Statistical test on random exploration on the results

February 10, 2017

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

0.0.2 First we will start by "pruning the weights using Epsilon greedy and Win-Stay, Lose-Shift"

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 = "epsilon_wsls.csv"
        datafileLeNet = "LecunPruningWeights.csv"
        df1 = pd.read_csv(datafile)
        dfLcun = pd.read_csv(datafileLeNet)
        df1
```

Out[5]:

	Dataset	Model	E.Greedy	WSLS	OBD	OBS	Magnitude	\
0	Banknote Authentication	0.01	0.01	0.04	0.01	0.02	3.23	
1	Blood Tra. Service Centre	0.08	0.08	0.20	0.08	0.08	0.44	
2	Credit Approval	0.08	0.10	0.08	0.08	8.62	2.55	
3	Haberman's Survival	0.09	0.08	0.09	0.08	0.08	0.63	
4	Liver Disorders	0.10	0.10	0.11	0.10	0.85	0.62	
5	MAGIC Gamma Tele.	0.06	0.10	0.32	0.06	0.12	2.49	
6	Mammographic Mass	0.09	0.09	0.09	0.09	0.09	2.59	
7	MONK's Problems	0.10	0.12	0.29	0.10	5.28	0.15	
8	Connectionist Bench	0.12	0.29	0.73	0.12	0.12	0.16	
9	Spambase	0.08	0.10	0.09	0.08	4.37	1.67	
10	SPECTF Heart	0.06	0.07	0.60	0.06	0.14	12.25	
11	Tic-Tac-Toe Endgame	0.06	0.06	1.58	0.06	0.07	21.30	

```
    Random
0     5.13
1     0.08
2    22.19
3     0.65
4     0.15
5     0.43
6     0.13
7     0.13
8     0.16
9     5.01
10    0.06
11   11.92
```

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

Out[6]:

	Layer	Model	EG Prune half the weights
0	FC	0.9906	0.9908
1	Conv	0.9906	0.9907

In [7]:
```
p = Bar(df1, label='Dataset',
         values = blend('Model', 'E.Greedy', 'WSLS','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', 'EG 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=(7,1), 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('WSLS', 'E.Greedy',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 [20]: p = Bar(df1, label='Dataset',
                 values = blend('Model', 'E.Greedy',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 [21]: # Perform the ANOVA
         stats.f_oneway(df1['Model'],df1['E.Greedy'],df1['WSLS'],df1['OBD'],
                        df1['OBS'],df1['Magnitude'],df1['Random'])
```

```
Out[21]: F_onewayResult(statistic=2.807329980502892, pvalue=0.015954482732966159)
```

2.1.2 p-value = 0.0159 < 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 2.807 and a p-value of 0.01595, 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 epsilon greedy

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 [22]:
```
# Get all models pairs
interstModel = ['WSLS', 'E.Greedy']
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 E.Greedy
Ttest_indResult(statistic=-1.0863390126158263, pvalue=0.28908909325816412)

 Model WSLS
Ttest_indResult(statistic=-2.1310645376660728, pvalue=0.04450377652658024)

 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)

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

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

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

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

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

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

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

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

 WSLS Random
Ttest_indResult(statistic=-1.7692841630786056, pvalue=0.090708752916927926)

 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 [23]: 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 ('E.Greedy', 'WSLS') is 0.0654621076476 > 0.05 then FAIL to REJECT the NULL H
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', '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', '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', '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
```

```
In [24]: 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[24]: <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 epsilon Greedy

```
In [25]: 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 )
```

