Statistical test on Pruning the weights using different MAB

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 UCB"

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 statsmodels.stats.weightstats import  ttest_ind as t_test
        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 = "all.csv"
datafileLeNet = "LecunPruningWeights.csv"
df1 = pd.read_csv(datafile)
dfLcun = pd.read_csv(datafileLeNet)
df1
```

Out[5]:

	Dataset	Model	E.Greedy	WSLS	UCB1	KLUCB	BayUCB \
0	banknote authentication	0.01	0.01	0.04	0.01	0.01	0.01
1	Blood Tra. Service Centre	0.08	0.08	0.20	0.08	0.08	0.08
2	Credit Approval	0.08	0.10	0.08	0.08	0.11	0.11
3	Haberman's Survival	0.09	0.08	0.09	0.08	0.08	0.08
4	Liver Disorders	0.10	0.10	0.11	0.10	0.10	0.10
5	MAGIC Gamma Tele.	0.06	0.10	0.32	0.06	0.06	0.06
6	Mammographic Mass	0.09	0.09	0.09	0.09	0.09	0.09
7	MONK's Problems	0.10	0.12	0.29	0.10	0.10	0.10
8	Connectionist Bench	0.12	0.29	0.73	0.40	0.50	0.50
9	Spambase	0.08	0.10	0.09	0.64	0.64	0.64
10	SPECTF Heart	0.06	0.07	0.60	0.41	0.41	0.41
11	Tic-Tac-Toe Endgame	0.06	0.06	1.58	0.06	0.06	0.06

```
     OBD   OBS  Thom. Sam  Magnitude  random
0   0.01  0.02       0.01       3.23    5.13
1   0.08  0.08       0.08       0.44    0.08
2   0.08  8.62       0.78       2.55   22.19
3   0.08  0.08       0.08       0.63    0.65
4   0.10  0.85       0.10       0.62    0.15
5   0.06  0.12       0.06       2.49    0.43
6   0.09  0.09       0.09       2.59    0.13
7   0.10  5.28       0.10       0.15    0.13
8   0.12  0.12       1.07       0.16    0.16
9   0.08  4.37       0.09       1.67    5.01
10  0.06  0.14       0.08      12.25    0.06
11  0.06  0.07       0.06      21.30   11.92
```

In [6]:
```
dfLcun
```

Out[6]:
```
  Layer   Model  TS Prune half the weights  EG Prune half the weights  \
0    FC  0.9906                      0.993                     0.9908
1  Conv  0.9906                      0.991                     0.9907

   UCB1 Prune half the weights
0                        0.994
1                        0.992
```

In [7]:
```
p = Bar(df1, label='Dataset',
         values = blend('Model', 'E.Greedy', 'WSLS', ''
                        'UCB1', 'BayUCB', 'KLUCB' ,'OBD','OBS','Thom. Sam',
                        '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', 'UCB1 Prune half the weights',
                        'TS Prune half the weights',
                        'EG Prune half the weights',
                        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', '@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=(11,1), 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>

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 [19]: # Perform the ANOVA
         stats.f_oneway(df1['Model'],df1['UCB1'],df1['BayUCB'], df1['KLUCB'] , df1['OBD'],df1['W
                        df1['Thom. Sam'], df1['E.Greedy'],df1['OBS'],df1['Magnitude'],df1['rando

Out[19]: F_onewayResult(statistic=3.1925913247206141, pvalue=0.0011373174675345626)
```

2.1.2 p-value = 0.0011373174675345626 < 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 3.1925913247206141 and a p-value of 0.0011373174675345626, 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 UCB 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 [20]: # Get all models pairs
         interstModel = ['BayUCB', 'UCB1', 'KLUCB', 'E.Greedy', 'Thom. Sam', 'WSLS']
         lst = list(df1.columns.values)
         lst.remove('Dataset')
         model_pairs = []

         for m1 in range(len(df1.columns)-2):
```

```python
            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 E.Greedy
Ttest_indResult(statistic=-1.0863390126158263, pvalue=0.28908909325816412)

 Model WSLS
Ttest_indResult(statistic=-2.1310645376660728, pvalue=0.044503776526580246)

 Model UCB1
Ttest_indResult(statistic=-1.7230408979574796, pvalue=0.098910045643490457)

 Model KLUCB
Ttest_indResult(statistic=-1.8131754322518554, pvalue=0.083472336134944675)

 Model BayUCB
Ttest_indResult(statistic=-1.8131754322518554, pvalue=0.083472336134944675)

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

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

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

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

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

 E.Greedy WSLS
Ttest_indResult(statistic=-1.9387820876814461, pvalue=0.065462107647573597)
```

```
 E.Greedy UCB1
Ttest_indResult(statistic=-1.2718201185394462, pvalue=0.2167176892417968)

 E.Greedy KLUCB
Ttest_indResult(statistic=-1.3836622103155838, pvalue=0.1803415261164272)

 E.Greedy BayUCB
Ttest_indResult(statistic=-1.3836622103155838, pvalue=0.1803415261164272)

 E.Greedy OBD
Ttest_indResult(statistic=1.1281521496355338, pvalue=0.27140894558164996)

 E.Greedy OBS
Ttest_indResult(statistic=-1.888299105876851, pvalue=0.072243959317878442)

 E.Greedy Thom. Sam
Ttest_indResult(statistic=-1.1753054501177005, pvalue=0.25243618558117398)

 E.Greedy Magnitude
Ttest_indResult(statistic=-2.1281556221539026, pvalue=0.044769625143042405)

 E.Greedy random
Ttest_indResult(statistic=-1.9010056030891738, pvalue=0.070481380115548525)

 WSLS UCB1
Ttest_indResult(statistic=1.2534698148026449, pvalue=0.2231926779682383)

 WSLS KLUCB
Ttest_indResult(statistic=1.1653952788136439, pvalue=0.25633950460919164)

 WSLS BayUCB
Ttest_indResult(statistic=1.1653952788136439, pvalue=0.25633950460919164)

 WSLS OBD
Ttest_indResult(statistic=2.1376193962434629, pvalue=0.043909932659192359)

 WSLS OBS
Ttest_indResult(statistic=-1.5638421537150378, pvalue=0.13212617808704541)

 WSLS Thom. Sam
Ttest_indResult(statistic=0.83762097924736978, pvalue=0.41125134560703669)

 WSLS Magnitude
Ttest_indResult(statistic=-1.9863143004376276, pvalue=0.059596000149265083)

 WSLS random
Ttest_indResult(statistic=-1.7692841630786056, pvalue=0.090708752916927926)
```

```
 UCB1 KLUCB
Ttest_indResult(statistic=-0.13184741989650636, pvalue=0.89630336594080373)

 UCB1 BayUCB
Ttest_indResult(statistic=-0.13184741989650636, pvalue=0.89630336594080373)

 UCB1 OBD
Ttest_indResult(statistic=1.7379630047688599, pvalue=0.096196942065215202)

 UCB1 OBS
Ttest_indResult(statistic=-1.79237145166225, pvalue=0.086837648263052875)

 UCB1 Thom. Sam
Ttest_indResult(statistic=-0.36260140096266436, pvalue=0.72036291654799067)

 UCB1 Magnitude
Ttest_indResult(statistic=-2.0859703385742958, pvalue=0.048789062213186053)

 UCB1 random
Ttest_indResult(statistic=-1.861744659668247, pvalue=0.076052383383690136)

 KLUCB BayUCB
Ttest_indResult(statistic=0.0, pvalue=1.0)

 KLUCB OBD
Ttest_indResult(statistic=1.8273188218411842, pvalue=0.081249447699195135)

 KLUCB OBS
Ttest_indResult(statistic=-1.778747839421666, pvalue=0.089104351651629526)

 KLUCB Thom. Sam
Ttest_indResult(statistic=-0.26261794758191148, pvalue=0.79528867847788343)

 KLUCB Magnitude
Ttest_indResult(statistic=-2.0799578877780593, pvalue=0.049387581828979586)

 KLUCB random
Ttest_indResult(statistic=-1.8561469589913955, pvalue=0.076877194167909751)

 BayUCB OBD
Ttest_indResult(statistic=1.8273188218411842, pvalue=0.081249447699195135)

 BayUCB OBS
Ttest_indResult(statistic=-1.778747839421666, pvalue=0.089104351651629526)

 BayUCB Thom. Sam
Ttest_indResult(statistic=-0.26261794758191148, pvalue=0.79528867847788343)
```

```
 BayUCB Magnitude
Ttest_indResult(statistic=-2.0799578877780593, pvalue=0.049387581828979586)

 BayUCB random
Ttest_indResult(statistic=-1.8561469589913955, pvalue=0.076877194167909751)

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

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

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

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

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

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

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

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

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

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

