Statistical test on TS on the results

February 12, 2017

0.0.1 This report shows applyining statistical tests of the results of Multi armed bandit of pruning the parameters

0.0.2 "pruning the weights using TS"

0.0.3 Here, we are showing two kinds of testing ANOVA test and Nonparametric tests

1 Import needed libraries

1.1 Import libraries for manipulating the data and statistic

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import scipy.stats as stats
        from scipy.stats import ttest_1samp, wilcoxon, ttest_ind, mannwhitneyu
        import scipy.special as special
        import emoji
        from math import pi
        from statsmodels.stats.multicomp import pairwise_tukeyhsd, MultiComparison
        from statsmodels.formula.api import ols
        import statsmodels.stats.api as sms
```

1.2 Import libraries for static ploting

```
In [2]: import matplotlib.pyplot as plt
        import matplotlib.gridspec as gridspec
        %matplotlib inline
        from IPython.display import set_matplotlib_formats
        set_matplotlib_formats('png', 'pdf')
        # some nice colors from http://colorbrewer2.org/
        COLOR1 = '#7fc97f'
        COLOR2 = '#beaed4'
        COLOR3 = '#fdc086'
        COLOR4 = '#ffff99'
        COLOR5 = '#386cb0'
```

1.3 Import libraries for interactive ploting Plotly

```
In [3]: import plotly.plotly as py
        from plotly.graph_objs import *
        import plotly.graph_objs as go
        #from plotly.tools import FigureFactory as FF
        import plotly.figure_factory as FF
        import cufflinks as cf
        cf.go_offline()
```

```
<IPython.core.display.HTML object>
```

1.4 Import libraries for interactive ploting BOKEH

```
In [4]: from bokeh.charts import Bar, Area, defaults, Donut
        from bokeh.layouts import row, gridplot
        from bokeh.charts.attributes import cat, color
        from bokeh.charts.operations import blend
        from bokeh.plotting import figure, output_notebook, show
        from bokeh.models import Legend
        TOOLS = 'box_zoom,box_select,crosshair,resize,reset,lasso_select,pan,save,poly_select,ta
        #defaults.width = 1000
        #defaults.height = 800
        output_notebook()
```

2 Statring the test and visulize the data

2.1 Load the data for pruning the weights using random expoloration

```
In [5]: datafile = "TS.csv"
        datafileLeNet = "LecunPruningWeights.csv"
        df1 = pd.read_csv(datafile)
        dfLcun = pd.read_csv(datafileLeNet)
        df1
```

```
Out[5]:                        Dataset  Model  Thom. Sam   OBD   OBS  Magnitude  random
        0     banknote authentication   0.01       0.01  0.01  0.02       3.23    5.13
        1   Blood Tra. Service Centre   0.08       0.08  0.08  0.08       0.44    0.08
        2             Credit Approval   0.08       0.78  0.08  8.62       2.55   22.19
        3          Haberman's Survival  0.09       0.08  0.08  0.08       0.63    0.65
        4             Liver Disorders   0.10       0.10  0.10  0.85       0.62    0.15
        5            MAGIC Gamma Tele.  0.06       0.06  0.06  0.12       2.49    0.43
        6            Mammographic Mass  0.09       0.09  0.09  0.09       2.59    0.13
        7             MONK's Problems   0.10       0.10  0.10  5.28       0.15    0.13
        8          Connectionist Bench  0.12       1.07  0.12  0.12       0.16    0.16
        9                    Spambase   0.08       0.09  0.08  4.37       1.67    5.01
        10               SPECTF Heart   0.06       0.08  0.06  0.14      12.25    0.06
        11         Tic-Tac-Toe Endgame  0.06       0.06  0.06  0.07      21.30   11.92
```

```
In [6]: dfLcun

Out[6]:   Layer   Model  TS Prune half the weights
        0    FC  0.9906                      0.993
        1  Conv  0.9906                      0.991

In [7]: p = Bar(df1, label='Dataset',
                values = blend('Model', 'Thom. Sam' ,'OBD','OBS',
                               'Magnitude',
                               'random',name='Scores', labels_name='Score'),
               group=cat(columns='Score', sort=False),
               title="Compare the performance", legend='top_center',
               tools=TOOLS, plot_width=900, plot_height=600,
               tooltips=[('Score', '@Score'), ('Model', '@Dataset')],
               xlabel='List of datasets', ylabel='Error')
        p.title.align = "center"
        #p.yaxis.major_label_orientation = "vertical"
        p.xaxis.major_label_orientation = pi/2
        show(p)

In [8]: p = Bar(dfLcun, label='Layer',
                values = blend('Model', 'TS Prune half the weights',name='Scores', labels_name='
               group=cat(columns='Score', sort=False),
               title="Compare the performance", legend='bottom_center',
               tools=TOOLS, plot_width=900, plot_height=600,
               tooltips=[('Score', '@Score'), ('Model', '@Layer')],
               xlabel='List of Layers', ylabel='Accuracy')
        p.title.align = "center"
        #p.yaxis.major_label_orientation = "vertical"
        p.xaxis.major_label_orientation = pi/2
        show(p)

In [9]: df=df1.copy()
        df.set_index('Dataset', inplace=True)
        py.iplot([{
            'x': df.index,
            'y': df[col],
            'name': col
        }  for col in df.columns])

Out[9]: <plotly.tools.PlotlyDisplay object>

In [10]: # Lecun Model
         dflc=dfLcun.copy()
         dflc.set_index('Layer', inplace=True)
         py.iplot([{
             'x': dflc.index,
             'y': dflc[col],
             'name': col
         }  for col in dflc.columns])
```

```
Out[10]: <plotly.tools.PlotlyDisplay object>

In [11]: df.iplot(subplots=True, subplot_titles=True, legend=False )

<IPython.core.display.HTML object>


In [12]: df.iplot(subplots=True, shape=(8,1), shared_xaxes=True, fill=True)

<IPython.core.display.HTML object>


In [13]: df.iplot(kind='bar')

<IPython.core.display.HTML object>


In [14]: df.iplot(kind='bar', barmode='stack')

<IPython.core.display.HTML object>


In [15]: df.iplot(kind='barh',barmode='stack', bargap=.2)

<IPython.core.display.HTML object>


In [16]: df.iplot(kind='histogram')