2. Win-Stay; Lose-Shift

```
In [26]: 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 [27]: df_for_Tukey = df1.copy()
         del df_for_Tukey['Dataset']
```

```
In [28]: # 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 [29]: # make two groups
         data = np.rec.array(lst,
                             dtype = [('Model','|U5'),('Score', '<f2')])
```

```
In [30]: # 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
--------------------------------------------
E.Gre  Magni   3.9065   -0.649 8.4621 False
E.Gre  Model  -0.0225   -4.578 4.5331 False
E.Gre   OBD   -0.0233  -4.5789 4.5322 False
E.Gre   OBS    1.5533  -3.0022 6.1089 False
E.Gre  Rando   3.7367  -0.8189 8.2922 False
E.Gre   WSLS   0.2517  -4.3039 4.8072 False
Magni  Model   -3.929  -8.4846 0.6265 False
Magni   OBD   -3.9298  -8.4854 0.6257 False
```

```
Magni   OBS   -2.3532  -6.9087 2.2023 False
Magni  Rando  -0.1699  -4.7254 4.3857 False
Magni   WSLS  -3.6548  -8.2104 0.9007 False
Model   OBD   -0.0008  -4.5564 4.5547 False
Model   OBS    1.5758  -2.9797 6.1314 False
Model  Rando   3.7592  -0.7964 8.3147 False
Model   WSLS   0.2742  -4.2814 4.8297 False
 OBD    OBS    1.5766  -2.9789 6.1322 False
 OBD   Rando    3.76   -0.7956 8.3155 False
 OBD    WSLS   0.275   -4.2805 4.8306 False
 OBS   Rando   2.1834  -2.3722 6.7389 False
 OBS    WSLS  -1.3016  -5.8572 3.2539 False
Rando   WSLS   -3.485  -8.0405 1.0706 False
--------------------------------------------
```

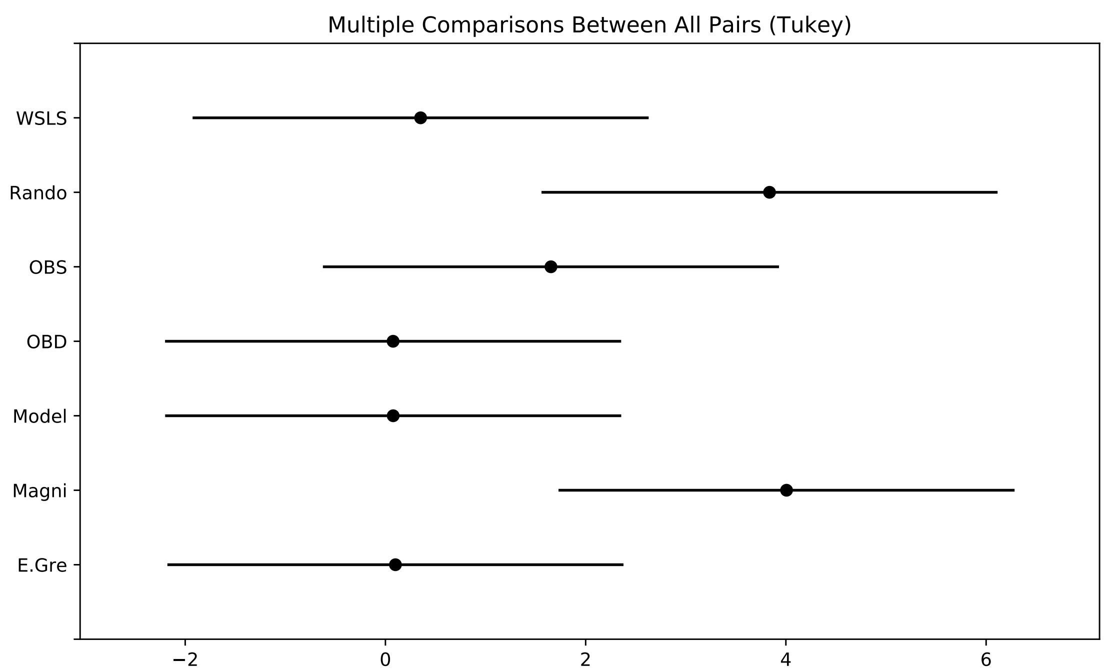
2.2.1 If we looking to Epsilon greedy and Win-stay, Lose-shift we conclude that

