EE 559 Project

05/04/2015

Classification of Adults' Salary into >50k and <=50k

Wenzhen Gong

wenzheng@usc.edu

Individual project

Prior or parallel work: None

1. Abstract:

The project is based on a survey done more than two decades ago. The survey collected some information about 48842 people, which serves as features and classes in this project. My goal is to use the features of part of the instances to learn a satisfying model that can classify the rest of instances into their correct classes. I used random assignment, random assignment with prior, random assignment with maximum prior and minimum distance to class mean classifier as my default systems. Then I used pseudoinverse learning and perceptron learning as my distribution free classifiers. I used K nearest neighbors and parzen window as my statistical classifiers. Finally I used support vector machine to be my ultimate classifier, expecting the best performance. It turned out that SVM did give me the best result under some conditions.

2. Name and brief introduction of the dataset:

The dataset contains some information about lots of adults. There are 48842 samples, 32561 of which are used as training data, 16281 of which are used as test data. Each sample has 14 features, which are age, workclass, fnlwgt, education, education-num, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country. There are 2 classes for salary, which are class 0: >50k and class 1: <=50k.

3. Preprocessing:

1. Categorical features

There are 8 categorical features for each sample. So it is necessary to deal with these features in order to do further work. I used the traditional binary feature method to convert the categorical features into numeric features. Take sex for example, instead of using sex itself, I created two new features called male and female, if the value is male, the resulting numeric representation is [1, 0]. Of course, some categorical features has many more than 2 options, thus making the final numeric feature space has 105 dimensions.

2. Missing data

Since I do not have lots of missing data for any single feature, all features are preserved for training process. However, a small part of samples have missing data. I deleted these samples.

3. Normalization

I treat normalization as an option for training process. I will compare the result of using original dataset and normalized dataset. Actually the best result I got involves normalization.

4. Data formatting

The original format of data is clear and recognizable. No further formatting needed.

4. Classifiers used:

1. Random assignment without prior
2. Random assignment with prior
3. Random assignment with maximum prior
4. Minimum to class mean classifier

5. Pseudoinverse learning
6. Perceptron learning
7. K nearest neighbor
8. Parzen window
9. Support vector machine

I used Python3 as platform and scikit learn as the main toolbox, see more details in **8. About coding**.

5. Model selection/ parameter optimization:

Some parameters can be tuned for some classifiers:
1. Learning rate and iterations of perceptron learning
2. Number of neighbors in K nearest neighbors
3. Window size of parzen window

I tuned those parameters so that each classifier achieves best performance. The following results displayed reflect the best performance for each classifier.

6. Dimensionality reduction:

I used PCA for dimensionality. The rationale is to project the data points on a hyperplane with lower dimensionality. The hyperplane should be chosen so that the variance of the new data points should be largest. PCA can save some computation time and delete some useless features. In my project, PCA does not change the error rate so much, so it is meaningful to be used to save time. Using FLD from scikit learn has a restriction that the dimension we want to reduce to has to be less than the class number. Therefore, 1D is the only choice, which is not interesting. I did not use FLD though I coded the function. Results and interpretations are included in the following parts. (See **11. Dimensionality reduction interpretation**)

7. Data usage:

I divided training dataset into 5 equal sized parts. I used 4 parts to do training and test the model with the 5^{th} part. Then I found the combination with lowest error rate and used the corresponding model to test on test dataset. However, for most of the classifier, the results using cross validation is not as good as using the whole training dataset to train the model (except for perceptron learning). So I abandoned cross validation and used the whole training dataset to train the model.

8. About coding:

I used Python3 as my programming language. Scikit learn is my toolbox. I used sklearn.preprocessing for normalization; I used DummyClassifier to do random assignments and my baseline system; I used Perceptron for perceptron learning; I used KNeighborsClassifier, RadiusNeighborsClassifier for K nearest neighbors and parzen window; I used SVC for support vector machine. I coded the functions for each classifier by myself. I coded pseudoinverse learning by myself. I coded read_data by myself. I coded the main program by myself. I coded plot by myself. Unexpected issue was the difficulty of dealing with categorical features. However, I found that scikit learn also supports converting categorical features into numeric features after I coded read_data, but I did not redo my finished work.

9. Results for random assignments and baseline system:

1. No data population, No normalization

```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5651  5709
1  1911  1789
        0       1
0  49.74%  50.26%
1  51.65%  48.35%
The rate of accuracy is  49.40%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  8564  2796
1  2770   930
        0       1
0  75.39%  24.61%
1  74.86%  25.14%
The rate of accuracy is  63.04%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
       0  1
0  11360  0
1   3700  0
         0      1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  5220  6140
1  1598  2102
        0       1
0  45.95%  54.05%
1  43.19%  56.81%
The rate of accuracy is  48.62%
================================================
```

2. No data population, With normalization

```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5705  5655
1  1827  1873
        0       1
0  50.22%  49.78%
1  49.38%  50.62%
The rate of accuracy is  50.32%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  8480  2880
1  2833   867
        0       1
0  74.65%  25.35%
1  76.57%  23.43%
The rate of accuracy is  62.07%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
       0  1
0  11360  0
1   3700  0
         0      1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  8322  3038
1   537  3163
        0       1
0  73.26%  26.74%
1  14.51%  85.49%
The rate of accuracy is  76.26%
================================================
```

3. With data population, No normalization

```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5645  5715
1  1837  1863
        0       1
0  49.69%  50.31%
1  49.65%  50.35%
The rate of accuracy is  49.85%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  5680  5680
1  1817  1883
        0       1
0  50.00%  50.00%
1  49.11%  50.89%
The rate of accuracy is  50.22%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
       0  1
0  11360  0
1   3700  0
         0      1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  5219  6141
1  1598  2102
        0       1
0  45.94%  54.06%
1  43.19%  56.81%
The rate of accuracy is  48.61%
```

4. With data population, With normalization

```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5639  5721
1  1862  1838
        0       1
0  49.64%  50.36%
1  50.32%  49.68%
The rate of accuracy is  49.65%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  5766  5594
1  1832  1868
        0       1
0  50.76%  49.24%
1  49.51%  50.49%
The rate of accuracy is  50.69%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
       0  1
0  11360  0
1   3700  0
         0      1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  7337  4023
1   308  3392
        0       1
0  64.59%  35.41%
1   8.32%  91.68%
The rate of accuracy is  71.24%
```

