Assignment 1

March 5, 2018

```
In [4]: from IPython.display import HTML
        HTML('''<style>html, body{overflow-y: visible !important} .CodeMirror{min-width:105% !i

Out[4]: <IPython.core.display.HTML object>
```

1 Foundations of Data Mining: Assignment 1

Please complete all assignments in this notebook. You should submit this notebook, as well as a PDF version (See File > Download as).

```
In [5]: # Please fill in your names here
        NAME_STUDENT_1 = "Bram van der Pol"
        NAME_STUDENT_2 = "Joris van der Heijden"
```

```
In [6]: %matplotlib inline
        from preamble import *
        plt.rcParams['savefig.dpi'] = 200 # This controls the size of your figures
        # Comment out and restart notebook if you only want the last output of each cell.
        InteractiveShell.ast_node_interactivity = "all"
```

1.1 MoneyBall (5 points, 1+2+1+1)

In the early 2000s, 2 baseball scouts completely changed the game of baseball by analysing the available data about baseball players and hiring the best ones. The MoneyBall dataset contains this data (click the link for more details). The goal is to accurately predict the number of 'runs' each player can score.

```
In [7]: moneyball = oml.datasets.get_dataset(41021); # Download MoneyBall data
        # Get the predictors X and the target y
        X, y, attribute_names = moneyball.get_data(target=moneyball.default_target_attribute, r
        # Describe the data with pandas, just to get an overview
        ballframe = pd.DataFrame(X, columns=attribute_names);
        ballframe.describe();

        import matplotlib;
        #n, bins, patches = ax.hist(X, num_bins, normed=1)
```

```
for index in range(len(X.T)):
    #if (index < 8):
    print(index);
    column=X[:,index];
    Filtered_column1=column[~np.isnan(column)];

    Filtered_column2 = [np.nan if np.isnan(x) else x for x in column];

    fig1 = plt.figure(); #Generate new figure
    matplotlib.pyplot.subplot(1,2,1);
    matplotlib.pyplot.hist(Filtered_column1);
    matplotlib.pyplot.title('Histogram',fontweight='bold',fontsize=15);


    matplotlib.pyplot.subplot(1,2,2);
    matplotlib.pyplot.scatter(Filtered_column2,y);
    matplotlib.pyplot.title('Scatter',fontweight='bold',fontsize=15);
    matplotlib.pyplot.xlabel('X');
    matplotlib.pyplot.ylabel('y');

    matplotlib.pyplot.suptitle(attribute_names[index]+" Index "+str(index) ,fontweight
    matplotlib.pyplot.subplots_adjust(left=0.2,wspace=0.8,top=0.8);
```

Out[7]:

```
          Team  League     Year       RA  ...  RankPlayoffs        G  \
count  1232.00  1232.0  1232.00  1232.00  ...        244.00  1232.00
mean     15.67     0.5  1988.96   715.08  ...          1.72     3.92
std       9.72     0.5    14.82    93.08  ...          1.10     0.62
min       0.00     0.0  1962.00   472.00  ...          0.00     0.00
25%       7.00     0.0  1976.75   649.75  ...          1.00     4.00
50%      16.00     0.5  1989.00   709.00  ...          2.00     4.00
75%      23.00     1.0  2002.00   774.25  ...          3.00     4.00
max      38.00     1.0  2012.00  1103.00  ...          4.00     7.00

         OOBP    OSLG
count  420.00  420.00
mean     0.33    0.42
std      0.02    0.03
min      0.29    0.35
25%      0.32    0.40
50%      0.33    0.42
75%      0.34    0.44
max      0.38    0.50

[8 rows x 14 columns]
```

```
0
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef0bb0e48>

```
Out[7]: (array([152., 188., 152.,  98., 141., 175., 132.,  95.,  78.,  21.]),
         array([ 0. ,  3.8,  7.6, 11.4, 15.2, 19. , 22.8, 26.6, 30.4, 34.2, 38. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef0edfdd8>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef0f47ac8>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'Team Index 0')

1


Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef0c95a90>

Out[7]: (array([616.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 616.]),
         array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef0f8b550>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef0fde1d0>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'League Index 1')

2


Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef0fdec88>

Out[7]: (array([100., 112.,  96., 104., 130., 130.,  82., 148., 150., 180.]),
         array([1962., 1967., 1972., 1977., 1982., 1987., 1992., 1997., 2002.,
                2007., 2012.]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef102fef0>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef10662b0>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'Year Index 2')
```

3

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1066d68>

Out[7]: (array([ 20., 110., 239., 327., 270., 163.,  77.,  24.,   1.,   1.]),
         array([ 472. ,  535.1,  598.2,  661.3,  724.4,  787.5,  850.6,  913.7,
                 976.8, 1039.9, 1103. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef10a2da0>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef10f2a58>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'RA Index 3')
```

4

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef10f2f28>

Out[7]: (array([  2.,  17.,  50., 167., 243., 291., 286., 134.,  39.,   3.]),
         array([ 40. ,  47.6,  55.2,  62.8,  70.4,  78. ,  85.6,  93.2, 100.8,
                108.4, 116. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1122048>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef1177e48>
```

```
Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'W Index 4')

5


Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1180a90>

Out[7]: (array([  7.,  22.,  64., 193., 322., 266., 225.,  81.,  45.,   7.]),
         array([0.277, 0.287, 0.296, 0.306, 0.315, 0.325, 0.335, 0.344, 0.354,
                0.363, 0.373]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef11b8e10>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef1208fd0>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'OBP Index 5')

6


Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef122abe0>

Out[7]: (array([ 13.,  31., 100., 186., 276., 258., 202.,  93.,  57.,  16.]),
         array([0.301, 0.32 , 0.339, 0.358, 0.377, 0.396, 0.415, 0.434, 0.453,
                0.472, 0.491]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1244940>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef1292cc0>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')
```

```
Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'SLG Index 6')

7


Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef12988d0>

Out[7]: (array([  4.,  14.,  41., 102., 242., 329., 229., 186.,  58.,  27.]),
         array([0.214, 0.222, 0.23 , 0.238, 0.246, 0.254, 0.262, 0.27 , 0.278,
                0.286, 0.294]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef12d2b70>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef1262c18>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'BA Index 7')

8


Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef133c908>

Out[7]: (array([988.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 244.]),
         array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef135cc88>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef13aae80>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'Playoffs Index 8')
```

9

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef13b4be0>

Out[7]: (array([52., 53., 44.,  0., 44., 21.,  0., 20.,  9.,  1.]),
         array([0. , 0.7, 1.4, 2.1, 2.8, 3.5, 4.2, 4.9, 5.6, 6.3, 7. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef13e7fd0>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef142fcc0>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'RankSeason Index 9')
```

10

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1437898>

Out[7]: (array([47.,  0., 47.,  0.,  0., 80.,  0., 68.,  0.,  2.]),
         array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1471c88>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef14bff98>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'RankPlayoffs Index 10')
```

11

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef14c9be0>
```

```
Out[7]: (array([  1.,  10.,  23.,   0., 139., 954.,   0.,  93.,  10.,   2.]),
         array([0. , 0.7, 1.4, 2.1, 2.8, 3.5, 4.2, 4.9, 5.6, 6.3, 7. ]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef14faa90>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef154ecc0>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'G Index 11')
```

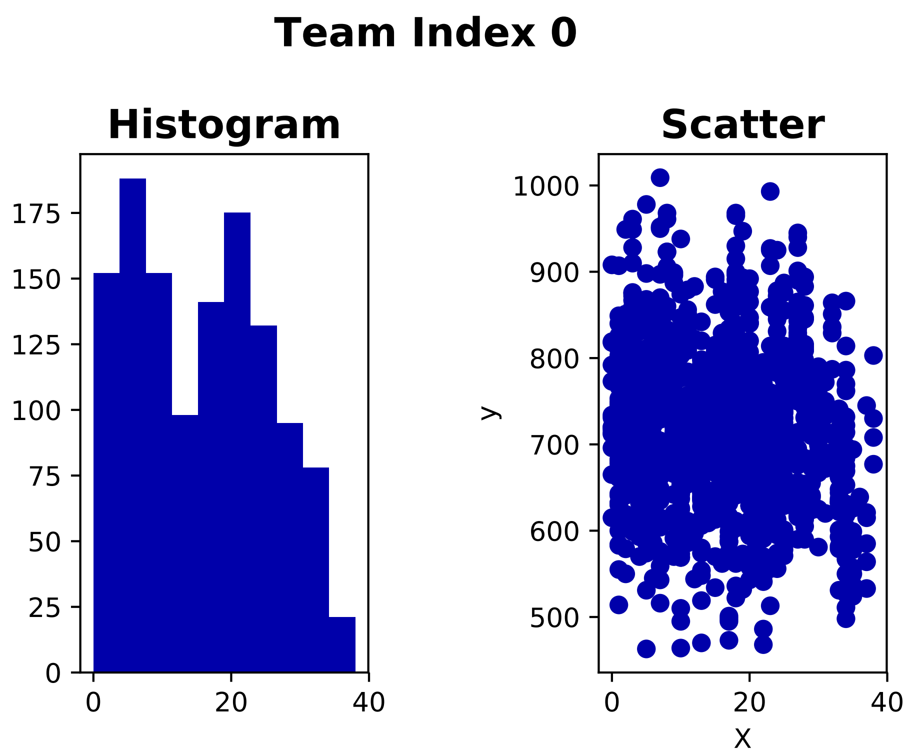
12

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1558978>

Out[7]: (array([ 4., 33., 64., 89., 94., 74., 41., 15.,  5.,  1.]),
         array([0.294, 0.303, 0.312, 0.321, 0.33 , 0.339, 0.348, 0.357, 0.366,
                0.375, 0.384]),
         <a list of 10 Patch objects>)

Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef158cbe0>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef15d9eb8>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'OOBP Index 12')
```

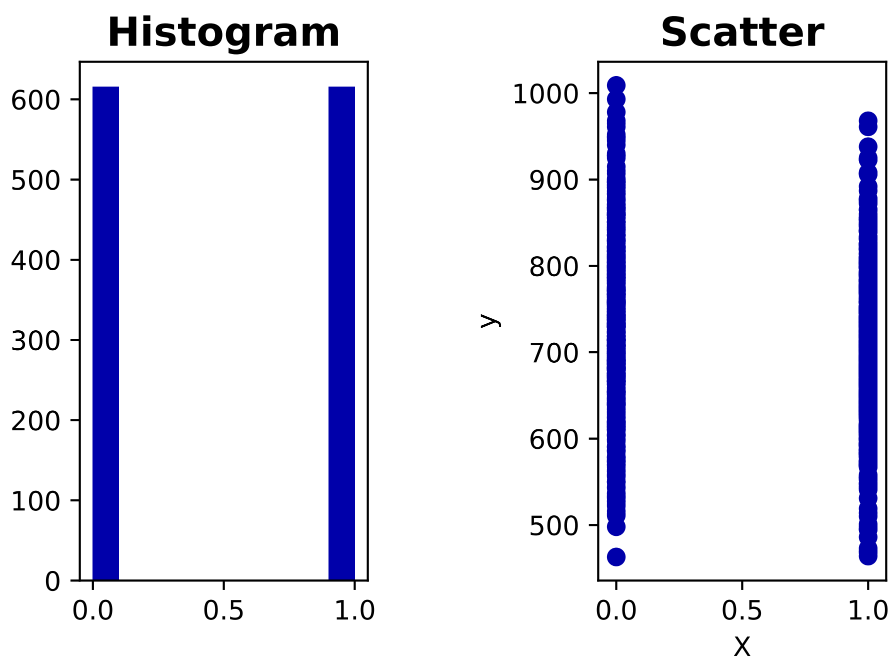
13

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef1563b70>

Out[7]: (array([ 6., 14., 36., 92., 82., 82., 61., 31., 14.,  2.]),
         array([0.346, 0.361, 0.377, 0.392, 0.407, 0.422, 0.438, 0.453, 0.468,
                0.484, 0.499]),
         <a list of 10 Patch objects>)
```

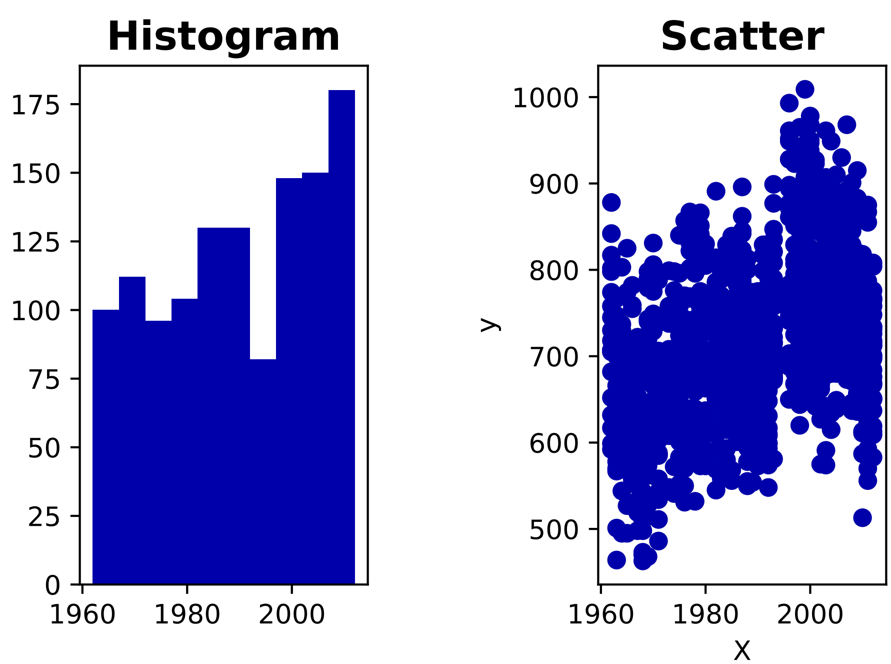
```
Out[7]: Text(0.5,1,'Histogram')

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2cef16158d0>

Out[7]: <matplotlib.collections.PathCollection at 0x2cef1664ac8>

Out[7]: Text(0.5,1,'Scatter')

Out[7]: Text(0.5,0,'X')

Out[7]: Text(0,0.5,'y')

Out[7]: Text(0.5,0.98,'OSLG Index 13')
```