<IPython.core.display.HTML object>


In [17]: df.scatter_matrix(world_readable=True)

<IPython.core.display.HTML object>


In [18]: df.iplot(kind='box')

<IPython.core.display.HTML object>


In [19]: p = Bar(df1, label='Dataset',
                 values = blend('Model', 'Thom. Sam',name='Scores', labels_name='Score'),
                group=cat(columns='Score', sort=False),
                title="Compare the performance", legend='top_center',
                tools=TOOLS, plot_width=900, plot_height=600,
                tooltips=[('Score', '@Score'), ('Model', '@Dataset')],
                xlabel='List of datasets', ylabel='Error')
         p.title.align = "center"
         #p.yaxis.major_label_orientation = "vertical"
         p.xaxis.major_label_orientation = pi/2
         ################################################################################
         show(p)
```

2.1.1 We will use alpha 0.05 to do ANOVA test. The null hypothesis there is no difference between the all methods and the alternative hypothesis there is a difference. According to p-value we see if there is a difference.

```
In [20]: # Perform the ANOVA
         stats.f_oneway(df1['Model'],df1['Thom. Sam'] , df1['OBD'],
                        df1['OBS'],df1['Magnitude'],df1['random'])
```

```
Out[20]: F_onewayResult(statistic=2.6273655974573149, pvalue=0.031563475049194982)
```

2.1.2 p-value = 0.035020053547419529 < 0.05 where small p-values suggest that the null hypothesis is unlikely to be true then we reject the null hypothesis which's mean there is a difference.

2.1.3 The test output yields an F-statistic of 2.40 and a p-value of 0.035020053547419529, indicating that there is significant difference between the means of each group.

The test result suggests the groups don't have the same sample means in this case, since the p-value is significant at a 95% confidence level.

We want to test the best pruning model which is this case is TS family

To check which groups differ after getting a positive ANOVA result, we can perform a follow up test or "post-hoc test".

2.1.4 One post-hoc test is to perform a separate t-test for each pair of groups. We can perform a t-test between all pairs using by running each pair through the stats.ttest_ind() we covered in the following to do t-tests:

```
In [21]: # Get all models pairs
         interstModel = ['Thom. Sam']
         lst = list(df1.columns.values)
         lst.remove('Dataset')
         model_pairs = []

         for m1 in range(len(df1.columns)-2):
             for m2  in range(m1+1,len(df1.columns)-1):
                 model_pairs.append((lst[m1], lst[m2]))

         # Conduct t-test on each pair
         pvalueList = []
         new_model_pairs = []
         for m1, m2 in model_pairs:
             print('\n',m1, m2)
             pvalue = stats.ttest_ind(df1[m1], df1[m2])
             #print(pvalue[1])
             if (m1 in interstModel or m2 in interstModel):
                 new_model_pairs.append((m1,m2))
                 pvalueList.append(pvalue[1])
             print(pvalue)
```

```
 Model Thom. Sam
Ttest_indResult(statistic=-1.4236866987486101, pvalue=0.16856608350251728)

 Model OBD
Ttest_indResult(statistic=0.073234127598741677, pvalue=0.94228157972204629)

 Model OBS
Ttest_indResult(statistic=-1.9160734438661973, pvalue=0.068440210215287733)

 Model Magnitude
Ttest_indResult(statistic=-2.1405072319282352, pvalue=0.043650582535484338)

 Model random
Ttest_indResult(statistic=-1.9125261657982566, pvalue=0.068916013437619064)

 Thom. Sam OBD
Ttest_indResult(statistic=1.4323016719620232, pvalue=0.16611391988577526)

 Thom. Sam OBS
Ttest_indResult(statistic=-1.7348147949885926, pvalue=0.096763985488854717)

 Thom. Sam Magnitude
Ttest_indResult(statistic=-2.0618114351977597, pvalue=0.051234112020019824)

 Thom. Sam random
Ttest_indResult(statistic=-1.8394809707113433, pvalue=0.079379195796686619)

 OBD OBS
Ttest_indResult(statistic=-1.9170884030883408, pvalue=0.068304603881402803)

 OBD Magnitude
Ttest_indResult(statistic=-2.140961591206707, pvalue=0.043609903648211962)

 OBD random
Ttest_indResult(statistic=-1.9129504322071127, pvalue=0.068858953230263545)

 OBS Magnitude
Ttest_indResult(statistic=-1.1699913498827907, pvalue=0.254523709674912)

 OBS random
Ttest_indResult(statistic=-1.0247290048069007, pvalue=0.3166273271391854)

 Magnitude random
Ttest_indResult(statistic=0.063211556114818712, pvalue=0.95016886148726365)
```

```
In [22]: for pair, p in zip(new_model_pairs, pvalueList):
    if p < 0.05:
        print('The pvalue between',pair, 'is', p, '< 0.05 then',
              emoji.emojize('REJECT the NULL Hypothesis :thumbs_up_sign:'))
    else:
        print('The pvalue between',pair, 'is', p, '> 0.05 then',
              emoji.emojize('FAIL to REJECT the NULL Hypothesis :thumbs_down_sign:'))
```

```
The pvalue between ('Model', 'Thom. Sam') is 0.168566083503 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('Thom. Sam', 'OBD') is 0.166113919886 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('Thom. Sam', 'OBS') is 0.0967639854889 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('Thom. Sam', 'Magnitude') is 0.05123411202 > 0.05 then FAIL to REJECT the NU
The pvalue between ('Thom. Sam', 'random') is 0.0793791957967 > 0.05 then FAIL to REJECT the NUL
```

```
In [23]: matrix_twosample = []
         matrix_twosample.append(['Methods', 'P value', 'Null Hypothesis', 'EMOJI'])
         for pair, p in zip(new_model_pairs, pvalueList):
             if p < 0.05:
                 matrix_twosample.append((pair, p, 'REJECT', emoji.emojize(':thumbs_up_sign:')))
             else:
                 matrix_twosample.append((pair, p, 'ACCEPT (FAIL TO REJECT)', emoji.emojize(':th
         colorscale = [[0, '#4d004c'],[.5, '#f2e5ff'],[1, '#ffffff']]
         #colorscale = [[0, '#272D31'],[.5, '#ffffff'],[1, '#ffffff']]
         #font=['#FCFCFC', '#00EE00', '#008B00', '#004F00', '#660000', '#CD0000', '#FF3030']
         #font=['#FCFCFC', '#00EE00', '#008B00']
         #table.layout.width=250
         twosample_table = FF.create_table(matrix_twosample, index=True, colorscale=colorscale)
         py.iplot(twosample_table)
```