```
In [21]: 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', 'E.Greedy') is 0.289089093258 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('Model', 'WSLS') is 0.0445037765266 < 0.05 then REJECT the NULL Hypothesis
```

```
The pvalue between ('Model', 'UCB1') is 0.0989100456435 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('Model', 'KLUCB') is 0.0834723361349 > 0.05 then FAIL to REJECT the NULL Hyp
The pvalue between ('Model', 'BayUCB') is 0.0834723361349 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('Model', 'Thom. Sam') is 0.168566083503 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('E.Greedy', 'WSLS') is 0.0654621076476 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('E.Greedy', 'UCB1') is 0.216717689242 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('E.Greedy', 'KLUCB') is 0.180341526116 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('E.Greedy', 'BayUCB') is 0.180341526116 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('E.Greedy', 'OBD') is 0.271408945582 > 0.05 then FAIL to REJECT the NULL Hyp
The pvalue between ('E.Greedy', 'OBS') is 0.0722439593179 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('E.Greedy', 'Thom. Sam') is 0.252436185581 > 0.05 then FAIL to REJECT the NU
The pvalue between ('E.Greedy', 'Magnitude') is 0.044769625143 < 0.05 then REJECT the NULL Hypot
The pvalue between ('E.Greedy', 'random') is 0.0704813801155 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('WSLS', 'UCB1') is 0.223192677968 > 0.05 then FAIL to REJECT the NULL Hypoth
The pvalue between ('WSLS', 'KLUCB') is 0.256339504609 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('WSLS', 'BayUCB') is 0.256339504609 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('WSLS', 'OBD') is 0.0439099326592 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('WSLS', 'OBS') is 0.132126178087 > 0.05 then FAIL to REJECT the NULL Hypothe
The pvalue between ('WSLS', 'Thom. Sam') is 0.411251345607 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('WSLS', 'Magnitude') is 0.0595960001493 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('WSLS', 'random') is 0.0907087529169 > 0.05 then FAIL to REJECT the NULL Hyp
The pvalue between ('UCB1', 'KLUCB') is 0.896303365941 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('UCB1', 'BayUCB') is 0.896303365941 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('UCB1', 'OBD') is 0.0961969420652 > 0.05 then FAIL to REJECT the NULL Hypoth
The pvalue between ('UCB1', 'OBS') is 0.0868376482631 > 0.05 then FAIL to REJECT the NULL Hypoth
The pvalue between ('UCB1', 'Thom. Sam') is 0.720362916548 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('UCB1', 'Magnitude') is 0.0487890622132 < 0.05 then REJECT the NULL Hypothes
The pvalue between ('UCB1', 'random') is 0.0760523833837 > 0.05 then FAIL to REJECT the NULL Hyp
The pvalue between ('KLUCB', 'BayUCB') is 1.0 > 0.05 then FAIL to REJECT the NULL Hypothesis
The pvalue between ('KLUCB', 'OBD') is 0.0812494476992 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('KLUCB', 'OBS') is 0.0891043516516 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('KLUCB', 'Thom. Sam') is 0.795288678478 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('KLUCB', 'Magnitude') is 0.049387581829 < 0.05 then REJECT the NULL Hypothes
The pvalue between ('KLUCB', 'random') is 0.0768771941679 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('BayUCB', 'OBD') is 0.0812494476992 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('BayUCB', 'OBS') is 0.0891043516516 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('BayUCB', 'Thom. Sam') is 0.795288678478 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('BayUCB', 'Magnitude') is 0.049387581829 < 0.05 then REJECT the NULL Hypothe
The pvalue between ('BayUCB', 'random') is 0.0768771941679 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('OBD', 'Thom. Sam') is 0.166113919886 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('OBS', 'Thom. Sam') 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 [22]:
```
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[22]: <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 UCB1

In [23]:
```
dd = df1.copy()
dd['diff'] = dd['UCB1'] - 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 = 0.0983333333333

Margin of Error = 0.119724614009

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.0983333333333 ś 0.119724614009
Confidence Intervals = ( -0.0213912806758 , 0.218057947342 )
```

2. Bayesian UCB

```
In [24]: dd = df1.copy()
         dd['diff'] = dd['BayUCB'] - 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 = 0.109166666667

Margin of Error = 0.125578019764

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.109166666667 ś 0.125578019764
Confidence Intervals = ( -0.0164113530974 , 0.234744686431 )
```

3. KLUCB

```
In [25]: dd = df1.copy()
         dd['diff'] = dd['KLUCB'] - 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 = 0.109166666667

Margin of Error = 0.125578019764

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.109166666667 ś 0.125578019764
Confidence Intervals = ( -0.0164113530974 , 0.234744686431 )
```

4. Thompson Sampling

```
In [26]: 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 )
```

5. Epsilon Greedy

```
In [27]: dd = df1.copy()
         dd['diff'] = dd['E.Greedy'] - 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 = 0.0225

Margin of Error = 0.0304756602557

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.0225 ś 0.0304756602557
Confidence Intervals = ( -0.00797566025569 , 0.0529756602557 )
```

6. WSLS

In [28]:
```
dd = df1.copy()
dd['diff'] = dd['WSLS'] - 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 = 0.274166666667

Margin of Error = 0.280726267812

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.274166666667 ś 0.280726267812
Confidence Intervals = ( -0.00655960114521 , 0.554892934479 )
```

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