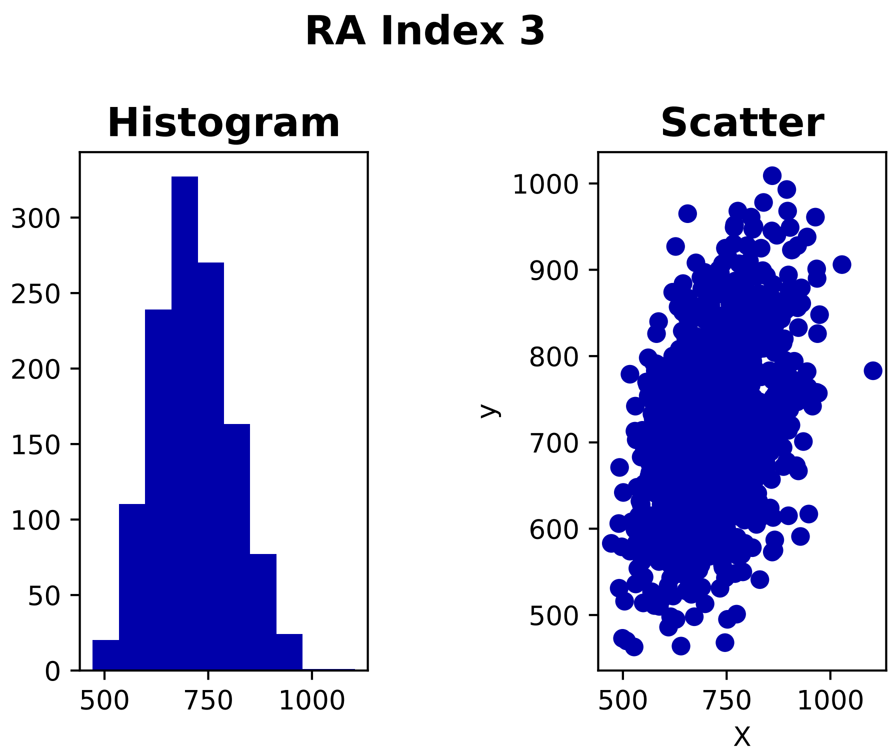


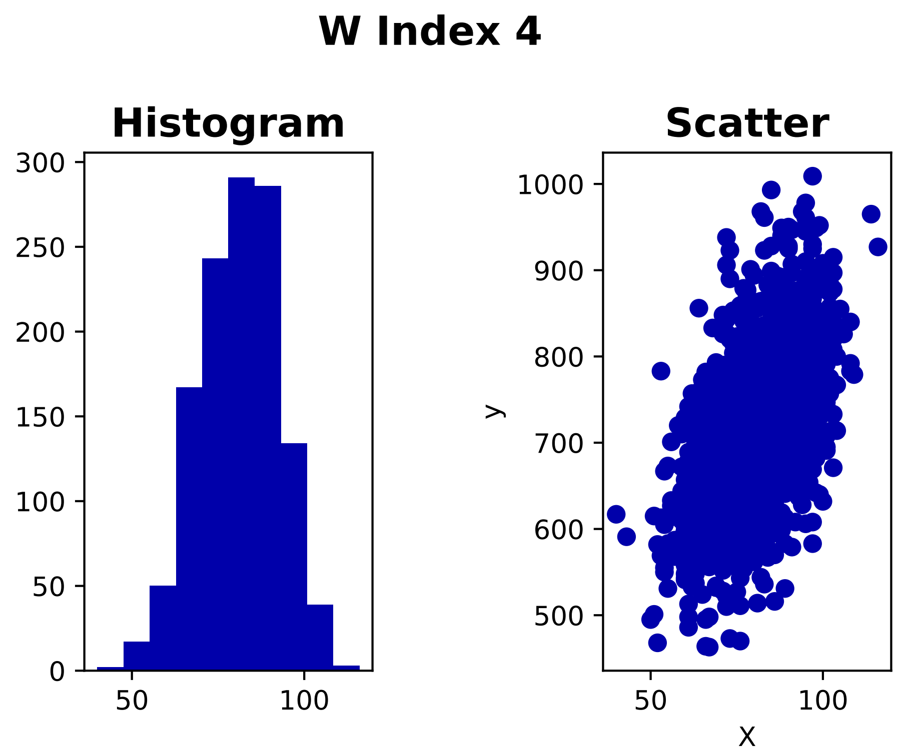
League Index 1

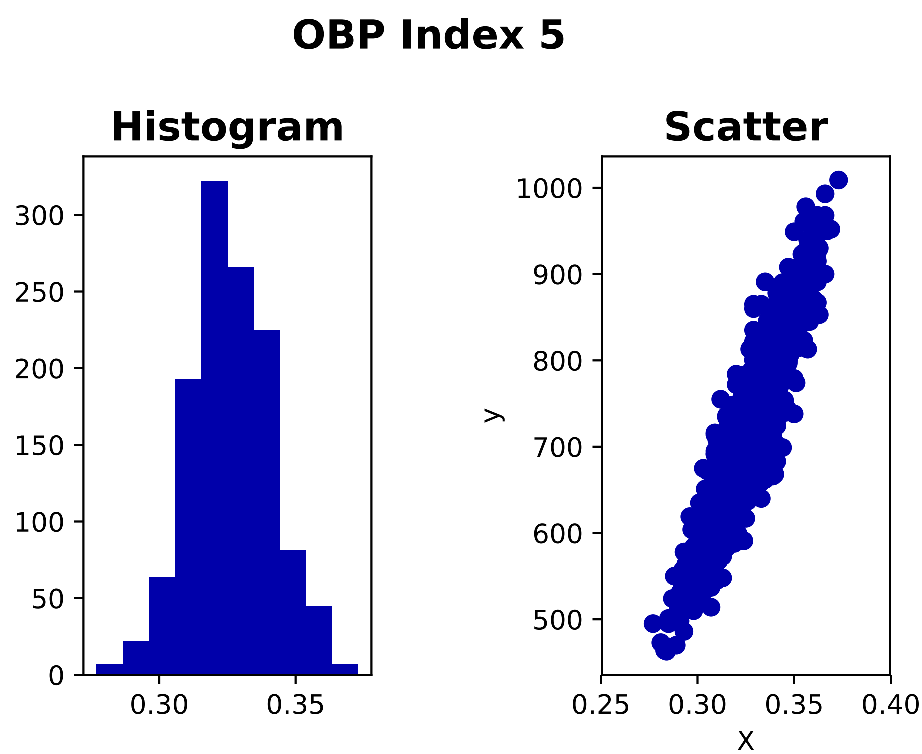


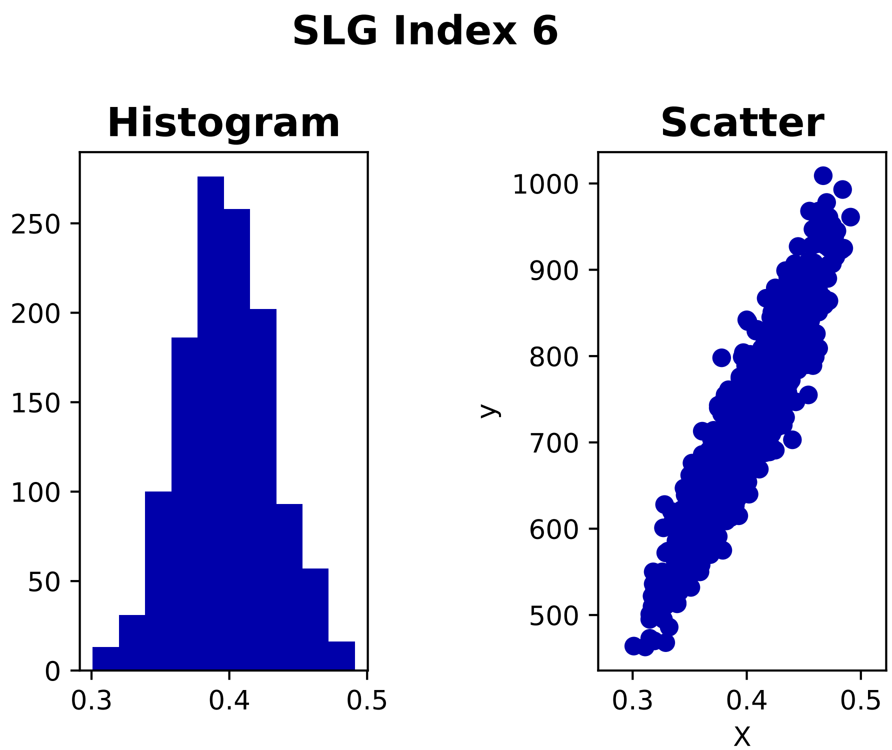
Year Index 2

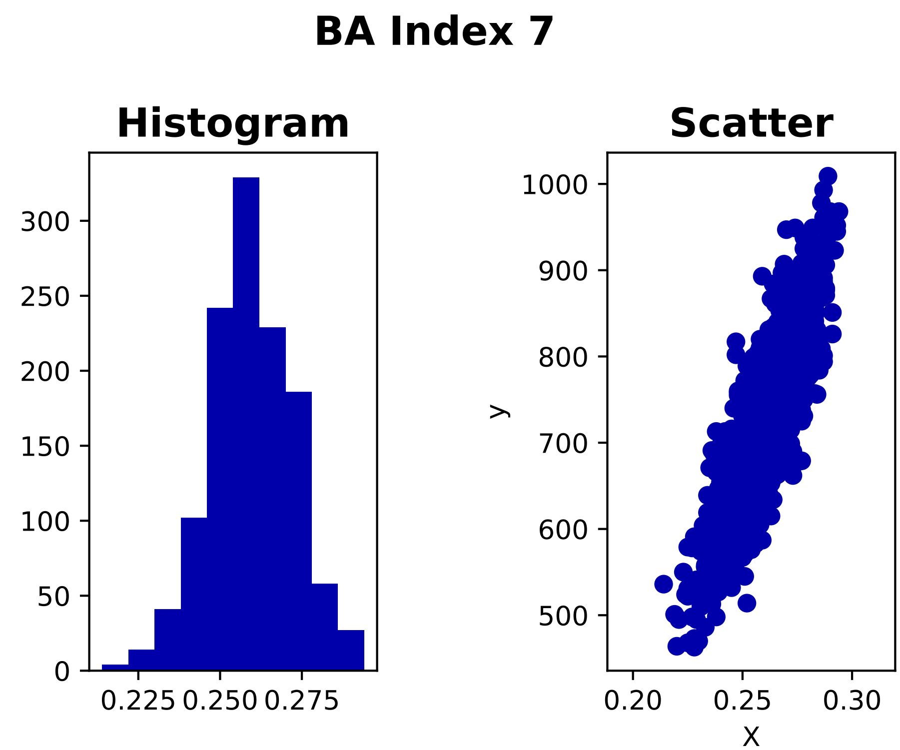


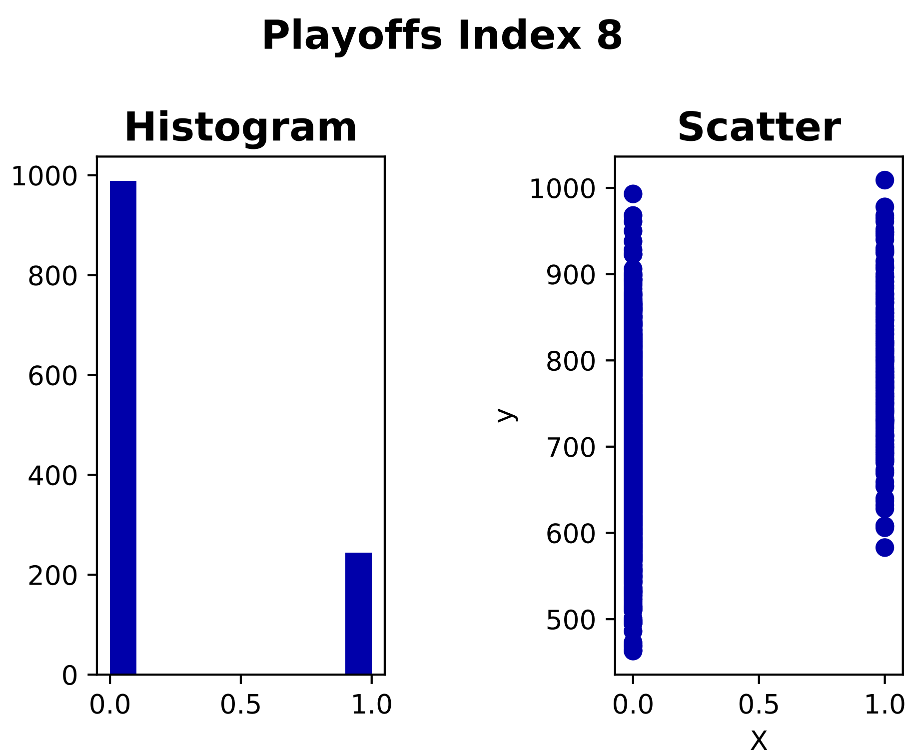

RA Index 3
Histogram
300
250
200
150
100
50
0
500
750
1000
Scatter
1000
900
800
700
600
500
y
500
750
1000
X


W Index 4
Histogram
Scatter
300
250
200
150
100
50
0
50
100
1000
900
800
700
600
500
y
X

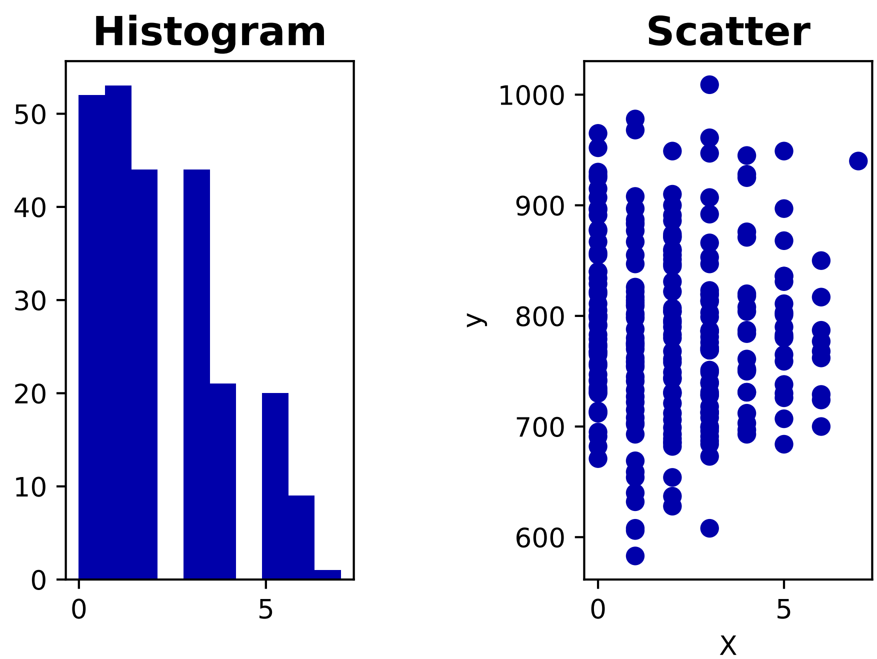

OBP Index 5
Histogram
Scatter
300
250
200
150
100
50
0
0.30
0.35
1000
900
800
700
600
500
y
0.25
0.30
0.35
0.40
X


SLG Index 6
Histogram
Scatter
0
50
100
150
200
250
0.3
0.4
0.5
1000
900
800
700
600
500
y
X

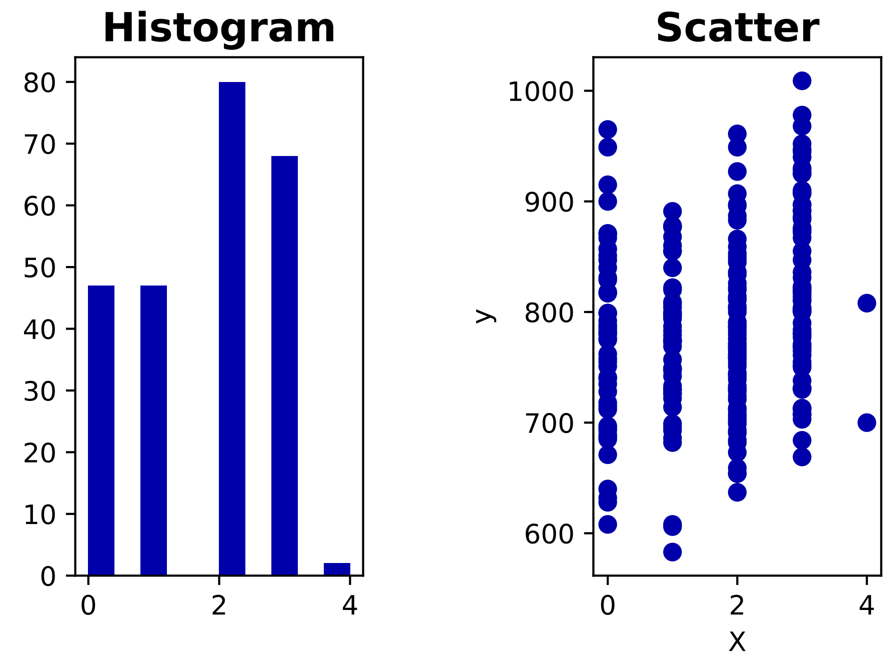

BA Index 7
Histogram
Scatter
300
250
200
150
100
50
0
0.225
0.250
0.275
1000
900
800
700
600
500
y
0.20
0.25
0.30
X


Playoffs Index 8
Histogram
Scatter
1000
800
600
400
200
0
0.0
0.5
1.0
1000
900
800
700
600
500
y
0.0
0.5
1.0
X