```
Out[23]: <plotly.tools.PlotlyDisplay object>
```

2.1.5 Margin of Error and Confidence Intervals

margin of error = Tcritical*SE
Confidence Intervals = point estimate ś Margin of Error

1. For TS

```
In [24]: dd = df1.copy()
         dd['diff'] = dd['Thom. Sam'] - dd['Model']
         n = len(dd['diff'])
         t = stats.t.ppf(1-0.025, n)
         def interval_margin(d, t):
             mn = d.mean()
             sd = d.std()
             se = sd/np.sqrt(len(d))
             m = se * t
             ci_lower = mn - m
             ci_upper = mn + m
```

```
                return mn, ci_lower, ci_upper, m

        Pint_Estimate, Lower_CI, Upper_CI, Margin_of_Error = interval_margin(dd['diff'], t)
        print('Point Estimate =', Pint_Estimate )
        print('\nMargin of Error =', Margin_of_Error )
        print('\nConfidence Intervals = point estimate ś Margin of Error')
        print('Confidence Intervals = ', Pint_Estimate, 'ś', Margin_of_Error)
        print('Confidence Intervals = (', Lower_CI,',', Upper_CI, ')' )
```

```
Point Estimate = 0.139166666667

Margin of Error = 0.204310831595

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.139166666667 ś 0.204310831595
Confidence Intervals = ( -0.065144164928 , 0.343477498261 )
```

2.2 Perform Tukey's range test (Tukey's Honestly Significant Difference)

Create a set of confidence intervals on the differences between the means of the levels of a factor with the specified family-wise probability of coverage. The intervals are based on the Studentized range statistic, Tukey's 'Honest Significant Difference' method. [Wekipedia]

```
In [25]: df_for_Tukey = df1.copy()
         del df_for_Tukey['Dataset']
```

```
In [26]: # group the data as tukeyhsd is needed
         lst = []
         for c in df_for_Tukey.columns:
             for r in df_for_Tukey[c]:
                 lst.append((c,r))
```

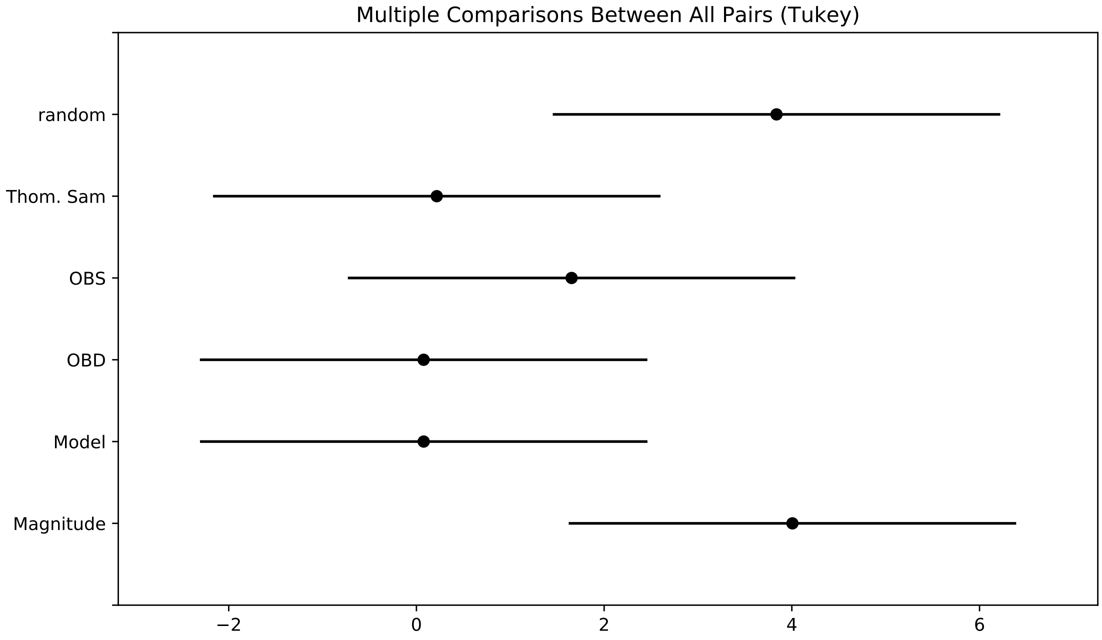
```
In [27]: # make two groups
         data = np.rec.array(lst,
                             dtype = [('Model','|U10'),('Score', '<f2')])
```

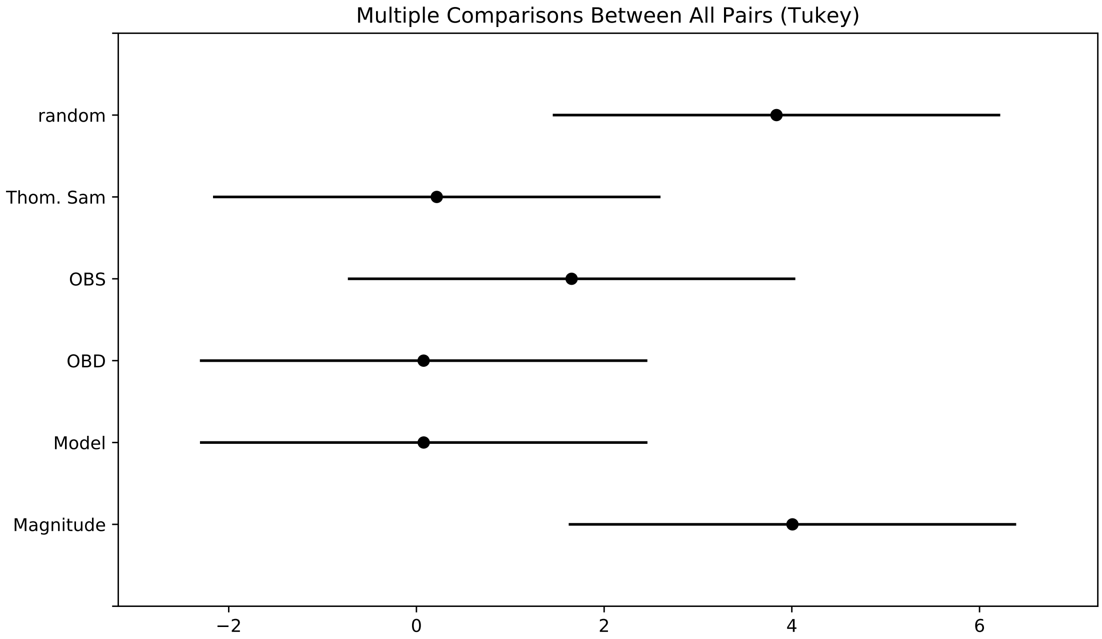
```
In [28]: # perform the test
         mc = MultiComparison(data['Score'], data['Model'])
         result = mc.tukeyhsd()
         print(result)
```