```
In [30]: # 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 [31]: # make two groups
         data = np.rec.array(lst,
                             dtype = [('Model','|U10'),('Score', '<f2')])
```

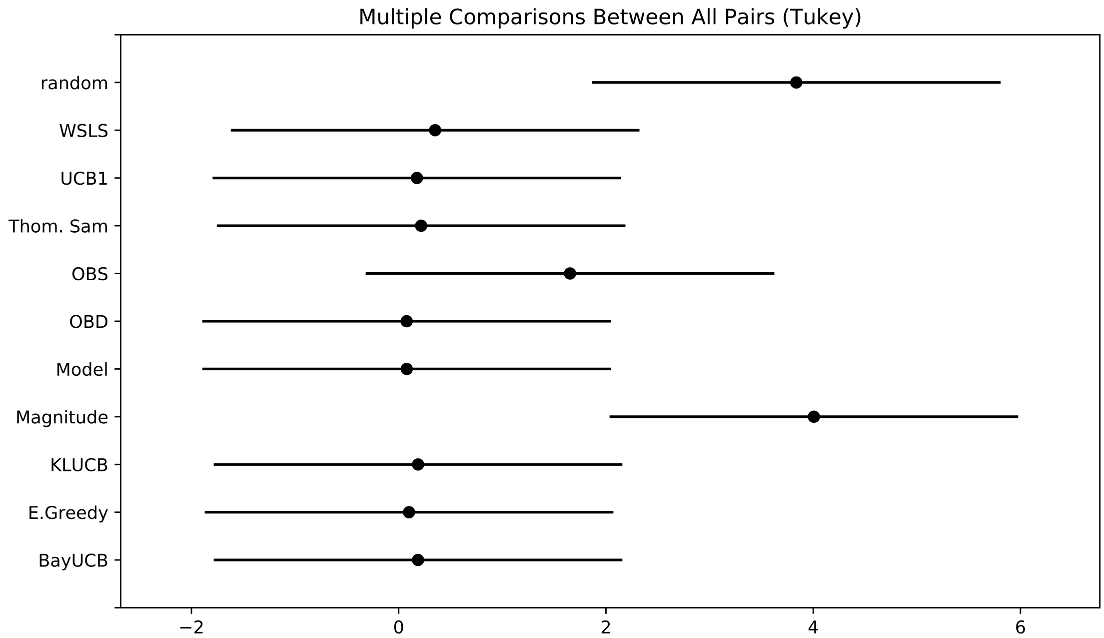
```
In [32]: # perform the test
         mc = MultiComparison(data['Score'], data['Model'])
         result = mc.tukeyhsd()
         print(result)
```

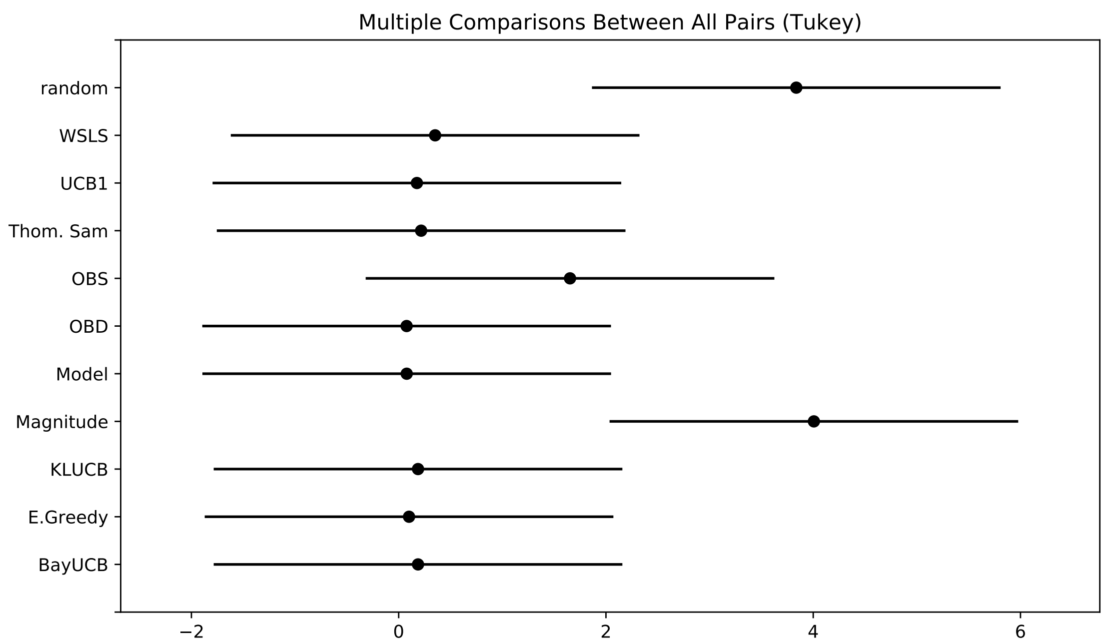
```
Multiple Comparison of Means - Tukey HSD,FWER=0.05
==================================================
  group1    group2  meandiff  lower  upper  reject
--------------------------------------------------
  BayUCB   E.Greedy -0.0867  -4.0301 3.8568 False
  BayUCB    KLUCB     0.0    -3.9435 3.9435 False
  BayUCB  Magnitude  3.8198  -0.1236 7.7633 False
  BayUCB    Model   -0.1092  -4.0526 3.8343 False
  BayUCB     OBD     -0.11   -4.0535 3.8335 False
  BayUCB     OBS     1.4666  -2.4768 5.4101 False
  BayUCB  Thom. Sam   0.03   -3.9134 3.9735 False
  BayUCB     UCB1   -0.0108  -3.9543 3.9326 False
  BayUCB     WSLS    0.165   -3.7784 4.1085 False
  BayUCB    random    3.65   -0.2935 7.5934 False
 E.Greedy   KLUCB    0.0867  -3.8568 4.0301 False
 E.Greedy Magnitude  3.9065  -0.0369  7.85  False
```

```
 E.Greedy   Model   -0.0225   -3.966 3.921  False
 E.Greedy    OBD    -0.0233  -3.9668 3.9201 False
 E.Greedy    OBS     1.5533  -2.3901 5.4968 False
 E.Greedy Thom. Sam  0.1167  -3.8268 4.0601 False
 E.Greedy    UCB1    0.0758  -3.8676 4.0193 False
 E.Greedy    WSLS    0.2517  -3.6918 4.1951 False
 E.Greedy   random   3.7367  -0.2068 7.6801 False
  KLUCB   Magnitude  3.8198  -0.1236 7.7633 False
  KLUCB     Model   -0.1092  -4.0526 3.8343 False
  KLUCB      OBD     -0.11   -4.0535 3.8335 False
  KLUCB      OBS     1.4666  -2.4768 5.4101 False
  KLUCB   Thom. Sam   0.03   -3.9134 3.9735 False
  KLUCB      UCB1   -0.0108  -3.9543 3.9326 False
  KLUCB      WSLS    0.165   -3.7784 4.1085 False
  KLUCB     random    3.65   -0.2935 7.5934 False
Magnitude   Model    -3.929  -7.8725 0.0145 False
Magnitude    OBD    -3.9298  -7.8733 0.0136 False
Magnitude    OBS    -2.3532  -6.2967 1.5903 False
Magnitude Thom. Sam -3.7898  -7.7333 0.1536 False
Magnitude    UCB1   -3.8307  -7.7741 0.1128 False
Magnitude    WSLS   -3.6548  -7.5983 0.2886 False
Magnitude   random  -0.1699  -4.1133 3.7736 False
  Model      OBD    -0.0008  -3.9443 3.9426 False
  Model      OBS     1.5758  -2.3677 5.5193 False
  Model   Thom. Sam  0.1392  -3.8043 4.0826 False
  Model      UCB1    0.0983  -3.8451 4.0418 False
  Model      WSLS    0.2742  -3.6693 4.2176 False
  Model     random   3.7592  -0.1843 7.7026 False
   OBD       OBS     1.5766  -2.3668 5.5201 False
   OBD    Thom. Sam   0.14   -3.8034 4.0835 False
   OBD       UCB1    0.0992  -3.8443 4.0426 False
   OBD       WSLS    0.275   -3.6684 4.2185 False
   OBD      random    3.76   -0.1835 7.7034 False
   OBS    Thom. Sam -1.4366  -5.3801 2.5068 False
   OBS       UCB1   -1.4775  -5.4209 2.466  False
   OBS       WSLS   -1.3016  -5.2451 2.6418 False
   OBS      random   2.1834  -1.7601 6.1268 False
Thom. Sam    UCB1   -0.0408  -3.9843 3.9026 False
Thom. Sam    WSLS    0.135   -3.8084 4.0785 False
Thom. Sam   random    3.62   -0.3235 7.5634 False
   UCB1      WSLS    0.1759  -3.7676 4.1193 False
   UCB1     random   3.6608  -0.2826 7.6043 False
   WSLS     random   3.485   -0.4585 7.4284 False
--------------------------------------------------
```