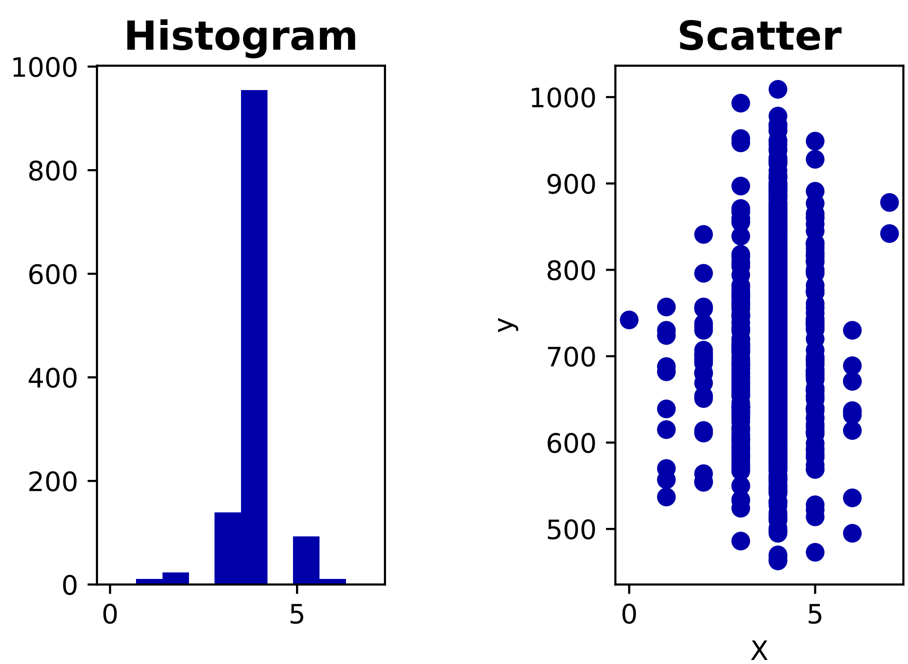
RankSeason Index 9

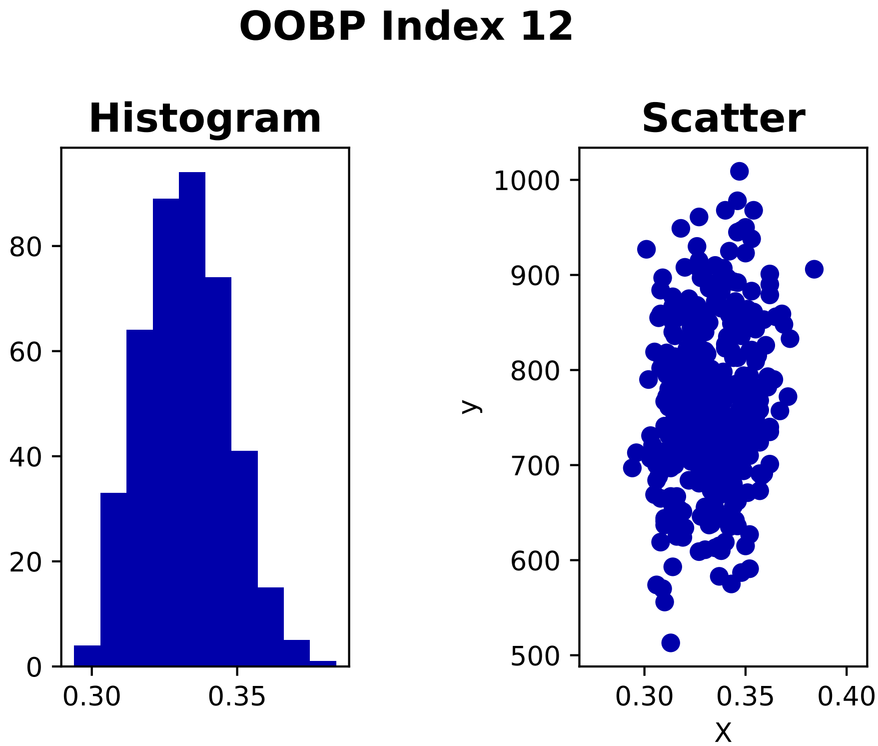


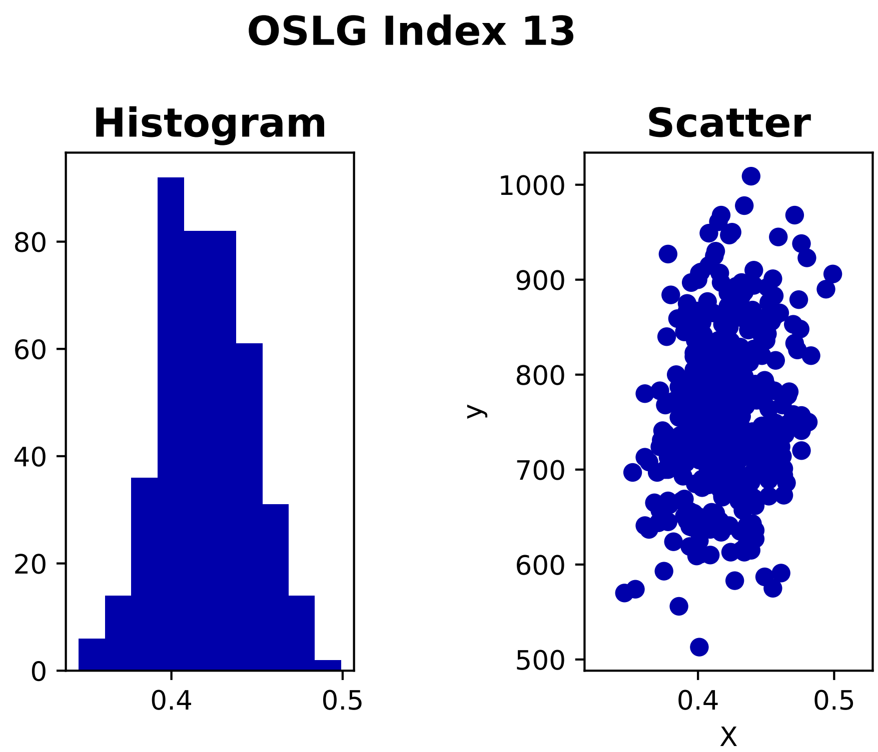
RankPlayoffs Index 10



G Index 11




OOBP Index 12
Histogram
Scatter
0
20
40
60
80
0.30
0.35
500
600
700
800
900
1000
0.30
0.35
0.40
X
y




```
In [5]: '''
        1 . Visually explore the data. Plot the distribution of each feature (e.g. histograms)
        - Feel free to create additional plots that help you understand the data
        - Only visualize the data, you don't need to change it (yet)

        **Answers:**
        The first column of figures shows the histograms of each column of the data in X.
        The second column shows the relationship between y and x.
        *Is there anything that stands out?*
        The dataset containts non real numbers, so the data set must first be cleaned to visua
        Beside the NaN the data the figures that do not show a clear distrubution are: 1, 8, 1


        Is there something that you think might require special treatment?
        '''

Out[5]: "\n1 . Visually explore the data. Plot the distribution of each feature (e.g. histogra
```

2 . Compare all linear regression algorithms that we covered in class (Linear Regression, Ridge, Lasso and ElasticNet), as well as kNN. Evaluate using cross-validation and the R^2 score, with the default parameters. Does scaling the data with StandardScaler help? Provide a concise but meaningful interpretation of the results. - Preprocess the data as needed (e.g. are there nominal features that are not ordinal?). If you don't know how to proceed, remove the feature and continue.

3 . Do a default, shuffled train-test split and optimize the linear models for the degree of regularization (*alpha*) and choice of penalty (L1/L2). For Ridge and Lasso, plot a curve showing the effect of the training and test set performance (R^2) while increasing the degree of regularization for different penalties. For ElasticNet, plot a heatmap $alpha \times l1\_ratio \rightarrow R^2$ using test set performance. Report the optimal performance. Again, provide a concise but meaningful interpretation. What does the regularization do? Can you get better results? - Think about how you get the L1/L2 loss. This is not a hyperparameter in regression. - We've seen how to generate such heatmaps in Lecture 3.

4 . Visualize the coefficients of the optimized models. Do they agree on which features are important? Compare the results with the feature importances returned by a RandomForest. Does it agree with the linear models? What would look for when scouting for a baseball player?

1.2 Nepalese character recognition (5 points, 1+2+2)

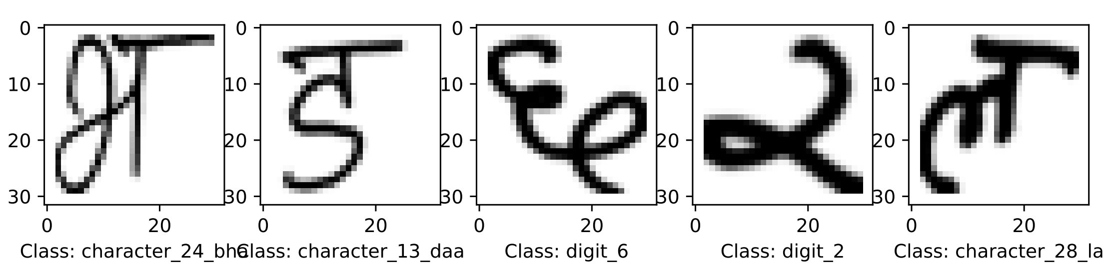
The Devnagari-Script dataset contains 92,000 images (32x32 pixels) of 46 characters from Devanagari script. Your goal is to learn to recognize the right letter given the image.

```
In [56]: # Initial the setting so the code can be run from this point
         from IPython.display import HTML
         HTML('''<style>html, body{overflow-y: visible !important} .CodeMirror{min-width:105%
         %matplotlib inline
         from preamble import *
         plt.rcParams['savefig.dpi'] = 200 # This controls the size of your figures
         # Comment out and restart notebook if you only want the last output of each cell.
         InteractiveShell.ast_node_interactivity = "all"
```

```
Out[56]: <IPython.core.display.HTML object>
```

```
In [57]: devnagari = oml.datasets.get_dataset(40923) # Download Devnagari data
         # Get the predictors X and the labels y
         X, y = devnagari.get_data(target=devnagari.default_target_attribute);
         if not 'classes' in locals():
             classes = devnagari.retrieve_class_labels(target_name='character') # This one tak
```

```
In [58]: from random import randint
         # Take some random examples, reshape to a 32x32 image and plot
         fig, axes = plt.subplots(1, 5,  figsize=(10, 5))
         for i in range(5):
             n = randint(0,90000)
             axes[i].imshow(X[n].reshape(32, 32), cmap=plt.cm.gray_r)
             axes[i].set_xlabel("Class: %s" % (classes[y[n]]))
         plt.show();
```



1. Evaluate k-Nearest Neighbors, Logistic Regression and RandomForests with their default settings.

    - Take a stratified 10% subsample of the data.
    - Use the default train-test split and predictive accuracy. Is predictive accuracy a good scoring measure for this problem?
    - Try to build the same models on increasingly large samples of the dataset (e.g. 10%, 20%,...). Plot the training time and the predictive performance for each. Stop when the training time becomes prohibitively large (this will be different for different models).