```
Multiple Comparison of Means - Tukey HSD,FWER=0.05
==================================================
  group1    group2  meandiff  lower  upper  reject
--------------------------------------------------
Magnitude   Model    -3.929   -8.697 0.839  False
Magnitude    OBD    -3.9298  -8.6978 0.8381 False
Magnitude    OBS    -2.3532  -7.1212 2.4148 False
Magnitude Thom. Sam -3.7898  -8.5578 0.9781 False
Magnitude   random  -0.1699  -4.9378 4.5981 False
  Model       OBD    -0.0008  -4.7688 4.7671 False
  Model       OBS     1.5758  -3.1922 6.3438 False
  Model    Thom. Sam  0.1392  -4.6288 4.9071 False
  Model     random    3.7592  -1.0088 8.5271 False
   OBD        OBS     1.5766  -3.1913 6.3446 False
   OBD     Thom. Sam   0.14    -4.628 4.908  False
   OBD      random     3.76    -1.008 8.528  False
   OBS     Thom. Sam -1.4366  -6.2046 3.3313 False
   OBS      random    2.1834  -2.5846 6.9513 False
Thom. Sam   random     3.62    -1.148 8.3879 False
--------------------------------------------------
```

```
In [29]: result.plot_simultaneous()
```

```
Out[29]:
```





From the figure we can coclude that deep compression, OBS and randon are the worest.

2.3 eta squared

proportion of total variation that is due to between group differences (explain variation) http://imaging.mrc-cbu.cam.ac.uk/statswiki/FAQ/effectSize

```
In [30]: def FPvalue( *args):

             df_btwn, df_within = __degree_of_freedom_( *args)

             mss_btwn = __ss_between_( *args) / float( df_btwn)
             mss_within = __ss_within_( *args) / float( df_within)

             F = mss_btwn / mss_within
             P = special.fdtrc( df_btwn, df_within, F)

             return( F, P)

         def EtaSquare( *args):

             return( float( __ss_between_( *args) / __ss_total_( *args)))

         def __concentrate_( *args):

             v = list( map( np.asarray, args))
             vec = np.hstack( np.concatenate( v))
```

```
    return( vec)

def __ss_total_( *args):

    vec = __concentrate_( *args)
    ss_total = sum( (vec - np.mean( vec)) **2)
    return( ss_total)

def __ss_between_( *args):

    grand_mean = np.mean( __concentrate_( *args))

    ss_btwn = 0
    for a in args:
        ss_btwn += ( len(a) * ( np.mean( a) - grand_mean) **2)

    return( ss_btwn)

def __ss_within_( *args):
    return( __ss_total_( *args) - __ss_between_( *args))

def __degree_of_freedom_( *args):
    args = list( map( np.asarray, args))
    # number of groups minus 1
    df_btwn = len( args) - 1

    # total number of samples minus number of groups
    df_within = len( __concentrate_( *args)) - df_btwn - 1

    return( df_btwn, df_within)
eta = EtaSquare(df1['OBS'], df1['Model'],df1['OBD'],
                df1['Thom. Sam'], df1['Magnitude'])
print('The Eta square of anova test is ', eta)
if eta>=0.14:
    print('This eta square consider to be Large')
elif 0.06<=eta<0.14:
    print('This eta square consider to be Medium')
elif 0.01<=eta<0.06:
    print('This eta square consider to be Small')
else:
    print('This eta square consider to be very Small')
```

```
The Eta square of anova test is  0.20596923876279902
This eta square consider to be Large
```

2.3.1 The eta Square is large which means 23% the difference based on the variation in the group mean

2.4 Cohen's d

if any two samples have a bsolute different greater that 2.505 the the different conseder honestly significant difference

```
In [31]: # Compute Cohen's d
         from numpy import std, mean, sqrt
         def cohen_d(x,y):
             if type(x)==list: # if the input data list
                 nx = len(x)
                 ny = len(y)
                 dof = nx + ny - 2
                 return (mean(x) - mean(y)) / sqrt(((nx-1)*std(x, ddof=1) ** 2 + (ny-1)*std(y, d
             else:   # if the input numpy array or series[pandas]
                 diff = x.mean() - y.mean()
                 n1, n2 = len(x), len(y)
                 var1 = x.var()
                 var2 = y.var()
                 pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
                 return (diff / np.sqrt(pooled_var))
```