```
In [33]: result.plot_simultaneous()
```

Out[33]:





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 [34]:
```
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['E.Greedy'], df1['Thom. Sam'],
                    df1['UCB1'], df1['KLUCB'], df1['BayUCB'], 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  0.23940863261242906
This eta square consider to be Large
```

2.3.1 The eta Square is large which means 24% 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 [35]: # 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 [36]: 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 [37]: 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
```

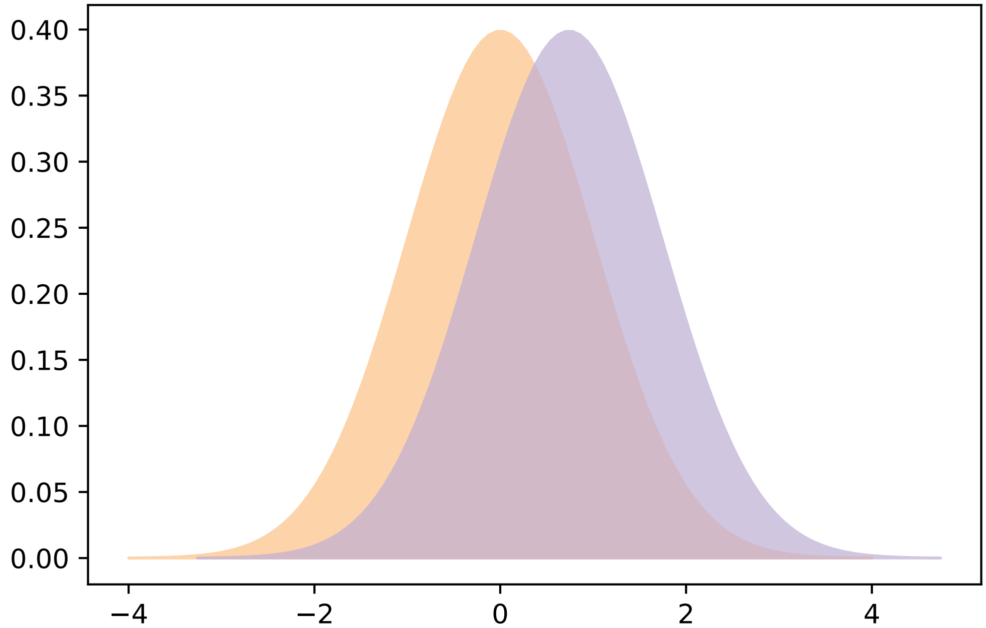
```
In [38]: def plot_pdfs(cohen_d=2):
             control = stats.norm(0, 1)
             treatment = stats.norm(cohen_d, 1)
             xs, ys = eval_pdf(control)
             plt.fill_between(xs, ys, label='control', color=COLOR3, alpha=0.7)

             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)
```

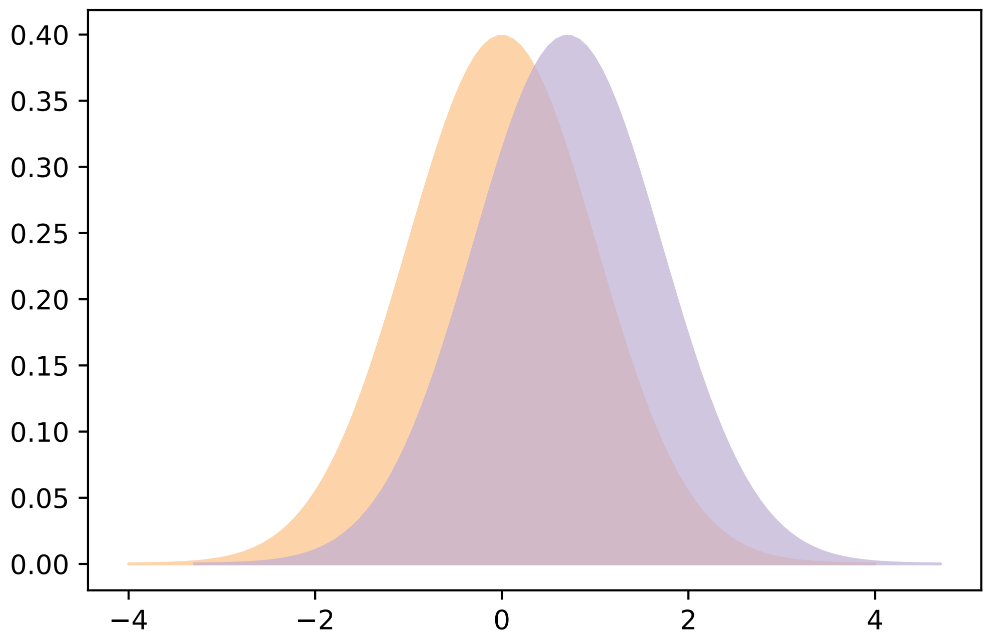
```
In [39]: print('The Cohen d')
         c1 = cohen_d(df1['BayUCB'], df1['Model'])
         if c1 >= 2.505:
             print('The Cohen d between BayUCB and Model is', c1, '>2.505 then',
                   emoji.emojize('honestly significant difference :thumbs_up_sign:'))
         else:
             print('The Cohen d between BayUCB and Model is', c1, '<2.505 then',
                   emoji.emojize('NO honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c1)
```

```
The Cohen d
The Cohen d between BayUCB and Model is 0.740225770528 <2.505 then NO honestly significant diffe
overlap 0.724
superiority 0.685
```



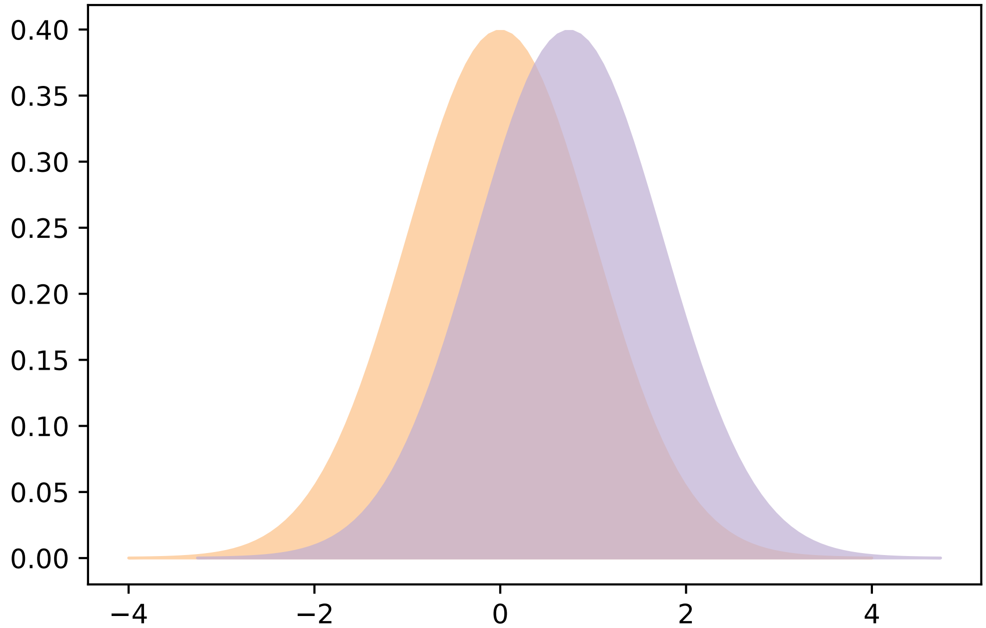

```
In [40]: c2 = cohen_d(df1['UCB1'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between UCB1 and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between UCB1 and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between UCB1 and Model is 0.70342850099 <2.505 then No honestly significant differen
overlap 0.705
superiority 0.707
```