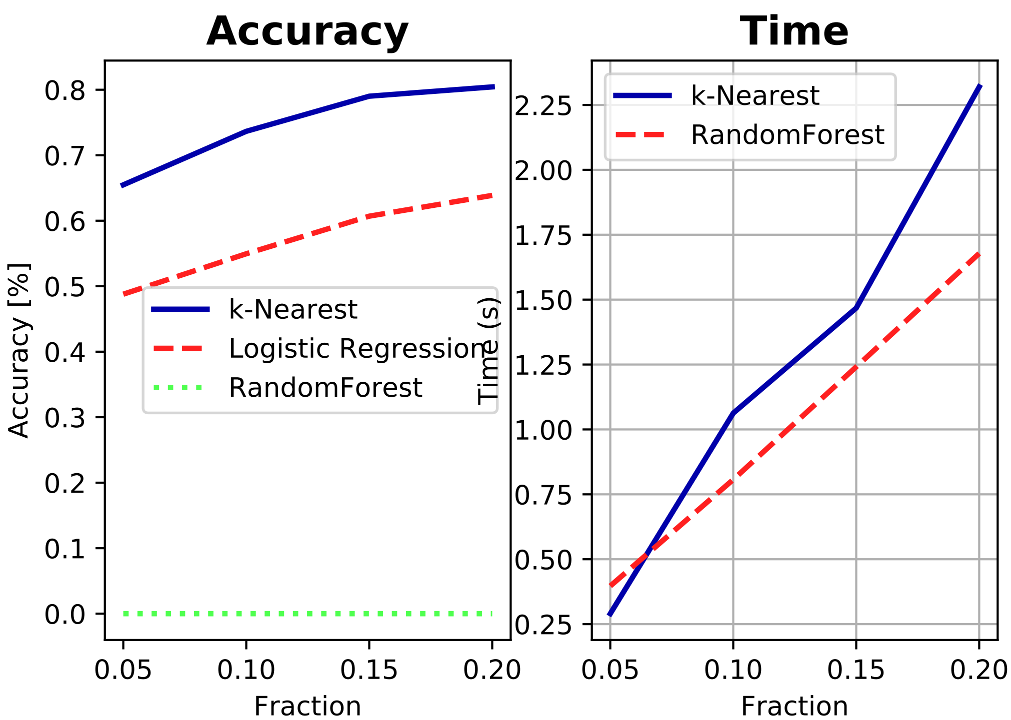
In [67]:
```
# Import the functions (standard variables are used)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
import time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt

def Question1a():

    Fraction=np.array([0.05,0.1,0.15,0.2])
                       #,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.80,0.85,
    A = np.zeros((len(Fraction), 2)) # Accuracy Matrix
    T = np.zeros((len(Fraction),2))  # Computation time Matrix

    # Split the data in test data and train data (stratified 10% subsample)
    from sklearn.model_selection import train_test_split

    #n1 = randint(0,len(y_test1))
    #n2 = randint(0,len(y_test1))

    for i in range(len(Fraction)):

        X_del, X_split, y_del, y_split = train_test_split(X, y, test_size=Fraction[i]
        X_train, X_test, y_train, y_test = train_test_split(X_split, y_split, test_si

        print(i)

        # Solve the learning problems if the solutions do not exist yet
        #if not 'classifier' in locals():
        tic = time.clock()
        knn = KNeighborsClassifier(n_neighbors=5)
        knn.fit(X_train, y_train)
```

```
        toc = time.clock()
        #y_predKNearest = knn.predict(X_test)
        Accuracy_KNearest=knn.score(X_test,y_test)
        A[i][0]=Accuracy_KNearest
        T[i][0]=toc-tic

        #if not 'Forest' in locals():
        tic = time.clock()
        Forest = RandomForestClassifier()
        Forest.fit(X_train,y_train)
        toc = time.clock()
        #y_predForest = Forest.predict(X_test)
        Accuracy_Forest=Forest.score(X_test,y_test) #accuracy_score(y_test,y_predFore
        A[i][1]=Accuracy_Forest
        T[i][1]=toc-tic

    xas=Fraction
    plot1=plt.subplot(1,2,1);
    plt.plot(xas,A[:,0],linewidth=2);
    plt.plot(xas,A[:,1],linewidth=2);
    plt.plot(xas,A[:,2],linewidth=2);
    plt.title('Accuracy',fontweight='bold',fontsize=15);
    plt.xlabel('Fraction');
    plt.ylabel('Accuracy [%]');
#     plot1.set_ylim([0, 1])

#     red_patch = mpatches.Patch(color='red', label='The red data')
    plt.legend(['k-Nearest','RandomForest'])
    #plt.grid()
    plt.subplot(1,2,2);
    plt.plot(xas,T[:,0],linewidth=2);
    plt.plot(xas,T[:,1],linewidth=2);
    plt.title('Time',fontweight='bold',fontsize=15);
    plt.xlabel('Fraction');
    plt.ylabel('Time (s)');
    plt.grid()
    plt.legend(['k-Nearest','RandomForest'])
    plt.show()
    return

def Question1b():

    length_data=len(y); #92000

    number_of_test=100; # To have more control of the computation time
    number_of_training=1000; # Not used

    Fractionb=np.array([0.02,0.04,0.06])
```

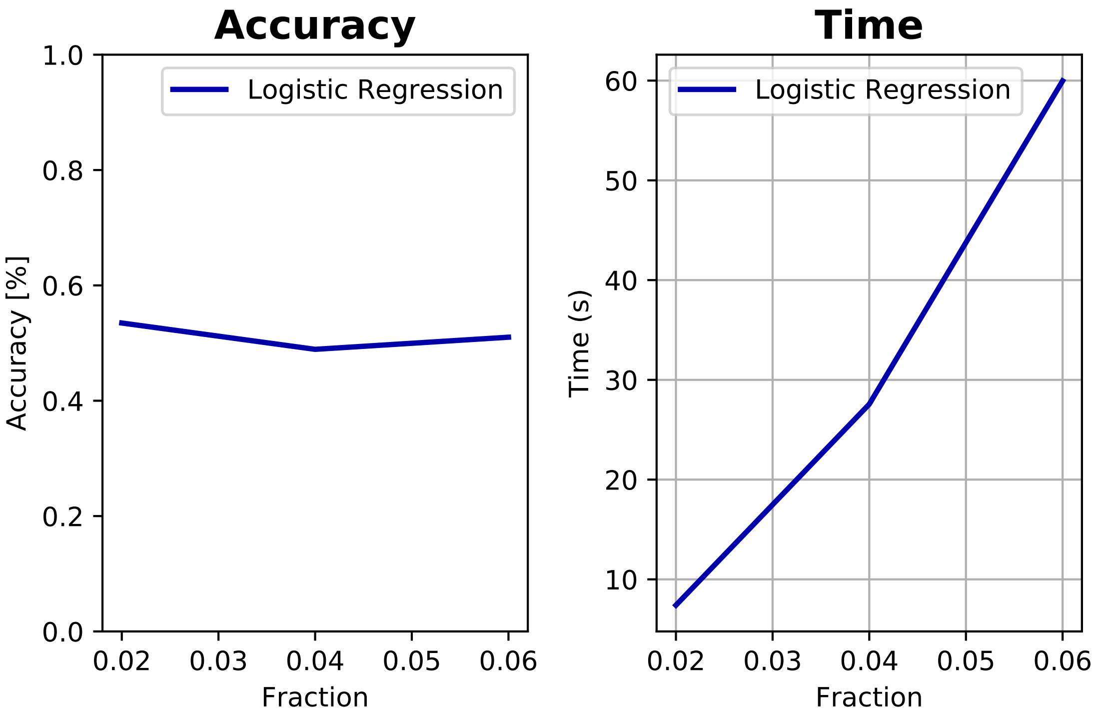
```
    A1 = np.zeros((len(Fractionb),1)) # Accuracy Matrix
    T1 = np.zeros((len(Fractionb),1))  # Computation time Matrix

    # Split the data in test data and train data (stratified 10% subsample)
    from sklearn.model_selection import train_test_split

    #n1 = randint(0,len(y_test1))
    #n2 = randint(0,len(y_test1))

    for i in range(len(Fractionb)):
        X_del, X_split, y_del, y_split = train_test_split(X, y, test_size=Fractionb[i]
        X_train, X_test, y_train, y_test = train_test_split(X_split, y_split, test_siz

        print(i)

        #if not 'logistic' in locals():
        tic = time.clock()
        logistic = LogisticRegression()
        logistic.fit(X_train,y_train)
        toc = time.clock()
#         y_predLogistic = logistic.predict(X_test)
        Accuracy_Logistic=logistic.score(X_test,y_test) #accuracy_score(y_test,y_pred
        A1[i][0]=Accuracy_Logistic
        T1[i][0]=toc-tic

    xas=Fractionb
    plt.figure()
    plot1=plt.subplot(1,2,1);
    plt.plot(xas,A1[:,0],linewidth=2);
    plt.title('Accuracy',fontweight='bold',fontsize=15);
    plt.xlabel('Fraction');
    plt.ylabel('Accuracy [%]');
    plot1.set_ylim([0, 1])

#   red_patch = mpatches.Patch(color='red', label='The red data')
    plt.legend(['Logistic Regression'])
    #plt.grid()
    plt.subplot(1,2,2);
    plt.plot(xas,T1[:,0],linewidth=2);
    plt.title('Time',fontweight='bold',fontsize=15);
    plt.xlabel('Fraction');
    plt.ylabel('Time (s)');
    plt.grid()
    plt.legend(['Logistic Regression'])
    plt.tight_layout()
    plt.show()
```

```
    return

Question1a()
Question1b()
```

```
0
1
2
3
```




```
0
1
2
```



