showcase_figures

March 6, 2016

1 Showcase figures

In this showcase we combine several tools to recreate the figures used for the thesis. For the creation of figure 5, refer to showcase_xgboost.
Actual *saving* of figures is commented out.

```
In [1]: import kaggleData as kD
        import toolbox as tb
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.patches as mpatches
        import matplotlib
        import os
```

We use /plots/ as main saving directory.

```
In [2]: scriptFolderPath = os.path.dirname(os.getcwd())
        mainFolderPath = os.path.dirname(scriptFolderPath)
        thesisImagePath = (mainFolderPath + "/docs/latex/thesis/src/images/")
        plotPath = (mainFolderPath + "/plots/")
        scatterPath = (plotPath + "/scatter/")
        histPath = (plotPath + "/hist/")
```

We import the Kaggle data.

```
In [3]: csv_data,csv_header = kD.csvToArray()
        train_data,train_header,test_data,test_header=kD.getOriginalKaggleSets(csv_data,csv_header)
        sol_data,sol_header = kD.getSolutionKey(csv_data,csv_header)
        test_events = kD._extractFeature("EventId",test_data,csv_header).astype(float)

In [4]: train_all = train_data[:,1:-2].astype(float)
        train_labels = kD.translateLabels(train_data[:,-1],["Label"]).astype(float)
        train_weights = train_data[:,-2].astype(float)
        test_all = test_data[:,1:].astype(float)
        header_all = test_header[1:]
```

1.1 Figures 1 and 2

We dont want to avoid outliers in the scatter plots and histograms. We do so by adjusting the axis' limits.

```
In [5]: def scaleAxis(data,cutPercent):

            sData = sorted(data)
            l = len(sData)
```

```
            cutPoint = int(cutPercent*l/100)
            axisMax = sData[l - cutPoint]
            axisMin = sData[cutPoint]
            #cut missing values
            if axisMin == -999.0:
                for x in sData:
                    if x > axisMin:
                        axisMin = x
                        break
            return axisMin,axisMax
```

We want background and signals to be colored equally for all plotting, we force this with following method.

```
In [6]: def generateLabelsColors(labels):
            colors = []
            for i in range(0,len(labels)):
                label = labels[i]
                if label == 1:
                    labelColor = "b" ##signal = blue
                elif label == 0:
                    labelColor = "r" ##background = red
                else:
                    print("ERROR in Labels!")
                colors.append(labelColor)
            return colors
```

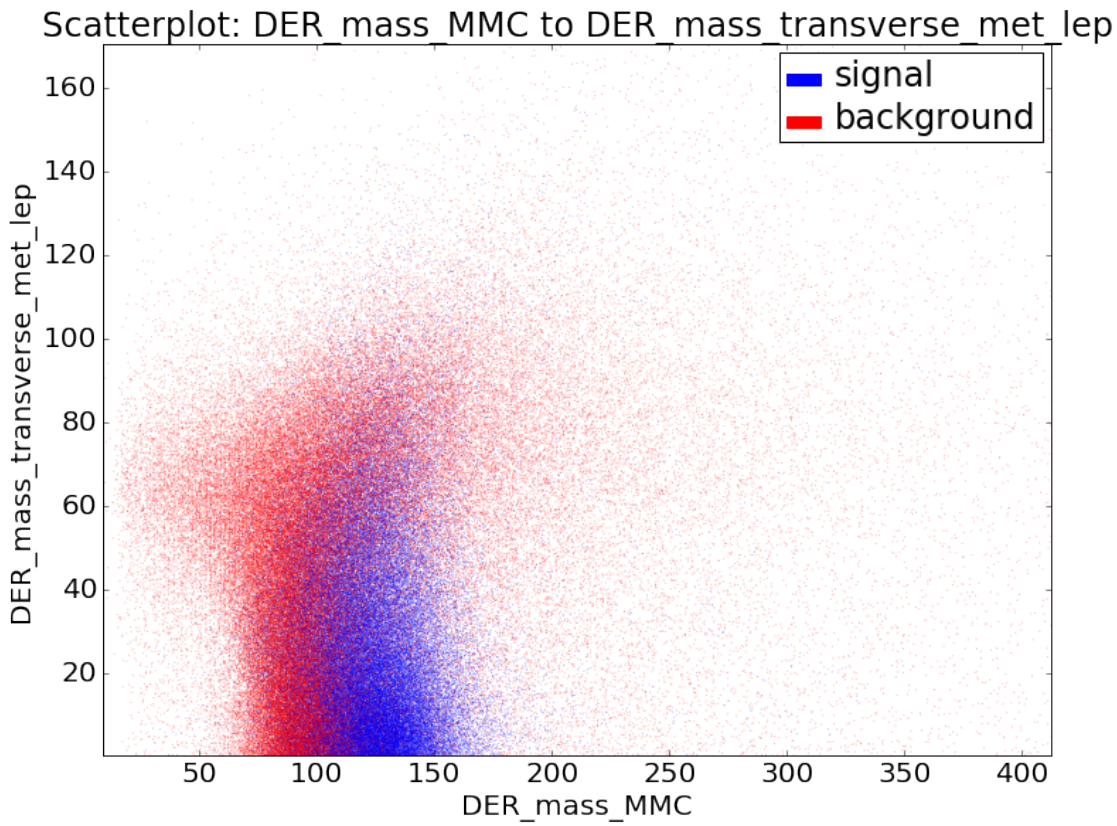
We construct a method for creating scatter plots.

```
In [7]: def scattered(xName,yName,xData,yData,labels, alph=0.2):
            font = {'size'   : 20}
            colors = generateLabelsColors(labels)

            scale = 0.5
            xmin,xmax = scaleAxis(xData,scale)
            ymin,ymax = scaleAxis(yData,scale)

            scat = plt.scatter(xData, yData, s=1, edgecolor="", c=colors, alpha=alph)

            plt.xlabel(xName)
            plt.ylabel(yName)

            plt.axis([xmin,xmax,ymin,ymax])

            title = ("Scatterplot: "+ xName+ " to "+ yName)

            blue_patch = mpatches.Patch(color='blue', label='signal')
            red_patch = mpatches.Patch(color='red', label='background')
            plt.legend(handles=[blue_patch,red_patch])
            matplotlib.rc('font', **font)
            plt.title(title)

            return plt
```