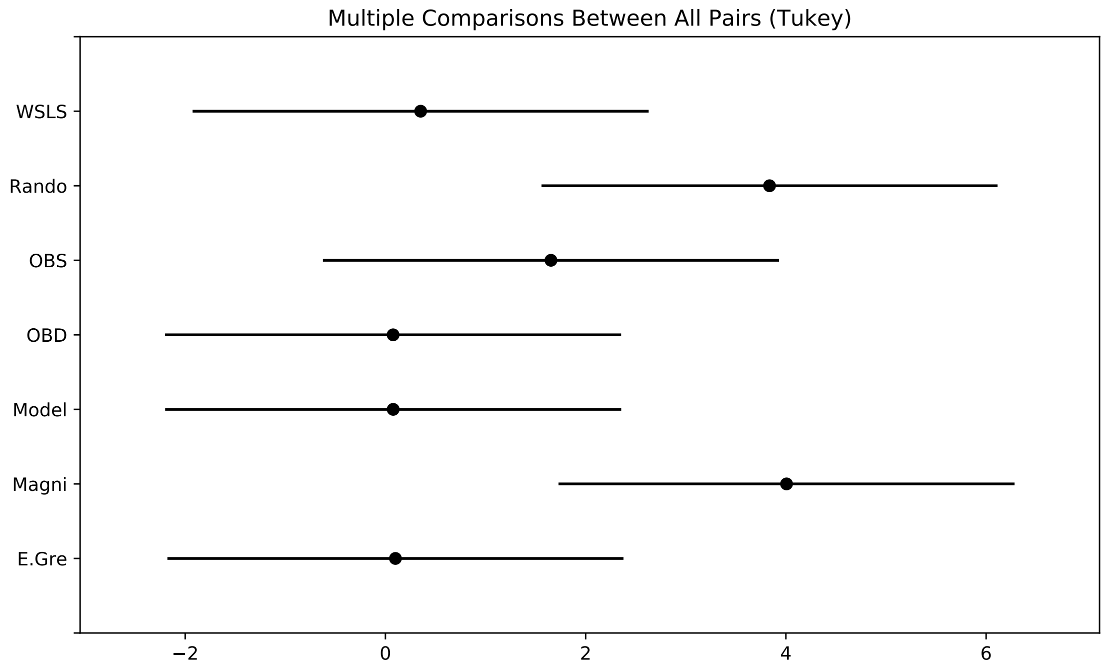
1. We can reject the NULL hypothesis that saying there is no difference between Epsilon greedy and (WSLS, Magnitude, Random prune) and in contrarst of that there is no effidence that Epsilon greedy is difference than the model, OBS and OBD.

2. WSLS is difference than OBD, Epsilon Greedy and the Model but there is no differnece comparining to Magnitude and Deep pruning

In [31]: `result.plot_simultaneous()`

Out[31]:





From the figure we can coclude that Epsilon greedy, Model and OBD, there is conflict on their confernet interval so mostly we can reject the null hypothesis. The same with OBS, Random and WSLS.

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 [32]:
```
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)))
```

```python
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['WSLS'], df1['WSLS'], df1['WSLS'], df1['WSLS'])
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  0.17765634557427643
This eta square consider to be Large
```

2.3.1 The eta Square is large which means 43% 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

Effect size d Reference
Very small 0.01 Sawilowsky, 2009
Small 0.20 Cohen, 1988
Medium 0.50 Cohen, 1988
Large 0.80 Cohen, 1988
Very large 1.20 Sawilowsky, 2009
Huge 2.0 Sawilowsky, 2009
https://en.wikipedia.org/wiki/Effect_size

```
In [33]: # 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 [34]: 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 [35]: 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
```

