assignment-4

March 17, 2018

1 Assignment 4

In [210]:
```
#Import packages
import pandas as pd
import numpy as np
import os
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classificat
from sklearn import grid_search
from sklearn import tree
from scipy.spatial.distance import cdist
from scipy.cluster.hierarchy import dendrogram, linkage
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

In [217]:
```
#UDFs

def grid_scores_to_df(grid_scores):
    rows = list()
    for grid_score in grid_scores:
        for fold, score in enumerate(grid_score.cv_validation_scores):
            row = grid_score.parameters.copy()
            row['fold'] = fold
            row['score'] = score
            rows.append(row)
    df = pd.DataFrame(rows)
    return df
```

Problem 1 (20 Points): Problem 1 (20 points): Download the letter recognition data from: http://archive.ics.uci.edu/ml/datasets/Letter+Recognition The objective is to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the

English alphabet. The character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique stimuli. Each stimulus was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. Below is the attribute information, but more information on the data and how it was used for data mining research can be found in the paper:
P. W. Frey and D. J. Slate. "Letter Recognition Using Holland-style Adaptive Classifiers". (Machine Learning Vol 6 #2 March 91)
Attribute Information:
1. lettr capital letter (26 values from A to Z)
2. x-box horizontal position of box (integer)
3. y-box vertical position of box (integer)
4. width width of box (integer)
5. high height of box (integer)
6. onpix total # on pixels (integer)
7. x-bar mean x of on pixels in box (integer)
8. y-bar mean y of on pixels in box (integer)
9. x2bar mean x variance (integer)
10. y2bar mean y variance (integer)
11. xybar mean x y correlation (integer)
12. x2ybr mean of x * x * y (integer)
13. xy2br mean of x * y * y (integer)
14. x-ege mean edge count left to right (integer)
15. xegvy correlation of x-ege with y (integer)
16. y-ege mean edge count bottom to top (integer)
17. yegvx correlation of y-ege with x (integer)

Create a classification model for letter recognition using decision trees as a classification method with a holdout partitioning technique for splitting the data into training versus testing.

a. Changing the values for the depth, number of cases per parent and number of cases per leaf produces different tree configurations with different accuracies for training and testing. Choose at least five different configurations and report the accuracy for training and testing for each one of them. Which configuration will you choose as the best model? Explain your answer.

b. For the best tree configuration, report the misclassification matrix and interpret it. In your opinion, is accuracy a good way to interpret the performance of the model? If not, suggest other measures.

c. What are the most important three attributes for recognizing the letters?

Problem 1-a Given the parameters for max_depth, min_samples_split, and min_samples_leaf, the best configuration for the decision tree classifier would be max_depth = 10, min_samples_leaf = 5, min_samples_split = 10 . This configuration had the highest mean_validation_score and cv_validation_score.

```
In [16]: #Import data
         data_url = 'https://raw.github.com/PixarJunkie/fundamentals-of-data-scienc
```

```
data_df = pd.read_csv(data_url, sep = ";")

data_df.rename(columns = {'lettr': 'label'}, inplace = True)
```

```
In [ ]: #Features and labels
        X = data_df.loc[:, data_df.columns != 'label']
        y = data_df.loc[:, data_df.columns == 'label']

        #Train test split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,

        print('X_train size: ' + str(X_train.shape))
        print('X_test size: ' + str(X_test.shape))
        print('y_train size: ' + str(y_train.shape))
        print('y_test size: ' + str(y_test.shape))

        labels = y_train.label
        feature_lst = list(X_train.columns)
```

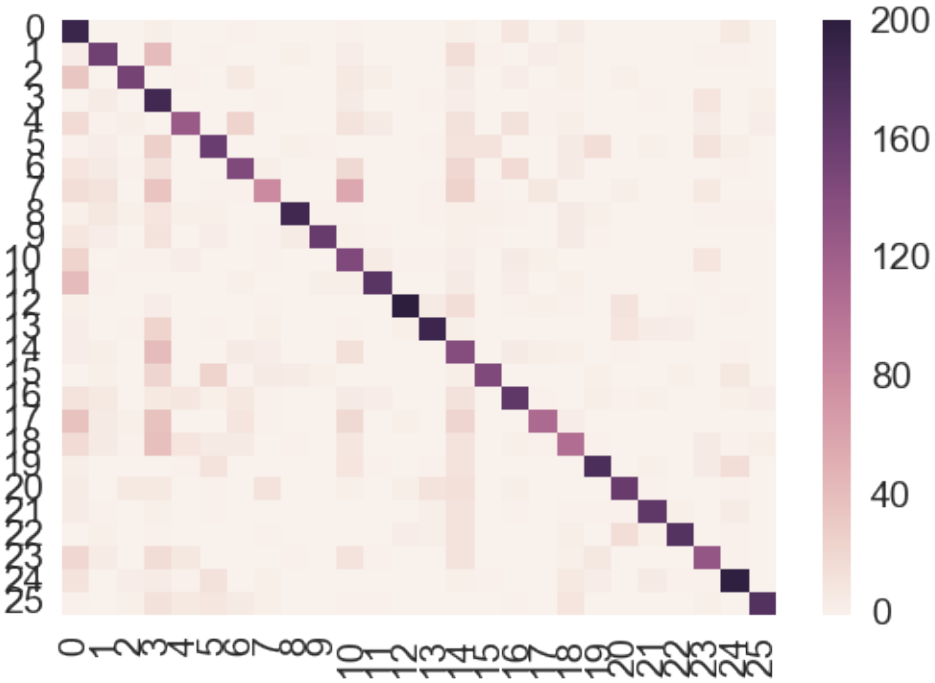
```
In [115]: #Base decision tree
          dt = DecisionTreeClassifier(criterion = 'gini', random_state = 23)