We reproduce Fig. 2:

```
In [8]: xName = "DER_mass_MMC"
        yName = "DER_mass_transverse_met_lep"

        xData = train_all[:,header_all.index(xName)]
        yData = train_all[:,header_all.index(yName)]
        labels = train_labels

        %pylab inline
        mainFig = plt.figure(figsize=(12,9))
        ax = mainFig.add_subplot(111)
        ax = scattered(xName,yName,xData,yData,labels, alph=0.3)
        savePath = (thesisImagePath + "scatter.png")

        #we use pdf-format only for the printed thesis, because the digital version experiences heavy l
        #mainFig.savefig(savePath,format="pdf",dpi=270)
        #mainFig.savefig(savePath,format="png",dpi=540)
```

```
Populating the interactive namespace from numpy and matplotlib
```



For data analysis, we created all possible two feature combinations.
Run following method with caution, as it takes several hours to terminate.

```
In [9]: def createAllScatterplots(header,data,labels,scatterPath,startX=1,x_size=3,y_size=2,n=0):
            for x in range(startX,len(header)):
                xName = header[x]
                xData = data[:,x]
```

```
            xFolder = (scatterPath + "/" + xName)
            if not os.path.exists(xFolder):
                os.makedirs(xFolder)
            y = 1
            while y in range(1,len(header)):
                mainFig = plt.figure(figsize=(20,15))
                for i in range(1,(1+(x_size*y_size))):
                    yName = header[y]
                    yData = data[:,y]
                    ax = mainFig.add_subplot(x_size,y_size,i)
                    y += 1
                    ax = scattered(xName,yName,xData,yData,labels)

                savePath = (xFolder + "/" + str(n))
                n += 1
                mainFig.savefig(savePath)
                mainFig = None

            plt.close("all")
```

```
In [10]: start = header_all.index("PRI_lep_pt")
         n = (start-1)*5
         #createAllScatterplots(header_all,test_all,test_labels,scatterPath,startX=start,n=n)
```

Following method creates histograms:

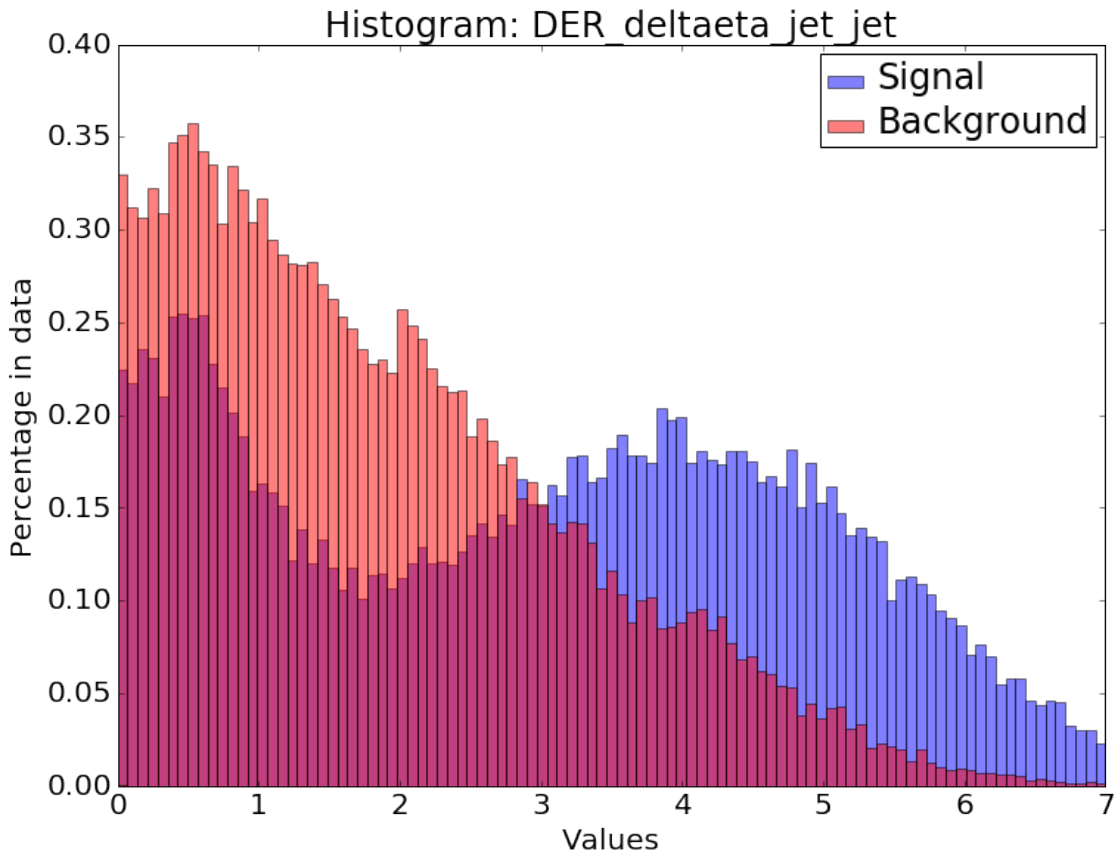
```
In [11]: def histo(featName,data,labels):
             font = {'size'   : 20}

             b = 100

             title = str("Histogram: "+ featName)
             sdata = []
             bdata = []
             for i in range(0,len(data)):
                 if labels[i] == 0:
                     bdata.append(data[i])
                 else:
                     sdata.append(data[i])

             colors = generateLabelsColors(labels)

             scale = 0.1
             xmin,xmax = scaleAxis(data,scale)

             xmin = int(xmin)
             xmax = int(xmax)

             shist = plt.hist(sdata,bins = np.linspace(xmin,xmax,b), normed=1, facecolor='blue', alpha=
             bhist = plt.hist(bdata,bins = np.linspace(xmin,xmax,b), normed=1, facecolor='red', alpha=0

             plt.legend(('Signal', 'Background'))

             plt.ylabel('Percentage in data')
             plt.xlabel('Values')
```

```
            plt.title(title)
            matplotlib.rc('font', **font)
            return plt
```