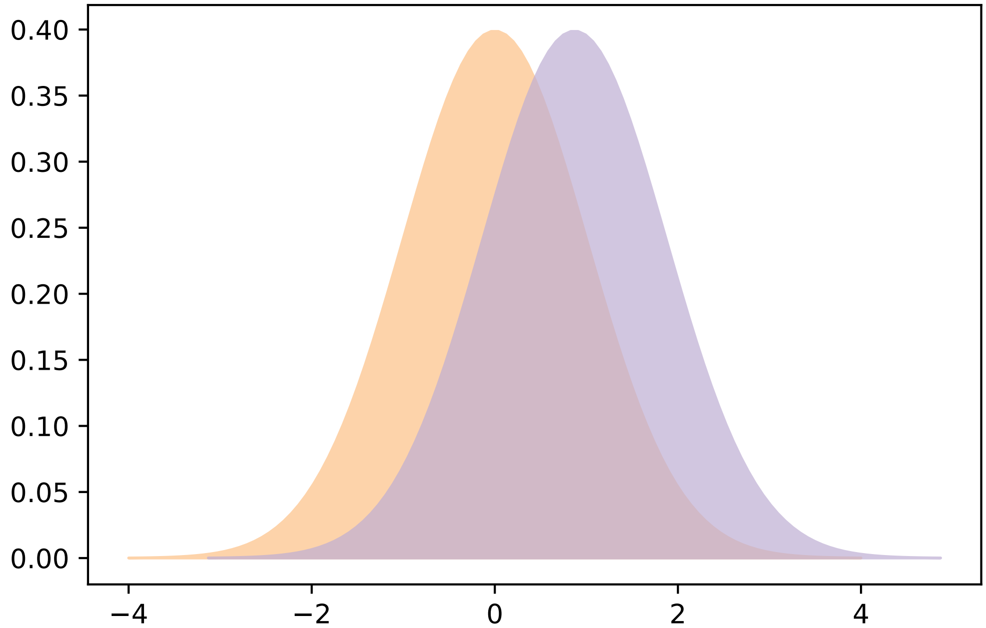
```
In [36]: def plot_pdfs(cohen_d=2):
             control = stats.norm(0, 1)
             treatment = stats.norm(cohen_d, 1)
             xs, ys = eval_pdf(control)
             plt.fill_between(xs, ys, label='control', color=COLOR3, alpha=0.7)

             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)
```

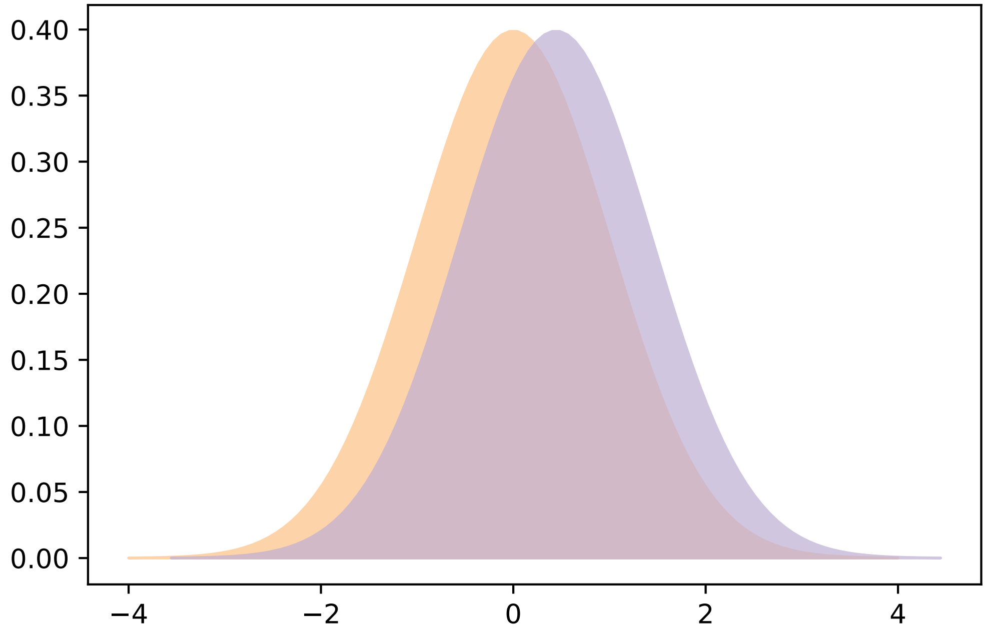
```
In [37]: print('The Cohen d')
         c1 = cohen_d(df1['WSLS'], df1['Model'])
         if c1 >= 2.505:
             print('The Cohen d between WSLs and Model is', c1, '>2.505 then',
                   emoji.emojize('honestly significant difference :thumbs_up_sign:'))
         else:
             print('The Cohen d between WSLs 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 WSLs and Model is 0.87000345437 <2.505 then NO honestly significant differen
overlap 0.669
superiority 0.708
```




```
In [38]: c2 = cohen_d(df1['E.Greedy'], df1['Model'])
         if c2 >= 2.505:
             print('The Cohen d between Epsilon Greedy and Model is', c2, '>2.505 then',
                   emoji.emojize('honestly significant differences :thumbs_up_sign:'))
         else:
             print('The Cohen d between Epsilon Greed and Model is', c2, '<2.505 then',
                   emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
         plot_pdfs(c2)
```

```
The Cohen d between Epsilon Greed and Model is 0.443496044765 <2.505 then No honestly significan
overlap 0.847
superiority 0.624
```