10. Results for pseudoinverse/ Perceptron/ KNN/ Parzen window/ SVM

1. No data population, No normalization

```
Press any key to launch PSEUDOINVERSE LEARNING...
       0     1
0  10694   666
1   1769  1931
        0       1
0  94.14%   5.86%
1  47.81%  52.19%
The rate of accuracy is  83.83%
============================================
Press any key to launch PERCEPTRON LEARNING...
       0     1
0  10781   579
1   2650  1050
        0       1
0  94.90%   5.10%
1  71.62%  28.38%
The rate of accuracy is  78.56%
============================================
Press any key to launch K NEAREST NEIGHBORS...
       0    1
0  11354    6
1   3369  331
        0      1
0  99.95%  0.05%
1  91.05%  8.95%
The rate of accuracy is  77.59%
============================================
Press any key to launch PARZEN WINDOW...
       0   1
0  11359   1
1   3619  81
        0      1
0  99.99%  0.01%
1  97.81%  2.19%
The rate of accuracy is  75.96%
============================================
        0     1
0  10768   592
1   2586  1114
         0       1
0  94.79%   5.21%
1  69.89%  30.11%
The rate of accuracy is  78.90%
```

(SVM)

2. No data population, With normalization

```
Press any key to launch PSEUDOINVERSE LEARNING...
      0     1
0  4921  6439
1  1614  2086
        0       1
0  43.32%  56.68%
1  43.62%  56.38%
The rate of accuracy is  46.53%
============================================
Press any key to launch PERCEPTRON LEARNING...
       0     1
0  10127  1233
1   1870  1830
        0       1
0  89.15%  10.85%
1  50.54%  49.46%
The rate of accuracy is  79.40%
============================================
Press any key to launch K NEAREST NEIGHBORS...
       0     1
0  10452   908
1   1628  2072
        0       1
0  92.01%   7.99%
1  44.00%  56.00%
The rate of accuracy is  83.16%
============================================
Press any key to launch PARZEN WINDOW...
       0   1
0  11356   4
1   3688  12
        0      1
0  99.96%  0.04%
1  99.68%  0.32%
The rate of accuracy is  75.48%
============================================
        0      1
0  10611   749
1   1548  2152
         0       1
0  93.41%   6.59%
1  41.84%  58.16%
The rate of accuracy is  84.75%
```

(SVM)

3. With data population, No normalization

```
Press any key to launch PSEUDOINVERSE LEARNING...
      0     1
0  8776  2584
1   614  3086
        0       1
0  77.25%  22.75%
1  16.59%  83.41%
The rate of accuracy is  78.76%
============================================
Press any key to launch PERCEPTRON LEARNING...
   0      1
0  0  11360
1  0   3700
        0       1
0  0.00%  100.00%
1  0.00%  100.00%
The rate of accuracy is  24.57%
============================================
Press any key to launch K NEAREST NEIGHBORS...
       0     1
0  10040  1320
1   2590  1110
        0       1
0  88.38%  11.62%
1  70.00%  30.00%
The rate of accuracy is  74.04%
============================================
Press any key to launch PARZEN WINDOW...
       0     1
0  6088  5272
1  1786  1914
        0       1
0  53.59%  46.41%
1  48.27%  51.73%
The rate of accuracy is  53.13%
============================================
```

(SVM takes over 10 hours, gave up)

4. With data population, With normalization

```
Press any key to launch PSEUDOINVERSE LEARNING...
      0     1
0  8876  2484
1  2405  1295
        0       1
0  78.13%  21.87%
1  65.00%  35.00%
The rate of accuracy is  67.54%
============================================
Press any key to launch PERCEPTRON LEARNING...
      0      1
0  268  11092
1    3   3697
        0       1
0  2.36%  97.64%
1  0.08%  99.92%
The rate of accuracy is  26.33%
============================================
Press any key to launch K NEAREST NEIGHBORS...
       0     1
0  7858  3502
1   661  3039
        0       1
0  69.17%  30.83%
1  17.86%  82.14%
The rate of accuracy is  72.36%
============================================
Press any key to launch PARZEN WINDOW...
      0      1
0  166  11194
1   22   3678
        0       1
0  1.46%  98.54%
1  0.59%  99.41%
The rate of accuracy is  25.52%
============================================
```

(SVM takes over 10 hours, gave up)

(Data population will be discussed in the following part. An population algorithm called SMOTE was used.)

11. Dimensionality reduction interpretation:

I did PCA on the dataset. The results are very similar to the results of using original dataset. I will show the classifiers' performances for two PCA cases: reduce to 52D(half of original dimensionality) and 2D(for data visualization). Here I use No data population, No normalization and do not take SVM into account since it is very time-consuming. I will plot the dataset for 2D case to show how the samples are distributed.

1. 52D

```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5600  5760
1  1859  1841
        0       1
0  49.30%  50.70%
1  50.24%  49.76%
The rate of accuracy is  49.41%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  8549  2811
1  2773   927
        0       1
0  75.26%  24.74%
1  74.95%  25.05%
The rate of accuracy is  62.92%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
        0  1
0  11360  0
1   3700  0
          0      1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  5220  6140
1  1598  2102
        0       1
0  45.95%  54.05%
1  43.19%  56.81%
The rate of accuracy is  48.62%
================================================
Press any key to launch PSEUDOINVERSE LEARNING...
        0     1
0  10694   666
1   1769  1931
         0       1
0  94.14%   5.86%
1  47.81%  52.19%
The rate of accuracy is  83.83%
================================================
Press any key to launch PERCEPTRON LEARNING...
        0     1
0  10781   579
1   2650  1050
         0       1
0  94.90%   5.10%
1  71.62%  28.38%
The rate of accuracy is  78.56%
================================================
Press any key to launch K NEAREST NEIGHBORS...
        0    1
0  11354    6
1   3369  331
         0      1
0  99.95%  0.05%
1  91.05%  8.95%
The rate of accuracy is  77.59%
================================================
Press any key to launch PARZEN WINDOW...
        0   1
0  11359   1
1   3619  81
         0      1
0  99.99%  0.01%
1  97.81%  2.19%
The rate of accuracy is  75.96%
================================================
```