We reproduce Fig. 1:

```
In [12]: %pylab inline
         xName = "DER_deltaeta_jet_jet"

         xData = train_all[:,header_all.index(xName)]
         labels = train_labels

         mainFig = plt.figure(figsize=(12,9))
         ax = mainFig.add_subplot(111)
         ax = histo(xName,xData,labels)
         savePath = (thesisImagePath + "histogram.pdf")
         #mainFig.savefig(savePath,format="pdf",dpi=270)
```

```
Populating the interactive namespace from numpy and matplotlib
```



In contrast to scatter plots, creating all histograms terminates within a minute (on the system used for the thesis).

```
In [13]: def createAllHistograms(header,data,labels,histPath,x=3,y=2):
             n = 0
```

```
    p = 0
    while p in range(0,len(header)):
        mainFig = plt.figure(figsize=(30,20))
        for i in range(1,(1+(x*y))):
            if p < len(header):
                featName = header[p]
                feat_data = data[:,p]
                ax = mainFig.add_subplot(x,y,i)
                ax = histo(featName,feat_data,labels);
                p += 1
            else:
                break
        savePath = (histPath + "/hist_" + str(n))
        n += 1
        mainFig.savefig(savePath)

        mainFig = None
        plt.close("all")
```

```
In [14]: #createAllHistograms(header_all,train_all,train_labels,histPath)
```

1.2 Figures 8 and 10

First, we fetch data of the challenge's leaderboards.

```
In [15]: pubLB,privLB = kD.getLeaderBoards()
```

For simplicity, following placements were retrieved by hand, because `getLeaderBoards()` does not fetch submission names.

```
In [16]: benchmarkData = [["bm: random submission",privLB[1698,1],1699],
                          ["bm: simple window",privLB[1606,1],1607],
                          ["bm: naive Bayes starting kit",privLB[1462,1],1463],
                          ["bm: simple TMVA boosted trees",privLB[991,1],992],
                          ["bm: multiboost",privLB[902,1],903]
                         ]
```

We calculate placements of our best submissions per hand, real submissions are extracted from leaderboard data.

```
In [17]: amsData = [["logistic Regression",2.06934119373163,1429],
                   ["k Nearest Neighbors",3.18323451660961,996],
                   ["sklearn.GradientBoostingClassifier",3.45097313747456,838],
                   ["XGBoost",3.71268472156738,65],
                   ["XGBoost original sumission",privLB[44,1],45],
                   ["winner",privLB[0,1],1]
                  ]
```

The data is stacked, sorted and split into 3 seperate arrays for their use in following plots.

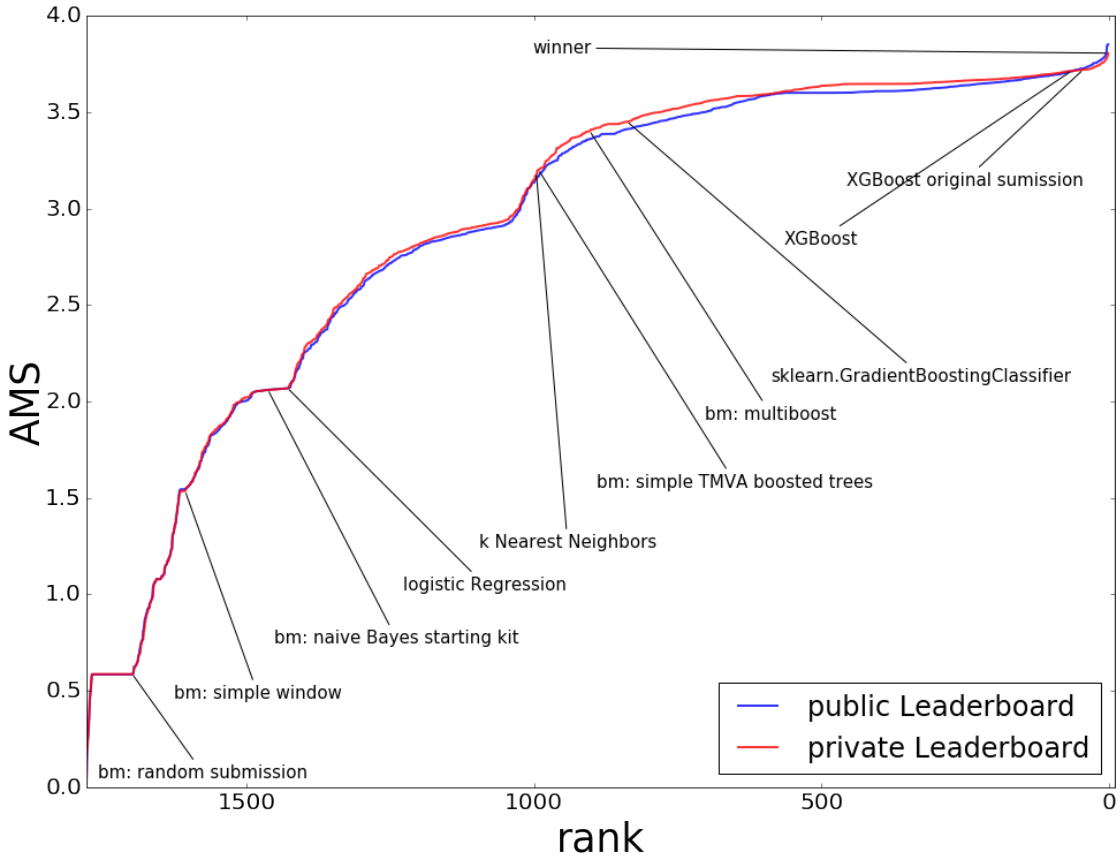
```
In [18]: data = vstack((benchmarkData,amsData))
         data = tb.sortByColumn(data,1)
```

```
In [19]: ranks = np.array(data[:,2],dtype=float)
         ams = np.array(data[:,1],dtype=float)
         names = np.array(data[:,0],dtype="<U40")
```

We reproduce Fig. 8. It is necessary to adjust the annotations manually.

