

Accuracy Deep Learning-After Feature Map

April 9, 2017

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

0.0.2 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 plotting

```
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 plotting 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 plotting 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,tap'
        #defaults.width = 1000
        #defaults.height = 800
        output_notebook()

```

2 Statring the test and visulize the data on small model

2.1 Load the data for pruning the weights using random expoloration

```

In [7]: datafile = "./results/results.xlsx"
        df_accuracy = pd.read_excel(datafile)
        df_accuracy

```

```

Out[7]:
  Model_Dataset_Layer  Lecun_MNIST_layer_1  Lecun_MNIST_layer_2 \
0                Model                0.98                0.98
1                UCB1                0.99                0.98
2                 TS                0.99                0.98

  Lecun_MNIST_layer_ALL  ConvNet_cifar10_layer_1  ConvNet_cifar10_layer_2 \
0                0.98                0.81                0.81
1                0.99                0.81                0.81
2                0.99                0.81                0.81

  ConvNet_cifar10_layer_3  ConvNet_cifar10_layer_4 \
0                0.81                0.81
1                0.81                0.81
2                0.81                0.81

  ConvNet_cifar10_layer_ALL  ConvNet_cifar100_layer_1 \
0                0.81                0.43
1                0.82                0.43

```

2	0.82	0.43	
---	------	------	--

	ConvNet_cifar100_layer_2	ConvNet_cifar100_layer_3	\
0	0.43	0.43	
1	0.43	0.43	
2	0.43	0.43	

	ConvNet_cifar100_layer_4	ConvNet_cifar100_layer_ALL	ConvNet_SVHN_layer_1	\
0	0.43	0.43	0.96	
1	0.43	0.44	0.96	
2	0.43	0.44	0.96	

	ConvNet_SVHN_layer_2	ConvNet_SVHN_layer_3	ConvNet_SVHN_layer_4	\
0	0.96	0.96	0.96	
1	0.96	0.96	0.96	
2	0.96	0.96	0.96	

	ConvNet_SVHN_layer_ALL
0	0.96
1	0.97
2	0.96

In [8]: df_accuracy.T

Out [8]:

	0	1	2
Model_Dataset_Layer	Model	UCB1	TS
Lecun_MNIST_layer_1	0.98	0.99	0.99
Lecun_MNIST_layer_2	0.98	0.98	0.98
Lecun_MNIST_layer_ALL	0.98	0.99	0.99
ConvNet_cifar10_layer_1	0.81	0.81	0.81
ConvNet_cifar10_layer_2	0.81	0.81	0.81
ConvNet_cifar10_layer_3	0.81	0.81	0.81
ConvNet_cifar10_layer_4	0.81	0.81	0.81
ConvNet_cifar10_layer_ALL	0.81	0.82	0.82
ConvNet_cifar100_layer_1	0.43	0.43	0.43
ConvNet_cifar100_layer_2	0.43	0.43	0.43
ConvNet_cifar100_layer_3	0.43	0.43	0.43
ConvNet_cifar100_layer_4	0.43	0.43	0.43
ConvNet_cifar100_layer_ALL	0.43	0.44	0.44
ConvNet_SVHN_layer_1	0.96	0.96	0.96
ConvNet_SVHN_layer_2	0.96	0.96	0.96
ConvNet_SVHN_layer_3	0.96	0.96	0.96
ConvNet_SVHN_layer_4	0.96	0.96	0.96
ConvNet_SVHN_layer_ALL	0.96	0.97	0.96

3 Starting with Accuracy

4 First, All methods

4.1 Visulize the Accuracy of all the models and methods

```
In [13]: p = Bar(df_accuracy, label= 'Model_Dataset_Layer',
                values = blend('Lecun_MNIST_layer_1', 'Lecun_MNIST_layer_2', 'Lecun_MNIST_layer_3',
                               'ConvNet_cifar10_layer_1', 'ConvNet_cifar10_layer_2', 'ConvNet_cifar10_layer_3',
                               'ConvNet_cifar10_layer_4', 'ConvNet_cifar10_layer_ALL',
                               'ConvNet_cifar100_layer_1', 'ConvNet_cifar100_layer_2', 'ConvNet_cifar100_layer_3',
                               'ConvNet_cifar100_layer_4', 'ConvNet_cifar100_layer_ALL',
                               'ConvNet_SVHN_layer_1', 'ConvNet_SVHN_layer_2', 'ConvNet_SVHN_layer_3',
                               'ConvNet_SVHN_layer_4', 'ConvNet_SVHN_layer_ALL',
                               name='Scores', labels_name='Score'),
                group=cat(columns='Score', sort=False),
                title="Compare the performance", legend='bottom_center',
                tools=TOOLS, plot_width=800, plot_height=1300,
                tooltips=[('Score', '@Score'), ('Model', '@Model_Dataset_Layer')],
                xlabel='List of Models', ylabel='Score')
p.title.align = "center"
#p.yaxis.major_label_orientation = "vertical"
p.xaxis.major_label_orientation = pi/2
show(p)
```

```
In [15]: df=df_accuracy.copy()
df.set_index('Model_Dataset_Layer', inplace=True)
py.iplot([{'x': df.index,
            'y': df[col],
            'name': col}
          for col in df.columns])
```

```
Out[15]: <plotly.tools.PlotlyDisplay object>
```

```
In [19]: df.T.iplot(kind='barh',barmode='stack', bargap=.2)
```