2. 2D

```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5565  5795
1  1821  1879
        0       1
0  48.99%  51.01%
1  49.22%  50.78%
The rate of accuracy is  49.43%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  8590  2770
1  2802   898
        0       1
0  75.62%  24.38%
1  75.73%  24.27%
The rate of accuracy is  63.00%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
        0  1
0  11360  0
1   3700  0
          0      1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  5220  6140
1  1598  2102
        0       1
0  45.95%  54.05%
1  43.19%  56.81%
The rate of accuracy is  48.62%
================================================
Press any key to launch PSEUDOINVERSE LEARNING...
        0     1
0  10694   666
1   1769  1931
         0       1
0  94.14%   5.86%
1  47.81%  52.19%
The rate of accuracy is  83.83%
================================================
Press any key to launch PERCEPTRON LEARNING...
        0     1
0  10781   579
1   2650  1050
         0       1
0  94.90%   5.10%
1  71.62%  28.38%
The rate of accuracy is  78.56%
================================================
Press any key to launch K NEAREST NEIGHBORS...
        0    1
0  11354    6
1   3369  331
         0      1
0  99.95%  0.05%
1  91.05%  8.95%
The rate of accuracy is  77.59%
================================================
Press any key to launch PARZEN WINDOW...
        0   1
0  11359   1
1   3619  81
         0      1
0  99.99%  0.01%
1  97.81%  2.19%
The rate of accuracy is  75.96%
================================================
```

It is easy to tell that when there is no data population and no normalization, the classifiers' performances does not change so much when the dimensionality was reduced. That means lots of our features are highly correlated so that some of them can be deleted safely. Actually the distribution free classifiers and statistical classifiers do not change at all, meaning that when we are using the original features, some of them are too trivial to affect the classification. This is what I expected because some features' values are much smaller than others'. For example, fnlwgt is 10^5 or 10^6 while the binary features are just 0 or 1. In order to let them have the same importance, I will normalize the dataset and show results again.

1. 52D

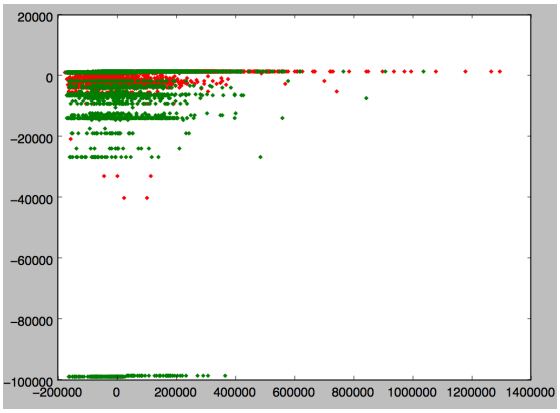
```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5743  5617
1  1805  1895
        0       1
0  50.55%  49.45%
1  48.78%  51.22%
The rate of accuracy is  50.72%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  8571  2789
1  2783   917
        0       1
0  75.45%  24.55%
1  75.22%  24.78%
The rate of accuracy is  63.00%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
       0  1
0  11360  0
1   3700  0
        0       1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  8322  3038
1   537  3163
        0       1
0  73.26%  26.74%
1  14.51%  85.49%
The rate of accuracy is  76.26%
================================================
Press any key to launch PSEUDOINVERSE LEARNING...
       0     1
0  4921  6439
1  1614  2086
        0       1
0  43.32%  56.68%
1  43.62%  56.38%
The rate of accuracy is  46.53%
================================================
Press any key to launch PERCEPTRON LEARNING...
       0     1
0  10127  1233
1   1870  1830
        0       1
0  89.15%  10.85%
1  50.54%  49.46%
The rate of accuracy is  79.40%
================================================
Press any key to launch K NEAREST NEIGHBORS...
        0     1
0  10452   908
1   1628  2072
        0       1
0  92.01%   7.99%
1  44.00%  56.00%
The rate of accuracy is  83.16%
================================================
Press any key to launch PARZEN WINDOW...
       0  1
0  11358  2
1   3700  0
         0       1
0   99.98%  0.02%
1  100.00%  0.00%
The rate of accuracy is  75.42%
================================================
```

2. 2D

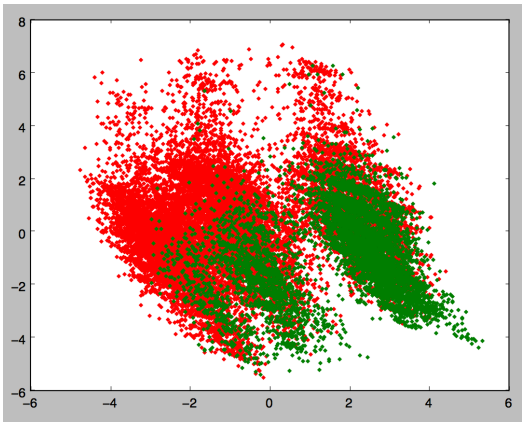
```
Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...
      0     1
0  5609  5751
1  1855  1845
        0       1
0  49.38%  50.62%
1  50.14%  49.86%
The rate of accuracy is  49.50%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...
      0     1
0  8497  2863
1  2817   883
        0       1
0  74.80%  25.20%
1  76.14%  23.86%
The rate of accuracy is  62.28%
================================================
Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...
       0  1
0  11360  0
1   3700  0
        0       1
0  100.00%  0.00%
1  100.00%  0.00%
The rate of accuracy is  75.43%
================================================
Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my baseline)...
      0     1
0  8322  3038
1   537  3163
        0       1
0  73.26%  26.74%
1  14.51%  85.49%
The rate of accuracy is  76.26%
================================================
Press any key to launch PSEUDOINVERSE LEARNING...
       0     1
0  4921  6439
1  1614  2086
        0       1
0  43.32%  56.68%
1  43.62%  56.38%
The rate of accuracy is  46.53%
================================================
Press any key to launch PERCEPTRON LEARNING...
       0     1
0  10127  1233
1   1870  1830
        0       1
0  89.15%  10.85%
1  50.54%  49.46%
The rate of accuracy is  79.40%
================================================
Press any key to launch K NEAREST NEIGHBORS...
        0     1
0  10452   908
1   1628  2072
        0       1
0  92.01%   7.99%
1  44.00%  56.00%
The rate of accuracy is  83.16%
================================================
Press any key to launch PARZEN WINDOW...
       0  1
0  11358  2
1   3700  0
         0       1
0   99.98%  0.02%
1  100.00%  0.00%
The rate of accuracy is  75.42%
================================================
```

The result is again similar to the No data population, with normalization results on the original dataset. The conclusion is that for this particular dataset, PCA works and will reduce the computation time. Here is the data visualization for 2D cases (No data population, without and with normalization). Red stands for class 0(major class), green stands for class1(minor class).