          #Params to pass to classification model
          params = {'max_depth': [2, 4, 6, 8, 10], 'min_samples_split': [10, 20, 30

          #Classification model
          dt_clf = grid_search.GridSearchCV(dt, param_grid = params, n_jobs = 4)
          dt_clf = dt_clf.fit(X = X_train, y = labels)

          predictions = dt_clf.predict(X_test)

          results_df = grid_scores_to_df(clf.grid_scores_).sort_values(by = ['score

          results_df.head()
```

Out[115]:

	fold	max_depth	min_samples_leaf	min_samples_split	score
302	2	10	5	10	0.700021
305	2	10	5	20	0.695942
308	2	10	5	30	0.689500
320	2	10	10	20	0.689285
317	2	10	10	10	0.689285

Problem 1-b The confusion matrix below shows the correctly classified cases for each label on the main diagnal, while everything else shows the count of misclassified labels for each label. Overall, the accuracy ranges from medium to high. I would say that accuracy is a good start for measuring model performance, though I would also consider exploring percision and AUC scores as well.

```
In [113]: #Decision tree model with best configuration
          dt = DecisionTreeClassifier(criterion = 'gini', random_state = 23, max_de
          dt = dt.fit(X_train, y_train)

          predictions = dt.predict(X_test)

          #Confusion matrix plot
          conf_arr = confusion_matrix(y_test, predictions)
          sns.heatmap(conf_arr)

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x155c5d31ef0>
```



Problem 1-c The three most important feature in the model are x-ege mean, y2bar mean, and y-ege mean.

```
In [116]: features_df = pd.DataFrame(list(zip(X_train.columns, clf.best_estimator_.

          features_df.head()

Out[116]:    feature label  feature importance
          12    x-ege mean            0.167458
          8     y2bar mean            0.129838
          14    y-ege mean            0.113367
          13         xegvy            0.108774
          11    xy2br mean            0.095337
```

Problem 2 (20 Points): On the same data from Problem 1, apply a K-nearest neighbor (kNN) classifier to classify the data. Report the following:

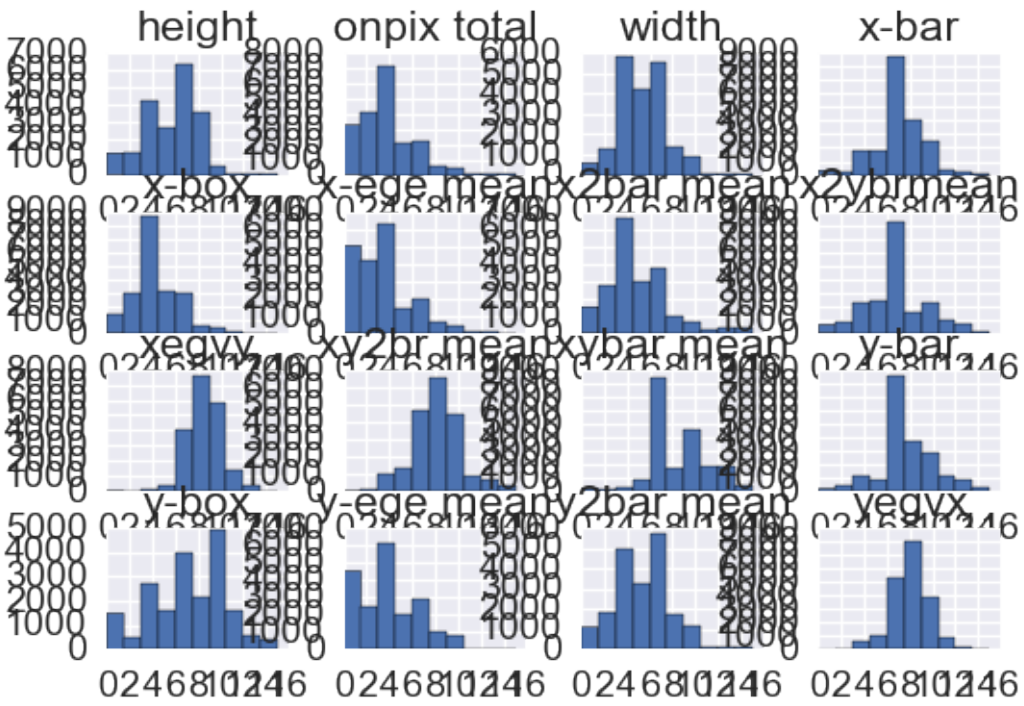
a. If you are doing any data transformation, explain the transformation and why it is needed.

b. Report the misclassification matrix and the appropriate performance metrics for different values of K (K=1, 3, 5, and 7).

c. Interpret the results and also compare them with the ones obtained by using the decision trees.

Problem 2-a From looking at the histograms for the features, I'm not going to transform anything initially.