```
In [32]: def eval_pdf(rv, num=4):
             mean, std = rv.mean(), rv.std()
             xs = np.linspace(mean - num*std, mean + num*std, 100)
             ys = rv.pdf(xs)
             return xs, ys
```

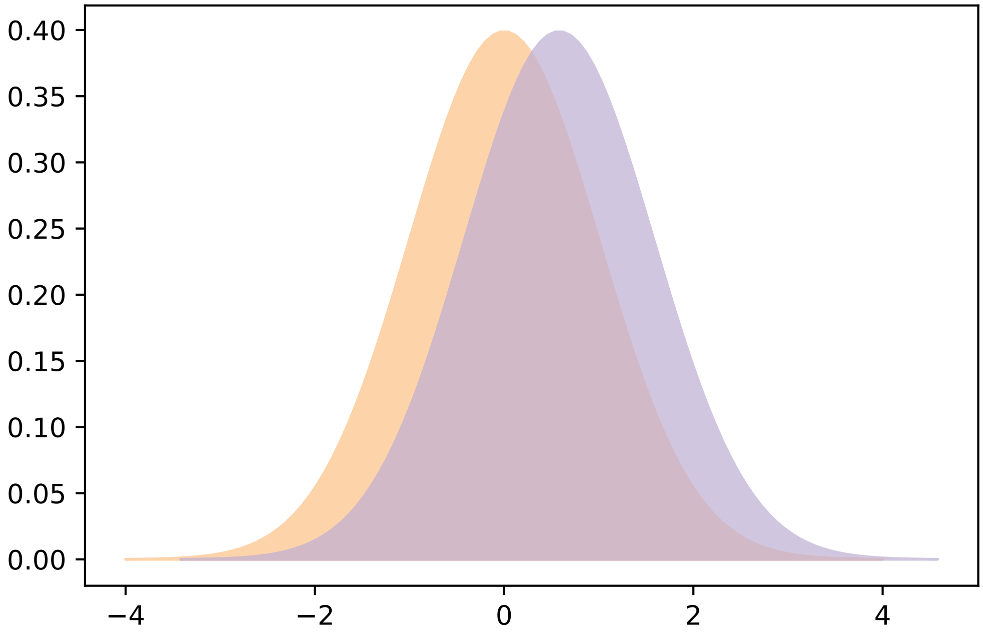
```
In [33]: def overlap_superiority(control, treatment, n=1000):
             control_sample = control.rvs(n)
             treatment_sample = treatment.rvs(n)
             thresh = (control.mean() + treatment.mean()) / 2

             control_above = sum(control_sample > thresh)
             treatment_below = sum(treatment_sample < thresh)
             overlap = (control_above + treatment_below) / n

             superiority = sum(x > y for x, y in zip(treatment_sample, control_sample)) / n
             return overlap, superiority
```

```
In [34]: def plot_pdfs(cohen_d=2):
             control = stats.norm(0, 1)
             treatment = stats.norm(cohen_d, 1)
             xs, ys = eval_pdf(control)
             plt.fill_between(xs, ys, label='control', color=COLOR3, alpha=0.7)

             xs, ys = eval_pdf(treatment)
```

```
            plt.fill_between(xs, ys, label='treatment', color=COLOR2, alpha=0.7)

            o, s = overlap_superiority(control, treatment)
            print('overlap', o)
            print('superiority', s)
```

```
In [35]: c2 = cohen_d(df1['Thom. Sam'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between TS and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between TS and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between TS and Model is 0.58121766092 <2.505 then No honestly significant difference
overlap 0.776
superiority 0.655
```



2.5 t student test in Lecun Model

```
In [36]: dfLcun
```

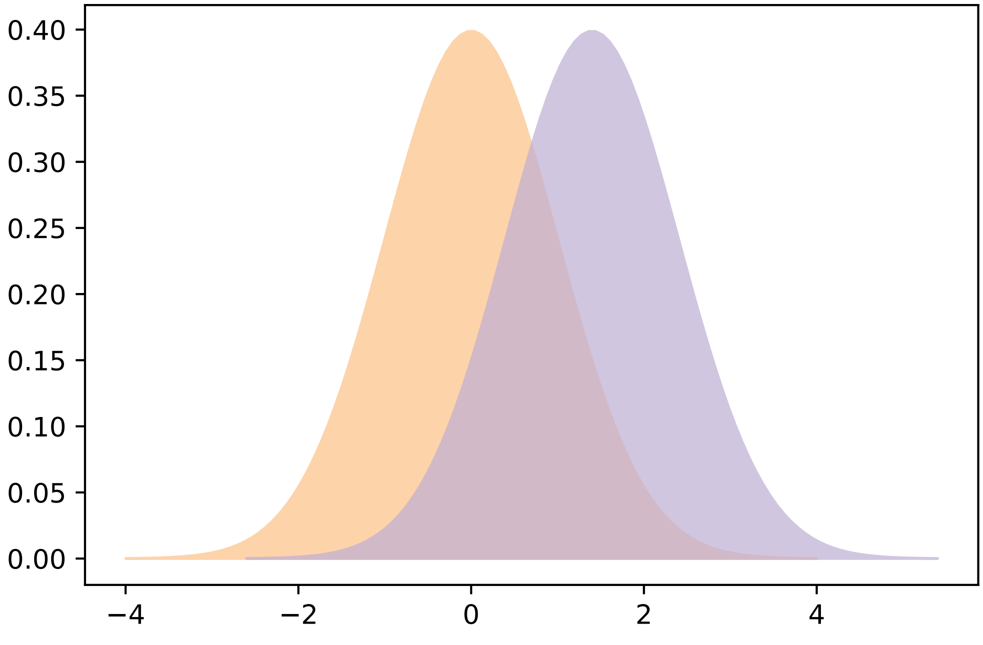
```
Out[36]:   Layer   Model  TS Prune half the weights
         0    FC  0.9906                      0.993
         1  Conv  0.9906                      0.991
```

```
In [37]: print('TS vs random Pruning')
         H, pval = stats.ttest_ind(dfLcun['TS Prune half the weights'], dfLcun['Model'])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval/2)))
         if pval/2 < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval/2 > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
TS vs random Pruning
H-statistic:        1.4
P-value:        0.148236764659
Accept NULL hypothesis - No significant difference between groups.
```

```
In [38]: cL = cohen_d(dfLcun['TS Prune half the weights'], dfLcun['Model'])
         if cL >= 2.505:
             print('The Cohen d between TS and Model is', cL, '>2.505 then',
                   emoji.emojize('honestly significant difference :thumbs_up_sign:'))
         else:
             print('The Cohen d between TS and Model is', cL, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(cL)
```

```
The Cohen d between TS and Model is 1.4 <2.505 then No honestly significant difference
overlap 0.49
superiority 0.829
```



2.5.1 Margin of Error and Confidence Intervals of Lecun Model

margin of error = Tcritical*SE
Confidence Intervals = point estimate ś Margin of Error

```
In [39]: dd = dfLcun.copy()
         dd['diff'] = dd['TS Prune half the weights'] - dd['Model']
         n = len(dd['diff'])
         t = stats.t.ppf(1-0.025, n)
         def interval_margin(d, t):
             mn = d.mean()
             sd = d.std()
             se = sd/np.sqrt(len(d))
             m = se * t
             ci_lower = mn - m
             ci_upper = mn + m
             return mn, ci_lower, ci_upper, m