In [20]:
```
mainFig = plt.figure(figsize=(16,12))
ax = mainFig.add_subplot(111)

#font size of annotations
font = {'size'   : 15}
matplotlib.rc('font', **font)

xmax = pubLB[:,1].max()
plt.axis([1779,-10,0.,4.])

plt.plot(pubLB[:,2],pubLB[:,1], linewidth=2, alpha = 0.8)
plt.plot(privLB[:,2],privLB[:,1], "r", linewidth=2, alpha = 0.8)

#annotations made by hand for useful positioning
#random submission
plt.annotate(names[0],xy = (ranks[0],ams[0]), xytext=(-30,-90),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#simple windows
plt.annotate(names[1],xy = (ranks[1],ams[1]), xytext=(-10,-180),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#naive bayes
plt.annotate(names[2],xy = (ranks[2],ams[2]), xytext=(5,-220),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#logistic regression
plt.annotate(names[3],xy = (ranks[3],ams[3]), xytext=(100,-175),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#kNN
plt.annotate(names[4],xy = (ranks[4],ams[4]), xytext=(-50,-325),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#simple TMVA
plt.annotate(names[5],xy = (ranks[5],ams[5]), xytext=(50,-275),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#multiboost
plt.annotate(names[6],xy = (ranks[6],ams[6]), xytext=(100,-250),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#sklearn gbc
plt.annotate(names[7],xy = (ranks[7],ams[7]), xytext=(125,-225),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#xgboost
plt.annotate(names[8],xy = (ranks[8],ams[8]), xytext=(-250,-150),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#original xgboost
plt.annotate(names[9],xy = (ranks[9],ams[9]), xytext=(-205,-100),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-
#winner
plt.annotate(names[10],xy = (ranks[10],ams[10]), xytext=(-500,0),
                 xycoords = "data", textcoords='offset points',arrowprops = dict(arrowstyle="-

#text formatting of axis
overridex = {
    'fontsize'            : 'xx-large',
    'verticalalignment'   : 'top',
```

```
    'horizontalalignment' : 'center'
    }

overridey = {
   'fontsize'            : 'xx-large',
   'verticalalignment'   : 'baseline',
   'horizontalalignment' : 'center',
   'rotation'            : 'vertical'
}

#font size of labels and legend
font = {'size'    : 20}
matplotlib.rc('font', **font)

plt.xlabel("rank",overridex)
plt.ylabel("AMS",overridey)

plt.legend(('public Leaderboard', 'private Leaderboard'),loc=4)

savePath = (thesisImagePath + "amscompare.pdf")
#mainFig.savefig(savePath,format="pdf",dpi=270)
```



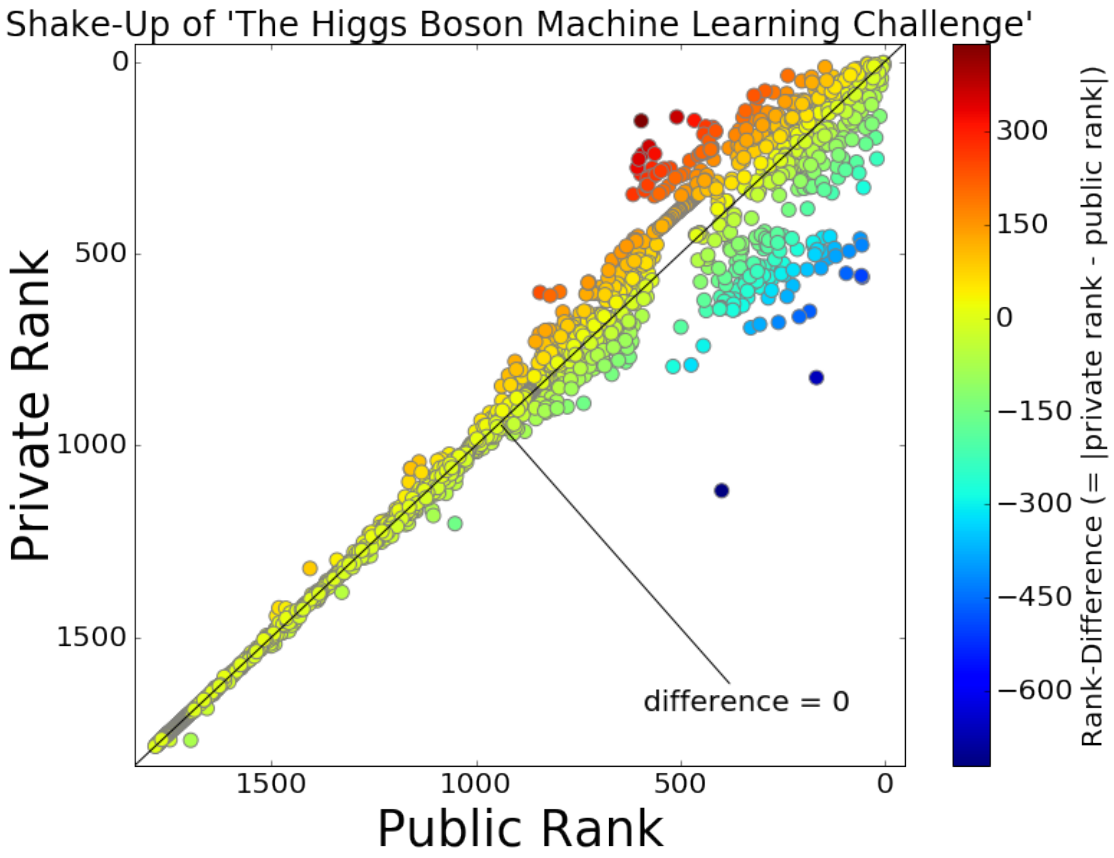
We produce the Shake-up plot twice and only use different data. It makes sense to create a method:

```
In [21]: def plotShakeup(pubLB,privLB,fname,title = "a Kaggle competition"):
```

```
#prepare data
sortedPriv=tb.sortByColumn(privLB,0)
sortedPub=tb.sortByColumn(pubLB,0)

rankdiff = np.copy(sortedPub[:,2]-sortedPriv[:,2])

#font = {'size'   : 20}

mainFig = plt.figure(figsize=(12,9))
ax = mainFig.add_subplot(111)

xData = sortedPub[:,2]
yData = sortedPriv[:,2]

plt.Normalize()
#actual plotting
scat = plt.scatter(xData, yData, s=100, edgecolor="gray",  c=rankdiff, cmap="jet")

xName = "Public Rank"
yName = "Private Rank"

#text format of x and y axis
overridex = {
    'fontsize'            : 'xx-large',
    'verticalalignment'   : 'top',
    'horizontalalignment' : 'center'
    }

overridey = {
   'fontsize'            : 'xx-large',
   'verticalalignment'   : 'baseline',
   'horizontalalignment' : 'center',
   'rotation'            : 'vertical'
}

plt.xlabel(xName,overridex)
plt.ylabel(yName,overridey)

xmax = pubLB[-1,-1]+50
xmin = -50

plt.axis([xmax,xmin,xmax,xmin])

#mark "zero difference"-line
x = np.arange(xmin,xmax)
xy = plt.plot(x,x,color="black")

coord = int(abs(xmax-xmin)/2)
plt.annotate("difference = 0",xy = (coord,coord), xytext=(+100,-200),
             xycoords = "data", textcoords='offset points',
             arrowprops = dict(arrowstyle="-"))

#font size of title and labels
```

```
    matplotlib.rc('font', **font)

    head = ("Shake-Up of " + title)
    plt.title(head)

    cbar = plt.colorbar()
    cbar.set_label("Rank-Difference (= |private rank - public rank|)")

    savePath = (thesisImagePath + fname)

    #comment in to save created plots
    #mainFig.savefig(savePath,format="pdf",dpi=270)
```