```
In [102]: data_df.hist()

Out[102]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000155600905
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155634FB5
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000015563531
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000015563568
                 [<matplotlib.axes._subplots.AxesSubplot object at 0x00000155635B34
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155635F30
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000015563639
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155C3ED5
                 [<matplotlib.axes._subplots.AxesSubplot object at 0x00000155C3F254
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155C3F6BE
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000015562EACE
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155C3FC83
                 [<matplotlib.axes._subplots.AxesSubplot object at 0x00000155C40075
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155C40538
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155C4086E
                  <matplotlib.axes._subplots.AxesSubplot object at 0x00000155C40D78
```



Problem 2-b Confusion matrix below. The classification_report for K = 1,3,5,7 show a high level of accuracy, precision, and recall.

```
In [117]: #Base KNN
          knn = KNeighborsClassifier()

          #Params to pass to classification model
          params = {'n_neighbors': [1, 3, 5, 7], 'weights': ['uniform', 'distance']

          #Classification model
          knn_clf = grid_search.GridSearchCV(knn, param_grid = params)
          knn_clf = knn_clf.fit(X = X_train, y = labels)

          predictions = knn_clf.predict(X_test)

          results_df = grid_scores_to_df(knn_clf.grid_scores_).sort_values(by = ['s

          results_df.head()

Out[117]:    fold  n_neighbors     score   weights
          9     0            3  0.945466  distance
          0     0            1  0.945252   uniform
          3     0            1  0.945252  distance
```

```
4     1            1  0.944933  distance
1     1            1  0.944933   uniform
```

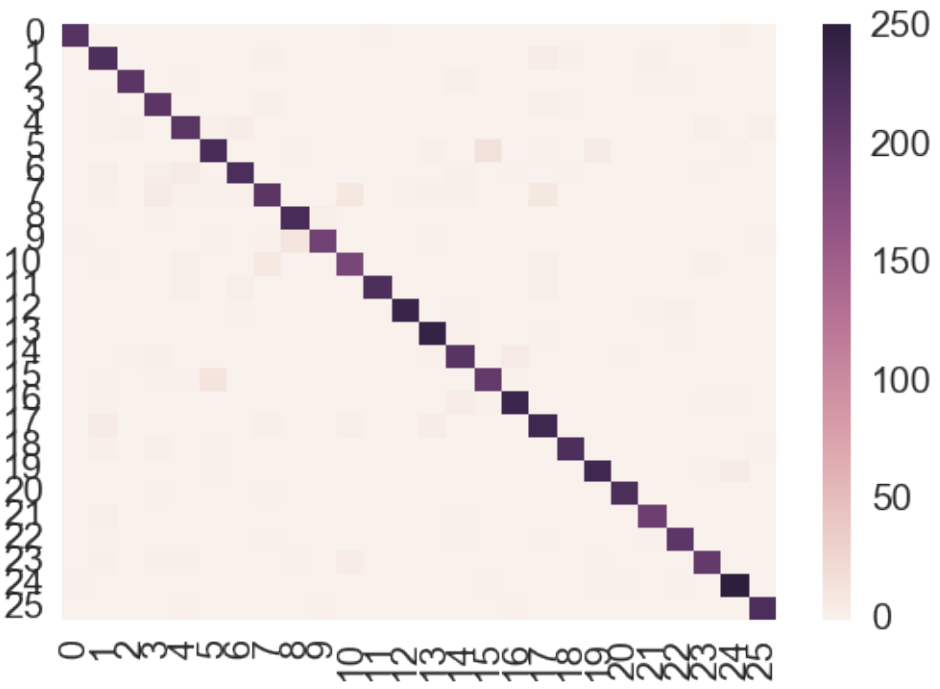
In [226]:
```
k = [1, 3, 5, 7]

for n in k:
    knn = KNeighborsClassifier(n_neighbors = n, weights = 'distance')
    knn.fit(X_train, y_train)
    prediction = knn.predict(X_test)

    print('k = %d: ' %(n) + 'accuracy: ' + str(accuracy_score(y_test, pre
#     print('k = %d: ' %(n) + 'mertics:')
#     print(str(classification_report(y_test, predictions))) #Exluded the
```

```
k = 1: accuracy: 0.9495
k = 3: accuracy: 0.9495
k = 5: accuracy: 0.9495
k = 7: accuracy: 0.9495
```

In [111]:
```
#KNN with best configuration
knn = KNeighborsClassifier(n_neighbors = 3, weights = 'distance')
knn.fit(X_train, y_train)

#KNN predictions
predictions = knn.predict(X_test)

#Confusion matrix plot

print(accuracy_score(y_test, predictions))

conf_arr = confusion_matrix(y_test, predictions)
sns.heatmap(conf_arr)
```

```
0.954333333333
```

Out[111]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x155c5f07358>
```



Probelm 2-c The results from the KNN classifier show better performance than the decision tree classifier. KNN has higher accuracy, and the confusion matrix shows that it is also more successful in correct predictions.