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

```
In [21]: df.T.iplot(kind='box')
```

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

4.1.1 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 [22]: df.T.columns
```

```
Out[22]: Index(['Model', 'UCB1', 'TS'], dtype='object', name='Model_Dataset_Layer')
```

```
In [23]: # Perform the ANOVA
df1 = df.T
stats.f_oneway(df1['Model'], df1['UCB1'], df1['TS'])
```

```
Out[23]: F_onewayResult(statistic=0.00073353401070507353, pvalue=0.99926674550209138)
```

4.1.2 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 [36]: # Get all models pairs
interstModel = ['Model', 'UCB1', 'TS']
lst = list(df1.columns.values)
#lst.remove('Methods')
model_pairs = []

for m1 in range(len(df1.columns)-1):
    for m2 in range(m1+1, len(df1.columns)):
        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 UCB1

```
Ttest_indResult(statistic=-3.3891719470228243, pvalue=0.0017884109954953058)
```

Model TS

```
Ttest_indResult(statistic=-2.8284271247461912, pvalue=0.0077883740345155066)
```

UCB1 TS

```
Ttest_indResult(statistic=0.89973541084243414, pvalue=0.37458967660855147)
```

```
In [38]: 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', 'UCB1') is 0.0017884109955 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('Model', 'TS') is 0.00778837403452 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('UCB1', 'TS') is 0.374589676609 > 0.05 then FAIL to REJECT the NULL Hypothesis

```

In [39]: 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(':thumbs_down_sign:')))
        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 [39]: <plotly.tools.PlotlyDisplay object>

4.1.3 Margin of Error and Confidence Intervals

margin of error = Tcritical*SE

Confidence Intervals = point estimate \pm Margin of Error

4.2 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. [Wikipedia]

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

5 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))
```

6 make two groups

```
data = np.rec.array(lst, dtype=[('Model','<U5'),('Score','<f2')])
```

7 perform the test

```
mc = MultiComparison(data['Score'], data['Model']) result = mc.tukeyhsd() print(result)
result.plot_simultaneous()
```

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

7.1 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>

```
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['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')

```

7.1.1 The eta Square is large which means 43% the difference based on the variation in the group mean

7.2 Cohen's d

if any two samples have a absolute 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

8 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, ddof=1) ** 2) / dof) 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))

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

def overlap_superiority(control, treatment, n=1000): control_sample = control.rvs(n) treat-
ment_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

```



```

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)

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)
c2 = cohen_d(df1['E.Greedy'], df1['Model']) if c2 >= 2.505: print("The Cohen d between Ep-
silon 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)

```

9 Second Ranking the elements

```
In [27]: dfbuck = df1.copy()
```

```
In [28]: 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=1, axis=1)
df_ranked_count = df_ranked.copy()
#df_ranked['Methods'] = df1['Methods']
# ranked table
df_ranked
```

```
Out[28]:
```

Model_Dataset_Layer	Model	UCB1	TS
Lecun_MNIST_layer_1	1.0	2.5	2.5
Lecun_MNIST_layer_2	2.0	2.0	2.0
Lecun_MNIST_layer_ALL	1.0	2.5	2.5
ConvNet_cifar10_layer_1	2.0	2.0	2.0
ConvNet_cifar10_layer_2	2.0	2.0	2.0
ConvNet_cifar10_layer_3	2.0	2.0	2.0
ConvNet_cifar10_layer_4	2.0	2.0	2.0
ConvNet_cifar10_layer_ALL	1.0	2.5	2.5
ConvNet_cifar100_layer_1	2.0	2.0	2.0
ConvNet_cifar100_layer_2	2.0	2.0	2.0
ConvNet_cifar100_layer_3	2.0	2.0	2.0
ConvNet_cifar100_layer_4	2.0	2.0	2.0
ConvNet_cifar100_layer_ALL	1.0	2.5	2.5
ConvNet_SVHN_layer_1	2.0	2.0	2.0
ConvNet_SVHN_layer_2	2.0	2.0	2.0

ConvNet_SVHN_layer_3	2.0	2.0	2.0
ConvNet_SVHN_layer_4	2.0	2.0	2.0
ConvNet_SVHN_layer_ALL	1.5	3.0	1.5