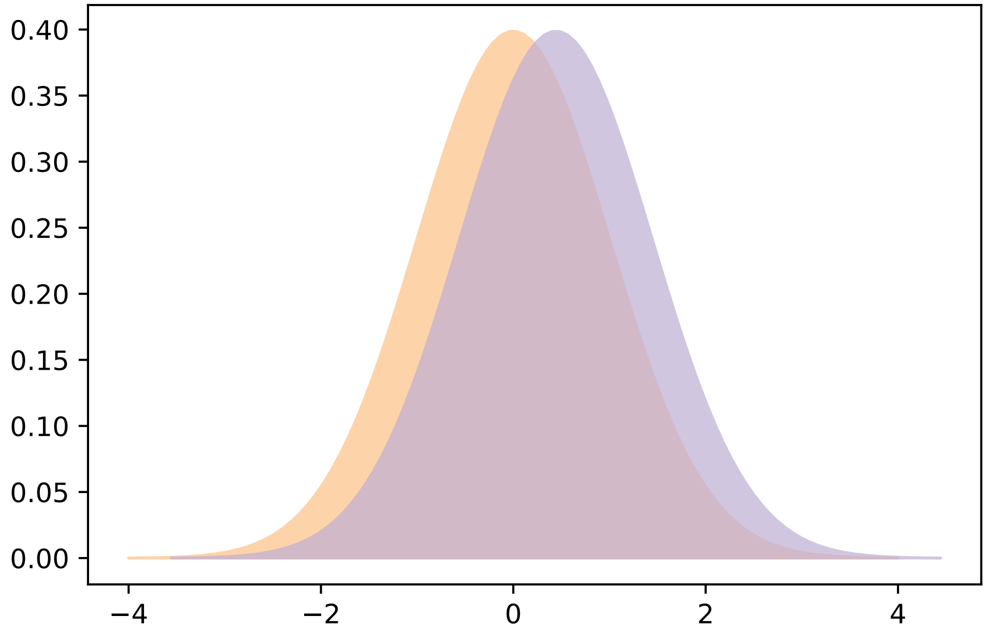

```
In [41]: c2 = cohen_d(df1['KLUCB'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between KLUCB and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between KLUCB and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between KLUCB and Model is 0.740225770528 <2.505 then No honestly significant differ
overlap 0.732
superiority 0.69
```



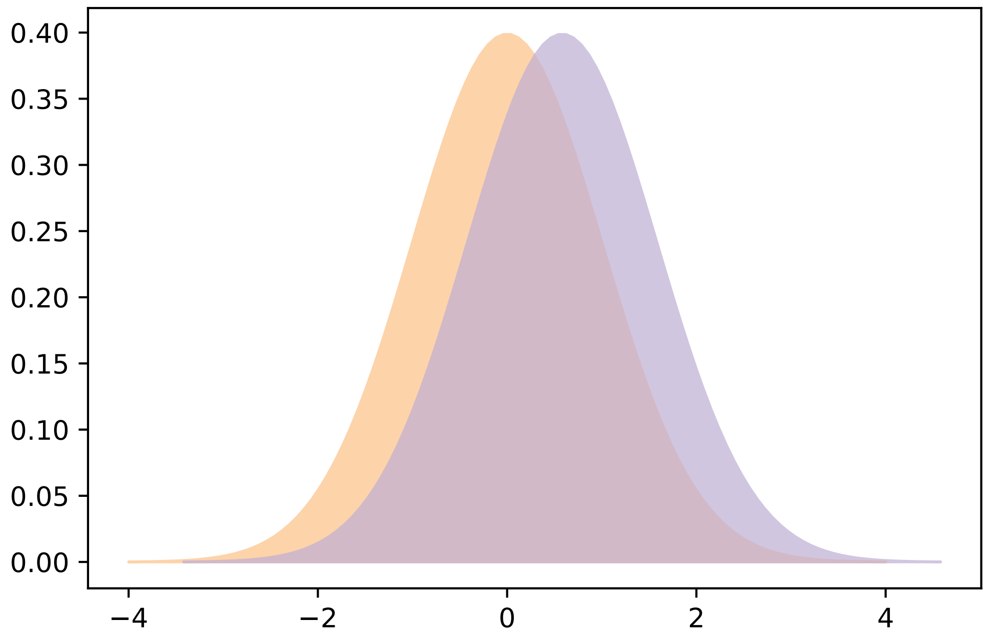

```
In [42]: c2 = cohen_d(df1['E.Greedy'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between E.Greedy and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between E.Greedy and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between E.Greedy and Model is 0.443496044765 <2.505 then No honestly significant dif
overlap 0.793
superiority 0.646
```



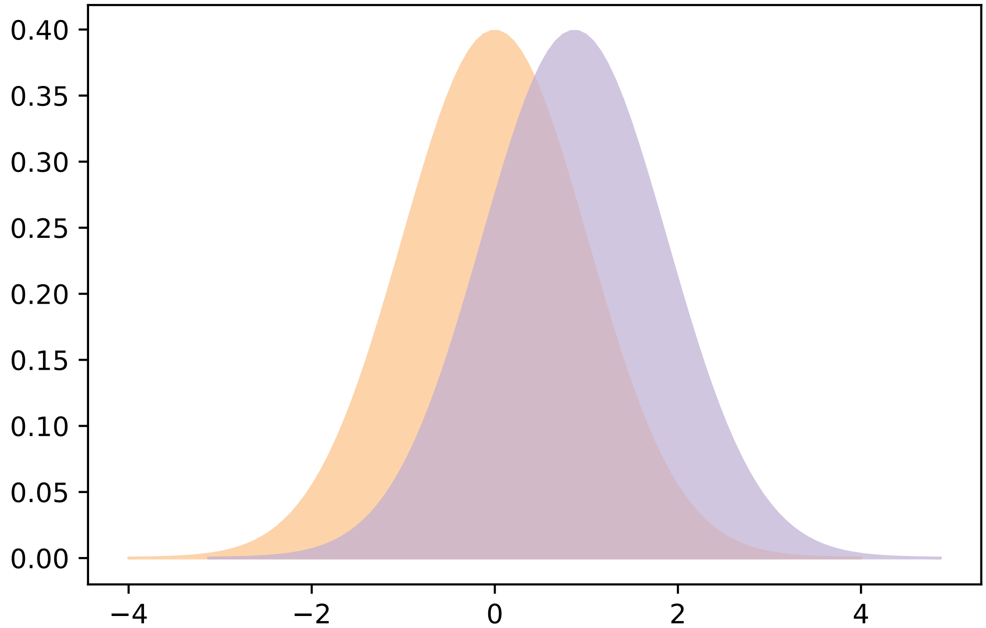

```
In [43]: c2 = cohen_d(df1['Thom. Sam'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between Thom. Sam and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between Thom. Sam and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between Thom. Sam and Model is 0.58121766092 <2.505 then No honestly significant dif
overlap 0.791
superiority 0.636
```




```
In [44]: c2 = cohen_d(df1['WSLS'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between WSLS and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between WSLS and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between WSLS and Model is 0.87000345437 <2.505 then No honestly significant differen
overlap 0.643
superiority 0.757
```



2.5 LeCun Model

```
In [45]: dfLcun
```

```
Out[45]:   Layer   Model  TS Prune half the weights  EG Prune half the weights  \
        0    FC  0.9906                      0.993                     0.9908
        1  Conv  0.9906                      0.991                     0.9907

           UCB1 Prune half the weights
        0                        0.994
        1                        0.992
```

2.5.1 Starting with ANOVA test

```
In [46]: # Perform the ANOVA
         stats.f_oneway(dfLcun['Model'],dfLcun['TS Prune half the weights'],
                        dfLcun['EG Prune half the weights'], dfLcun['UCB1 Prune half the weights
```

```
Out[46]: F_onewayResult(statistic=2.5580524344568021, pvalue=0.19309778199502767)
```

pvalue=0.19309778199502767 > 0.05 there is no significant difference between the methods