2.5 t student test in Lecun Model

```
In [39]: dfLcun
```

```
Out[39]:   Layer   Model  EG Prune half the weights
         0    FC  0.9906                     0.9908
         1  Conv  0.9906                     0.9907
```

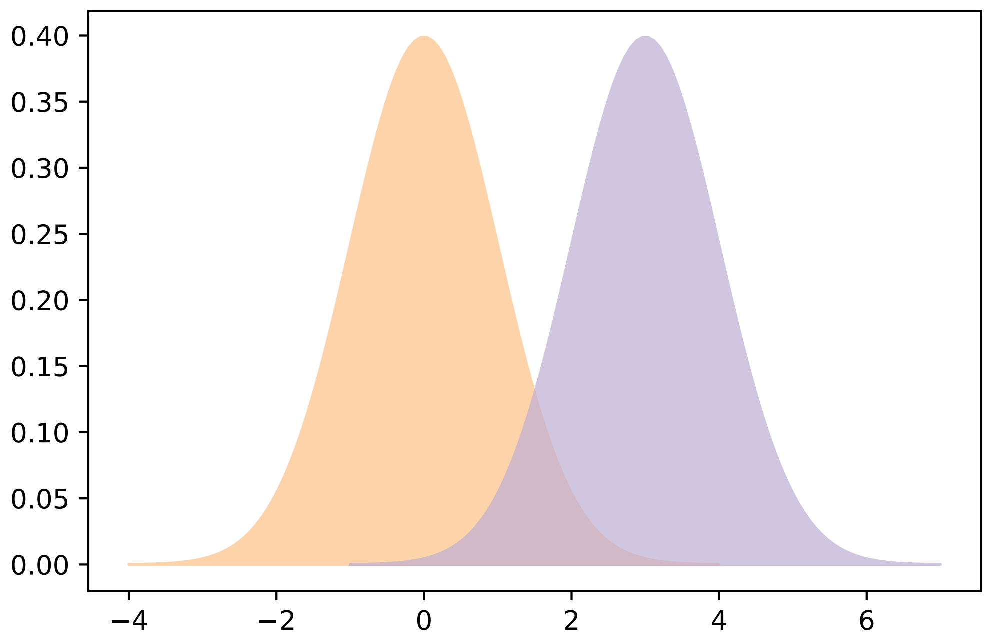
```
In [40]: # Onesided test as we assume there better performcance by using Epsilon Greedy
         print('Epsilon Greedy vs Random Pruning')
         H, pval = stats.ttest_ind(dfLcun['EG Prune half the weights'], dfLcun['Model'])
         print ("H-statistic:\t%s\nP-value:\t%s" % (str(H),str(pval/2)))
         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.")
```

```
Epsilon Greedy vs Random Pruning
H-statistic:        3.0
P-value:        0.0477329831334
Reject NULL hypothesis - Significant differences exist between groups.
```

```
In [41]: cL = cohen_d(dfLcun['EG Prune half the weights'], dfLcun['Model'])
         if cL >= 2.505:
             print('The Cohen d between Epsilon Greedy and Model is', cL, '>2.505 then',
```

```
                emoji.emojize('honestly significant difference :thumbs_up_sign:'))
        else:
            print('The Cohen d between Epsilon Greed and Model is', cL, '<2.505 then',
                  emoji.emojize('No honestly significant difference :thumbs_down_sign:'))
        plot_pdfs(cL)
```

```
The Cohen d between Epsilon Greedy and Model is 3.0 >2.505 then honestly significant difference
overlap 0.133
superiority 0.99
```



