Assignment 4

It will very clear if you can read my assignment4 in iPython notebook.

If you want to use it:
You can check the ipynb file in my submission

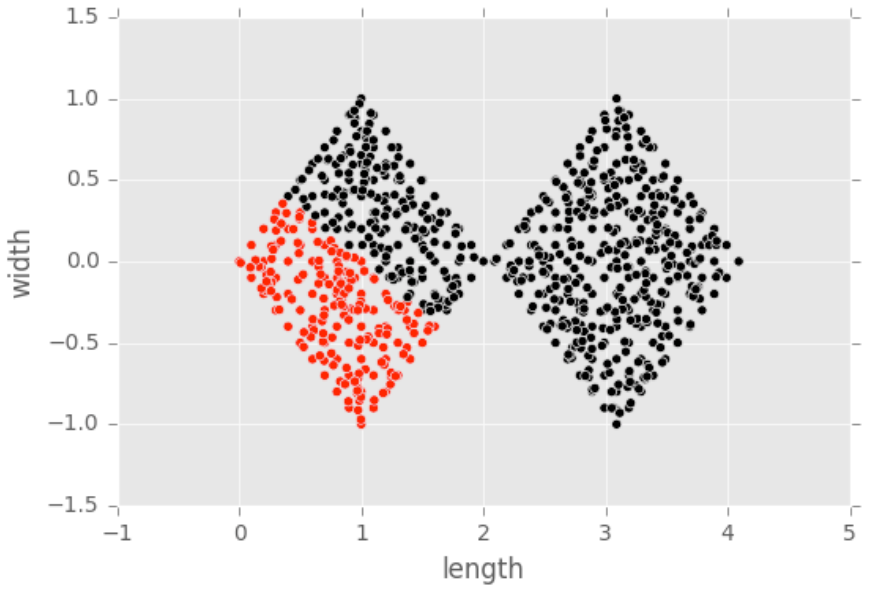
If you do not use iPython notebook but still want to check:
You can check them from my github:
Problem1: https://github.com/boyazhou1993/exercise_ai/blob/master/A4_Problem1.ipynb
Problem2: https://github.com/boyazhou1993/exercise_ai/blob/master/A4_Problem2.ipynb
Problem3: https://github.com/boyazhou1993/exercise_ai/blob/master/A4_Problem3.ipynb
Problem4: https://github.com/boyazhou1993/exercise_ai/blob/master/A4_Problem4.ipynb

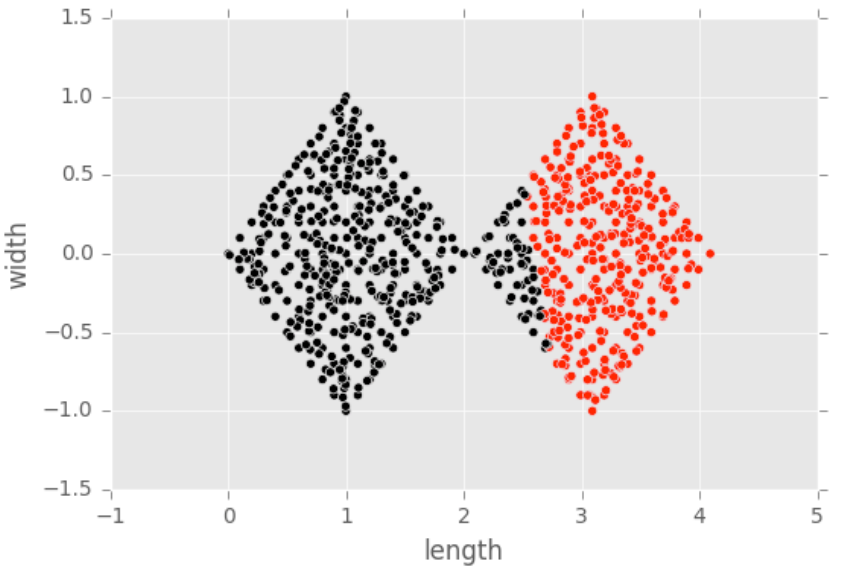
If you do not want to check it, you can run the .py file in my submission, however, it will not generate any picture since I use %matplotlib inline in my code to generate plot and this can not be implement in .py file. I will directly put all the picture I generate in this word file.

