Project # 3

Original Dataset URL: https://www.kaggle.com/dansbecker/nba-shot-logs

Implementation

The dataset "shot_logs.csv" of the link above has 21 attributes and 128069 data points. The dataset I used in this project is "NBA_shot_logs_2014_2015_reduced_at1", a modified version of the original dataset. Its size of the data points is reduced to 945 by clustering (project#2).

The following 10 attributes are selected among 21 attributes:

W, FINAL_MARGIN, SHOT_NUMBER, PERIOD, SHOT_CLOCK, DRIBBLES, TOUCH_TIME, SHOT_DIST(ft), PTS_TYPE, CLOSE_DEF_DIST(ft)

"6. Biplot" is missing because I failed to implement it.

1. Bivariate Scatterplot

This is the only part that I didn't use python.

The selection of variables x and y are provided by dropdown menu ("select" tag in html).

One of the problems I encountered was the conversion of categorical data to numerical equivalent data. If I use 0 and 1 for the categorical variable that has only two value types, the scaling will be domain(0, 1) to range(0, canvasWidth). This causes a problem in that the plot output doesn't look good because points are on the edge of canvas. So I decided to use domain(0, 3) scale instead, so that I can assign 1 or 2 for two-value categorical data (0 and 3 will be left edge and right edge of the canvas).

Selection of new variables by the dropdown will trigger an event handler that runs to remove all current d3 drawings and draw a new plot with the two variable selected:

```
xDropDown.on("change", xVarSelectHandler);
function xVarSelectHandler() {
    varSelectionChanged();
}
```

```
yDropDown.on("change", yVarSelectHandler);
function yVarSelectHandler() {
    varSelectionChanged();
}
```

```
function varSelectionChanged() {
    var xSelectionIndex = xDropDown.property("selectedIndex");
    var ySelectionIndex = yDropDown.property("selectedIndex");
```

I made 3 different functions for [x=Numeric, y=Numeric], [x=Numeric, y=Categorical], [x=Categorical, y=Numeric], since numeric and categorical data have different ways to be scaled to the plot as I explained above. I have omitted the scatterplot of [x=Categorical, y=Categorical] because the plot looks not very interesting.

```
else {
    var domXSelect = document.getElementById(xDropDownId);
    var xAttrName = domXSelect.options[xSelectionIndex].text;
    var domYSelect = document.getElementById(yDropDownId);
    var yAttrName = domYSelect.options[ySelectionIndex].text;

    // x=categorical, y=categorical
    if( (xAttrName in cateDataTable) && (yAttrName in cateDataTable) ) {
        // not done. not interesting
    }
    // x=categorical, y=numeric
    else if( (xAttrName in cateDataTable) && !(yAttrName in cateDataTable) ) {
        drawPlot_xCat_yNum(xAttrName, yAttrName);
    }
    // x=numeric, y=categorical
    else if( !(xAttrName in cateDataTable) && (yAttrName in cateDataTable) ) {
        drawPlot_xNum_yCat(xAttrName, yAttrName);
    }
    // x=numeric, y=numeric
    else if( !(xAttrName in cateDataTable) && !(yAttrName in cateDataTable) ) {
        drawPlot_Numerics(xAttrName, yAttrName);
    }
}
```

2. 10x10 Correlation Matrix

In this part, I made one python program “gen_10x10correlation_mat_csv.py” to write csv file of correlation matrix. The program calculate correlation by the following equation:

r(x,y) = sigma((xi-xavg)*(yi-yavg)) / sqrt(sigma((xi-xavg)^2)*sigma((yi-yavg)^2))

```
def calculateCorrelation(xlist, ylist):
    # formula of correlation calcultation:
    # r(x,y) = sigma((xi-xavg)*(yi-yavg)) / sqrt(sigma((xi-xavg)^2)*sigma((yi-yavg)^2))
    xavg = sum(xlist) / len(xlist)
    yavg = sum(ylist) / len(ylist)

    # numerator
    numerator = 0.0
    for i in range(len(xlist)):
        numerator += (xlist[i]-xavg) * (ylist[i]-yavg)

    # denominator
    x_part_sigma = 0.0
    for i in range(len(xlist)):
        xi_xavg_2 = (xlist[i] - xavg) ** 2
        x_part_sigma += xi_xavg_2

    y_part_sigma = 0.0
    for i in range(len(ylist)):
        yi_yavg_2 = (ylist[i] - yavg) ** 2
        y_part_sigma += yi_yavg_2

    denominator = math.sqrt( x_part_sigma * y_part_sigma )

    corr = numerator / denominator

    return corr
```

The program's output csv file name is "NBA_shot_10by10CorrMat.csv".

