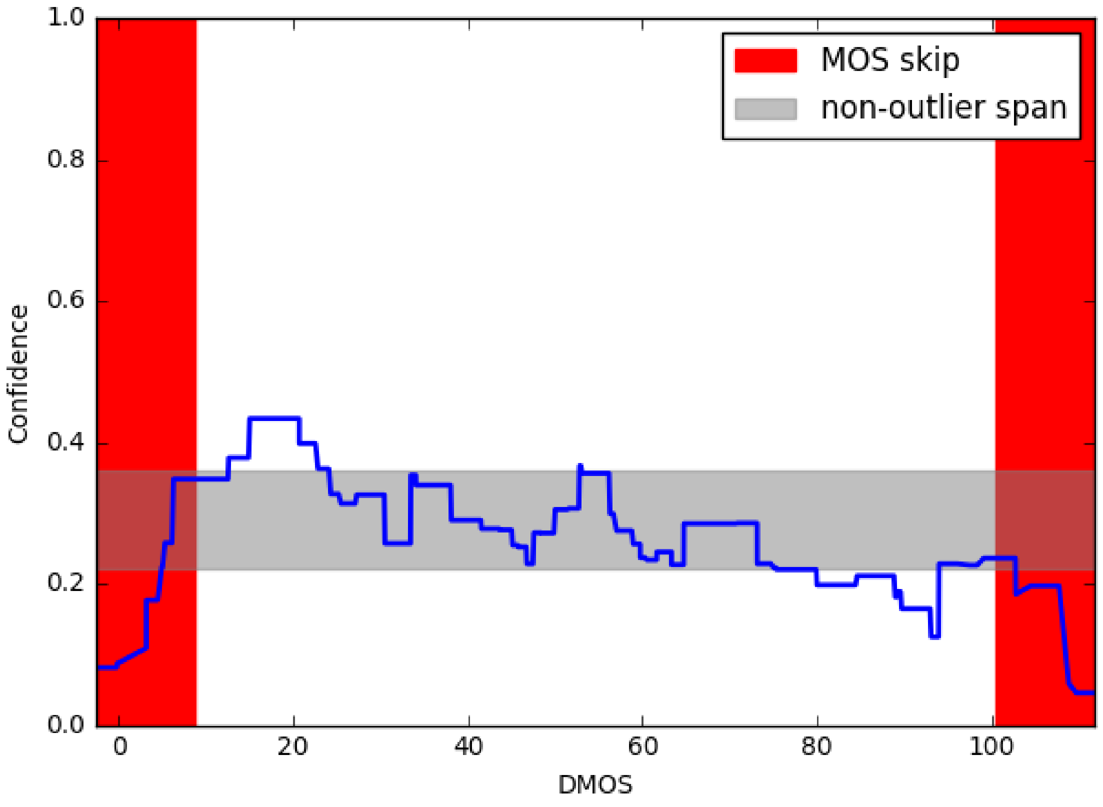
In our case the signal shape is defined by outliers, basically values that are outside of the range $\mu \pm \sigma$. When looking at the plot of the confidence value however, it is clear that at the beginning and end of the graph they confidence tips steeply. This is due to the metric values being bound by 0 and 1. Meaning the spread of values naturally decreases.

In order to remove these outliers, which will happen for practically every image metric we will skip the first and last 10% of the MOS values.

The following figure illustrates the $\mu \pm \sigma$ range as well as the parts that are skipped for outlier detection.

```
MOSmin = min(x)
MOSmax = max(x)
MOSrange = MOSmax - MOSmin
MOSskip = MOSrange*0.1

plot.ylim(0,1)
plot.xlim(MOSmin,MOSmax)
plot.xlabel("DMOS")
plot.ylabel("Confidence")
plot.axvspan(MOSmin, MOSmin+MOSskip, color="red");
plot.axvspan(MOSmax-MOSskip, MOSmax, color="red", label="MOS skip");
plot.axhspan(Dmu-Dsigma,Dmu+Dsigma, color="gray", alpha=0.5, label="non-outlier span");
plot.plot( x, Dnorm, linewidth=2)
plot.legend();
```



3.3.1 Outlier and z-Score

In order to calculate the outliers we simply normalize the confidence value d in such a way that the value gives the distance to the average μ in multiple of σ. This is called the z-Score:

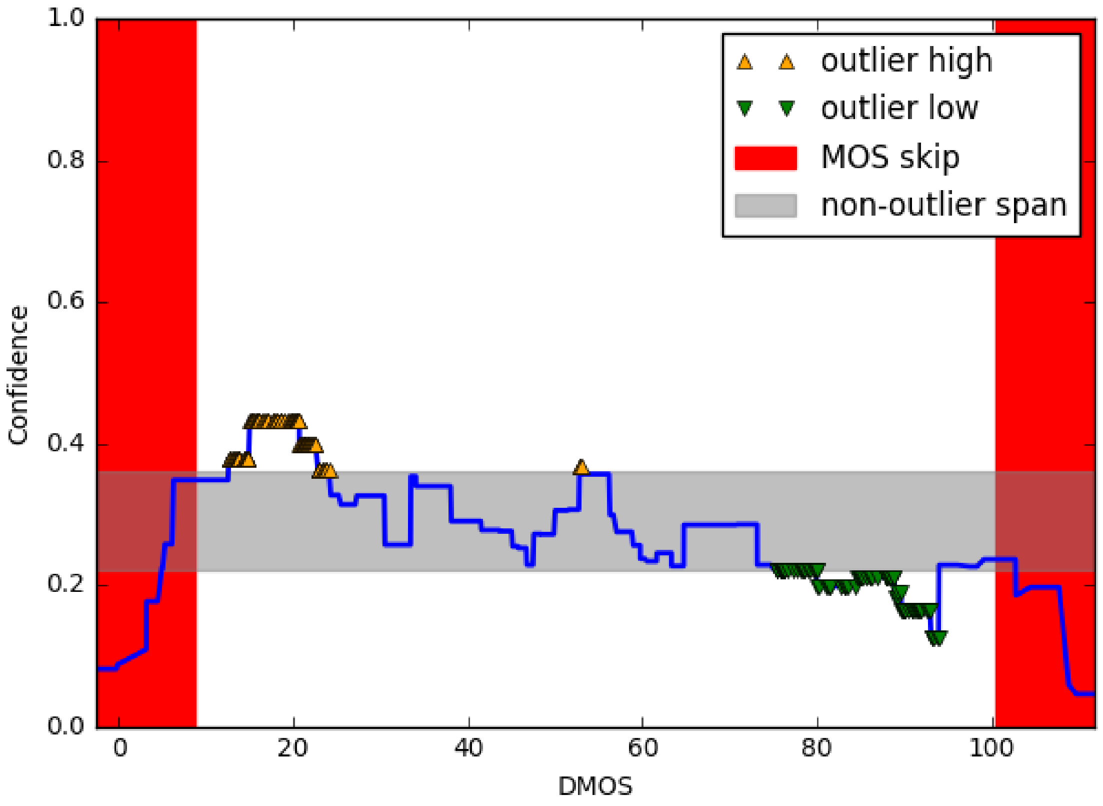
$$z(d, \mu, \sigma) = \frac{d - \mu}{\sigma}.$$

Since we assume values d are outliers if they are more than one σ away from μ we define a z-Score $> +1$ as a high outlier and a z-Score < -1 as a low outlier.

```
zscore=[ (d-Dmu)/Dsigma for d in Dnorm ]
outlier = []
for idx in range(len(x)):
    if x[idx] < MOSmin+MOSskip or x[idx] > MOSmax-MOSskip:
        outlier.append(0)
    elif zscore[idx] < -1:
        outlier.append(-1)
    elif zscore[idx] >1:
        outlier.append(1)
    else:
        outlier.append(0)

#for plotting
outlier_low_x = []
outlier_low_y = []
outlier_high_x = []
outlier_high_y = []
for idx in range(len(outlier)):
    if outlier[idx] == 1:
        outlier_high_x.append( x[idx])
        outlier_high_y.append( Dnorm[idx])
    if outlier[idx] == -1:
        outlier_low_x.append( x[idx])
        outlier_low_y.append( Dnorm[idx])

plot.ylim(0,1)
plot.xlim(MOSmin,MOSmax)
plot.xlabel("DMOS")
plot.ylabel("Confidence")
plot.axvspan(MOSmin, MOSmin+MOSskip, color="red");
plot.axvspan(MOSmax-MOSskip, MOSmax, color="red", label="MOS skip");
plot.axhspan(Dmu-Dsigma,Dmu+Dsigma, color="gray", alpha=0.5, label="non-outlier span");
plot.plot( x, Dnorm, linewidth=2)
plot.plot(outlier_high_x, outlier_high_y, '^', color="orange", label="outlier high")
plot.plot(outlier_low_x, outlier_low_y, 'v', color="green", label="outlier low")
plot.legend();
```



3.3.2 Signal Shape

The signal shape depens on the outlier and relative position of the outliers to each other.

If there are not outliers, which means the all confidence values fall into the range of $\mu \pm \sigma$ then the **signal is stable**.

If there is a clear threshold separating high and low outliers then the **signal is biased**. The signal is biased towards the low outliers, that is, the metric prediction of the MOS values is better where the low outliers are.

If the outliers are intermixed, i.e., they can not be separated clearly, then the **signal is unstable**.

```
def getBias(outlier, MOS_order):
    #We are only interested in true outliers, i.e. != 0
    true_outlier = [o for o in outlier if o != 0]

    #no outlier -> stable
    if len(true_outlier) == 0: return "Stable"

    start = true_outlier[0]
    flipped = False
    for i in true_outlier:
        if not flipped:
            if i == start: continue
            # found a outlier in the other direction
            flipped = True
        else:
            ## found a flipped, now flipped back -> unstable
            if i == start: return "Unstable"

    #neither stable nor unstable, which bias?
    if start < 0:
        #low mos
        if MOS_order == "OHQ":
            return "Bias High"
        else:
            return "Bias Low"
    else:
        #high mos
        if MOS_order == "OHQ":
            return "Bias Low"
        else:
            return "Bias High"

getBias(outlier, LIVE_order)
```

```
'Bias Low'
```