Problem 3 (30 Points): For this problem, we will be doing clustering on the seeds dataset, which you must download from http://archive.ics.uci.edu/ml/datasets/seeds# The examined data group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment. High quality imaging of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.
Attribute information – seven geometric parameters of wheat kernels were measured:
1. area A,
2. perimeter P,
3. compactness C = 4*pi*A/P^2,
4. length of kernel,
5. width of kernel,
6. asymmetry coefficient
7. length of kernel groove.

All of these attributes are numeric, continuously varied variables. The last attribute in the data file represents the class label.

a. (15 points)

1. In k-means how are the cluster centers calculated?

2. Name two similarity measures (or distance functions) and what type of data you would use them for.

3. Perform k-means clustering using all the attributes except the class label. Use multiple numbers of clusters (k=3, 4, 5, and 6). For each k: i. Report the final cluster centers ii. Report the number of elements in each cluster iii. Report the class distribution within each cluster (use crosstab between labels and cluster membership) iv. Export to a variable (but do not report) the distance of each point from its cluster.

4. Using the information you gathered (particularly (iv)), choose the best number of clusters by doing an unsupervised analysis. Specifically, create a graph (in Excel or Google Docs, etc.) of the total error per number of clusters and find the 'knee'. See the last lab for details. Which k do you choose?

5. Also choose the best number of clusters by doing an analysis that takes advantage of the labels, by looking at the results of (iii) above. Which k would you choose?

6. For the selected k in (iv), analyze and report if the normalization of the attributes will influence the clustering results.

b. (10 points) Perform hierarchical clustering using all attributes except the class label as follows:

    1. Apply single linkage algorithm and report i. The dendogram (copy and paste the image and scale it to a single page)
    ii. The class distribution at the level of the dendogram where there are only three clusters.
    2. Apply complete linkage and report i. The dendogram (copy and paste the image and scale it to a single page) ii. The class distribution at level of the dendogram where there are only three clusters.

Problem 3-a

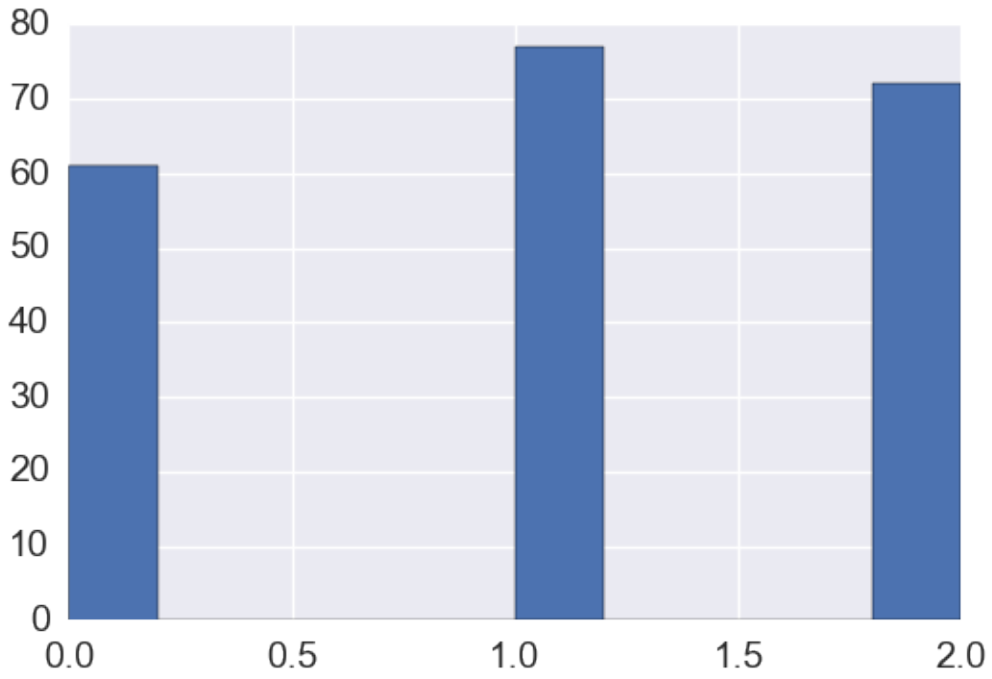
1. Cluster centers are calculated as the mean value of the points within the cluster.
2. Two main similariy measures are the euclidean and cosine similarities. Cosine similarity works best when used in situations where magnitude of data points within a cluster do not matter.

```
In [152]: #Import data
          seeds_url = 'https://github.com/PixarJunkie/fundamentals-of-data-science/
          seeds_df = pd.read_table(seeds_url, header = None)
          seeds_df.rename(columns = {0: 'area', 1: 'permeter', 2: 'compactness', 3:
```