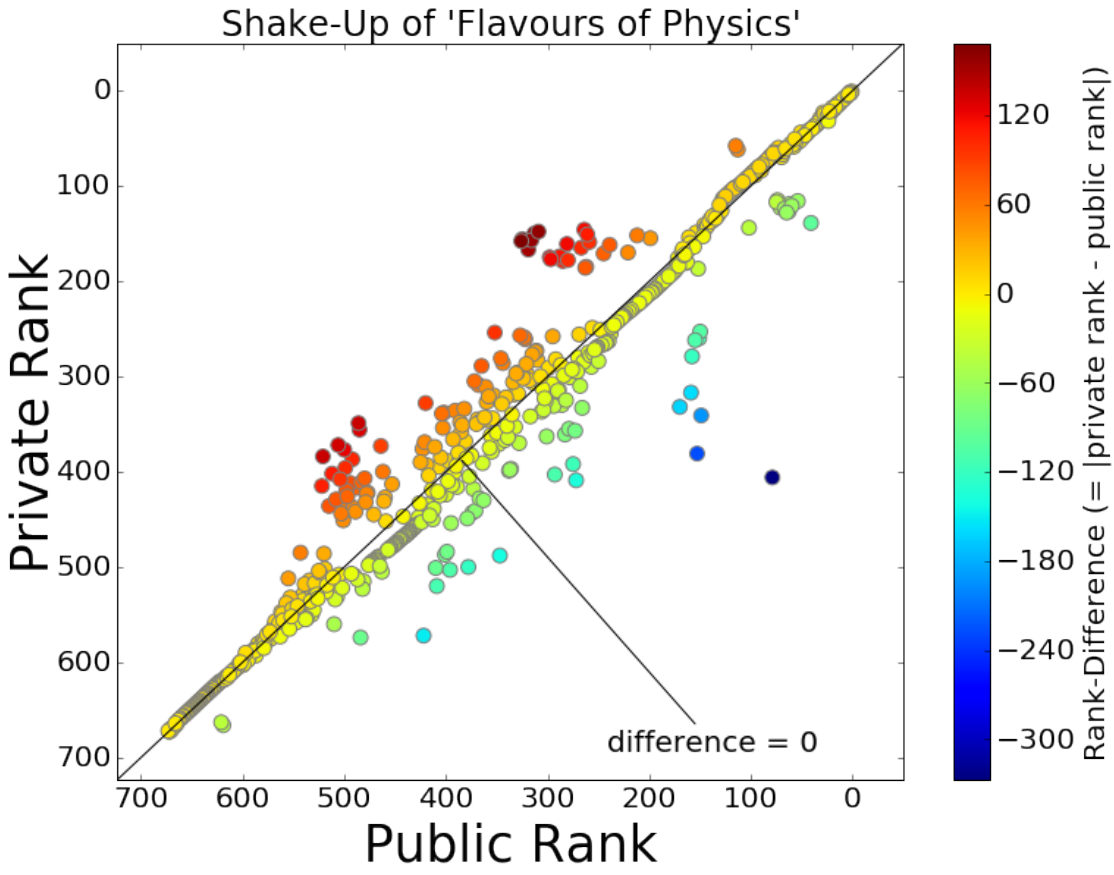
In [22]:
```
plotShakeup(pubLB,privLB,"shakeup1.pdf",title = "'The Higgs Boson Machine Learning Challenge'")
```



We create the Shake-Up plot for 'Flavours of Physics'

In [23]:
```
pub_url = "https://www.kaggle.com/c/flavours-of-physics/leaderboard/public"
priv_url = "https://www.kaggle.com/c/flavours-of-physics/leaderboard/private"
flav_pubLB = kD.getLeaderBoard(pub_url)
flav_privLB = kD.getLeaderBoard(priv_url)
```

In [24]:
```
plotShakeup(flav_pubLB,flav_privLB,"shakeup2.pdf",title = "'Flavours of Physics'")
```



1.3 Figures 4,7 and 9

We extract the data we recorded during testing.

```
In [25]: rec_data ,rec_header = tb.getRecord()
         rec_data_sorted = tb.sortByColumn(rec_data,0)
```

As we want to extract more specific data, we need a method:

```
In [26]: def extractRecord(nameList, record, r_header):

             #if the classifiers name is in nameList, extract this recorded run
             array = []
             for row in record:
                 if row[0] in nameList:
                     array.append(row)
             n_array = np.array(array)

             #create a classifier-specific header
             n_header = r_header[:7]
             for i in range(7,len(r_header)):
                 if len(n_array[0,i].split("=")) < 2:
                     break
```

```
            else:
                n_header.append(n_array[0,i].split("=")[0])

        #extract the test parameters of this run, cut "None" entries
        n_array = n_array[:,:len(n_header)]
        for row in n_array:
            for i in range(7,len(n_header)):
                row[i] = row[i].split("=")[1]


        return n_array,n_header
```