```
In [29]: df_ranked_countS = df_ranked_count.sum()
pie_chart = Donut(df_ranked_countS, tools=TOOLS )
print('On classification dataset')
show(pie_chart)
```

On classification dataset

```
In [30]: labels = df_ranked_countS.index.tolist()
values = df_ranked_countS.tolist()
trace=go.Pie(labels=labels,values=values)
py.iplot([trace])
```

Out[30]: <plotly.tools.PlotlyDisplay object>

```
In [31]: df1 = df_ranked.copy()
df=df1.copy()
py.iplot([
    'x': df.index,
    'y': df[col],
    'name': col
} for col in df.columns])
```

Out[31]: <plotly.tools.PlotlyDisplay object>

```
In [32]: df.iplot(subplots=True, subplot_titles=True, legend=False )
```

<IPython.core.display.HTML object>

```
In [33]: df.iplot(kind='bar', barmode='stack')
```

<IPython.core.display.HTML object>

```
In [34]: df.iplot(kind='barh',barmode='stack', bargap=.2)
```

<IPython.core.display.HTML object>

```
In [35]: df.iplot(kind='box')
```

<IPython.core.display.HTML object>

9.1 Using Nonparametric tests

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

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

9.1.2 normally distributed

9.2 Start comparing 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. [Wikipedia]

9.2.1 Compute Kruskal–Wallis test by ranks between All methods

```
In [40]: from scipy.stats import mstats
         H, pval = mstats.kruskalwallis(df1['Model'], df1['UCB1'], df1['TS'])

         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: 11.3031536149

P-Value: 0.00351197469885

Reject NULL hypothesis - Significant differences exist between groups.

9.3 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. [Wikipedia]

Second 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]

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

```
In [41]: # Get all models pairs
interstModel = ['Model', 'UCB1', 'TS']
lst = list(df1.columns.values)
#lst.remove('Methods')
model_pairs = []

for m1 in range(len(df1.columns)-1):
    for m2 in range(m1+1, len(df1.columns)):
        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 UCB1

Ttest_indResult(statistic=-3.3891719470228243, pvalue=0.0017884109954953058)

Model TS

Ttest_indResult(statistic=-2.8284271247461912, pvalue=0.0077883740345155066)

UCB1 TS

Ttest_indResult(statistic=0.89973541084243414, pvalue=0.37458967660855147)

```
In [42]: 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', 'UCB1') is 0.0017884109955 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('Model', 'TS') is 0.00778837403452 < 0.05 then REJECT the NULL Hypothesis
The pvalue between ('UCB1', 'TS') is 0.374589676609 > 0.05 then FAIL to REJECT the NULL Hypothesis

```

In [43]: 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(':thumbs_down_sign:')))
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[43]: <plotly.tools.PlotlyDisplay object>

10 Second paper UCB

```

In [44]: df1 = dfbuck.copy()
df_copy = df1.copy()
df_ranked = df_copy.rank(ascending=1, axis=1)
del (df_ranked['TS'])
df_ranked = df_ranked.rank(ascending=1, axis=1)
df_ranked_count = df_ranked.copy()
df_ranked

```

```

Out[44]: Model_Dataset_Layer      Model  UCB1
Lecun_MNIST_layer_1              1.0    2.0
Lecun_MNIST_layer_2              1.5    1.5
Lecun_MNIST_layer_ALL            1.0    2.0
ConvNet_cifar10_layer_1          1.5    1.5
ConvNet_cifar10_layer_2          1.5    1.5
ConvNet_cifar10_layer_3          1.5    1.5
ConvNet_cifar10_layer_4          1.5    1.5
ConvNet_cifar10_layer_ALL        1.0    2.0

```

ConvNet_cifar100_layer_1	1.5	1.5
ConvNet_cifar100_layer_2	1.5	1.5
ConvNet_cifar100_layer_3	1.5	1.5
ConvNet_cifar100_layer_4	1.5	1.5
ConvNet_cifar100_layer_ALL	1.0	2.0
ConvNet_SVHN_layer_1	1.5	1.5
ConvNet_SVHN_layer_2	1.5	1.5
ConvNet_SVHN_layer_3	1.5	1.5
ConvNet_SVHN_layer_4	1.5	1.5
ConvNet_SVHN_layer_ALL	1.0	2.0