2.6 t student test in Lecun Model

```
In [47]: # Get all models pairs
         interstModel = ['TS Prune half the weights', 'EG Prune half the weights',
```

```
                 'UCB1 Prune half the weights']
lst = list(dfLcun.columns.values)
lst.remove('Layer')
model_pairs = []

for m1 in range(len(dfLcun.columns)-2):
    for m2  in range(m1+1,len(dfLcun.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(dfLcun[m1], dfLcun[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 TS Prune half the weights
Ttest_indResult(statistic=-1.3999999999999555, pvalue=0.29647352931856286)

 Model EG Prune half the weights
Ttest_indResult(statistic=-3.0, pvalue=0.095465966266709112)

 Model UCB1 Prune half the weights
Ttest_indResult(statistic=-2.3999999999999333, pvalue=0.13845020965872043)

 TS Prune half the weights EG Prune half the weights
Ttest_indResult(statistic=1.2484404235972781, pvalue=0.33819808102579296)

 TS Prune half the weights UCB1 Prune half the weights
Ttest_indResult(statistic=-0.70710678118660641, pvalue=0.55278640450001226)

 EG Prune half the weights UCB1 Prune half the weights
Ttest_indResult(statistic=-2.2471927624751098, pvalue=0.15365075810028025)
```

```
In [48]: 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', 'TS Prune half the weights') is 0.296473529319 > 0.05 then FAIL to
The pvalue between ('Model', 'EG Prune half the weights') is 0.0954659662667 > 0.05 then FAIL to
The pvalue between ('Model', 'UCB1 Prune half the weights') is 0.138450209659 > 0.05 then FAIL t
The pvalue between ('TS Prune half the weights', 'EG Prune half the weights') is 0.338198081026
The pvalue between ('TS Prune half the weights', 'UCB1 Prune half the weights') is 0.5527864045
The pvalue between ('EG Prune half the weights', 'UCB1 Prune half the weights') is 0.1536507581
```

2.6.1 Margin of Error and Confidence Intervals of Lecun Model

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

```
In [49]: dd = dfLcun.copy()
         dd['diff'] = dd['UCB1 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 = 0.0024

Margin of Error = 0.00430265272991

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.0024 ś 0.00430265272991
Confidence Intervals = ( -0.00190265272991 , 0.00670265272991 )
```

```
In [50]: 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 )
```

```
In [51]: dd = dfLcun.copy()
         dd['diff'] = dd['EG 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 = 0.00015

Margin of Error = 0.000215132636496

Confidence Intervals = point estimate ś Margin of Error
Confidence Intervals =  0.00015 ś 0.000215132636496
```

```
Confidence Intervals = ( -6.51326364956e-05 , 0.000365132636496 )
```

3 Second Ranking the elements

In [52]:
```python
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[52]:

	Model	E.Greedy	WSLS	UCB1	KLUCB	BayUCB	OBD	OBS	Thom. Sam	Magnitude	random	Dataset
0	8.0	8.0	3.0	8.0	8.0	8.0	8.0	4.0	8.0	2.0	1.0	banknote authentication
1	7.0	7.0	2.0	7.0	7.0	7.0	7.0	7.0	7.0	1.0	7.0	Blood Tra. Service Centre
2	9.5	7.0	9.5	9.5	5.5	5.5	9.5	2.0	4.0	3.0	1.0	Credit Approval
3	3.5	8.0	3.5	8.0	8.0	8.0	8.0	8.0	8.0	2.0	1.0	Haberman's Survival
4	8.0	8.0	4.0	8.0	8.0	8.0	8.0	1.0	8.0	2.0	3.0	Liver Disorders
5	8.5	5.0	3.0	8.5	8.5	8.5	8.5	4.0	8.5	1.0	2.0	MAGIC Gamma Tele.
6	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0	1.0	2.0	Mammographic Mass
7	8.5	5.0	2.0	8.5	8.5	8.5	8.5	1.0	8.5	3.0	4.0	MONK's Problems
8	10.0	6.0	2.0	5.0	3.5	3.5	10.0	10.0	1.0	7.5	7.5	Connectionist Bench
9	10.5	7.0	8.5	5.0	5.0	5.0	10.5	2.0	8.5	3.0	1.0	Spambase
10	10.0	8.0	2.0	4.0	4.0	4.0	10.0	6.0	7.0	1.0	10.0	SPECTF Heart
11	8.0	8.0	3.0	8.0	8.0	8.0	8.0	4.0	8.0	1.0	2.0	Tic-Tac-Toe Endgame

In [53]:
```python
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[53]:    Model  TS Prune half the weights  EG Prune half the weights  \
        0    1.0                        3.0                        2.0
        1    1.0                        3.0                        2.0

           UCB1 Prune half the weights Layer
        0                          4.0    FC
        1                          4.0  Conv
```

```
In [54]: # old table
         df1.head()
```

```
Out[54]:                     Dataset  Model  E.Greedy  WSLS  UCB1  KLUCB  BayUCB  \
        0    banknote authentication   0.01      0.01  0.04  0.01   0.01    0.01
        1  Blood Tra. Service Centre   0.08      0.08  0.20  0.08   0.08    0.08
        2            Credit Approval   0.08      0.10  0.08  0.08   0.11    0.11
        3         Haberman's Survival   0.09      0.08  0.09  0.08   0.08    0.08
        4            Liver Disorders   0.10      0.10  0.11  0.10   0.10    0.10

            OBD   OBS  Thom. Sam  Magnitude  random
        0  0.01  0.02       0.01       3.23    5.13
        1  0.08  0.08       0.08       0.44    0.08
        2  0.08  8.62       0.78       2.55   22.19
        3  0.08  0.08       0.08       0.63    0.65
        4  0.10  0.85       0.10       0.62    0.15
```

```
In [55]: df_ranked_coumtS = df_ranked_coumt.sum()
         dfLcun_ranked_coumtS = dfLcun_ranked_coumt.sum()
```

```
In [56]: 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 [57]: labels = df_ranked_coumtS.index.tolist()
         values =   df_ranked_coumtS.tolist()
         trace=go.Pie(labels=labels,values=values)
         py.iplot([trace])
```

Out[57]: <plotly.tools.PlotlyDisplay object>

```
In [58]: labelsLcun = dfLcun_ranked_coumtS.index.tolist()
         valuesLcun =   dfLcun_ranked_coumtS.tolist()
         traceLcun=go.Pie(labels=labelsLcun,values=valuesLcun)
         py.iplot([traceLcun])
```

Out[58]: <plotly.tools.PlotlyDisplay object>

```
In [59]: p = Bar(df_ranked, label='Dataset',
                 values = blend('Model', 'UCB1', 'BayUCB', 'KLUCB'  , 'OBD','OBS',
                               'Magnitude', 'Thom. Sam', 'E.Greedy', 'WSLS',
                               '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 [60]: p = Bar(df_ranked, label='Dataset',
                 values = blend('BayUCB', 'UCB1', 'KLUCB'  , 'Thom. Sam', 'E.Greedy', 'WSLS',
                               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 [61]: p = Bar(df_ranked, label='Dataset',
                 values = blend('Model', 'BayUCB', 'UCB1', 'KLUCB'  , 'Thom. Sam', 'E.Greedy', 'W
                               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 [62]: p = Bar(dfLcun_ranked, label='Layer',
                  values = blend('Model', 'UCB1 Prune half the weights',
                                 'TS Prune half the weights', 'EG Prune half the weights',
                                 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', '@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 [63]: 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[63]: <plotly.tools.PlotlyDisplay object>