         Pint_Estimate, Lower_CI, Upper_CI, Margin_of_Error = interval_margin(dd['diff'], t)
         print('Point Estimate =', Pint_Estimate )
         print('\nMargin of Error =', Margin_of_Error )
         print('\nConfidence Intervals = point estimate ś Margin of Error')
         print('Confidence Intervals = ', Pint_Estimate, 'ś', Margin_of_Error)
         print('Confidence Intervals = (', Lower_CI,',', Upper_CI, ')' )
```

```
Point Estimate = 0.0014

Margin of Error = 0.00430265272991

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.0014 ś 0.00430265272991
Confidence Intervals = ( -0.00290265272991 , 0.00570265272991 )
```

3 Second Ranking the elements

```
In [40]: df_copy = df1.copy()
         #del df_copy['Dataset']
         #df_ranked = df_copy.rank(ascending=0, axis=1, method='min')
         df_ranked = df_copy.rank(ascending=0, axis=1)
         df_ranked_coumt = df_ranked.copy()
         df_ranked['Dataset'] = df1['Dataset']
         # ranked table
         df_ranked
```

```
Out[40]:     Model  Thom. Sam  OBD  OBS  Magnitude  random                    Dataset
         0     5.0        5.0  5.0  3.0        2.0     1.0    banknote authentication
         1     4.0        4.0  4.0  4.0        1.0     4.0  Blood Tra. Service Centre
```

```
2     5.5        4.0  5.5  2.0        3.0      1.0            Credit Approval
3     3.0        5.0  5.0  5.0        2.0      1.0        Haberman's Survival
4     5.0        5.0  5.0  1.0        2.0      3.0            Liver Disorders
5     5.0        5.0  5.0  3.0        1.0      2.0          MAGIC Gamma Tele.
6     4.5        4.5  4.5  4.5        1.0      2.0          Mammographic Mass
7     5.0        5.0  5.0  1.0        2.0      3.0            MONK's Problems
8     5.0        1.0  5.0  5.0        2.5      2.5        Connectionist Bench
9     5.5        4.0  5.5  2.0        3.0      1.0                  Spambase
10    5.0        3.0  5.0  2.0        1.0      5.0               SPECTF Heart
11    5.0        5.0  5.0  3.0        1.0      2.0        Tic-Tac-Toe Endgame
```

```
In [41]: dfLcun
         dfLcun_copy = dfLcun.copy()
         #del df_copy['Dataset']
         #df_ranked = df_copy.rank(ascending=0, axis=1, method='min')
         dfLcun_ranked = dfLcun_copy.rank(ascending=1, axis=1)
         dfLcun_ranked_coumt = dfLcun_ranked.copy()
         dfLcun_ranked['Layer'] = dfLcun['Layer']
         # ranked table
         dfLcun_ranked.head()
```

```
Out[41]:    Model  TS Prune half the weights Layer
         0    1.0                        2.0    FC
         1    1.0                        2.0  Conv
```

```
In [42]: # old table
         df1.head()
```

```
Out[42]:                      Dataset  Model  Thom. Sam   OBD   OBS  Magnitude  random
         0    banknote authentication   0.01       0.01  0.01  0.02       3.23    5.13
         1  Blood Tra. Service Centre   0.08       0.08  0.08  0.08       0.44    0.08
         2            Credit Approval   0.08       0.78  0.08  8.62       2.55   22.19
         3        Haberman's Survival   0.09       0.08  0.08  0.08       0.63    0.65
         4            Liver Disorders   0.10       0.10  0.10  0.85       0.62    0.15
```

```
In [43]: df_ranked_coumtS = df_ranked_coumt.sum()
         dfLcun_ranked_coumtS = dfLcun_ranked_coumt.sum()
```

```
In [44]: pie_chart = Donut(df_ranked_coumtS, tools=TOOLS )
         pieLcun_chart = Donut(dfLcun_ranked_coumtS, tools=TOOLS )
         print('On classification dataswt')
         show(pie_chart)
         print('On Lecun model')
         show(pieLcun_chart)
```

```
On classification dataswt


On Lecun model
```

```
In [45]: labels = df_ranked_coumtS.index.tolist()
         values =   df_ranked_coumtS.tolist()
         trace=go.Pie(labels=labels,values=values)
         py.iplot([trace])

Out[45]: <plotly.tools.PlotlyDisplay object>

In [46]: labelsLcun = dfLcun_ranked_coumtS.index.tolist()
         valuesLcun =   dfLcun_ranked_coumtS.tolist()
         traceLcun=go.Pie(labels=labelsLcun,values=valuesLcun)
         py.iplot([traceLcun])

Out[46]: <plotly.tools.PlotlyDisplay object>

In [47]: p = Bar(df_ranked, label='Dataset',
                 values = blend('Model', 'Thom. Sam', 'BayTS', 'KLTS' , 'OBD','OBS',
                                'Magnitude',
                                'random',name='Scores', labels_name='Score'),
                group=cat(columns='Score', sort=False),
                title="Compare the performance", legend='bottom_center',
                tools=TOOLS, plot_width=900, plot_height=1600,
                tooltips=[('Score', '@Score'), ('Model', '@Dataset')],
                xlabel='List of datasets', ylabel='Ranked')
         p.title.align = "center"
         #p.yaxis.major_label_orientation = "vertical"
         p.xaxis.major_label_orientation = pi/2
         show(p)

In [48]: p = Bar(df_ranked, label='Dataset',
                 values = blend('Model', 'Thom. Sam', name='Scores', labels_name='Score'),
                group=cat(columns='Score', sort=False),
                title="Compare the performance", legend='bottom_center',
                tools=TOOLS, plot_width=900, plot_height=600,
                tooltips=[('Score', '@Score'), ('Model', '@Dataset')],
                xlabel='List of datasets', ylabel='Ranked')
         p.title.align = "center"
         #p.yaxis.major_label_orientation = "vertical"
         p.xaxis.major_label_orientation = pi/2
         ################################################################################
         show(p)