1. 2D without normalization



2. 2D with normalization



However, in order to be safe, when I tried to find the best classifier, I still used the original dataset. So the interpretation is based on the results of **Part 9 and Part 10**.

12. Introduction to SMOTE algorithm:

A special case of my dataset is that it is highly imbalanced. The ratio of number of label 0 and label 1 samples is more than 3:1, which has a great effect on the final model. From the results in Part 9 and 10, we can see that no matter what the final accuracy rate is, class 1's accuracy is always much lower than that of class 0 when no data population is involved. Even when the final accuracy is not bad, for example, the final accuracy is 83.83% using pseudoinverse learning, the accuracy for class 1 is just a little bit higher than 50%, while that for class 0 is higher than 90%. The cause is that I have too many samples for class 0 while I put the same weight for samples from both classes, which will cause the classification greatly in favor of the class with more samples. Although the final accuracy is not bad, I still want to reach a balance because in some practical situation, misclassification might cause serious consequences. For example, in casinos, if you misclassify a rich guy as a not-so-rich guy, you may refuse to give him access to VIP, thus lose the chance to earn more money from that guy. Accuracy is not my only concern, cost of classifications is also important in practice. One way to deal with this situation is to tune the weights for classifiers. However, not every function supports weight tuning. The ultimate way is to change the dataset itself. A good option is to downsample the class with more samples. However, it might cause a problem. Suppose the samples were drawn according to a specific distribution, say normal distribution. When we downsample the dataset, we do not know the mean and variance of the distribution, so we downsample randomly, or with a uniform distribution. This process will break the nature of the dataset, thus cause the inaccuracy of the final result. Another option is to populate the minor class by copying the samples. However, this way will cause the data redundancy problem. SMOTE (Synthetic Minority Over-sampling Technique) algorithm gives us another option. It is basically means interpolation among the minor samples. This way avoided the disadvantages of downsampling and upsampling and is acceptable because it creates samples based on given samples.

13. Interpretation of the results:

(1) Random assignment

Since random assignment is just making guess with a half-half probability, for each case in Part9, their accuracy is around 50%. No much deep research needed.

(2). Random assignment with prior

When there is no data population, random assignment with prior gives each class the accuracy according to the priors, which is about P(S0) = 0.751 and P(S1) = 0.249. When I populate the minor dataset, I changed the priors to P(S0) ≈ P(S1) ≈ 0.5. In the first two cases, from the confusion matrices, values for first columns are around 0.751 and that for second columns are around 0.249. In the third and fourth cases, all values for first and second columns are around 0.5. This is simple probability problem and no deep research needed.

(3) Random assignment with maximum prior

This classifier assigns each test sample with class label with higher prior in the training dataset. It is even simpler than the first two classifiers. However, when we populate the data, if the minor class becomes major after population, the accuracy will be 1-75.43% instead of 75.43%, which is much lower. Luckily, the minor class is still minor after the population in my project, whose prior may be a little bit less than 50%, thus making the performance of this classifier still acceptable.

(4) Minimum to class mean classifier (my baseline)

Compare the cases with and without normalization. When there is no normalization, the accuracy is around 48.6%, while after normalization, the accuracy exceeded 70%. This is not hard to understand because when there is no normalization, only a few features with larger numeric values is affecting the classification. Those features with small values, such as binary values are not providing sufficient importance. However, when normalization is done, every feature is evaluated with the same importance. The difference between with and without normalization tells us that some features with small numeric values are important for the classification, especially when we consider the Euclidean distances. They do not come into play unless normalization is done.

(5) Pseudoinverse learning

The first conclusion is that normalization actually degrades the performance of pseudoinverse learning. This is an unexpected situation. The possible reason might be that I should choose another b for the normalized feature matrix. However, there is something interesting when I applied SMOTE algorithm with pseudoinver learning. When there is data population and no normalization, the accuracy for class 0 is 77.25% and the accuracy for class 1 is 83.14%. 83.14% actually is the best accuracy for class 1 among all classifiers I used and all restrictions involved. SMOTE algorithm changed the situation that classification will always in favor of the major class. However, SMOTE algorithm does not work so well with other classifiers. But one face is very important: SMOTE algorithm more or less improves the accuracy of the minor class. The tradeoff is the worse accuracy of the major class. So although the class 1 accuracy is raised from 52.19%(see first case) to 83.14%, the class 0 accuracy dropped from 94.14% to 77.25% and since class 0 is major, the overall performance actually became worse. SMOTE algorithm still has some practical meaning because if misclassification cost for minor is higher than that of major class,

pseudoinverse + SMOTE will outperform any other available classifiers in this project.

(6) Perceptron learning

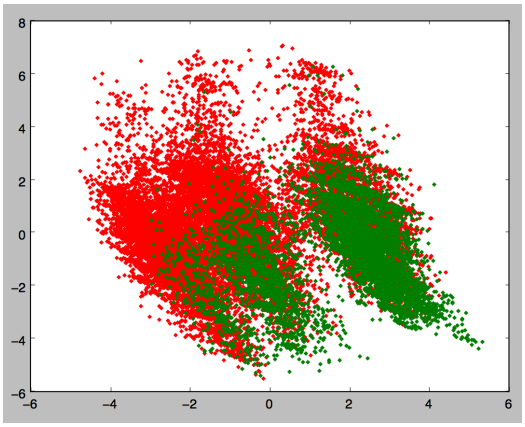
Data population greatly degraded the performance of perceptron learning. Since the samples are not completely linearly separable, perceptron learning will cause lots of iterations in order to minimize the criterion function. After population the data, the it makes the dataset more complex, thus makes it more difficult for the perceptron to converge. Actually after data population, the perceptron learning did not converge as it classified almost all samples into class 1. This actually is not the ideal effect of SMOTE algorithm. However, strictly speaking, it still improves the accuracy of class1, the tradeoff is just too large.