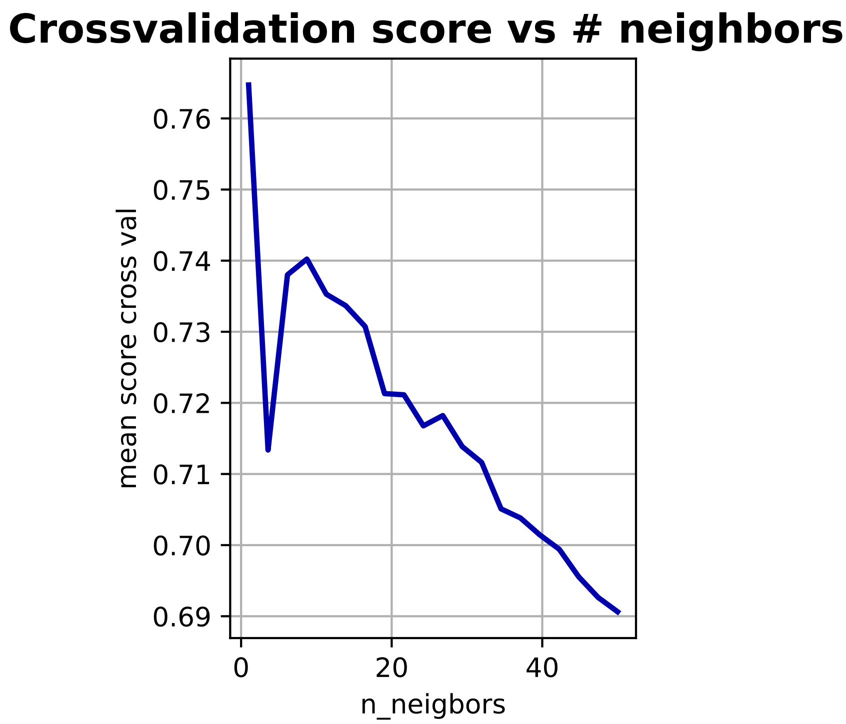
2 . Optimize the value for the number of neighbors k (keep $k < 50$) and the number of trees (keep $n\_estimators < 100$) on the stratified 10% subsample. Use 10-fold crossvalidation and plot k and $n\_estimators$ against the predictive accuracy. Which value of k, $n\_estimators$ should you pick?

```
In [68]: from sklearn.model_selection import cross_val_score

         def Question2():

         #    length_data=len(y); #92000

             Fraction=np.array([0.1])

             n_neighborsA=np.linspace(1,50,20)
             n_estimatorsA=np.linspace(1,100,20)

             A1 = np.zeros((len(n_neighborsA), 3)) # Accuracy Matrix
             T1 = np.zeros((len(n_neighborsA),3))  # Computation time Matrix

             A2 = np.zeros((len(n_estimatorsA), 3)) # Accuracy Matrix
             T2 = np.zeros((len(n_estimatorsA),3))  # Computation time Matrix

             # Split the data in test data and train data (stratified 10% subsample)
             from sklearn.model_selection import train_test_split

         #     X_del, X_train, y_del, y_train = train_test_split(X, y, test_size = Fraction[0]
```

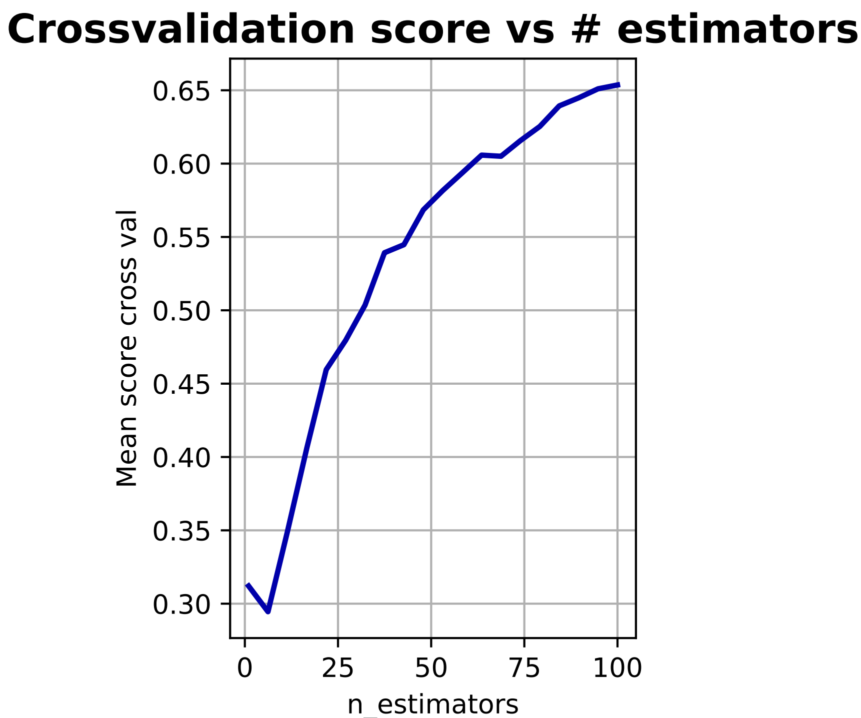
```
#     X_del, X_test, y_del, y_test = train_test_split(X, y, test_size = number_of_tes

    X_del, X_split, y_del, y_split = train_test_split(X, y, test_size=Fraction[0])
    X_train, X_test, y_train, y_test = train_test_split(X_split, y_split, test_size=0

    X_train.shape
    y_train.shape

    for i in range(len(n_neighborsA)):

        # Reduce the test data as well to 10 samples
        print("knn iteration: %d " % i)
        # Solve the learning problems if the solutions do not exist yet
        #if not 'classifier' in locals():
        tic = time.clock()
        knn = KNeighborsClassifier(n_neighbors=i+1)
        toc = time.clock()

        scores = cross_val_score(knn, X_train, y_train, cv=10)
        Accuracy_knn=np.mean(scores)

#         Accuracy_KNearest=accuracy_score(y_test,y_predKNearest)
        A1[i][0]=Accuracy_knn
        T1[i][0]=toc-tic

#       if not 'logistic' in locals(): # There is no n_neighbors or # of trees option
        '''
        tic = time.clock()
        logistic = LogisticRegression()
        logistic.fit(X_train,y_train)
        toc = time.clock()
        y_predLogistic = logistic.predict(X_test)
        Accuracy_Logistic=accuracy_score(y_test,y_predLogistic)
        A[i][1]=Accuracy_Logistic
        T[i][1]=toc-tic
        '''

        #if not 'Forest' in locals():

    plt.subplot(1,2,1);
    plt.plot(n_neighborsA,A1[:,0],linewidth=2);
    plt.title('Crossvalidation score vs # neighbors ',fontweight='bold',fontsize=15);
    plt.xlabel('n_neigbors');
    plt.ylabel('mean score cross val');
    plt.grid()
    plt.show()

    for i in range(len(n_estimatorsA)):
```

```python
            print("Random Forest iteration: %d " % i)
            tic = time.clock()
            Forest = RandomForestClassifier(n_estimators=i+1) # Number of trees
            toc = time.clock()

            scores = cross_val_score(Forest, X_train, y_train, cv=10)
            Accuracy_Forest=np.mean(scores)

            A2[i][0]=Accuracy_Forest
            T2[i][0]=toc-tic

        plt.subplot(1,2,2);
        plt.plot(n_estimatorsA,A2[:,0],linewidth=2);
        plt.title('Crossvalidation score vs # estimators',fontweight='bold',fontsize=15);
        plt.xlabel('n_estimators');
        plt.ylabel('Mean score cross val');
        plt.grid()
        plt.show()

        return

    Question2()
```

```
knn iteration: 0
knn iteration: 1
knn iteration: 2
knn iteration: 3
knn iteration: 4
knn iteration: 5
knn iteration: 6
knn iteration: 7
knn iteration: 8
knn iteration: 9
knn iteration: 10
knn iteration: 11
knn iteration: 12
knn iteration: 13
knn iteration: 14
knn iteration: 15
knn iteration: 16
knn iteration: 17
knn iteration: 18
knn iteration: 19
```




```
Random Forest iteration: 0
Random Forest iteration: 1
Random Forest iteration: 2
Random Forest iteration: 3
Random Forest iteration: 4
Random Forest iteration: 5
Random Forest iteration: 6
Random Forest iteration: 7
Random Forest iteration: 8
Random Forest iteration: 9
Random Forest iteration: 10
Random Forest iteration: 11
Random Forest iteration: 12
Random Forest iteration: 13
Random Forest iteration: 14
Random Forest iteration: 15
Random Forest iteration: 16
Random Forest iteration: 17
Random Forest iteration: 18
Random Forest iteration: 19
```