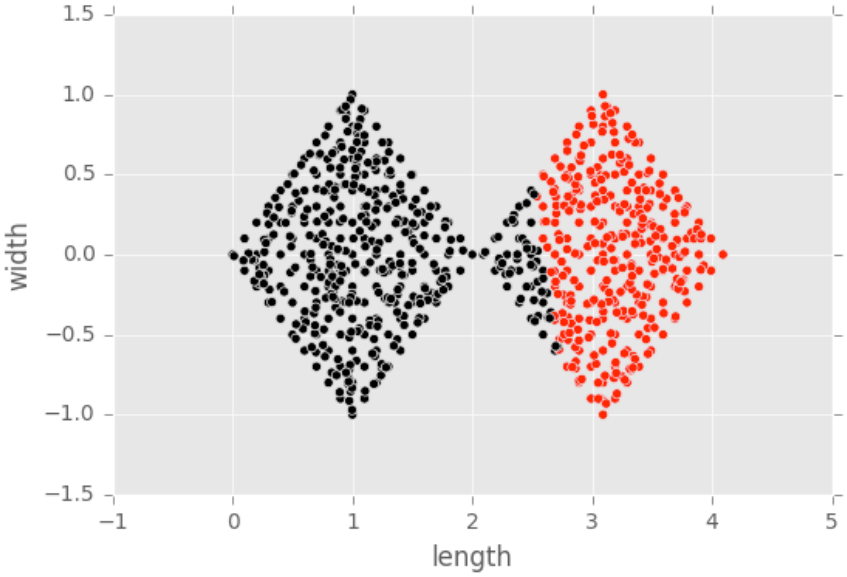
Note: All the inputs are from what Professor gave. Though I make a preprocessed chronic dataset, it's also in my submission, named clean_chronic.csv.

Problem1

I only use n_clusters = 2 in this problem, this is what I got after implement the k-means algorithm by myself:







Problem 1: findings

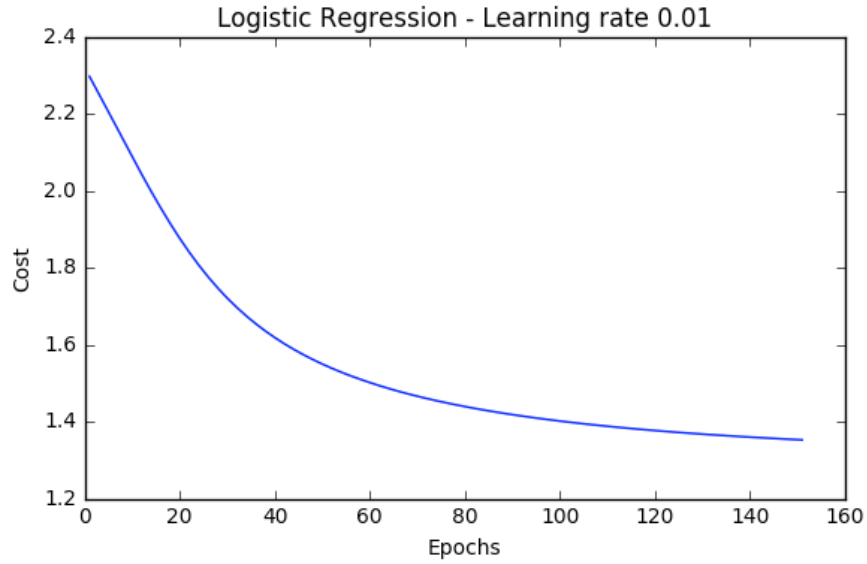
You can see from the 3 picture above, the results are different from each other, the black points are in one cluster and the red in other cluster.
If you run my code, maybe you can not get exactly same results as me, since I use random initial points and the results of k-means is dependent of initial points.