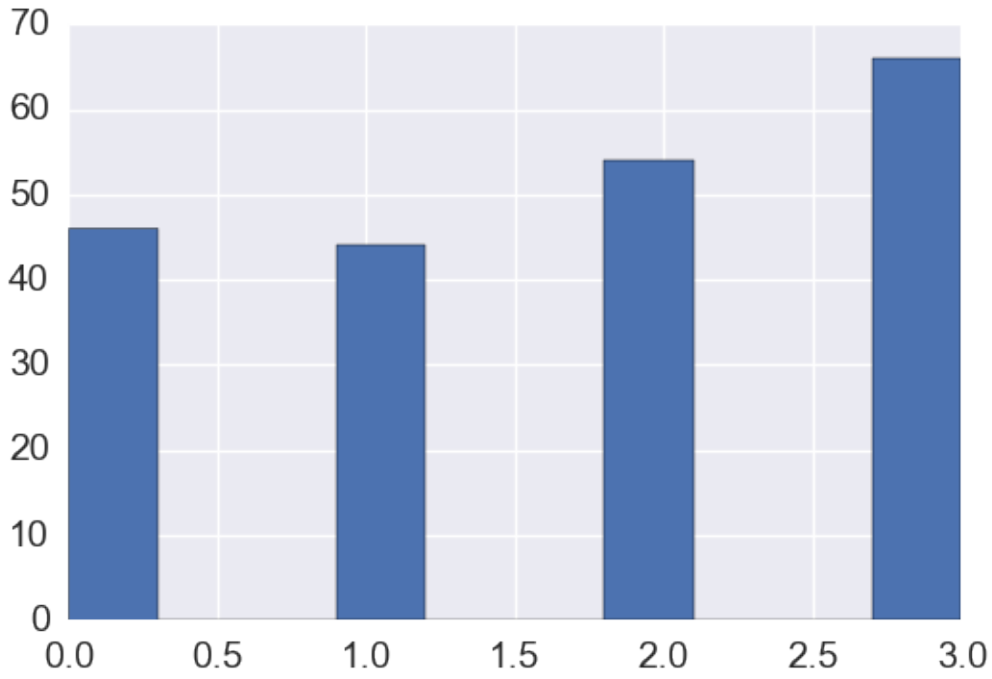
Problem 3-a - 3

```
In [208]: #Serparate labels and
          X = seeds_df.loc[:, seeds_df.columns != 'label'].values
          y = seeds_df.loc[:, seeds_df.columns == 'label']

          #Kmeans model
          k = [3, 4, 5, 6]
          dist = []

          for num in k:
              kmeans = KMeans(n_clusters = num).fit(X)
              kmeans_preds = kmeans.predict(X)
              dist.append(sum(np.min(cdist(X, kmeans.cluster_centers_, 'euclidean')
              print('k = %d' %(num) + ' ' + 'metrics')
              print('number of point in each cluster: ' + ' ' + str({i: len(np.wher
              print('cluster centers: ' + ' ' + str(kmeans.cluster_centers_))
              print('\n')
              print('class distribution')
              plt.hist(kmeans.labels_)
              plt.show()
```

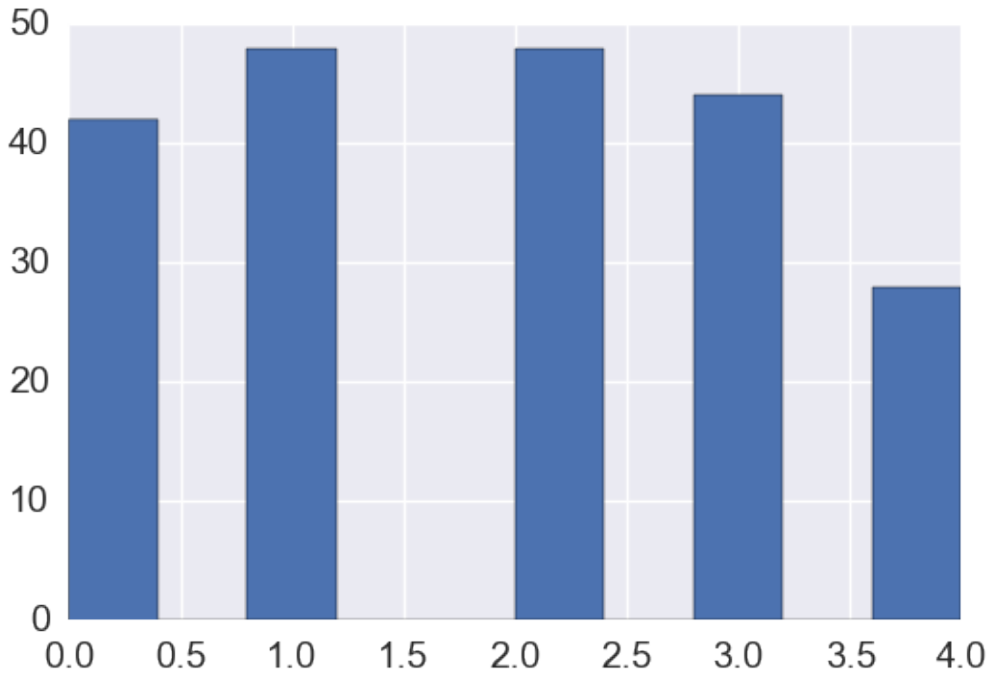
```
k = 3 metrics
number of point in each cluster:  {0: 61, 1: 77, 2: 72}
cluster centers:  [[ 18.72180328  16.29737705   0.88508689   6.20893443   3.726721
    3.60359016   6.06609836]
 [ 11.96441558  13.27480519   0.8522       5.22928571   2.87292208
    4.75974026   5.08851948]
 [ 14.64847222  14.46041667   0.87916667   5.56377778   3.27790278
    2.64893333   5.19231944]]


class distribution
```