In [49]: p = Bar(dfLcun_ranked, label='Layer',
                 values = blend('Model', 'TS Prune half the weights',name='Scores', labels_name=
                group=cat(columns='Score', sort=False),
                title="Compare the performance", legend='bottom_center',
                tools=TOOLS, plot_width=900, plot_height=600,
                tooltips=[('Score', '@Score'), ('Model', '@Layer')],
                xlabel='List of Layers', ylabel='Ranked')
         p.title.align = "center"
         #p.yaxis.major_label_orientation = "vertical"
         p.xaxis.major_label_orientation = pi/2
         #################################################################################
         show(p)
```

```
In [50]: df1 = df_ranked.copy()
         df=df1.copy()
         df.set_index('Dataset', inplace=True)
         py.iplot([{
             'x': df.index,
             'y': df[col],
             'name': col
         }  for col in df.columns])
```

```
Out[50]: <plotly.tools.PlotlyDisplay object>
```

```
In [51]: df.iplot(subplots=True, subplot_titles=True, legend=False )
```

```
<IPython.core.display.HTML object>
```

```
In [52]: df.iplot(kind='bar', barmode='stack')
```

```
<IPython.core.display.HTML object>
```

```
In [53]: df.iplot(kind='barh',barmode='stack', bargap=.2)
```

```
<IPython.core.display.HTML object>
```

```
In [54]: df.iplot(kind='box')
```

```
<IPython.core.display.HTML object>
```

3.1 Using Nonparametric tests

I am not sure the data comes from Guassian distribution and less than 30 sample

3.1.1 alternative to paired t-test when data has an ordinary scale or when not

3.1.2 normally distributed

3.2 Start comparining all pruning algorithms

The Kruskal–Wallis test by ranks, Kruskal–Wallis H test (named after William Kruskal and W. Allen Wallis), or One-way ANOVA on ranks is a non-parametric method for testing whether samples originate from the same distribution. It is used for comparing two or more independent samples of equal or different sample sizes. It extends the Mann–Whitney U test when there are more than two groups. The parametric equivalent of the Kruskal-Wallis test is the one-way analysis of variance (ANOVA). A significant Kruskal-Wallis test indicates that at least one sample stochastically dominates one other sample. The test does not identify where this stochastic dominance occurs or for how many pairs of groups stochastic dominance obtains. Dunn's test,or the more powerful but less well known Conover-Iman test would help analyze the specific sample pairs for stochastic dominance in post hoc tests.

Since it is a non-parametric method, the Kruskal–Wallis test does not assume a normal distribution of the residuals, unlike the analogous one-way analysis of variance. If the researcher can make the less stringent assumptions of an identically shaped and scaled distribution for all groups, except for any difference in medians, then the null hypothesis is that the medians of all groups are equal, and the alternative hypothesis is that at least one population median of one group is different from the population median of at least one other group. [Wekipedia]

3.2.1 Compute Kruskal–Wallis test by ranks between pruning methods

```
In [55]: from scipy.stats import mstats
         H, pval = mstats.kruskalwallis(df1['Thom. Sam'], df1['OBD'], df1['OBS'],
                                        df1['Magnitude'], df1['random'])
         print("H-statistic:", H)
         print("P-Value:", pval)
         if pval < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
H-statistic: 32.7883885388
P-Value: 1.31976379369e-06
Reject NULL hypothesis - Significant differences exist between groups.
```

3.2.2 Compute Kruskal–Wallis test by ranks between pruning methods including the model itself

```
In [56]: df_copy = df1.copy()
         del df_copy['Dataset']
         H, pval = mstats.kruskalwallis([df_copy[col] for col in df_copy.columns])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval)))
         if pval < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
H-statistic:        42.5703807161
P-value:        4.51513609877e-08
Reject NULL hypothesis - Significant differences exist between groups.
```

3.2.3 Both ways indicate that the p value is 3.43469461952e-08 which is less than 0.05 then there

3.2.4 is a difference between the methods

3.3 Between our method and other methods separately as both are independent

First method is used if Two Independent Samples,, the population is same, To test both location and shape, and samples greater than 20 In statistics, the Mann–Whitney U test (also called the Mann–Whitney–Wilcoxon (MWW), Wilcoxon rank-sum test, or Wilcoxon–Mann–Whitney test) is a nonparametric test of the null hypothesis that it is equally likely that a randomly selected value from one sample will be less than or greater than a randomly selected value from a second sample.

Unlike the t-test it does not require the assumption of normal distributions. It is nearly as efficient as the t-test on normal distributions. [Wekipedia]

First method is used if Two Independent Samples,, the population is same and To test any kind of sample in the distribution In statistics, the Kolmogorov–Smirnov test (K–S test or KS test) is a nonparametric test of the equality of continuous, one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one-sample K–S test), or to compare two samples (two-sample K–S test). The Kolmogorov–Smirnov statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution, or between the empirical distribution functions of two samples. The null distribution of this statistic is calculated under the null hypothesis that the sample is drawn from the reference distribution (in the one-sample case) or that the samples are drawn from the same distribution (in the two-sample case). In each case, the distributions considered under the null hypothesis are continuous distributions but are otherwise unrestricted. [Wekipedia]

3.3.1 Number of samples less than 20, we will use second method

3.4 Kolmogorov–Smirnov test between TS and other pruning methods.

3.5 Kolmogorov-Smirnov test for goodness of fit.

3.6 Computes the Kolmogorov-Smirnov statistic on 2 samples.

https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.ks_2samp.html

```
In [57]: print('TS vs random Pruning')
         H, pval = stats.ks_2samp(df1['Thom. Sam'], df1['random'])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval)))
         if pval < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
TS vs random Pruning
H-statistic:        0.666666666667
P-value:        0.00459644384608
Reject NULL hypothesis - Significant differences exist between groups.
```

```
In [58]: print('TS vs Optimal Brain Damage')
         H, pval = stats.ks_2samp(df1['Thom. Sam'], df1['OBD'])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval)))
         if pval < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
TS vs Optimal Brain Damage
H-statistic:        0.333333333333
P-value:        0.43330893681
Accept NULL hypothesis - No significant difference between groups.
```

```
In [59]: print('TS vs Optimal Brain Surgeon')
         H, pval = stats.ks_2samp(df1['Thom. Sam'], df1['OBS'])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval)))
         if pval < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
TS vs Optimal Brain Surgeon
H-statistic:        0.5
P-value:        0.0655839639188
Accept NULL hypothesis - No significant difference between groups.
```