In the coding of index.html, the first problem was to decide a rgb color of each corr value. I first made a rgb color matrix whose value is scaled according to the correlation:

```
// corr = [-1.0 ~ 0.0] --> rgb = (0 ~ 255, 0 ~ 255, 255 fix)
// corr = [0.0 ~ 1.0] --> rgb = (255 fix, 255 ~ 0, 255 ~ 0)
// corr = [-1.0 ~ 1.0] --> _scale = [-255 ~ 255]
// if _scale < 0, use blue color
var rgbMatrix = []
var corrToRgbScale = d3.scale.linear()
                        .domain([-1.0, 1.0])
                        .range([-255, 255]);
for( i=0; i<corrMat.length; i++) {
    var row_i = corrMat[i];
    var rgb_row_i = []
    for( j=0; j<corrMat[0].length; j++) {
        var elem_i_j = row_i[j];
        var rgb_i_j = corrToRgbScale(elem_i_j);
        rgb_row_i.push(rgb_i_j);
    }
    rgbMatrix.push(rgb_row_i);
}
```

And here are the lines that the rect of d3 is actually filled:

```
.style("fill", function(rgbBoxObj) {
    return genFillStringRGB(rgbBoxObj);
});
```

```
function genFillStringRGB(rgbBoxObj) {
    var valueFromRGBMat = rgbBoxObj["rgbValue"];
    // blue
    if(valueFromRGBMat < 0.0) {
        var r_g_int = 255 - Math.floor( -valueFromRGBMat );
        return "rgb(" + r_g_int + "," + r_g_int + "," + 255 + ")";
    }
    // red
    else {
        var g_b_int = 255 - Math.floor(valueFromRGBMat);
        return "rgb(" + 255 + "," + g_b_int + "," + g_b_int + ")";
    }
}
```

3. 5x5 Scatter Plot Matrix

The python program "print_console_abs_corr_sum.py" reads the file "NBA_shot_10by10CorrMat.csv" and prints the sum of |correlation| of the 10 attributes. The console output shows us that DRIBBLES, TOUCH_TIME, SHOT_DIST(ft), PTS_TYPE, and CLOSE_DEF_DIST(ft) are the top 5 attributes. (check "Screen Capture of print_console_abs_corr_sum run.png" file).

The file "NBA_shot_logs_top5_corr.csv" is the dataset of the 5 attributes.

In index.html, I only use one canvas with 25 rect. Each rect represents one scatter plot. The position of each rect is stored in the object matrix:

```
var scatterplotInfoMatrix = [];
for( i=0; i<attrArray.length; i++) {
    var row = [];
    for( j=0; j<attrArray.length; j++) {
        var spInfoObj = {};     // scatterplot info object
        spInfoObj["xpos"] = leftMarginSize + j*squareSize;
        spInfoObj["ypos"] = topMarginSize + i*squareSize;
        // for example, [1][3] of the scatterplot matrix is drawn (x=attr3, y=attr1)
        spInfoObj["xVarName"] = attrArray[j];
        spInfoObj["yVarName"] = attrArray[i];
        row.push(spInfoObj);
    }
    scatterplotInfoMatrix.push(row);
}

for( row=0; row<attrArray.length; row++){
    for( col=0; col<attrArray.length; col++){
        drawPlot(scatterplotInfoMatrix[row][col]);
    }
}
```

4. Parallel Coordinates Display

From this point of the project, I started use “NBA_shot_logs_10attr.csv” file that contains only 10 attributes I selected, instead of “NBA_shot_logs_2014_2015_reduced_at1.csv”.

The ordering of the parallel coordinates is decided by my eyes seeing through “NBA_shot_10by10CorrMat.csv” file. The process is described in the txt file “Decision of ordering axes.txt”.