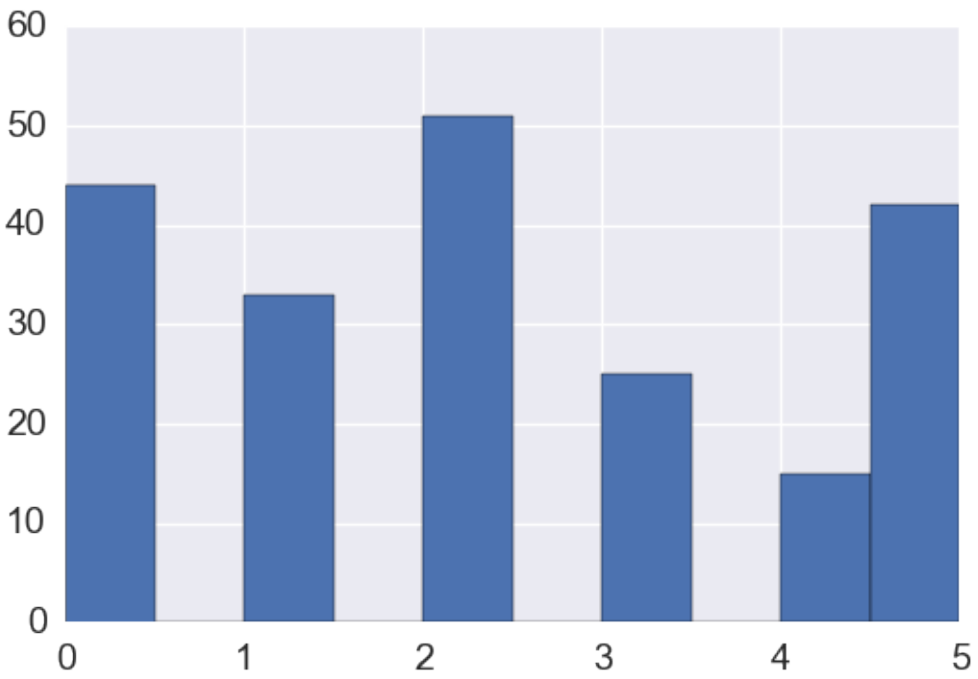

```
k = 4 metrics
number of point in each cluster:  {0: 46, 1: 44, 2: 54, 3: 66}
cluster centers:  [[ 13.39108696  13.86521739   0.87424565   5.366        3.1142826
    2.41580652   4.99067391]
 [ 15.71909091  14.97659091   0.88038636   5.74518182   3.3985       3.18704773
    5.44838636]
 [ 18.96296296  16.39666667   0.88595185   6.24272222   3.74992593
    3.54033333   6.10077778]
 [ 11.91439394  13.26151515   0.85024242   5.23122727   2.86393939
    5.06828788   5.10534848]]


class distribution
```




```
k = 5 metrics
number of point in each cluster:  {0: 42, 1: 48, 2: 48, 3: 44, 4: 28}
cluster centers:  [[ 11.9847619   13.29357143   0.85079524   5.24138095   2.8797381
    5.6732619    5.12197619]
 [ 14.625625    14.444375     0.88059792   5.56377083   3.27639583
    2.37129583   5.14375    ]
 [ 19.15104167  16.46916667   0.88708958   6.26885417   3.7729375
    3.46041667   6.12725    ]
 [ 12.09045455  13.30977273   0.85708636   5.21740909   2.90065909
    3.34375      5.00531818]
 [ 16.47714286  15.34428571   0.879125     5.86864286   3.47717857
    3.98992857   5.69035714]]


class distribution
```