In [64]: df.iplot(subplots=True, subplot_titles=True, legend=False )

<IPython.core.display.HTML object>

In [65]: df.iplot(kind='bar', barmode='stack')

<IPython.core.display.HTML object>

In [66]: df.iplot(kind='barh',barmode='stack', bargap=.2)

<IPython.core.display.HTML object>

In [67]: 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 [68]: from scipy.stats import mstats
         H, pval = mstats.kruskalwallis(df1['UCB1'], df1['BayUCB'], df1['KLUCB'], df1['OBD'], df
                                        df1['E.Greedy'], df1['WSLS'], df1['Thom. Sam'],
                                        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: 50.5051020949
P-Value: 8.65500205012e-08
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 [69]: 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:        56.9371760505
P-value:        1.3696376434e-08
Reject NULL hypothesis - Significant differences exist between groups.
```

3.2.3 Both ways indicate that the p value is 1.3696376434e-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 UCB1 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 [70]: print('UCB vs random Pruning')
         H, pval = stats.ks_2samp(df1['UCB1'], 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.")
```

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

```
In [71]: print('UCB vs Optimal Brain Damage')
         H, pval = stats.ks_2samp(df1['UCB1'], 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.")
```

```
UCB vs Optimal Brain Damage
H-statistic:        0.25
P-value:        0.786417162175
Accept NULL hypothesis - No significant difference between groups.
```

```
In [72]: print('UCB vs Optimal Brain Surgeon')
         H, pval = stats.ks_2samp(df1['UCB1'], 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.")
```

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

```
In [73]: print('UCB vs Deep Compression')
         H, pval = stats.ks_2samp(df1['UCB1'], 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.")
```

```
UCB vs Deep Compression
H-statistic:        0.916666666667
P-value:        2.05310748316e-05
Reject NULL hypothesis - Significant differences exist between groups.
```

3.7 Kolmogorov–Smirnov test between KLUCB and other pruning methods.

```
In [74]: print('KLUCB vs random Pruning')
         H, pval = stats.ks_2samp(df1['KLUCB'], 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.")
```

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

```
In [75]: print('KLUCB vs Optimal Brain Damage')
         H, pval = stats.ks_2samp(df1['KLUCB'], 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.")
```

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

```
In [76]: print('KLUCB vs Optimal Brain Surgeon')
         H, pval = stats.ks_2samp(df1['KLUCB'], 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.")
```

```
KLUCB vs Optimal Brain Surgeon
H-statistic:        0.416666666667
P-value:        0.186196839004
Accept NULL hypothesis - No significant difference between groups.
```

```
In [77]: print('KLUCB vs Deep Compression')
         H, pval = stats.ks_2samp(df1['KLUCB'], 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.")
```

```
KLUCB vs Deep Compression
H-statistic:        0.916666666667
P-value:        2.05310748316e-05
Reject NULL hypothesis - Significant differences exist between groups.
```

3.8 Kolmogorov–Smirnov test between BayUCB and other pruning methods.

```
In [78]: print('BayUCB vs random Pruning')
         H, pval = stats.ks_2samp(df1['BayUCB'], 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.")
```

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

```
In [79]: print('BayUCB vs Optimal Brain Damage')
         H, pval = stats.ks_2samp(df1['BayUCB'], 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.")
```

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

```
In [80]: print('BayUCB vs Optimal Brain Surgeon')
         H, pval = stats.ks_2samp(df1['BayUCB'], 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.")
```

```
BayUCB vs Optimal Brain Surgeon
H-statistic:        0.416666666667
P-value:        0.186196839004
Accept NULL hypothesis - No significant difference between groups.
```

```
In [81]: print('BayUCB vs Deep Compression')
         H, pval = stats.ks_2samp(df1['BayUCB'], 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.")
```

```
BayUCB vs Deep Compression
H-statistic:        0.916666666667
P-value:        2.05310748316e-05
Reject NULL hypothesis - Significant differences exist between groups.
```

```
In [82]: # Get all models pairs
         interstModel = ['BayUCB', 'UCB1', 'KLUCB',
                         'E.Greedy', 'WSLS', '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 E.Greedy
Ks_2sampResult(statistic=0.5, pvalue=0.065583963918802238)

 Model WSLS
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)

 Model UCB1
Ks_2sampResult(statistic=0.25, pvalue=0.78641716217514468)

 Model KLUCB
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)
```

```
 Model BayUCB
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)

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

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

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

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

 Model random
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)

 E.Greedy WSLS
Ks_2sampResult(statistic=0.75, pvalue=0.00091525414760188016)

 E.Greedy UCB1
Ks_2sampResult(statistic=0.25, pvalue=0.78641716217514468)

 E.Greedy KLUCB
Ks_2sampResult(statistic=0.16666666666666666, pvalue=0.99133252540492089)

 E.Greedy BayUCB
Ks_2sampResult(statistic=0.16666666666666666, pvalue=0.99133252540492089)

 E.Greedy OBD
Ks_2sampResult(statistic=0.5, pvalue=0.065583963918802238)

 E.Greedy OBS
Ks_2sampResult(statistic=0.58333333333333337, pvalue=0.019091732631329447)

 E.Greedy Thom. Sam
Ks_2sampResult(statistic=0.25, pvalue=0.78641716217514468)

 E.Greedy Magnitude
Ks_2sampResult(statistic=0.91666666666666663, pvalue=2.0531074831625211e-05)

 E.Greedy random
Ks_2sampResult(statistic=0.75, pvalue=0.00091525414760188016)

 WSLS UCB1
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)
```

```
 WSLS KLUCB
Ks_2sampResult(statistic=0.58333333333333337, pvalue=0.019091732631329447)

 WSLS BayUCB
Ks_2sampResult(statistic=0.58333333333333337, pvalue=0.019091732631329447)

 WSLS OBD
Ks_2sampResult(statistic=0.75, pvalue=0.00091525414760188016)

 WSLS OBS
Ks_2sampResult(statistic=0.33333333333333331, pvalue=0.43330893681048638)

 WSLS Thom. Sam
Ks_2sampResult(statistic=0.58333333333333337, pvalue=0.019091732631329447)

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

 WSLS random
Ks_2sampResult(statistic=0.33333333333333331, pvalue=0.43330893681048638)

 UCB1 KLUCB
Ks_2sampResult(statistic=0.083333333333333337, pvalue=0.99999999994070854)

 UCB1 BayUCB
Ks_2sampResult(statistic=0.083333333333333337, pvalue=0.99999999994070854)

 UCB1 OBD
Ks_2sampResult(statistic=0.25, pvalue=0.78641716217514468)

 UCB1 OBS
Ks_2sampResult(statistic=0.5, pvalue=0.065583963918802238)

 UCB1 Thom. Sam
Ks_2sampResult(statistic=0.083333333333333337, pvalue=0.99999999994070854)

 UCB1 Magnitude
Ks_2sampResult(statistic=0.91666666666666663, pvalue=2.0531074831625211e-05)

 UCB1 random
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)

 KLUCB BayUCB
Ks_2sampResult(statistic=0.0, pvalue=1.0)

 KLUCB OBD
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)
```

```
 KLUCB OBS
Ks_2sampResult(statistic=0.41666666666666674, pvalue=0.18619683900417583)

 KLUCB Thom. Sam
Ks_2sampResult(statistic=0.16666666666666666, pvalue=0.99133252540492089)

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

 KLUCB random
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)

 BayUCB OBD
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)

 BayUCB OBS
Ks_2sampResult(statistic=0.41666666666666674, pvalue=0.18619683900417583)

 BayUCB Thom. Sam
Ks_2sampResult(statistic=0.16666666666666666, pvalue=0.99133252540492089)

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

 BayUCB random
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)

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

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

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

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

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

 OBS Magnitude
Ks_2sampResult(statistic=0.58333333333333326, pvalue=0.019091732631329478)

 OBS random
Ks_2sampResult(statistic=0.33333333333333331, pvalue=0.43330893681048638)
```

```
 Thom. Sam Magnitude
Ks_2sampResult(statistic=0.83333333333333326, pvalue=0.00015073182112711414)

 Thom. Sam random
Ks_2sampResult(statistic=0.58333333333333337, pvalue=0.019091732631329447)

 Magnitude random
Ks_2sampResult(statistic=0.25, pvalue=0.78641716217514468)
```

```
In [83]: 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', 'E.Greedy') is 0.0655839639188 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('Model', 'WSLS') is 0.00459644384608 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('Model', 'UCB1') is 0.786417162175 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('Model', 'KLUCB') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('Model', 'BayUCB') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('Model', 'Thom. Sam') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('E.Greedy', 'WSLS') is 0.000915254147602 < 0.05 then REJECT the NULL Hypothe
The pvalue between ('E.Greedy', 'UCB1') is 0.786417162175 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('E.Greedy', 'KLUCB') is 0.991332525405 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('E.Greedy', 'BayUCB') is 0.991332525405 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('E.Greedy', 'OBD') is 0.0655839639188 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('E.Greedy', 'OBS') is 0.0190917326313 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('E.Greedy', 'Thom. Sam') is 0.786417162175 > 0.05 then FAIL to REJECT the NU
The pvalue between ('E.Greedy', 'Magnitude') is 2.05310748316e-05 < 0.05 then REJECT the NULL Hy
The pvalue between ('E.Greedy', 'random') is 0.000915254147602 < 0.05 then REJECT the NULL Hypot
The pvalue between ('WSLS', 'UCB1') is 0.00459644384608 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('WSLS', 'KLUCB') is 0.0190917326313 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('WSLS', 'BayUCB') is 0.0190917326313 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('WSLS', 'OBD') is 0.000915254147602 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('WSLS', 'OBS') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypothes
The pvalue between ('WSLS', 'Thom. Sam') is 0.0190917326313 < 0.05 then REJECT the NULL Hypothes
The pvalue between ('WSLS', 'Magnitude') is 0.186196839004 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('WSLS', 'random') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('UCB1', 'KLUCB') is 0.999999999941 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('UCB1', 'BayUCB') is 0.999999999941 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('UCB1', 'OBD') is 0.786417162175 > 0.05 then FAIL to REJECT the NULL Hypothe
The pvalue between ('UCB1', 'OBS') is 0.0655839639188 > 0.05 then FAIL to REJECT the NULL Hypoth
The pvalue between ('UCB1', 'Thom. Sam') is 0.999999999941 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('UCB1', 'Magnitude') is 2.05310748316e-05 < 0.05 then REJECT the NULL Hypoth
```

```
The pvalue between ('UCB1', 'random') is 0.00459644384608 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('KLUCB', 'BayUCB') is 1.0 > 0.05 then FAIL to REJECT the NULL Hypothesis
The pvalue between ('KLUCB', 'OBD') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypothe
The pvalue between ('KLUCB', 'OBS') is 0.186196839004 > 0.05 then FAIL to REJECT the NULL Hypoth
The pvalue between ('KLUCB', 'Thom. Sam') is 0.991332525405 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('KLUCB', 'Magnitude') is 2.05310748316e-05 < 0.05 then REJECT the NULL Hypot
The pvalue between ('KLUCB', 'random') is 0.00459644384608 < 0.05 then REJECT the NULL Hypothesi
The pvalue between ('BayUCB', 'OBD') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypoth
The pvalue between ('BayUCB', 'OBS') is 0.186196839004 > 0.05 then FAIL to REJECT the NULL Hypot
The pvalue between ('BayUCB', 'Thom. Sam') is 0.991332525405 > 0.05 then FAIL to REJECT the NULL
The pvalue between ('BayUCB', 'Magnitude') is 2.05310748316e-05 < 0.05 then REJECT the NULL Hypo
The pvalue between ('BayUCB', 'random') is 0.00459644384608 < 0.05 then REJECT the NULL Hypothes
The pvalue between ('OBD', 'Thom. Sam') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hyp
The pvalue between ('OBS', 'Thom. Sam') 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 0.0190917326313 < 0.05 then REJECT the NULL Hypoth
```