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 [42]: 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 Conclustion From t test

1. Epsilon greedy did better and generlize better than the model as the model get bigger
2. There is improve in the performcane from 0.0001 to 0.00019 over the orginal model
3. Then pruning the neural networks causes to improve the performance

4 Second Ranking the elements

In [43]:
```
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[43]:

	Model	E.Greedy	WSLS	OBD	OBS	Magnitude	Random	\
0	6.0	6.0	3.0	6.0	4.0	2.0	1.0	
1	5.0	5.0	2.0	5.0	5.0	1.0	5.0	
2	6.0	4.0	6.0	6.0	2.0	3.0	1.0	
3	3.5	6.0	3.5	6.0	6.0	2.0	1.0	
4	6.0	6.0	4.0	6.0	1.0	2.0	3.0	
5	6.5	5.0	3.0	6.5	4.0	1.0	2.0	
6	5.0	5.0	5.0	5.0	5.0	1.0	2.0	
7	6.5	5.0	2.0	6.5	1.0	3.0	4.0	
8	6.0	2.0	1.0	6.0	6.0	3.5	3.5	

```
9     6.5       4.0    5.0   6.5   2.0        3.0      1.0
10    6.0       4.0    2.0   6.0   3.0        1.0      6.0
11    6.0       6.0    3.0   6.0   4.0        1.0      2.0

                       Dataset
0      Banknote Authentication
1    Blood Tra. Service Centre
2              Credit Approval
3          Haberman's Survival
4              Liver Disorders
5            MAGIC Gamma Tele.
6            Mammographic Mass
7              MONK's Problems
8          Connectionist Bench
9                     Spambase
10                 SPECTF Heart
11         Tic-Tac-Toe Endgame
```

In [44]:
```
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[44]:

	Model	EG Prune half the weights	Layer
0	1.0	2.0	FC
1	1.0	2.0	Conv

In [45]:
```
# old table
df1.head()
```

Out[45]:

	Dataset	Model	E.Greedy	WSLS	OBD	OBS	Magnitude	\
0	Banknote Authentication	0.01	0.01	0.04	0.01	0.02	3.23	
1	Blood Tra. Service Centre	0.08	0.08	0.20	0.08	0.08	0.44	
2	Credit Approval	0.08	0.10	0.08	0.08	8.62	2.55	
3	Haberman's Survival	0.09	0.08	0.09	0.08	0.08	0.63	
4	Liver Disorders	0.10	0.10	0.11	0.10	0.85	0.62	

	Random
0	5.13
1	0.08
2	22.19
3	0.65
4	0.15

In [46]:
```
df_ranked_coumtS = df_ranked_coumt.sum()
dfLcun_ranked_coumtS = dfLcun_ranked_coumt.sum()
```