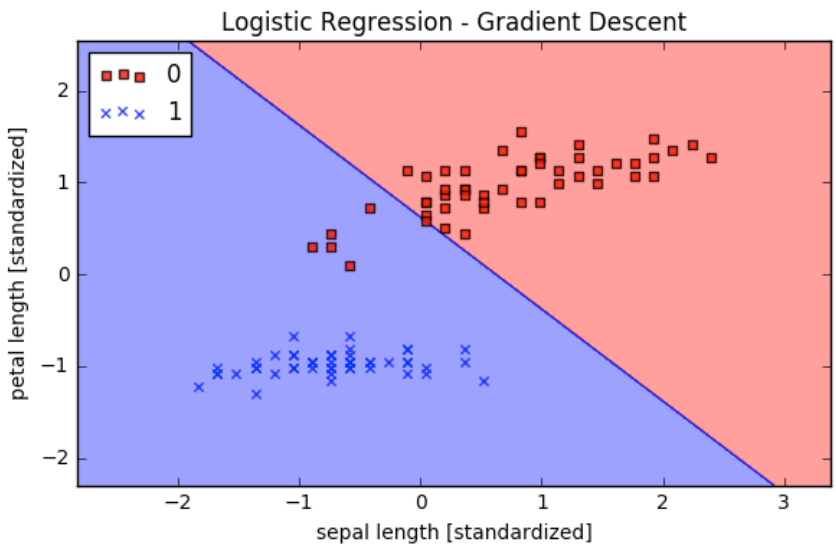
Problem2

(a)

I read the .arff file in weka and transfer it in csv format. And I use pandas to do the preprocessing stuff. Over than half of the instances in dataset got at least one empty feature, so it's not wise to delete these instances. So I use mean value to impute numeric feature and most frequent to impute the nominal feature.

After I implement the logistic regression, I use the famous iris dataset from https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data and code from https://github.com/rasbt/python-machine-learning-book/blob/master/code/bonus/logistic_regression.ipynb to make sure my algorithm works.





It seems works!

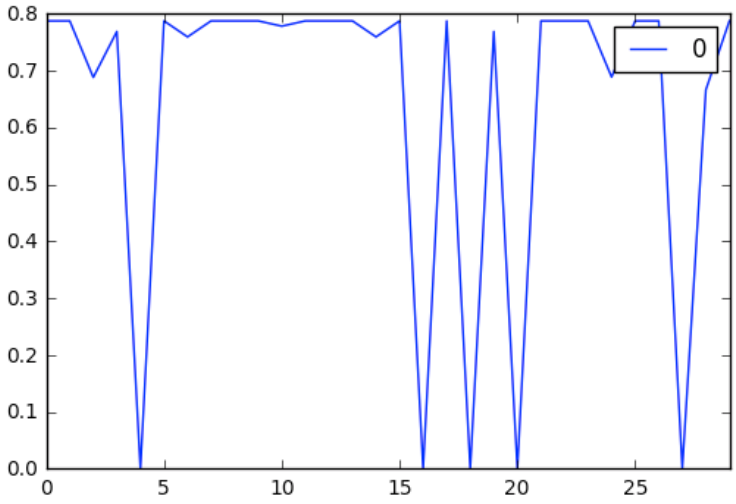
(b)

So I put the chronic dataset in my logistic regression algorithm:

Here I got 2 parameter: one is reg as said in assignment instructions, the other is tolerance to tell algorithm when treat it means converge and stop the algorithm.

The tolerance I pick for not normalized data is 10, and if the epoch larger than 40000, no matter converge or not, the algorithm stops.

This is the line plot I got:



The x-axis is reg from -2 to 4 step 0.2. The y-axis is f score.

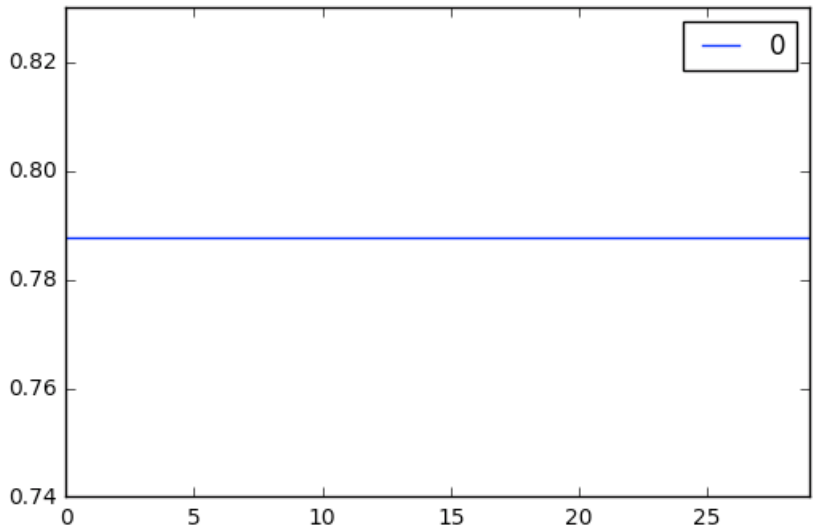
It seems the change of reg do not help logistic regression much in this case. And sometimes the logistic regression is trapped in local minimum.