(7) K nearest neighbors

This classifier is another example of performance improvement by SMOTE algorithm. However, whether normalize or not seems to have nothing to do with the accuracy. Although the overall performance became worse after population the minor class, we can still see that the accuracy for minor class improved. Especially in the case with normalization, the accuracy for class 1 increased from 56.00% to 82.14%. The accompanying tradeoff is that the accuracy for class 0 dropped from 92.01% down to 69.17%. That is why the performance of K nearest neighbors is not as good as pseudoinverse learning, the tradeoff is simply too big. The reason why SMOTE works well on K nearest neighbors is that: SMOTE populates the samples using interpolation. When we fix a K, if there is no data population, when we deal with the minor class, the volume will be comparatively large because the samples are far away from each other. While there is data population, when we find the same K nearest neighbors, the volume will become much smaller(since we populated 2 times the number of original samples, theoretically the volume should be 1/3 of the original case on average). Using $p = \frac{k_n}{n * V}$, since V becomes smaller, p becomes larger. That is the mathematical explanation of the improvement of accuracy of the minor class.

(8) Parzen window

Normalization does not affect the overall accuracy so much for parzen window. However, SMOTE does not work well with parzen window. Actually when working with SMOTE, parzen window performed poorly. However, SMOTE still achieved what it is supposed to do, increasing the accuracy of minor class. We can see that for no normalization case, the accuracy of class 1 improved from 2.19% to 51.73%, for normalization case, the accuracy improved from 0.32% to 99.41%. The tradeoff is obvious, the accuracy of class 0 dropped from 99.99% and 99.96% down to 53.59% and 1.46% respectively. The reason is that when we fix the volume, the data population makes the number of class 1 samples increased a lot. This problem involves the process of interpolation. SMOTE algorithm is based on the given minor samples instead of its distribution. So the data population only occurs in between the given samples instead of drawing another sets of samples from the actual distribution. The original minor class samples' distribution is not as broad as that of the major class samples'. We can see from the data visualization plots in Part11.

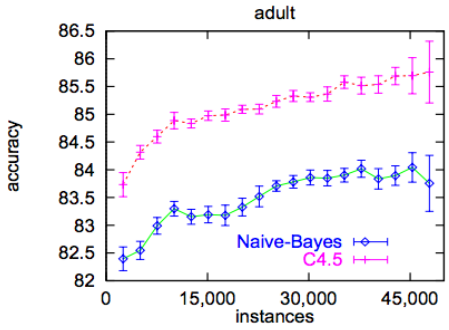


(red area is large, however, after interpolation by SMOTE, both areas contains same number of samples)

It is obvious that green samples occupy less area than red samples. However, we still force the population process to proceed by SMOTE. The Red area is about twice the size of green area while both areas contains almost the same number of samples. When a new sample comes in for classification, as long as it is close to the green samples, the probability calculated will be so high that the test samples was easier to be classified into the originally minor class. And this is the theory of SMOTE algorithm. Since the parzen window fixes the volume, there is no wonder that the classifier will be in favor of the originally minor class.

(9) Support vector machine

I achieved the best performance by SVM after normalized the dataset, which is 84.75%. Despite of the imbalanced dataset, SVM still got 58.16% accuracy on minor class and keeps high accuracy of 93.41% for major class. This is very close to the performance mentioned in *Scaling Up the Accuracy of Naïve Bayes Classifiers a DecisionTree Hybrid* by Ron Kohavi. I have 30162 samples after deleting the samples with missing values. The paper's accuracy is a little bit higher than 85% by C4.5. 84.75% is not a bad performance when compared to the paper's result.



However, if the real case requires some costs on misclassification, I think the best choice should be pseudoinverse learning with SMOTE and no normalization. It reaches a balance of accuracy for both major and minor class. Since SVM takes a lots of time, I did not get the result for SVM working with SMOTE algorithm(still running after 10 hours). I am confident that SVM should work better with SMOTE if given more time.

14. References:

1. Ron Kohavi, *Scaling Up the Accuracy of Naïve Bayes Classifiers a DecisionTree Hybrid*, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 1996
2. Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip, Kegelmeyer *SMOTE: Synthetic Minority Over-sampling Technique*, Journal of Artificial Intelligence Research, 2002
3. http://pydoc.net/Python/costcla/0.02-git/costcla.sampling._smote/

15. Codes:

Please put main.py, adult-data, adult-test in the same folder. Type in "python3 main.py" to run the program. Follow the instruction popped on the screen.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas
import math
from random import randrange, choice
from sklearn.preprocessing import scale, add_dummy_feature
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.lda import LDA
from sklearn.cross_validation import KFold
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import Perceptron
from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier,
NearestNeighbors
from sklearn.svm import SVC

# read datasets, return feature matrix and label vector
def read_data(filename):
  f = open(filename)
  tr_data = []
  tr_label = []
  for line in f:
    readline = line.split(', ')
    instance = []
    if not(any("?" in s for s in readline)):# do not pick up samples with missing
values
      # create feature matrix
      instance.append(readline[0])
      # categorical feature: workclass
      if readline[1] == 'Private':
        temp = [0] * 8
        temp[0] = 1
        instance[1:9]= temp
```

```
elif readline[1] == 'Self-emp-not-inc':
    temp = [0] * 8
    temp[1] = 1
    instance[1:9]= temp
elif readline[1] == 'Self-emp-inc':
    temp = [0] * 8
    temp[2] = 1
    instance[1:9]= temp
elif readline[1] == 'Federal-gov':
    temp = [0] * 8
    temp[3] = 1
    instance[1:9]= temp
elif readline[1] == 'Local-gov':
    temp = [0] * 8
    temp[4] = 1
    instance[1:9]= temp
elif readline[1] == 'State-gov':
    temp = [0] * 8
    temp[5] = 1
    instance[1:9]= temp
elif readline[1] == 'Without-pay':
    temp = [0] * 8
    temp[6] = 1
    instance[1:9]= temp
elif readline[1] == 'Never-worked':
    temp = [0] * 8
    temp[7] = 1
    instance[1:9]= temp
instance.append(readline[2])
# categorical feature: education
if readline[3] == 'Bachelors':
    temp = [0] * 16
    temp[0] = 1
    instance[10:26]= temp
elif readline[3] == 'Some-college':
    temp = [0] * 16
    temp[1] = 1
    instance[10:26]= temp
elif readline[3] == '11th':
    temp = [0] * 16
    temp[2] = 1
    instance[10:26]= temp
elif readline[3] == 'HS-grad':
    temp = [0] * 16
    temp[3] = 1
    instance[10:26]= temp
```