In [47]:
```
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 [48]:
```
labels = df_ranked_coumtS.index.tolist()
values =   df_ranked_coumtS.tolist()
trace=go.Pie(labels=labels,values=values)
py.iplot([trace])
```

Out[48]:
```
<plotly.tools.PlotlyDisplay object>
```

In [49]:
```
labelsLcun = dfLcun_ranked_coumtS.index.tolist()
valuesLcun =   dfLcun_ranked_coumtS.tolist()
traceLcun=go.Pie(labels=labelsLcun,values=valuesLcun)
py.iplot([traceLcun])
```

Out[49]:
```
<plotly.tools.PlotlyDisplay object>
```

In [50]:
```
p = Bar(df_ranked, label='Dataset',
         values = blend('Model', 'E.Greedy', 'WSLS','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 [51]:
```
p = Bar(df_ranked, label='Dataset',
         values = blend('WSLS', 'E.Greedy',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 [52]: p = Bar(df_ranked, label='Dataset',
                 values = blend('Model', 'E.Greedy',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 [53]: p = Bar(dfLcun_ranked, label='Layer',
                 values = blend('Model', 'EG 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 [54]: 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[54]: <plotly.tools.PlotlyDisplay object>
```

```
In [55]: df.iplot(subplots=True, subplot_titles=True, legend=False )
```

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

In [ ]:

```
In [56]: df.iplot(kind='bar', barmode='stack')
```

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

```
In [57]: df.iplot(kind='barh',barmode='stack', bargap=.2)
```

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

```
In [58]: df.iplot(kind='box')
```

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

4.1 Using Nonparametric tests

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

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

4.1.2 normally distributed

4.2 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]

4.2.1 Compute Kruskal–Wallis test by ranks between pruning methods

```
In [59]: from scipy.stats import mstats
         H, pval = mstats.kruskalwallis(df1['E.Greedy'], df1['WSLS'], 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: 37.4556754247
P-Value: 4.85286479066e-07
Reject NULL hypothesis - Significant differences exist between groups.
```

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

```
In [60]: 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:        48.8601119541
P-value:        7.95238105509e-09
Reject NULL hypothesis - Significant differences exist between groups.
```

4.2.3 Both ways indicate that the p value is 1.71919266103e-06 which is less than 0.05 then there

4.2.4 is a difference between the methods

4.3 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]

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

4.4 Kolmogorov–Smirnov test between Epsilon Greedy and other pruning methods.

4.5 Kolmogorov-Smirnov test for goodness of fit.

4.6 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 [61]: print('Epsilon Greedy vs Random Pruning')
         H, pval = stats.ks_2samp(df1['E.Greedy'], 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.")
```

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

```
In [62]: print('Epsilon Greedy vs Optimal Brain Damage')
         H, pval = stats.ks_2samp(df1['E.Greedy'], 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.")
```

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

```
In [63]: print('Epsilon Greedy vs Optimal Brain Surgeon')
         H, pval = stats.ks_2samp(df1['E.Greedy'], 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.")
```

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

```
In [64]: print('Epsilon Greedy vs Magnitude')
         H, pval = stats.ks_2samp(df1['E.Greedy'], 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.")
```

```
Epsilon Greedy vs Magnitude
H-statistic:        0.916666666667
P-value:        2.05310748316e-05
Reject NULL hypothesis - Significant differences exist between groups.
```

```
In [65]: # Get all models pairs
         interstModel = ['WSLS', 'E.Greedy']
         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=0.41666666666666663, pvalue=0.186196839004176)

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

 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)

 E.Greedy WSLS
Ks_2sampResult(statistic=0.58333333333333326, pvalue=0.019091732631329478)

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

 E.Greedy OBS
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)

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

 E.Greedy Random
Ks_2sampResult(statistic=0.66666666666666663, pvalue=0.0045964438460830287)

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

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

 WSLS Magnitude
Ks_2sampResult(statistic=0.33333333333333337, pvalue=0.43330893681048599)

 WSLS Random
Ks_2sampResult(statistic=0.25000000000000006, pvalue=0.78641716217514468)

 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=0.58333333333333326, pvalue=0.019091732631329478)