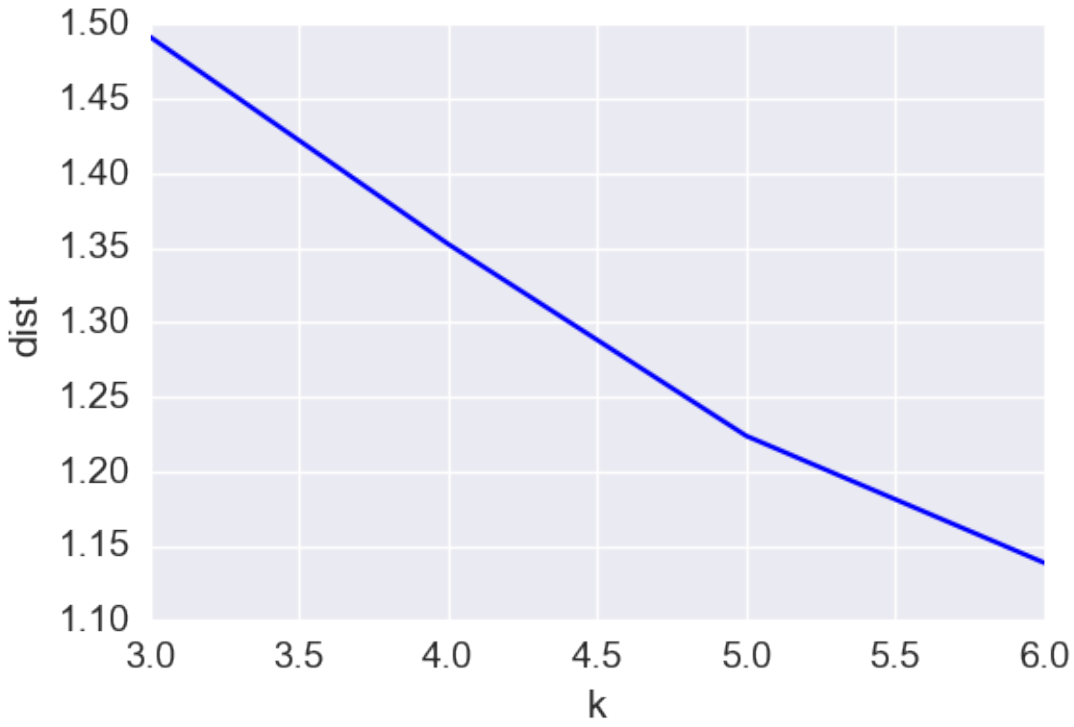



```
k = 6 metrics
number of point in each cluster:  {0: 44, 1: 33, 2: 51, 3: 25, 4: 15, 5: 42}
cluster centers:  [[ 12.09045455  13.30977273   0.85708636   5.21740909   2.900659
    3.34375      5.00531818]
 [ 18.95454545  16.38878788   0.8868       6.24748485   3.74469697
    2.72354545   6.11945455]
 [ 14.69294118  14.47411765   0.88094314   5.57213725   3.28643137
    2.40790588   5.15882353]
 [ 16.562       15.3916       0.878244     5.88816      3.4808       4.10948
    5.7252    ]
 [ 19.58333333  16.646        0.88772667   6.31586667   3.83506667
    5.08153333   6.1444    ]
 [ 11.9847619   13.29357143   0.85079524   5.24138095   2.8797381
    5.6732619    5.12197619]]


class distribution
```



Problem 3-a - 4 I would choose 5 clusters based on the knee chart since that is the point where improvemnets declines.

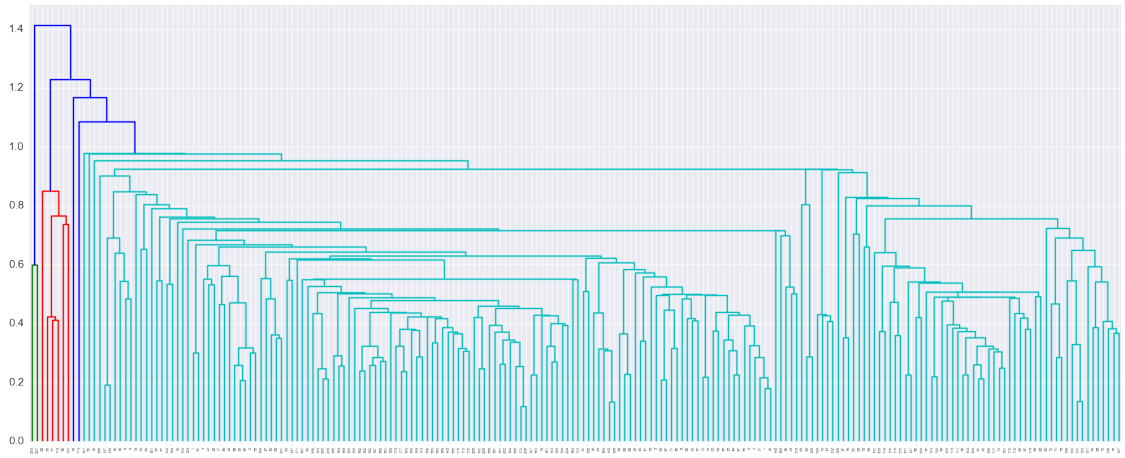
```
In [209]: #Knee plot
          plt.plot(k, dist, 'bx-')
          plt.xlabel('k')
          plt.ylabel('dist')
          plt.show()
```