```
elif readline[3] == 'Prof-school':
    temp = [0] * 16
    temp[4] = 1
    instance[10:26]= temp
elif readline[3] == 'Assoc-acdm':
    temp = [0] * 16
    temp[5] = 1
    instance[10:26]= temp
elif readline[3] == 'Assoc-voc':
    temp = [0] * 16
    temp[6] = 1
    instance[10:26]= temp
elif readline[3] == '9th':
    temp = [0] * 16
    temp[7] = 1
    instance[10:26]= temp
elif readline[3] == '7th-8th':
    temp = [0] * 16
    temp[8] = 1
    instance[10:26]= temp
elif readline[3] == '12th':
    temp = [0] * 16
    temp[9] = 1
    instance[10:26]= temp
elif readline[3] == 'Masters':
    temp = [0] * 16
    temp[10] = 1
    instance[10:26]= temp
elif readline[3] == '1st-4th':
    temp = [0] * 16
    temp[11] = 1
    instance[10:26]= temp
elif readline[3] == '10th':
    temp = [0] * 16
    temp[12] = 1
    instance[10:26]= temp
elif readline[3] == 'Doctorate':
    temp = [0] * 16
    temp[13] = 1
    instance[10:26]= temp
elif readline[3] == '5th-6th':
    temp = [0] * 16
    temp[14] = 1
    instance[10:26]= temp
elif readline[3] == 'Preschool':
    temp = [0] * 16
```

```
    temp[15] = 1
    instance[10:26]= temp
instance.append(readline[4])
# categorical feature: marital-status
if readline[5] == 'Married-civ-spouse':
    temp = [0] *7
    temp[0] = 1
    instance[27:34]= temp
elif readline[5] == 'Divorced':
    temp = [0] *7
    temp[1] = 1
    instance[27:34]= temp
elif readline[5] == 'Never-married':
    temp = [0] *7
    temp[2] = 1
    instance[27:34]= temp
elif readline[5] == 'Separated':
    temp = [0] *7
    temp[3] = 1
    instance[27:34]= temp
elif readline[5] == 'Widowed':
    temp = [0] *7
    temp[4] = 1
    instance[27:34]= temp
elif readline[5] == 'Married-spouse-absent':
    temp = [0] *7
    temp[5] = 1
    instance[27:34]= temp
elif readline[5] == 'Married-AF-spouse':
    temp = [0] *7
    temp[6] = 1
    instance[27:34]= temp
# categorical feature: occupation
if readline[6] == 'Tech-support':
    temp = [0] *14
    temp[0] = 1
    instance[34:48]= temp
elif readline[6] == 'Craft-repair':
    temp = [0] *14
    temp[1] = 1
    instance[34:48]= temp
elif readline[6] == 'Other-service':
    temp = [0] *14
    temp[2] = 1
    instance[34:48]= temp
elif readline[6] == 'Sales':
```

```
    temp = [0] *14
    temp[3] = 1
    instance[34:48]= temp
elif readline[6] == 'Exec-managerial':
    temp = [0] *14
    temp[4] = 1
    instance[34:48]= temp
elif readline[6] == 'Prof-specialty':
    temp = [0] *14
    temp[5] = 1
    instance[34:48]= temp
elif readline[6] == 'Handlers-cleaners':
    temp = [0] *14
    temp[6] = 1
    instance[34:48]= temp
elif readline[6] == 'Machine-op-inspct':
    temp = [0] *14
    temp[7] = 1
    instance[34:48]= temp
elif readline[6] == 'Adm-clerical':
    temp = [0] *14
    temp[8] = 1
    instance[34:48]= temp
elif readline[6] == 'Farming-fishing':
    temp = [0] *14
    temp[9] = 1
    instance[34:48]= temp
elif readline[6] == 'Transport-moving':
    temp = [0] *14
    temp[10] = 1
    instance[34:48]= temp
elif readline[6] == 'Priv-house-serv':
    temp = [0] *14
    temp[11] = 1
    instance[34:48]= temp
elif readline[6] == 'Protective-serv':
    temp = [0] *14
    temp[12] = 1
    instance[34:48]= temp
elif readline[6] == 'Armed-Forces':
    temp = [0] *14
    temp[13] = 1
    instance[34:48]= temp
# categorical feature: relationship
if readline[7] == 'Wife':
    temp = [0] * 6
```

```
    temp[0] = 1
    instance[48:54]= temp
elif readline[7] == 'Own-child':
    temp = [0] * 6
    temp[1] = 1
    instance[48:54]= temp
elif readline[7] == 'Husband':
    temp = [0] * 6
    temp[2] = 1
    instance[48:54]= temp
elif readline[7] == 'Not-in-family':
    temp = [0] * 6
    temp[3] = 1
    instance[48:54]= temp
elif readline[7] == 'Other-relative':
    temp = [0] * 6
    temp[4] = 1
    instance[48:54]= temp
elif readline[7] == 'Unmarried':
    temp = [0] * 6
    temp[5] = 1
    instance[48:54]= temp
# categorical feature: race
if readline[8] == 'White':
    temp = [0] * 5
    temp[0] = 1
    instance[54:59]= temp
elif readline[8] == 'Asian-Pac-Islander':
    temp = [0] * 5
    temp[1] = 1
    instance[54:59]= temp
elif readline[8] == 'Amer-Indian-Eskimo':
    temp = [0] * 5
    temp[2] = 1
    instance[54:59]= temp
elif readline[8] == 'Other':
    temp = [0] * 5
    temp[3] = 1
    instance[54:59]= temp
elif readline[8] == 'Black':
    temp = [0] * 5
    temp[4] = 1
    instance[54:59]= temp
# categorical feature: sex
if readline[9] == 'Female':
    temp = [0] * 2
```