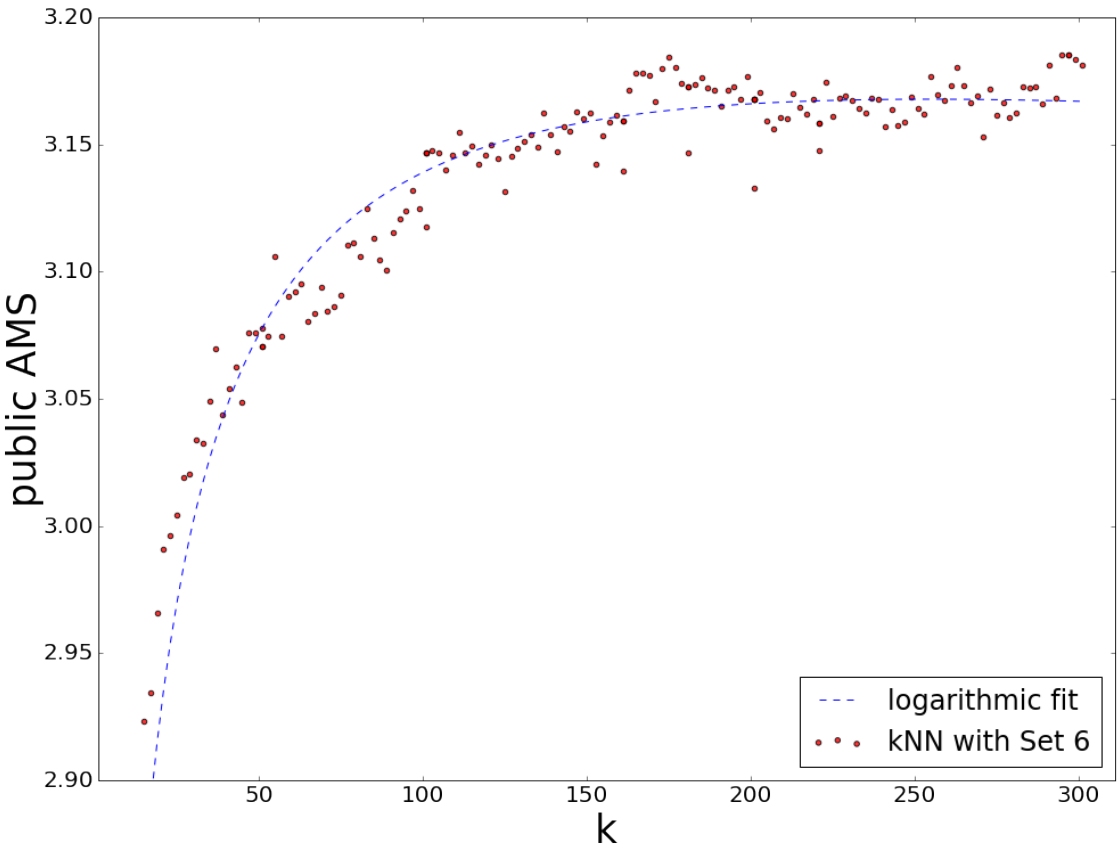
We split the recorded data corresponding to the used classifier.

```
In [27]: rec_xgb,xgb_header = extractRecord(["xgboost"],rec_data_sorted,rec_header)
         rec_gbc,gbc_header = extractRecord(["gbc"],rec_data_sorted,rec_header)
         rec_log,log_header = extractRecord(["log Reg","log Reg CV"],rec_data_sorted,rec_header)
         rec_knn,knn_header = extractRecord(["kNN"],rec_data_sorted,rec_header)
```

We reproduce Fig. 4a with kNN runs performed on feature set 6:

```
In [28]: rec_knn_6 = []
         for row in rec_knn:
             if row[1] == "header_6":
                 rec_knn_6.append(row)
         rec_knn_6 = np.array(rec_knn_6)

In [29]: xydata=np.vstack((rec_knn_6[:,3],rec_knn_6[:,8])).transpose().astype(float)

In [30]: mainFig = plt.figure(figsize=(16,12))
         ax = mainFig.add_subplot(111)

         font = {'size'   : 15}
         matplotlib.rc('font', **font)

         # set the axis manually
         plt.axis([1,311,2.9,3.2])

         # set up data
         xydata=tb.sortByColumn(xydata,1)
         x=xydata[:,1]
         y=xydata[:,0]

         # create the scatter plot
         plt.scatter(x,y,c="r", alpha = 0.8)

         # Fit a trend
         coefficients = np.polyfit(np.log(x),y,3) # Use log(x) as the input to polyfit.
         fit = np.poly1d(coefficients)
         plt.plot(x,fit(np.log(x)),"--", label="fit")

         # formating x and y axis
         font = {'size'   : 20}
         matplotlib.rc('font', **font)
```