3 . For the RandomForest, optimize both *n_estimators* and *max_features* at the same time on the entire dataset. - Use a nested cross-validation and a random search over the possible values, and measure the accuracy. Explore how fine-grained this grid/random search can be, given your computational resources. What is the optimal performance you find? - Hint: choose a nested cross-validation that is feasible. Don't use too many folds in the outer loop. - Repeat the grid search and visualize the results as a plot (heatmap) $n\_estimators \times max\_features \rightarrow ACC$ with ACC visualized as the color of the data point. Try to make the grid as fine as possible. Interpret the results. Can you explain your observations? What did you learn about tuning RandomForests?

```
In [72]: from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import GridSearchCV
         from sklearn.pipeline import Pipeline
         from sklearn.pipeline import make_pipeline
         import matplotlib.patches as mpatches
         plt.rcParams['savefig.dpi'] = 1000 # This controls the size of your figures
         def Question3():
         #    length_data=len(y); #92000

             number_of_test=100; # To have more control of the computation time
             Fraction=np.array([0.01])

             n_estimatorsAF=np.linspace(1,100,20)
             n_estimatorsAFR=np.round(n_estimatorsAF)
             n_estimatorsA=n_estimatorsAFR.astype(int)
```

```
print(n_estimatorsA)
max_featuresAF=np.linspace(1,100,10)
max_featuresAFR=np.round(max_featuresAF)
max_featuresA=max_featuresAFR.astype(int)
print(max_featuresA)
A1 = np.zeros((len(n_estimatorsA),len(max_featuresA),1)) # Accuracy Matrix
A2 = np.zeros((len(n_estimatorsA),len(max_featuresA),1)) # Accuracy Matrix
T1 = np.zeros((len(n_estimatorsA),len(max_featuresA),1))  # Computation time Matr

# Split the data in test data and train data (stratified 10% subsample)
from sklearn.model_selection import train_test_split

X_del, X_split, y_del, y_split = train_test_split(X, y, test_size=Fraction[0])
X_train, X_test, y_train, y_test = train_test_split(X_split, y_split, test_size=0

p_grid = {"n_estimators": n_estimatorsA,
      "max_features": max_featuresA}

for i in range(len(n_estimatorsA)):
    print("Random Forest iteration: %d " % i)
    for n in range(len(max_featuresA)):

        inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
        outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
        svc=            Forest = RandomForestClassifier() # Number of trees
        clf = GridSearchCV(svc, param_grid=p_grid, cv=inner_cv)
        clf.fit(X_train, y_train)
        non_nested_scores = clf.best_score_

        # Nested CV with parameter optimization
        nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
        A2[i][n][0] = nested_score.mean()


        tic = time.clock()
        Forest = RandomForestClassifier(n_estimators=n_estimatorsA[i], max_feature
        Forest_outer = RandomForestClassifier(n_estimators=n_estimatorsA[i], max_
        toc = time.clock()

        Forest.fit(X_train, y_train)
        scores = cross_val_score(Forest, X_train, y_train, cv=10)
        Accuracy_Forest=scores.mean()

        A1[i][n][0]=Accuracy_Forest
        T1[i][n][0]=toc-tic

A1.shape=(len(n_estimatorsA),len(max_featuresA))
param_grid = [{'n_estimatorsA': n_estimatorsA,
```

```
                 'max_featuresA': max_featuresA}]
    im1=mglearn.tools.heatmap(A1, xlabel='n_estimators', xticklabels=n_estimatorsA,
                              ylabel='max_features', yticklabels=max_featuresA, cmap="viridis"


#     plt.xlabel("# estimators")
#     plt.ylabel("# features")
    plt.title('Accuracy for # estimators and # features',fontweight='bold',fontsize=15
    plt.xticks(range(len(n_estimatorsA)))
    plt.yticks(range(len(max_featuresA)))
    values = np.unique(A1.ravel())
    colors = [ im1.cmap(im1.norm(value)) for value in values]

    plt.figure()
    A2.shape=(len(n_estimatorsA),len(max_featuresA))
    param_grid = [{'n_estimatorsA': n_estimatorsA,
                 'max_featuresA': max_featuresA}]
    im1=mglearn.tools.heatmap(A1, xlabel='n_estimators', xticklabels=n_estimatorsA,
                              ylabel='max_features', yticklabels=max_featuresA, cmap="viridis"
    plt.title('Accuracy for # estimators and # features',fontweight='bold',fontsize=15
    plt.xticks(range(len(n_estimatorsA)))
    plt.yticks(range(len(max_featuresA)))
    values = np.unique(A2.ravel())
    colors = [ im1.cmap(im1.norm(value)) for value in values]
    # create a patch (proxy artist) for every color
#      =[ mpatches.Patch(color=colors[i], label="{l}".format(l=np.round(values[i],3))
    # put those patched as legend-handles into the legend
#     plt.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0. )
    print(np.min(A1))
    print(np.max(A1))
    return


Question3()
```

```
[  1   6  11  17  22  27  32  37  43  48  53  58  64  69  74  79  84  90
  95 100]
[  1  12  23  34  45  56  67  78  89 100]
Random Forest iteration: 0
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-72-62019a690a66> in <module>()
 95
```

```
     96
---> 97 Question3()
     98


    <ipython-input-72-62019a690a66> in Question3()
     44
     45             # Nested CV with parameter optimization
---> 46             nested_score = cross_val_score(clf, X=X_train, y=y_train, cv=outer_cv)
     47             A2[i][n][0] = nested_score.mean()
     48


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selection\_validati
    340                                   n_jobs=n_jobs, verbose=verbose,
    341                                   fit_params=fit_params,
--> 342                                   pre_dispatch=pre_dispatch)
    343     return cv_results['test_score']
    344


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selection\_validati
    204             fit_params, return_train_score=return_train_score,
    205             return_times=True)
--> 206         for train, test in cv.split(X, y, groups))
    207
    208     if return_train_score:


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    777             # was dispatched. In particular this covers the edge
    778             # case of Parallel used with an exhausted iterator.
--> 779             while self.dispatch_one_batch(iterator):
    780                 self._iterating = True
    781             else:


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batch), self)
```

```
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\_paralle
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\_paralle
    330         # Don't delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333
    334     def get(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\parallel
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\parallel
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selection\_validati
    456             estimator.fit(X_train, **fit_params)
    457         else:
--> 458             estimator.fit(X_train, y_train, **fit_params)
    459
    460     except Exception as e:


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selection\_search.
    637                                   error_score=self.error_score)
    638           for parameters, (train, test) in product(candidate_params,
```

```
--> 639                                                    cv.split(X, y, groups)))
    640
    641         # if one choose to see train score, "out" will contain train score info


~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\parallel
    777             # was dispatched. In particular this covers the edge
    778             # case of Parallel used with an exhausted iterator.
--> 779             while self.dispatch_one_batch(iterator):
    780                 self._iterating = True
    781             else:


~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\parallel
    623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627


~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\parallel
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batch), self)
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590


~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\_paralle
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)


~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\_paralle
    330         # Don't delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333
    334     def get(self):


~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\parallel
    129
    130     def __call__(self):
```

```
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\model_selection\_validat
    456             estimator.fit(X_train, **fit_params)
    457         else:
--> 458             estimator.fit(X_train, y_train, **fit_params)
    459
    460     except Exception as e:


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ensemble\forest.py in fi
    326                     t, self, X, y, sample_weight, i, len(trees),
    327                     verbose=self.verbose, class_weight=self.class_weight)
--> 328                 for i, t in enumerate(trees))
    329
    330             # Collect newly grown trees


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    777             # was dispatched. In particular this covers the edge
    778             # case of Parallel used with an exhausted iterator.
--> 779             while self.dispatch_one_batch(iterator):
    780                 self._iterating = True
    781             else:


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    623                 return False
    624             else:
--> 625                 self._dispatch(tasks)
    626                 return True
    627


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    586         dispatch_timestamp = time.time()
    587         cb = BatchCompletionCallBack(dispatch_timestamp, len(batch), self)
```

```
--> 588         job = self._backend.apply_async(batch, callback=cb)
    589         self._jobs.append(job)
    590


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\_paralle
    109     def apply_async(self, func, callback=None):
    110         """Schedule a func to be run"""
--> 111         result = ImmediateResult(func)
    112         if callback:
    113             callback(result)


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\_paralle
    330         # Don't delay the application, to avoid keeping the input
    331         # arguments in memory
--> 332         self.results = batch()
    333
    334     def get(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\externals\joblib\paralle
    129
    130     def __call__(self):
--> 131         return [func(*args, **kwargs) for func, args, kwargs in self.items]
    132
    133     def __len__(self):


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ensemble\forest.py in _pa
    119             curr_sample_weight *= compute_sample_weight('balanced', y, indices)
    120
--> 121         tree.fit(X, y, sample_weight=curr_sample_weight, check_input=False)
    122     else:
    123         tree.fit(X, y, sample_weight=sample_weight, check_input=False)


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\tree\tree.py in fit(self
    788             sample_weight=sample_weight,
    789             check_input=check_input,
```

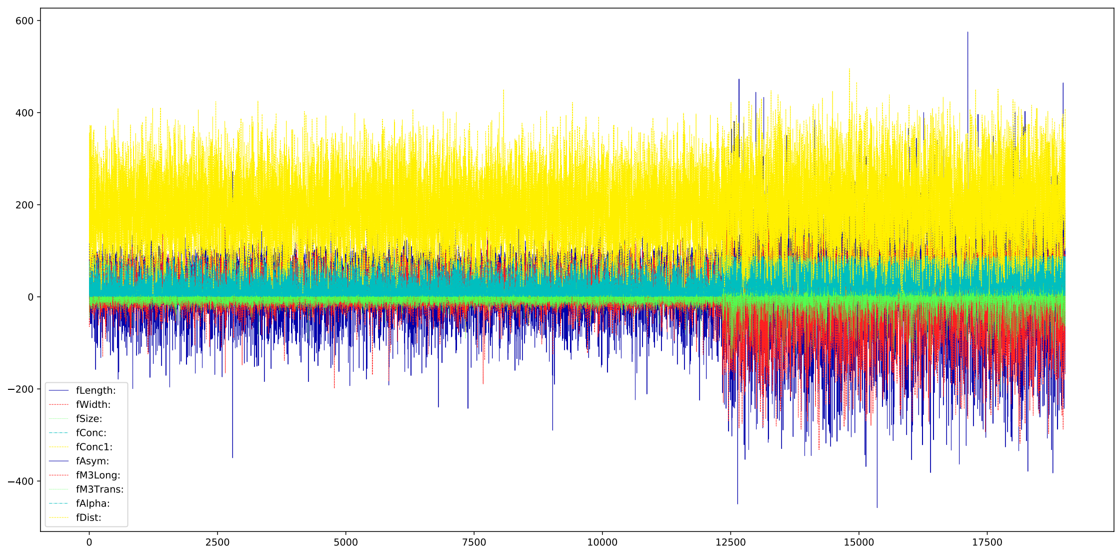
```
--> 790             X_idx_sorted=X_idx_sorted)
    791         return self
    792


    ~\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\tree\tree.py in fit(self
    360                                             min_impurity_split)
    361
--> 362         builder.build(self.tree_, X, y, sample_weight, X_idx_sorted)
    363
    364         if self.n_outputs_ == 1:


    KeyboardInterrupt:
```

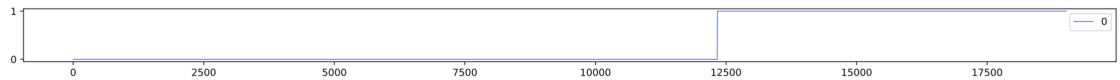
1.3 3. Understanding Ensembles (5 points (3+2))

Do a deeper analysis of how RandomForests and Gradient Boosting reduce their prediction error. We'll use the MAGIC telescope dataset (http://www.openml.org/d/1120). When high-energy particles hit the atmosphere, they produce chain reactions of other particles called 'showers', and you need to detect whether these are caused by gamma rays or cosmic rays.