```
    temp[0] = 1
    instance[59:61]= temp
elif readline[9] == 'Male':
    temp = [0] * 2
    temp[1] = 1
    instance[59:61]= temp
instance.append(readline[10])
instance.append(readline[11])
instance.append(readline[12])
# categorical feature: native-country
if readline[13] == 'United-States':
    temp = [0] * 41
    temp[0] = 1
    instance[64:105]= temp
elif readline[13] == 'Cambodia':
    temp = [0] * 41
    temp[1] = 1
    instance[64:105]= temp
elif readline[13] == 'England':
    temp = [0] * 41
    temp[2] = 1
    instance[64:105]= temp
elif readline[13] == 'Puerto-Rico':
    temp = [0] * 41
    temp[3] = 1
    instance[64:105]= temp
elif readline[13] == 'Canada':
    temp = [0] * 41
    temp[4] = 1
    instance[64:105]= temp
elif readline[13] == 'Germany':
    temp = [0] * 41
    temp[5] = 1
    instance[64:105]= temp
elif readline[13] == 'Outlying-US(Guam-USVI-etc)':
    temp = [0] * 41
    temp[6] = 1
    instance[64:105]= temp
elif readline[13] == 'India':
    temp = [0] * 41
    temp[7] = 1
    instance[64:105]= temp
elif readline[13] == 'Japan':
    temp = [0] * 41
    temp[8] = 1
    instance[64:105]= temp
```

```
elif readline[13] == 'Greece':
    temp = [0] * 41
    temp[9] = 1
    instance[64:105]= temp
elif readline[13] == 'South':
    temp = [0] * 41
    temp[10] = 1
    instance[64:105]= temp
elif readline[13] == 'China':
    temp = [0] * 41
    temp[11] = 1
    instance[64:105]= temp
elif readline[13] == 'Cuba':
    temp = [0] * 41
    temp[12] = 1
    instance[64:105]= temp
elif readline[13] == 'Iran':
    temp = [0] * 41
    temp[13] = 1
    instance[64:105]= temp
elif readline[13] == 'Honduras':
    temp = [0] * 41
    temp[14] = 1
    instance[64:105]= temp
elif readline[13] == 'Philippines':
    temp = [0] * 41
    temp[15] = 1
    instance[64:105]= temp
elif readline[13] == 'Italy':
    temp = [0] * 41
    temp[16] = 1
    instance[64:105]= temp
elif readline[13] == 'Poland':
    temp = [0] * 41
    temp[17] = 1
    instance[64:105]= temp
elif readline[13] == 'Jamaica':
    temp = [0] * 41
    temp[18] = 1
    instance[64:105]= temp
elif readline[13] == 'Vietnam':
    temp = [0] * 41
    temp[19] = 1
    instance[64:105]= temp
elif readline[13] == 'Mexico':
    temp = [0] * 41
```

```
    temp[20] = 1
    instance[64:105]= temp
elif readline[13] == 'Portugal':
    temp = [0] * 41
    temp[21] = 1
    instance[64:105]= temp
elif readline[13] == 'Ireland':
    temp = [0] * 41
    temp[22] = 1
    instance[64:105]= temp
elif readline[13] == 'France':
    temp = [0] * 41
    temp[23] = 1
    instance[64:105]= temp
elif readline[13] == 'Dominican-Republic':
    temp = [0] * 41
    temp[24] = 1
    instance[64:105]= temp
elif readline[13] == 'Laos':
    temp = [0] * 41
    temp[25] = 1
    instance[64:105]= temp
elif readline[13] == 'Ecuador':
    temp = [0] * 41
    temp[26] = 1
    instance[64:105]= temp
elif readline[13] == 'Taiwan':
    temp = [0] * 41
    temp[27] = 1
    instance[64:105]= temp
elif readline[13] == 'Haiti':
    temp = [0] * 41
    temp[28] = 1
    instance[64:105]= temp
elif readline[13] == 'Columbia':
    temp = [0] * 41
    temp[29] = 1
    instance[64:105]= temp
elif readline[13] == 'Hungary':
    temp = [0] * 41
    temp[30] = 1
    instance[64:105]= temp
elif readline[13] == 'Guatemala':
    temp = [0] * 41
    temp[31] = 1
    instance[64:105]= temp
```

```
            elif readline[13] == 'Nicaragua':
                temp = [0] * 41
                temp[32] = 1
                instance[64:105]= temp
            elif readline[13] == 'Scotland':
                temp = [0] * 41
                temp[33] = 1
                instance[64:105]= temp
            elif readline[13] == 'Thailand':
                temp = [0] * 41
                temp[34] = 1
                instance[64:105]= temp
            elif readline[13] == 'Yugoslavia':
                temp = [0] * 41
                temp[35] = 1
                instance[64:105]= temp
            elif readline[13] == 'El-Salvador':
                temp = [0] * 41
                temp[36] = 1
                instance[64:105]= temp
            elif readline[13] == 'Trinadad&Tobago':
                temp = [0] * 41
                temp[37] = 1
                instance[64:105]= temp
            elif readline[13] == 'Peru':
                temp = [0] * 41
                temp[38] = 1
                instance[64:105]= temp
            elif readline[13] == 'Hong':
                temp = [0] * 41
                temp[39] = 1
                instance[64:105]= temp
            elif readline[13] == 'Holand-Netherlands':
                temp = [0] * 41
                temp[40] = 1
                instance[64:105]= temp
            tr_data.append(instance)
            # create label vector
            if readline[14] == '<=50K\n':
                tr_label.append('0')
            elif readline[14] == '>50K\n':
                tr_label.append('1')
    return (np.array(tr_data).astype(float), np.array(tr_label).astype(float))
# data standardization
def standardize(X):
    # use built-in function
```