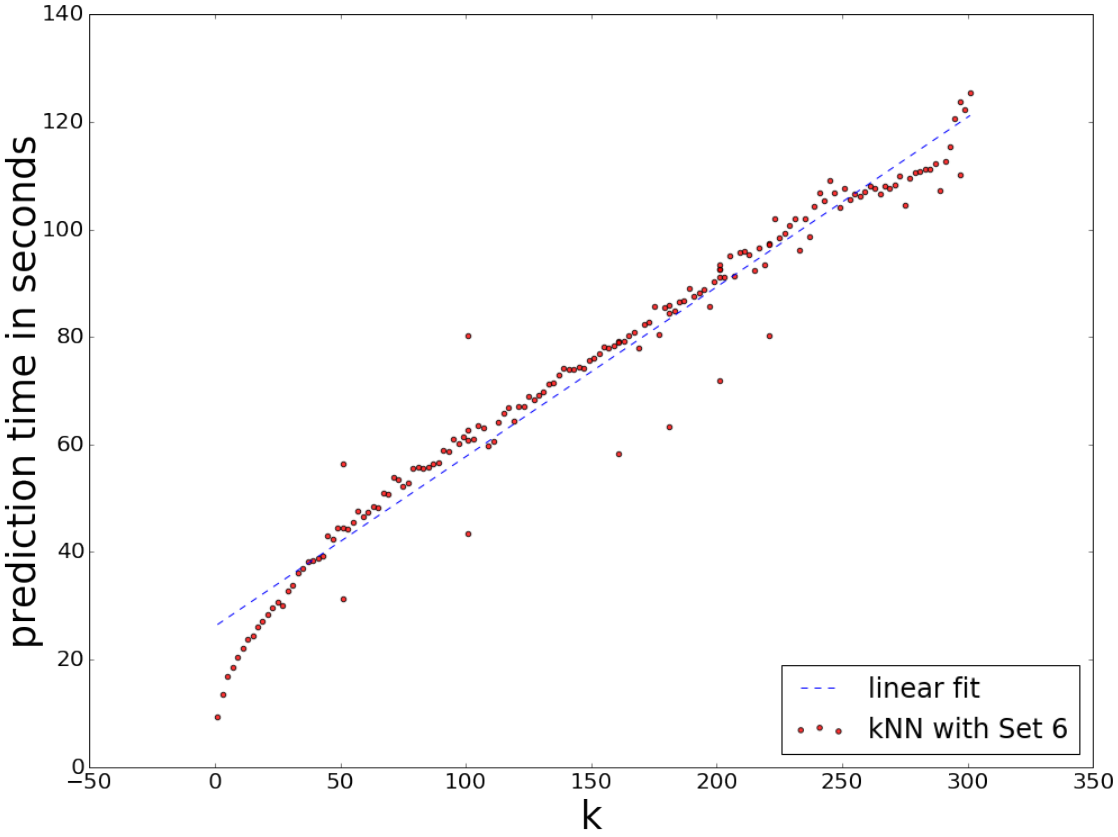
```
overridex = {
    'fontsize'            : 'xx-large',
    'verticalalignment'   : 'top',
    'horizontalalignment' : 'center'
    }

overridey = {
   'fontsize'            : 'xx-large',
   'verticalalignment'   : 'baseline',
   'horizontalalignment' : 'center',
   'rotation'            : 'vertical'
}

plt.xlabel("k",overridex)
plt.ylabel("public AMS",overridey)

# font size of legend
font = {'size'   : 20}
matplotlib.rc('font', **font)

# create legend
plt.legend(("logarithmic fit",'kNN with Set 6'),loc=4)

savePath = (thesisImagePath + "knn_ams.pdf")
#mainFig.savefig(savePath,format="pdf",dpi=270)
```



We reproduce Fig. 4b:

In [31]:
```
xydata=np.vstack((rec_knn_6[:,6],rec_knn_6[:,8])).transpose().astype(float)
```

In [32]:
```
mainFig = plt.figure(figsize=(16,12))
ax = mainFig.add_subplot(111)

font = {'size'   : 15}
matplotlib.rc('font', **font)

# set up data
xydata=tb.sortByColumn(xydata,1)
x=xydata[:,1]
y=xydata[:,0]

# create the scatter plot
plt.scatter(x,y,c="r", alpha = 0.8)

# Fit a trend
coefficients = np.polyfit(x,y,1)
fit = np.poly1d(coefficients)
plt.plot(x,fit(x),"--", label="fit")

# formating x and y axis
font = {'size'   : 30}
matplotlib.rc('font', **font)

overridex = {
    'fontsize'            : 'xx-large',
    'verticalalignment'   : 'top',
    'horizontalalignment' : 'center'
    }

overridey = {
   'fontsize'            : 'xx-large',
   'verticalalignment'   : 'baseline',
   'horizontalalignment' : 'center',
   'rotation'            : 'vertical'
}

plt.xlabel("k",overridex)
plt.ylabel("prediction time in seconds",overridey)

# font size of legend
font = {'size'   : 20}
matplotlib.rc('font', **font)

# create legend
plt.legend(('linear fit' , 'kNN with Set 6'),loc=4)

savePath = (thesisImagePath + "knn_speed.pdf")
#mainFig.savefig(savePath,format="pdf",dpi=270)
```