In [73]:
```
# Get the data
magic_data = oml.datasets.get_dataset(1120) # Download MAGIC Telescope data
X, y = magic_data.get_data(target=magic_data.default_target_attribute);
```

In [74]:
```
# Quick visualization
X, y, attribute_names = magic_data.get_data(target=magic_data.default_target_attribute
magic = pd.DataFrame(X, columns=attribute_names)
magic.plot(figsize=(20,10))
# Also plot the target: 1 = gamma, 0 = background
pd.DataFrame(y).plot(figsize=(20,1));
```





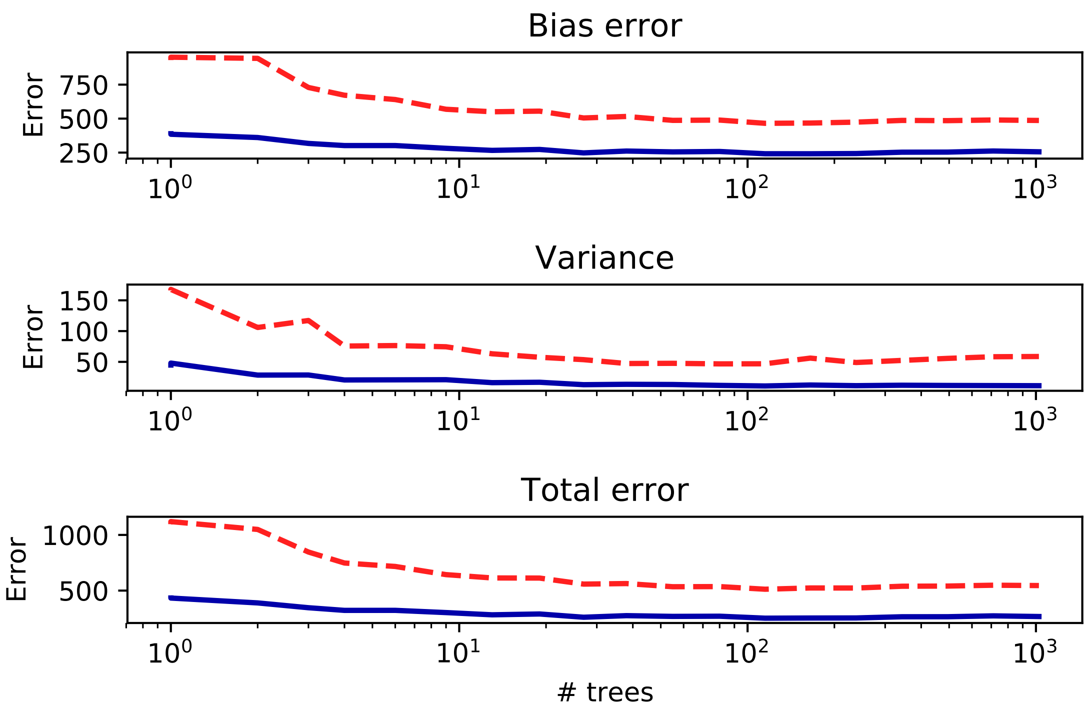
1 . Do a bias-variance analysis of both algorithms. For each, vary the number of trees on a log scale from 1 to 1024, and plot the bias error (squared), variance, and total error (in one plot per algorithm). Interpret the results. Which error is highest for small ensembles, and which reduced most by each algorithm as you use a larger ensemble? When are both algorithms under- or overfitting? Provide a detailed explanation of why random forests and gradient boosting behave this way. - See lecture 3 for an example on how to do the bias-variance decomposition - To save time, you can use a 10% stratified subsample in your initial experiments, but show the plots for the full dataset in your report.

```
In [75]: from sklearn.model_selection import ShuffleSplit
         from sklearn import ensemble


         def Question4():
             # Bootstraps
             n_treesAF=np.logspace(np.log10(1),np.log10(1024),20)
             n_treesAFR=np.round(n_treesAF)
             n_treesA=n_treesAFR.astype(int)
             print(n_treesA)
             n_repeat = 5
             shuffle_split = ShuffleSplit(train_size=0.1, n_splits=n_repeat)

             bias_sq=np.zeros((len(n_treesA),1))
             var=np.zeros((len(n_treesA),1))
             error=np.zeros((len(n_treesA),1))
             bias_sqG=np.zeros((len(n_treesA),1))
             varG=np.zeros((len(n_treesA),1))
             errorG=np.zeros((len(n_treesA),1))

             for n in range(len(n_treesA)):
                 # Store sample predictions
                 y_all_pred = [[] for _ in range(len(y))]

                 # Train classifier on each bootstrap and score predictions
                 for i, (train_index, test_index) in enumerate(shuffle_split.split(X)):
                     # Train and predict
                     clf =  RandomForestClassifier(n_estimators=n_treesA[n]) #ensemble.Gradien
                     clf.fit(X[train_index], y[train_index])
                     y_pred = clf.predict(X[test_index])
```

```
        # Store predictions
        for i,index in enumerate(test_index):
            y_all_pred[index].append(y_pred[i])

            # Compute bias, variance, error
        bias_sumi=0
        var_sumi=0
        error_sumi=0
        for i in range(len(y_all_pred)):
            x=y_all_pred[i]
            if not len(x)==0:
                bias_sumi=bias_sumi+(1 - x.count(y[i])/len(x))**2 * len(x)/n_repea
                var_sumi=var_sumi+((1 - ((x.count(0)/len(x))**2 + (x.count(1)/len
                error_sumi=error_sumi+(1 - x.count(y[i])/len(x)) * len(x)/n_repea
    bias_sq[n]=bias_sumi
    var[n]=var_sumi
    error[n]=error_sumi
    #print("Forest tree Bias squared: %.2f, Variance: %.2f, Total error: %.2f" %
    # Train classifier on each bootstrap and score predictions
    for i, (train_index, test_index) in enumerate(shuffle_split.split(X)):
        # Train and predict
        clf =  ensemble.GradientBoostingClassifier(n_estimators=n_treesA[n])
        clf.fit(X[train_index], y[train_index])
        y_pred = clf.predict(X[test_index])

        # Store predictions
        for i,index in enumerate(test_index):
            y_all_pred[index].append(y_pred[i])

            # Compute bias, variance, error
        bias_sumi=0
        var_sumi=0
        error_sumi=0
        for i in range(len(y_all_pred)):
            x=y_all_pred[i]
            if not len(x)==0:
                bias_sumi=bias_sumi+(1 - x.count(y[i])/len(x))**2 * len(x)/n_repea
                var_sumi=var_sumi+((1 - ((x.count(0)/len(x))**2 + (x.count(1)/len
                error_sumi=error_sumi+(1 - x.count(y[i])/len(x)) * len(x)/n_repea

        bias_sqG[n]=bias_sumi
        varG[n]=var_sumi
        errorG[n]=error_sumi
    #print("Gradient Boosting Bias squared: %.2f, Variance: %.2f, Total error: %.
    print('Iteration %d' %n)
plt.subplot(3,1,1);
plt.semilogx(n_treesA, bias_sq,linewidth=2);
plt.semilogx(n_treesA, bias_sqG,linewidth=2);
```

```python
    plt.ylabel('Error');
    plt.title('Bias error');
    plt.subplot(3,1,2);
    plt.semilogx(n_treesA, var,linewidth=2);
    plt.semilogx(n_treesA, varG,linewidth=2);
    plt.ylabel('Error');
    plt.title('Variance');
    plt.subplot(3,1,3);
    plt.semilogx(n_treesA, error,linewidth=2);
    plt.semilogx(n_treesA, errorG,linewidth=2);
    plt.ylabel('Error');
    plt.title('Total error')
    plt.xlabel('# trees');
    plt.tight_layout()


    plt.show()
    return

Question4()
```

```
[   1    1    2    3    4    6    9   13   19   27   38   55   80  115
  165  238  343  494  711 1024]
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
Iteration 16
Iteration 17
Iteration 18
Iteration 19
```



2 . A *validation curve* can help you understand when a model starts under- or overfitting. It plots both training and test set error as you change certain characteristics of your model, e.g. one or more hyperparameters. Build validation curves for gradient boosting, evaluated using AUROC, by varying the number of iterations between 1 and 500. In addition, use at least two values for the learning rate (e.g. 0.1 and 1), and tree depth (e.g. 1 and 4). This will yield at least 4 curves. Interpret the results and provide a clear explanation for the results. When is the model over- or underfitting? Discuss the effect of the different combinations learning rate and tree depth and provide a clear explanation. - While scikit-learn has a `validation_curve` function, we'll use a modified version (below) that provides a lot more detail and can be used to study more than one hyperparameter. You can use a default train-test split.

In [35]:
```
# Plots validation curves for every classifier in clfs.
# Also indicates the optimal result by a vertical line
# Uses 1-AUROC, so lower is better
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score

def validation_curve(clfs, X_test, y_test, X_train, y_train):
    for n,clf in enumerate(clfs):
        #print(vars(clf))
        #print(dir(clf))
        clf.fit(X_train,y_train)
        test_score = np.empty(len(clf.estimators_))
        train_score = np.empty(len(clf.estimators_))
```

```
        for i, pred in enumerate(clf.staged_decision_function(X_test)):
            test_score[i] = 1-roc_auc_score(y_test, pred)

        for i, pred in enumerate(clf.staged_decision_function(X_train)):
            train_score[i] = 1-roc_auc_score(y_train, pred)

        best_iter = np.argmin(test_score)
        learn = clf.get_params()['learning_rate']
        depth = clf.get_params()['max_depth']
        test_line = plt.plot(test_score,
                             label='learn=%.1f depth=%i (%.2f)'%(learn,depth,
                                                                  test_score[best_iter]

        colour = test_line[-1].get_color()
        plt.plot(train_score, '--', color=colour,linewidth=2)

        plt.xlabel("Number of boosting iterations")
        plt.ylabel("1 - area under ROC")
        plt.axvline(x=best_iter, color=colour)

    plt.legend(loc='best')


number_of_test=1000;
# To have more control of the computation time
Fraction=np.array([1])

n_estimatorsAF=np.linspace(1,10,2)
n_estimatorsAFR=np.round(n_estimatorsAF)
n_estimatorsA=n_estimatorsAFR.astype(int)

max_featuresAF=np.linspace(1,10,2)
max_featuresAFR=np.round(max_featuresAF)
max_featuresA=max_featuresAFR.astype(int)
# print(max_featuresA)
# A1 = np.zeros((len(n_estimatorsA),len(max_featuresA),1)) # Accuracy Matrix
# A2 = np.zeros((len(n_estimatorsA),len(max_featuresA),1)) # Accuracy Matrix
# T1 = np.zeros((len(n_estimatorsA),len(max_featuresA),1))  # Computation time Matrix

# Split the data in test data and train data (stratified 10% subsample)
from sklearn.model_selection import train_test_split


X_del, X_split, y_del, y_split = train_test_split(X, y, test_size = 0.5)
X_train, X_test, y_train, y_test = train_test_split(X_split, y_split, test_size = 0.5

classifiers = [
    GradientBoostingClassifier(n_estimators=500,learning_rate=0.1,max_depth=1, random_
```

```
    GradientBoostingClassifier(n_estimators=500,learning_rate=0.1,max_depth=4, random_
    GradientBoostingClassifier(n_estimators=500,learning_rate=1,max_depth=1, random_st
    GradientBoostingClassifier(n_estimators=500,learning_rate=1,max_depth=4, random_st
#     GradientBoostingClassifier(n_estimators=10,random_state=0, learning_rate=0.5)
    ]

validation_curve(classifiers,X_test=X_test,y_test=y_test,X_train=X_train,y_train=y_tra
#validation_curve(classifiers,X_train=,y_train,X_test,y_test)
```

Out[35]: '\ntrain_scores, test_scores = validation_curve(\n   SVC(), X_train, y_train, param_r

Out[35]: '\nimport plot_classifiers as pc\nfrom sklearn.ensemble import GradientBoostingClassi