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

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

```
In [66]: 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 0.186196839004 > 0.05 then FAIL to REJECT the NULL H
The pvalue between ('Model', 'WSLS') is 0.00459644384608 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('E.Greedy', 'WSLS') is 0.0190917326313 < 0.05 then REJECT the NULL Hypothesi
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 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hypo
The pvalue between ('E.Greedy', 'Magnitude') is 2.05310748316e-05 < 0.05 then REJECT the NULL Hy
The pvalue between ('E.Greedy', 'Random') is 0.00459644384608 < 0.05 then REJECT the NULL Hypoth
The pvalue between ('WSLS', 'OBD') is 0.000915254147602 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('WSLS', 'OBS') is 0.786417162175 > 0.05 then FAIL to REJECT the NULL Hypothe
The pvalue between ('WSLS', 'Magnitude') is 0.43330893681 > 0.05 then FAIL to REJECT the NULL Hy
The pvalue between ('WSLS', 'Random') is 0.786417162175 > 0.05 then FAIL to REJECT the NULL Hypo
```

```
In [67]: matrix_twosample = []
         matrix_twosample.append(['Methods', 'P value', 'Null Hypothesis', 'MOJI'])
         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[67]: <plotly.tools.PlotlyDisplay object>
```

5 Conclusion about Epsilon Greedy by doing two side Kolmogorov-Smirnov test

1. Epsilon greedy is better than Random Remove of the weights
2. Epsilon greedy is better than Magnitude method
3. There is no clear difference between Epsilon greedy and Optimal Brain Surgeon
4. There is no clear difference between Epsilon greedy and Optimal Brain Damage

5.1 Kolmogorov–Smirnov test between Win-Stay; Lose-Shift and other pruning methods.

5.2 Kolmogorov-Smirnov test for goodness of fit.

5.3 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 [68]: print('Win-Stay; Lose-Shift vs Random Pruning')
         H, pval = stats.ks_2samp(df1['WSLS'], 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.")
```

```
Win-Stay; Lose-Shift vs Random Pruning
H-statistic:        0.25
P-value:        0.786417162175
Accept NULL hypothesis - No significant difference between groups.
```

```
In [69]: print('Win-Stay; Lose-Shift vs Optimal Brain Damage')
         H, pval = stats.ks_2samp(df1['WSLS'], 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.")
```

```
Win-Stay; Lose-Shift vs Optimal Brain Damage
H-statistic:        0.75
P-value:        0.000915254147602
Reject NULL hypothesis - Significant differences exist between groups.
```

```
In [70]: print('Win-Stay; Lose-Shift vs Optimal Brain Surgeon')
         H, pval = stats.ks_2samp(df1['WSLS'], 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.")
```

```
Win-Stay; Lose-Shift vs Optimal Brain Surgeon
H-statistic:        0.25
P-value:        0.786417162175
Accept NULL hypothesis - No significant difference between groups.
```

```
In [71]: print('Win-Stay; Lose-Shift vs Magnitude')
         H, pval = stats.ks_2samp(df1['WSLS'], 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.")
```

```
Win-Stay; Lose-Shift vs Magnitude
H-statistic:        0.333333333333
P-value:        0.43330893681
Accept NULL hypothesis - No significant difference between groups.
```

6 Conclusion about Win-Stay; Lose-Shift by doing two side Kolmogorov-Smirnov test

1. Win-Stay; Lose-Shift is not good algorithm for pruning

6.1 Prune LeCun Model

```
In [72]: print('Epsilon Greedy pruned 50 vs Magnitude')
         H, pval = stats.ks_2samp(dfLcun_ranked['EG 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.")
```

```
Epsilon Greedy pruned 50 vs Magnitude
H-statistic:        0.5
P-value:        0.0485134487976
Reject NULL hypothesis - Significant differences exist between groups.
```

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

7 General Conclusion

Epsilon Greedy is better than Win-Stay; Lose-Shift

Epsilon Greedy better than Random pruning and Magnitude pruning

Epsilon greedy and Win-Stay; Lose-Shift 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 epsilon greedy imporove the model's performance like Lecum model