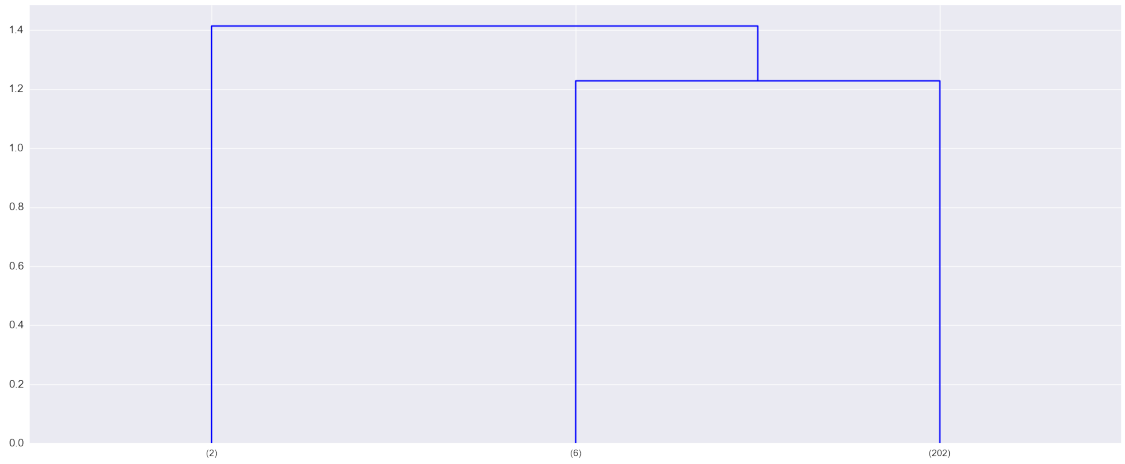
Problem 3-a - 5 Looking at the class distributions, I would also choose 5 clusters as the histogram is the most uniform.

Problem 3-b

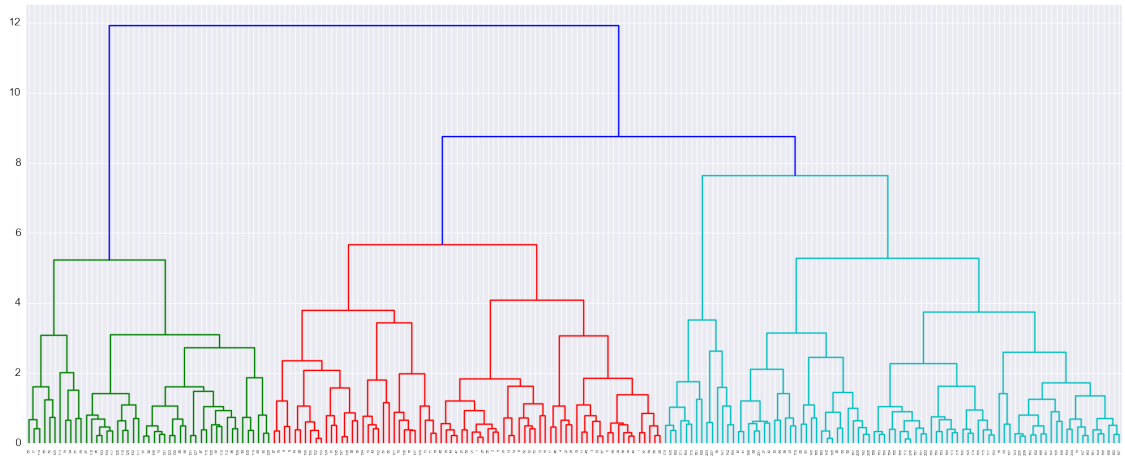
```
In [213]: #single linkage
          link = linkage(X, 'single')
          fig = plt.figure(figsize = (25, 10))
          d = dendrogram(link)
          plt.show()
```



In [215]:
```
#single linkage (3 clusters)
link = linkage(X, 'single')
fig = plt.figure(figsize = (25, 10))
d = dendrogram(link, truncate_mode = 'lastp', p = 3)
plt.show()
```



In [214]:
```
#complete linkage
link = linkage(X, 'complete')
fig = plt.figure(figsize = (25, 10))
d = dendrogram(link)
plt.show()
```



In [216]:
```
#complete linkage (3 clusters)
link = linkage(X, 'complete')
fig = plt.figure(figsize = (25, 10))
d = dendrogram(link, truncate_mode = 'lastp', p = 3)
plt.show()
```