The best f score is 0.78787878787878796.

(c)

After I normalize the data. I set tolerance as 0.000001. Here is what I got:

Notice my best score here is always 0.78787878787878796, and in very quick implementation time.



Problem3

k-means:

Here I change the algorithm in problem1 to make sure it can deal with all kinds of feature dimensions.

kmeans: confusion matrix

```
array([[171,   1,   0,   0,   1,   0,   0,   0,   5,   0],
       [  0,  56,   0,   0,  56,   0,   0,  10,  60,   0],
       [ 11,  16,  79,   4,   0,   1,   0,  54,  12,   0],
       [  2,  12,  47, 106,   0,   2,   0,   4,  10,   0],
       [  1,   0,   0,   0, 145,   0,   0,  24,  11,   0],
       [  0,  19,   2,   0,  25, 114,   0,  19,   3,   0],
       [  9,   0,   0,   0, 101,   7,   0,   0,  64,   0],
       [  0,  20,   1,   0,   1,   0,   0, 153,   4,   0],
       [  0,  43,   2,   2,   0,   7,   0,   7, 113,   0],
       [ 12,  29,  48,  37,   0,  32,   0,   7,  15,   0]])
```

kmeans: Fowlkes and Mallows index evaluation

0.41765609242192703

Agglomerative clustering with Ward linkage

Agglomerative clustering : majority vote

```
array([[178,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 155,  27,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0, 166,   0,   0,   0,   0,   1,  10,   0],
       [  0,   0,   0, 169,   0,   0,   0,   1,  13,   0],
       [  0,   0,   0,   0, 178,   0,   0,   3,   0,   0],
       [  0,   0,   0,   2,   0, 179,   1,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0, 180,   0,   1,   0],
       [  0,   0,   0,   0,   0,   0,   0, 179,   0,   0],
       [  0,   3,   4,   1,   0,   0,   0,   1, 165,   0],
       [  0,  20,   0, 145,   0,   2,   0,  11,   2,   0]])
```

Agglomerative clustering : Fowlkes and Mallows index evaluation¶

0.83213950467054931

AffinityPropagation

AffinityPropagation clustering : confusion matrix

```
array([[178,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [175,   7,   0,   0,   0,   0,   0,   0,   0,   0],
       [167,   0,   9,   1,   0,   0,   0,   0,   0,   0],
       [153,   0,   0,  29,   0,   1,   0,   0,   0,   0],
       [181,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [154,   0,   0,   0,   0,  28,   0,   0,   0,   0],
       [159,   0,   0,   0,   0,   0,  22,   0,   0,   0],
       [153,   0,   0,   0,   0,   0,   0,  26,   0,   0],
       [174,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [179,   0,   0,   0,   0,   1,   0,   0,   0,   0]])
```

AffinityPropagation clustering : Fowlkes and Mallows index evaluation

0.29757530338611876

Problem4

Support Vector Machine with the linear kernel and default parameters

In [10]:
```
from sklearn import svm
```

In [11]:
```
clf_linear = svm.SVC( kernel = 'linear')
clf_linear.fit(X_train_std, y_train)
```

Out[11]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [12]:
```
clf_linear_predict = clf_linear.predict(X_test_std)
```

Support Vector Machine : linear kernel f score

In [13]:
```
f1_score(y_test, clf_linear_predict)
```

Out[13]:
```
0.99029126213592222
```

Support Vector Machine with the rbf kernel and default parameters

In [17]:
```
clf_rbf = svm.SVC( kernel = 'rbf')
clf_rbf.fit(X_train_std, y_train)
```

Out[17]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [18]:
```
clf_rbf_predict = clf_rbf.predict(X_test_std)
```

Support Vector Machine : rbf kernel f score

In [19]:
```
f1_score(y_test, clf_rbf_predict)
```

Out[19]:
```
0.99029126213592222
```

Random forest with default parameters

In [20]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [21]:
```
clf_rf = RandomForestClassifier()
clf_rf.fit(X_train_std, y_train)
```

Out[21]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

In [22]:
```
clf_rf_predict = clf_rf.predict(X_test_std)
```

Random Forest : f score

In [23]:
```
f1_score(y_test, clf_rf_predict)
```

Out[23]:
```
0.99029126213592222
```