In index.html, I first made a list that tells the order of the axes:

```
var orderedAxesNames = ["TOUCH_TIME", "SHOT_DIST(ft)", "PTS_TYPE",
    "CLOSE_DEF_DIST(ft)", "DRIBBLES", "SHOT_NUMBER", "PERIOD",
    "SHOT_CLOCK", "W", "FINAL_MARGIN"];
```

And decide the x position of each axis:

```
// horizontal scaling for axis positions
var horizontalAxisPosScale = d3.scale.linear()
    .domain([0, orderedAxesNames.length-1])
    .range([leftMarginSize, canvasWidth-rightMarginSize]);

var axesXPoslist = [];
for( i=0; i<orderedAxesNames.length; i++ ) {
    var xPos = horizontalAxisPosScale(i);
    axesXPoslist.push(xPos);
}
```

Then make two types of objects: one contains the data point information and the other contains line information that can be generated by the first one.

```
// data points
var pointInfoObjListContainer = [];
for( i_axis=0 ; i_axis < orderedAxesNames.length ; i_axis++ ) {
    var ithAxisPointInfoList = [];
    var ithAttrName = orderedAxesNames[i_axis];
    var dataList = getDataListOfAttr(ithAttrName);
    var xPos = axesXPoslist[i_axis];    // it is absolute position in canvas

    var axisScale = d3.scale.linear()
        .domain([d3.min(dataList), d3.max(dataList)])
        .range([0, axisHeight]);

    for( j=0; j < numOfRecords; j++) {
        var yPos = axisScale(dataList[j]) + topMarginSize;  // scaling - not an absol
        var dataPointInfoObj = {};
        dataPointInfoObj["attrName"] = ithAttrName;
        dataPointInfoObj["xPos"] = xPos;
        dataPointInfoObj["yPos"] = yPos;

        ithAxisPointInfoList.push(dataPointInfoObj);
    }
    pointInfoObjListContainer.push(ithAxisPointInfoList);
}
```

```
// line infos
var lineInfoObjList = [];
var lineIndex = 0;
for( leftAttrIdx=0 ; leftAttrIdx < orderedAxesNames.length-1 ; leftAttrIdx++ ) {
    var rightAttrIdx = leftAttrIdx+1;
    var lineInfoObj = {};

    var leftPointInfoList = pointInfoObjListContainer[leftAttrIdx];
    var rightPointInfoList = pointInfoObjListContainer[rightAttrIdx];

    for( i=0 ; i < leftPointInfoList.length; i++){
        var lineInfoObj = {};
        lineInfoObj["x1"] = leftPointInfoList[i]["xPos"];
        lineInfoObj["y1"] = leftPointInfoList[i]["yPos"];
        lineInfoObj["x2"] = rightPointInfoList[i]["xPos"];
        lineInfoObj["y2"] = rightPointInfoList[i]["yPos"];
        lineInfoObjList.push(lineInfoObj);
    }
}
```

Then we just need to draw the lines based on the information from the objects.

5. PCA Plot

There are two python programs in this part.

The first program “gen_eig_value_and_vector_csv.py” writes two csv files: “eig_value.csv” and “eig_vector.csv”. I used numpy.cov() to get covariance matrix, and numpy.linalg.eig() to get eigenvalues and eignevectors.

If you check the eig_value.csv, you can easily figure out that the first and second eigenvectors on the list have top 2 eigenvalues in order. So I chose the two eigenvectors for the 10-d to 2-d projection.

The second program “gen_2d_projected_points_csv.py” reads two csv files: “eig_vector.csv” and

"NBA_shot_logs_10attr.csv". It writes one csv file "2d_projected_points.csv". The 10-d to 2-d projection matrix is formed by:

```
# datasets so far have string only. make them float
eigenvectors = floatifyDataset(eigenvectorsDataset)

top2eigenvectors = []
for i in range(len(top2eigVecIndex)):
    top2eigenvectors.append( eigenvectors[top2eigVecIndex[i]] )
print(top2eigenvectors)

# build projection matrix
projMat = numpy.array(top2eigenvectors).T
```

The d3 visualization of this part shows two plots: scree plot (bar chart) and PCA plot. The scree plot uses "eig_value.csv", and PCA plot uses "2d_projected_points.csv".

6. **Biplot -> missing**

7. **MDS display of the data**

There is one python program "gen_mds_points.py". It reads "NBA_shot_logs_10attr.csv", create a distance matrix, and create 2d projected points of MDS by the matrix. It writes a file "mds_points.csv".

Before creating the distance matrix, I did the normalization of the dataset. That means all the values in each point is scaled to 0.0 to 1.0.

```
# since each axis has different scales, we need to normalize the data.
# "column_vector" has all data of the axis
def vectorNormalization(column_vector):
    normalized_vector = []
    minVal = min(column_vector)
    maxVal = max(column_vector)
    for i in range(len(column_vector)):
        normedVal_i = (column_vector[i] - minVal) / (maxVal - minVal)
        normalized_vector.append(normedVal_i)
    return normalized_vector
```

This is because the Euclidean distance between two points seemed highly biased to large-scale attribute such as FINAL_MARGIN.