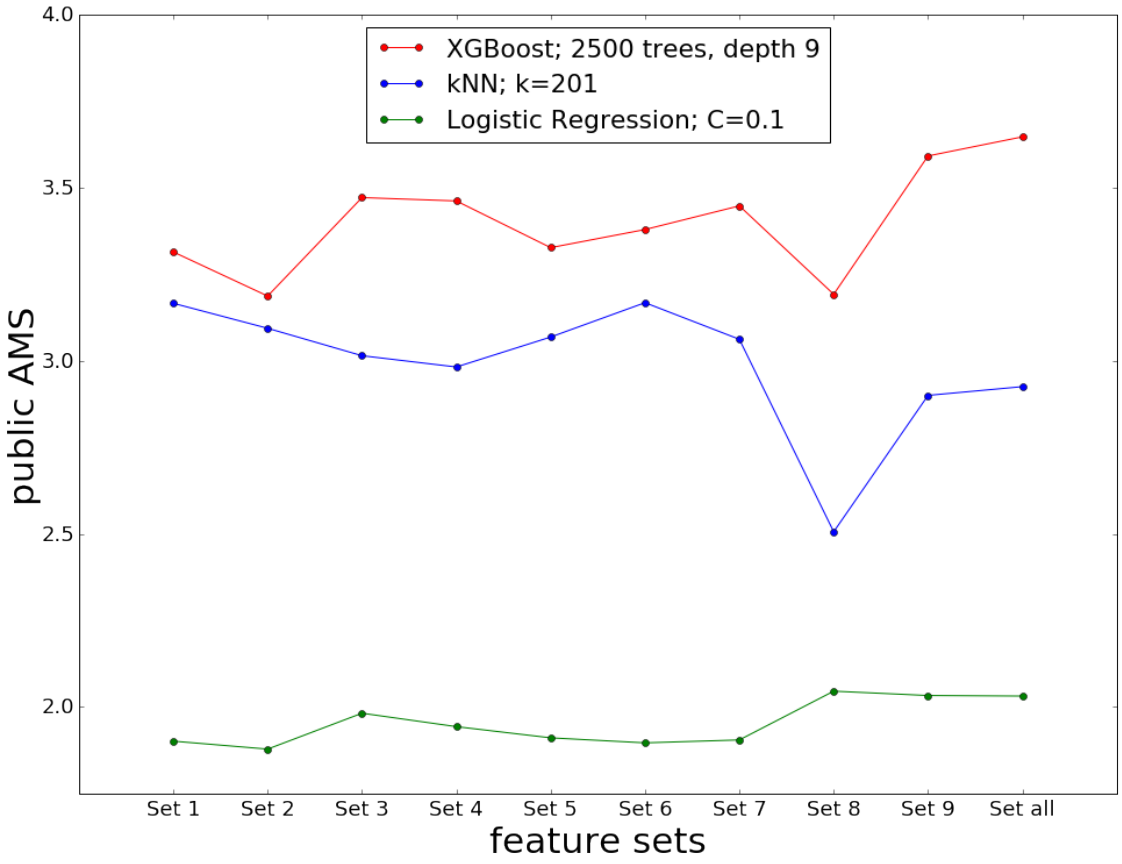


Sort recorded data w.r.t. public AMS.

```
In [33]: rec_pubams=tb.sortByColumn(rec_data,3)

In [34]: xgbData = []
         for row in rec_xgb:
             if row[7] == '0.14' and row[8] == '2500' and row[10] == '0.01':
                 xgbData.append([row[1],row[3]])
         xgbData = np.array(xgbData)
         xgbData = xgbData[:-1]

In [35]: knnData = []
         for row in rec_knn:
             if row[8] == '201' and row[9] == '1':
                 knnData.append([row[1],row[3]])
         knnData = np.array(knnData)
         knnData = np.vstack((tb.sortByColumn(knnData,0)[1:7],tb.sortByColumn(knnData,0)[-4:]))

In [36]: logData = []
         for row in rec_log:
             if row[-1] == '12' and row[-2] == '0.1':
                 logData.append([row[1],row[3]])
         logData = np.array(logData)
         logData = tb.sortByColumn(logData,0)
         logData = np.vstack((tb.sortByColumn(logData,0)[:-3],tb.sortByColumn(logData,0)[-2:]))
```

```
In [37]: def transform(data):
             for row in data:
                 setX = row[0].split("_")[1]
                 if setX == 'x':
                     setX = '9'
                 if setX == 'all':
                     setX = '10'
                 row[0] = setX
             return tb.sortByColumn(np.array(data).astype(float),0)

In [38]: xgbData = transform(xgbData)
         knnData = transform(knnData)
         logData = transform(logData)

In [39]: mainFig = plt.figure(figsize=(16,12))
         ax = mainFig.add_subplot(111)

         # font size
         font = {'size'   : 18}
         matplotlib.rc('font', **font)

         # create the plots
         plt.plot(xgbData[:,0],xgbData[:,1],"r-o")
         plt.plot(knnData[:,0],knnData[:,1],"b-o")
         plt.plot(logData[:,0],logData[:,1],"g-o")

         # formating x and y axis
         overridex = {
             'fontsize'            : 'xx-large',
             'verticalalignment'   : 'top',
             'horizontalalignment' : 'center'
             }

         overridey = {
            'fontsize'            : 'xx-large',
            'verticalalignment'   : 'baseline',
            'horizontalalignment' : 'center',
            'rotation'            : 'vertical'
         }

         plt.xlabel("feature sets",overridex)
         plt.ylabel("public AMS",overridey)

         # create x axis
         plt.xticks(np.arange(1,11),[
                     "Set 1",
                     "Set 2",
                     "Set 3",
                     "Set 4",
                     "Set 5",
                     "Set 6",
                     "Set 7",
                     "Set 8",
                     "Set 9",
                     "Set all"
```

```
        ])
plt.axis([0,11,1.75,4.0])

# create legend
plt.legend(('XGBoost; 2500 trees, depth 9','kNN; k=201','Logistic Regression; C=0.1'),loc=9)

savePath = (thesisImagePath + "setperformance.pdf")
#mainFig.savefig(savePath,format="pdf",dpi=270)
```



For creating Fig. 7 we extract specific recorded runs of gbc and xgboost.

```
In [40]: rec_gbc_all = []
         for row in rec_gbc:
             if row[1] == "header_all":
                 if row[10] == "0.01":
                     rec_gbc_all.append(row)
         rec_gbc_all = np.array(rec_gbc_all)

In [41]: rec_xgb_all_3000 = []
         for row in rec_xgb:
             if row[1] == "header_all":
                 if row[10] == "0.01" and row[8] == "3000":
                     rec_xgb_all_3000.append(row)
         rec_xgb_all_3000 = np.array(rec_xgb_all_3000)
```

```
In [42]: rec_xgb_all_1000 = []
         for row in rec_xgb:
             if row[1] == "header_all":
                 if row[10] == "0.01" and row[8] == "1000":
                     rec_xgb_all_1000.append(row)
         rec_xgb_all_1000 = np.array(rec_xgb_all_1000)
```

```
In [43]: rec_xgb_all_100 = []
         for row in rec_xgb:
             if row[1] == "header_all":
                 if row[9] in ['6','9','12']:
                     if row[10] == "0.01" and row[8] == "100":
                         rec_xgb_all_100.append(row)
         rec_xgb_all_100 = np.array(rec_xgb_all_100)
```

Create the plot with this data.

```
In [44]: mainFig = plt.figure(figsize=(12,10))
         ax = mainFig.add_subplot(111)

         font = {'size'   : 18}
         matplotlib.rc('font', **font)

         width = 0.25

         # create bar plots
         plt.bar(left=rec_gbc_all[:,9].astype(float),height = rec_gbc_all[:,3].astype(float), alpha = 1
         plt.bar(left=rec_xgb_all_100[:,9].astype(float)+width,height = rec_xgb_all_100[:,3].astype(flo
         plt.bar(left=rec_xgb_all_1000[:,9].astype(float)+2*width,height = rec_xgb_all_1000[:,3].astype
         plt.bar(left=rec_xgb_all_3000[:,9].astype(float)+3*width,height = rec_xgb_all_3000[:,3].astype

         # formating x and y axis
         overridex = {
             'fontsize'            : 'xx-large',
             'verticalalignment'   : 'top',
             'horizontalalignment' : 'center'
             }

         overridey = {
            'fontsize'            : 'xx-large',
            'verticalalignment'   : 'baseline',
            'horizontalalignment' : 'center',
            'rotation'            : 'vertical'
         }

         plt.xlabel("tree depth",overridex)
         plt.ylabel("public AMS",overridey)

         # create x axis
         plt.xticks([6,9,12])
         plt.axis([5,14,2.75,4.0])

         # create legend
         plt.legend(('sklearn.GradientBoostingClassifier, 100 trees','XGBoost, 100 trees','XGBoost, 100
```

```
savePath = (thesisImagePath + "xgb-gbc.pdf")
#mainFig.savefig(savePath,format="pdf",dpi=270)
```