```
In [60]: print('TS vs Deep Compression')
         H, pval = stats.ks_2samp(df1['Thom. Sam'], df1['Magnitude'])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval)))
         if pval < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
TS vs Deep Compression
H-statistic:        0.833333333333
P-value:        0.000150731821127
Reject NULL hypothesis - Significant differences exist between groups.
```

```
In [61]: # Get all models pairs
         interstModel = ['Thom. Sam']
         lst = list(df1.columns.values)
         lst.remove('Dataset')
         model_pairs = []

         for m1 in range(len(df1.columns)-2):
             for m2  in range(m1+1,len(df1.columns)-1):
                 model_pairs.append((lst[m1], lst[m2]))

         pvalueList = []
         new_model_pairs = []
         for m1, m2 in model_pairs:
             print('\n',m1, m2)
             pvalue = stats.ks_2samp(df1[m1], df1[m2])
             #print(pvalue[1])
             if (m1 in interstModel or m2 in interstModel):
                 new_model_pairs.append((m1,m2))
                 pvalueList.append(pvalue[1])
             print(pvalue)
```

```
 Model Thom. Sam
Ks_2sampResult(statistic=0.25, pvalue=0.78641716217514468)

 Model OBD
Ks_2sampResult(statistic=0.083333333333333343, pvalue=0.99999999994070876)

 Model OBS
Ks_2sampResult(statistic=0.58333333333333337, pvalue=0.019091732631329447)

 Model Magnitude
Ks_2sampResult(statistic=0.91666666666666663, pvalue=2.0531074831625211e-05)

 Model random
Ks_2sampResult(statistic=0.75, pvalue=0.00091525414760188016)

 Thom. Sam OBD
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)

 Thom. Sam OBS
Ks_2sampResult(statistic=0.5, pvalue=0.065583963918802238)

 Thom. Sam Magnitude
Ks_2sampResult(statistic=0.83333333333333337, pvalue=0.00015073182112711414)

 Thom. Sam random
Ks_2sampResult(statistic=0.66666666666666674, pvalue=0.0045964438460830122)

 OBD OBS
Ks_2sampResult(statistic=0.66666666666666674, pvalue=0.0045964438460830122)

 OBD Magnitude
Ks_2sampResult(statistic=1.0, pvalue=2.3129269928550027e-06)
```

```
 OBD random
Ks_2sampResult(statistic=0.83333333333333337, pvalue=0.00015073182112711414)

 OBS Magnitude
Ks_2sampResult(statistic=0.41666666666666669, pvalue=0.186196839004176)

 OBS random
Ks_2sampResult(statistic=0.24999999999999994, pvalue=0.7864171621751449)

 Magnitude random
Ks_2sampResult(statistic=0.16666666666666674, pvalue=0.99133252540492101)
```

```
In [62]: for pair, p in zip(new_model_pairs, pvalueList):
             if p < 0.05:
                 print('The pvalue between',pair, 'is', p, '< 0.05 then',
                       emoji.emojize('REJECT the NULL Hypothesis :thumbs_up_sign:'))
             else:
                 print('The pvalue between',pair, 'is', p, '> 0.05 then',
                       emoji.emojize('FAIL to REJECT the NULL Hypothesis :thumbs_down_sign:'))
```

```
The pvalue between ('Model', 'Thom. Sam') is 0.786417162175 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('Thom. Sam', 'OBD') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hyp
The pvalue between ('Thom. Sam', 'OBS') is 0.0655839639188 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('Thom. Sam', 'Magnitude') is 0.000150731821127 < 0.05 then REJECT the NULL H
The pvalue between ('Thom. Sam', 'random') is 0.00459644384608 < 0.05 then REJECT the NULL Hypot
```

```
In [63]: matrix_twosample = []
         matrix_twosample.append(['Methods', 'P value', 'Null Hypothesis', 'EMOJI'])
         for pair, p in zip(new_model_pairs, pvalueList):
             if p < 0.05:
                 matrix_twosample.append((pair, p, 'REJECT', emoji.emojize(':thumbs_up_sign:')))
             else:
                 matrix_twosample.append((pair, p, 'ACCEPT (FAIL TO REJECT)', emoji.emojize(':th
         colorscale = [[0, '#4d004c'],[.5, '#f2e5ff'],[1, '#ffffff']]
         #colorscale = [[0, '#272D31'],[.5, '#ffffff'],[1, '#ffffff']]
         #font=['#FCFCFC', '#00EE00', '#008B00', '#004F00', '#660000', '#CD0000', '#FF3030']
         #font=['#FCFCFC', '#00EE00', '#008B00']
         #table.layout.width=250
         twosample_table = FF.create_table(matrix_twosample, index=True, colorscale=colorscale)
         py.iplot(twosample_table)

Out[63]: <plotly.tools.PlotlyDisplay object>
```

4 Conclusion about TS family by doing two side Kolmogorov-Smirnov test

1. TS is better than random Remove of the weights

2. TS is better than Deep compression method
3. There is no clear difference between TS and Optimal Brain Surgeon
4. There is no clear difference between TS and Optimal Brain Damage
5. There is no clear difference between TS, KLTS and BayTS but TS has less computation

4.1 Prune LeCun Model

```
In [64]: print('TS pruned 50 vs Deep Compression')
         H, pval = stats.ks_2samp(dfLcun_ranked['TS Prune half the weights'], dfLcun_ranked['Mod
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H/2),str(pval/2)))
         if pval/2 < 0.05:
             print("Reject NULL hypothesis - Significant differences exist between groups.")
         if pval/2 > 0.05:
             print("Accept NULL hypothesis - No significant difference between groups.")
```

```
TS pruned 50 vs Deep Compression
H-statistic:        0.5
P-value:        0.0485134487976
Reject NULL hypothesis - Significant differences exist between groups.
```

4.1.1 In Lecume even though we prune have of the model, the model generalizw better

5 General Conclusion

TS better than random pruning and deep compression pruning

TS is faster than OBS and OBD as shown from the time consuming

There is no general improve in the model in all cases after prune 20% of the models as the orginal models very small

When the model becomes bigger the pruned based on TS imporove the model's performance like Lecum model

```
In [ ]:
```