```
In [45]: df_ranked_countS = df_ranked_count.sum()
pie_chart = Donut(df_ranked_countS, tools=TOOLS )
print('On classification dataset')
show(pie_chart)
```

On classification dataset

```
In [49]: labels = df_ranked_countS.index.tolist()
values = df_ranked_countS.tolist()
trace=go.Pie(labels=labels,values=values)
py.iplot([trace])
```

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

```
In [46]: df1 = df_ranked.copy()
df=df1.copy()
py.iplot([
    'x': df.index,
    'y': df[col],
    'name': col
} for col in df.columns])
```

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

```
In [47]: df.iplot(subplots=True, subplot_titles=True, legend=False )

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

```
In [48]: df.iplot(kind='bar', barmode='stack')

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

```
In [49]: df.iplot(kind='barh', barmode='stack', bargap=.2)

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

10.1 Using Nonparametric tests

10.1.1 Compute Kruskal–Wallis test by ranks between pruning methods with the model (No need as only 2 methods)

```
from scipy.stats import mstats
H, pval = mstats.kruskalwallis(df1['NN'], df1['UCB1'])
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.")
```

10.1.2 Kolmogorov–Smirnov test

```
In [50]: # Get all models pairs
interstModel = ['Model', 'UCB1']
lst = list(df1.columns.values)
#lst.remove('Methods')
model_pairs = []

for m1 in range(len(df1.columns)-1):
    for m2 in range(m1+1, len(df1.columns)):
        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 UCB1

Ttest_indResult(statistic=-3.616202853397894, pvalue=0.00095787665604187575)

```
In [51]: 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', 'UCB1') is 0.000957876656042 < 0.05 then REJECT the NULL Hypothesis

```
In [52]: 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(':thumbs_down_sign:')))
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[52]: <plotly.tools.PlotlyDisplay object>
```

11 Third paper Thompson Sampling

```
In [53]: df1 = dfbuck.copy()
df_copy = df1.copy()
df_ranked = df_copy.rank(ascending=1, axis=1)
del (df_ranked['UCB1'])
df_ranked = df_ranked.rank(ascending=1, axis=1)
df_ranked_count = df_ranked.copy()
# df_ranked

In [54]: df_ranked_countS = df_ranked_count.sum()
pie_chart = Donut(df_ranked_countS, tools=TOOLS )
print('On classification dataset')
show(pie_chart)
```

On classification dataset

```
In [60]: labels = df_ranked_countS.index.tolist()
values = df_ranked_countS.tolist()
trace=go.Pie(labels=labels, values=values)
py.iplot([trace])
```

Out[60]: <plotly.tools.PlotlyDisplay object>

```
In [55]: df1 = df_ranked.copy()
df=df1.copy()
py.iplot([
    'x': df.index,
    'y': df[col],
    'name': col
} for col in df.columns])
```

Out[55]: <plotly.tools.PlotlyDisplay object>

11.1 Using Nonparametric tests

11.1.1 Compute Kruskal–Wallis test by ranks between All methods (No need because two elements)

```
from scipy.stats import mstats
H, pval = mstats.kruskalwallis(df1['NN'], df1['Tomp. Sampling'])
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.")
```

11.1.2 Kolmogorov–Smirnov test

```
In [56]: df1.head()
```

```
Out [56]: Model_Dataset_Layer      Model    TS
Lecun_MNIST_layer_1                1.0    2.0
Lecun_MNIST_layer_2                1.5    1.5
Lecun_MNIST_layer_ALL              1.0    2.0
ConvNet_cifar10_layer_1            1.5    1.5
ConvNet_cifar10_layer_2            1.5    1.5
```

```
In [57]: # Get all models pairs
interstModel = ['Model', 'TS']
lst = list(df1.columns.values)
#lst.remove('Methods')
model_pairs = []

for m1 in range(len(df1.columns)-1):
    for m2 in range(m1+1, len(df1.columns)):
        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 TS

```
Ttest_indResult(statistic=-3.1167748898959196, pvalue=0.0037058574803140831)
```

```
In [58]: 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') is 0.00370585748031 < 0.05 then REJECT the NULL Hypothesis

```

In [59]: 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(':thumbs_down_sign:')))
        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[59]: <plotly.tools.PlotlyDisplay object>