The following lines show how I create 945x945 distance matrix with normalized dataset:

```
distMat = []
for i in range(numOfRecords):
    dist_ith_row = []
    vec_i = normed_dataset[i]
    for j in range(numOfRecords):
        vec_j = normed_dataset[j]
        vec_sub = numpy.subtract(vec_i, vec_j)
        dist_i_j = math.sqrt( sum(vec_sub**2) )
        dist_ith_row.append(dist_i_j)
    distMat.append(dist_ith_row)
```

Here is how I get the 2d data points:

```
mds = manifold.MDS(n_components=2, dissimilarity="precomputed", random_state=6)
results = mds.fit(distMat)
coords = results.embedding_
```

“coords” is the list of 2d data points.

The code of index.html is not very special. Just read “mds_points.csv” and draw them.

8. MDS display of the attributes

The python program “gen_mds_attr_points.py” reads “NBA_shot_logs_10attr.csv”, and writes “mds_attr_points.csv”. The making of distance matrix is same as before, but the difference is that the distance is between two attributes, not between two observations. Of course, I did the normalization before making the distance matrix. The distance matrix is 10 by 10 because we have 10 attributes.

Since I want to see attribute names of the points on the MDS plot, the output file has one more column “Attribute Name” in addition to “x” and “y”.

Report

1. What information does the display show well (Pros)?

    A. Bivariate Scatterplot

Bivariate scatterplot shows almost every relation between two variables: distribution, mean, correlation, and the number and the locations of clusters. If we only want to focus on two variables, this is the best choice.

    B. 10x10 correlation matrix

Since it uses colors, it is easier and faster to detect which correlation between two variables are high or low.

    C. 5x5 scatter plot matrix

It is convenient. We don’t need to select and change variables every time as we did at bivariate scatterplot.

    D. Parallel coordinates

It shows the relation of the two axes that are right next to each other. Also, it is effective when we want to know where in the data converges on a certain axis. This is the only possible way that we can see the actual value of each observation in high dimensional space.

    E. PCA plot

People cannot perceive high dimensional data. PCA allows us to understand the high dimensional data points by projecting them on the 2d plot. Some value of each data point can be lost, but at least we can assume the distribution and the clusters of the dataset.

F. Biplot -> missing

G. MDS display of the data

It is the best way to show how the clusters of high dimensional dataset are formed.

H. MDS display of the attributes

It shows us the distance between two attributes, and we can assume the correlation between them. If the attributes A and B are far in the display, it means that if A is high in some points, then B is low in them.

2. What information can't the display show (Cons)?

A. Bivariate Scatterplot

Categorical data with 2 or three value types makes the plot less interesting, unless we implement some additional functionalities such that concentrated data points in limited area change to be one big circle. Let's say you have attribute A (0 or 1) and B (0 or 1). Even if you have 100 or 1000 observations, the scatter plot of the two attributes may only show 4 small dots.

Bivariate scatterplot is ineffective if you want to compare multiple attributes.

B. 10x10 correlation matrix

We cannot get the exact amount of each correlation. Also, the only information we get is the correlation. We cannot get the distribution and cluster of the dataset.

C. 5x5 scatter plot matrix

It is still not an effective way to use for comparison between attributes. If we use this plot matrix when we have 20 or 30 attributes, it becomes same as using bivariate scatterplot: we will compare one by one.

D. Parallel coordinates

It is impossible to get the relation between axes that is not right next to each axis. Also, if the number of observation is high, it is hard to get anything from the plot.

E. PCA plot

It can provide some deceptive information: even if the two observations are far in high dimensional space, it looks really close in the projected 2d space.

PCA plot also doesn't provide any information about the relations between attributes.

F. Biplot -> missing

G. MDS display of the data

We lost the almost all information of each observation. It also does not provide any information about the relations between attributes.

H. MDS display of the attributes

It only shows which attributes are close or far. We don't even know how many observations we have.

3. What did you learn about your data from these plots?

I learned a few things by the correlation matrix. TOUCH_TIME and PTS_TYPE correlation is blue. It means that three-point shot is more likely done if the amount of the ball holding time is small. In other words, players are less likely to try 3-point if they hold ball for long. If they hold the ball a bit longer, they may try to drive in to the 2-point area.

Also, SHOT_DIST and SHOT_CLOCK correlation is blue. It shows that players like to try a shot in distant position when they have less time left for offense.

In the MDS display of attributes, I found that PERIOD is far away from PTS_TYPE. If PERIOD is big, PTS_TYPE is likely to be small. We can assume that players like to avoid 3-points if they are close to end-game.