In [84]:
```
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[84]:
```
<plotly.tools.PlotlyDisplay object>
```

3.9 Prune LeCun Model

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

In [85]:
```
dfLcun_ranked
```

Out[85]:
```
   Model  TS Prune half the weights  EG Prune half the weights  \
0    1.0                        3.0                        2.0
1    1.0                        3.0                        2.0

   UCB1 Prune half the weights Layer
0                          4.0    FC
1                          4.0  Conv
```

In [86]:
```
dfLcun_ranked_copy = dfLcun_ranked.copy()
del dfLcun_ranked_copy['Layer']
H, pval = mstats.kruskalwallis([dfLcun_ranked_copy[col] for col in dfLcun_ranked_copy.c
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:        7.0
P-value:        0.0718977724965
Accept NULL hypothesis - No significant difference between groups.
```

In [87]:
```
dfLcun_ranked
```

Out[87]:
```
   Model  TS Prune half the weights  EG Prune half the weights  \
0    1.0                        3.0                        2.0
1    1.0                        3.0                        2.0

   UCB1 Prune half the weights Layer
0                          4.0    FC
1                          4.0  Conv
```

In [88]:
```
# Get all models pairs
interstModel = ['TS Prune half the weights', 'EG Prune half the weights',
                'UCB1 Prune half the weights']
lst = list(dfLcun_ranked.columns.values)
lst.remove('Layer')
model_pairs = []

for m1 in range(len(dfLcun_ranked.columns)-2):
    for m2  in range(m1+1,len(dfLcun_ranked.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,'<--- VS --->', m2)
    pvalue = stats.ks_2samp(dfLcun_ranked[m1], dfLcun_ranked[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 <--- VS ---> TS Prune half the weights
```

```
Ks_2sampResult(statistic=1.0, pvalue=0.097026897595220707)

 Model <--- VS ---> EG Prune half the weights
Ks_2sampResult(statistic=1.0, pvalue=0.097026897595220707)

 Model <--- VS ---> UCB1 Prune half the weights
Ks_2sampResult(statistic=1.0, pvalue=0.097026897595220707)

 TS Prune half the weights <--- VS ---> EG Prune half the weights
Ks_2sampResult(statistic=1.0, pvalue=0.097026897595220707)

 TS Prune half the weights <--- VS ---> UCB1 Prune half the weights
Ks_2sampResult(statistic=1.0, pvalue=0.097026897595220707)

 EG Prune half the weights <--- VS ---> UCB1 Prune half the weights
Ks_2sampResult(statistic=1.0, pvalue=0.097026897595220707)
```

In [89]:
```
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', 'TS Prune half the weights') is 0.0970268975952 > 0.05 then FAIL to
The pvalue between ('Model', 'EG Prune half the weights') is 0.0970268975952 > 0.05 then FAIL to
The pvalue between ('Model', 'UCB1 Prune half the weights') is 0.0970268975952 > 0.05 then FAIL
The pvalue between ('TS Prune half the weights', 'EG Prune half the weights') is 0.0970268975952
The pvalue between ('TS Prune half the weights', 'UCB1 Prune half the weights') is 0.09702689759
The pvalue between ('EG Prune half the weights', 'UCB1 Prune half the weights') is 0.09702689759
```

In [90]:
```
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[90]: <plotly.tools.PlotlyDisplay object>

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

4 General Conclusion

UCB better than random pruning and deep compression pruning

UCB 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 UCB1 imporove the model's performance like Lecum model