```
    return scale(X)
# confusion matrix
def conf_mat(y_true, y_pred):
    cm1 = confusion_matrix(y_true, y_pred)
    df1 = pandas.DataFrame(cm1)
    tot0 = cm1[0, 0] + cm1[0, 1]
    tot1 = cm1[1, 0] + cm1[1, 1]
    cm2 = np.array([[format(cm1[0, 0]/tot0,'.2%'), format(cm1[0,
1]/tot0,'.2%')],[format(cm1[1, 0]/tot1,'.2%'), format(cm1[1, 1]/tot1,'.2%')]])
    df2 = pandas.DataFrame(cm2)
    print(df1)
    print(df2)
    correctrate = (cm1[0, 0] + cm1[1, 1])/(tot0 + tot1)
    print("The rate of accuracy is ", format(correctrate, '.2%'))
# returns training matrix after PCA
def PCA_r(X, dim):
    # can change n_components to desired value
    pca = PCA(n_components = dim)
    return pca.fit_transform(X)
# returns training matrix after FLD, however, dimsionality is restricted as 1, not
interesting
def FLD_r(X, y):
    # can change n_components to desired value
    fld = LDA()
    return fld.fit_transform(X, y)
# get training and test sets of k fold cross validation
def get_kfold_train_test(X, y, k):
    kf = KFold(len(X), k)
    for train, test in kf:
        # training sets and test sets
        yield X[train], y[train], X[test], y[test]
# random assignment without prior
def RA(X_tr, y_tr, X_te):
    clf = DummyClassifier(strategy='uniform').fit(X_tr, y_tr)
    y_pred = clf.predict(X_te)
    return y_pred
# random assignment with prior 1: assign with P(S_i)
def RA1(X_tr, y_tr, X_te):
    clf = DummyClassifier(strategy='stratified').fit(X_tr, y_tr)
    y_pred = clf.predict(X_te)
    return y_pred
# random assignment with prior 2: assign with the maximum prior
def RA2(X_tr, y_tr, X_te):
    clf = DummyClassifier(strategy='most_frequent').fit(X_tr, y_tr)
    y_pred = clf.predict(X_te)
    return y_pred
```

```
# baseline system: minimum distance to class mean
def baseline(X_tr, y_tr, X_te):
    y_pred = []
    m0 = X_tr[np.where(y_tr == 0)].mean(0)
    m1 = X_tr[np.where(y_tr == 1)].mean(0)
    for idx in range(len(X_te)):
        if np.linalg.norm(X_te[idx] - m0) < np.linalg.norm(X_te[idx] - m1):
            y_pred.append(0)
        else:
            y_pred.append(1)
    return(np.array(y_pred).astype(float))
# pseudoinverse learning
def pinv(X_tr, y_tr, X_te):
    # augment the feature space
    X_tr_aug = add_dummy_feature(X_tr)
    X_te_aug = add_dummy_feature(X_te)
    X_tr_aug[np.where(y_tr == 1)] = -X_tr_aug[np.where(y_tr == 1)]
    b = np.ones((len(X_tr_aug),))
    w = np.dot(np.linalg.pinv(X_tr_aug), b)
    indicator = np.dot(X_te_aug, w)
    for i in range(len(indicator)):
        if indicator[i] > 0:
            indicator[i] = 0
        else:
            indicator[i] = 1
    return indicator
# perceptron
def percep(X_tr, y_tr, X_te):
    clf = Perceptron(n_iter = 1000)
    X_tr_aug = add_dummy_feature(X_tr)
    X_te_aug = add_dummy_feature(X_te)
    clf.fit(X_tr_aug, y_tr)
    y_pred = clf.predict(X_te_aug)
    return y_pred
# k nearest neighbors
def knn(X_tr, y_tr, X_te):
    neigh = KNeighborsClassifier(n_neighbors = int(math.sqrt(len(X_tr))))
    neigh.fit(X_tr, y_tr)
    y_pred = neigh.predict(X_te)
    return y_pred
# parzen window
def par(X_tr, y_tr, X_te, r):
    neigh = RadiusNeighborsClassifier(radius = r)
    neigh.fit(X_tr, y_tr)
    y_pred = neigh.predict(X_te)
    return y_pred
```

```
# SVM, takes more than an hour, takes forever when working with SMOTE algorithm
def svm(X_tr, y_tr, X_te):
   clf = SVC(kernel = 'linear')
   clf.fit(X_tr, y_tr)
   y_pred = clf.predict(X_te)
   return y_pred
# SMOTE algorithm
def SMOTE(T, N, k):
n_minority_samples, n_features = T.shape

   if N < 100:
      #create synthetic samples only for a subset of T.
      #TODO: select random minortiy samples
      N = 100
      pass

   if (N % 100) != 0:
      raise ValueError("N must be < 100 or multiple of 100")

   N = int(N/100)
   n_synthetic_samples = N * n_minority_samples
   S = np.zeros(shape=(n_synthetic_samples, n_features))

   #Learn nearest neighbours
   neigh = NearestNeighbors(n_neighbors = k)
   neigh.fit(T)

   #Calculate synthetic samples
   for i in range(n_minority_samples):
      nn = neigh.kneighbors(T[i], return_distance=False)
      for n in range(N):
         nn_index = choice(nn[0])
         #NOTE: nn includes T[i], we don't want to select it
         while nn_index == i:
            nn_index = choice(nn[0])

         dif = T[nn_index] - T[i]
         gap = np.random.random()
         S[n + i * N, :] = T[i,:] + gap * dif[:]
   return S
# plot prototypes
def plot(X_tr, y_tr):
   X_tr0 = X_tr[np.where(y_tr == 0)]
   X_tr1 = X_tr[np.where(y_tr == 1)]
   plt.plot(X_tr0[:,0], X_tr0[:,1], 'r.')
   plt.plot(X_tr1[:,0], X_tr1[:,1], 'g.')
```

```
    plt.show()
# read datasets
input('Press any key to load adults dataset(both training and test)...')
X_tr, y_tr = read_data('adult-data')
X_te, y_te = read_data('adult-test')
while(True):
    yorn = input('Do you want to apply SMOTE algorithm? Enter y or n: ')
    if yorn == 'y':
        # populate the minority class by SMOTE algorithm
        X_tr0 = X_tr[np.where(y_tr == 0)]
        X_tr1 = SMOTE(X_tr[np.where(y_tr == 1)], 300, 2)
        X_tr = np.concatenate((X_tr0, X_tr1))
        y_tr0 = y_tr[np.where(y_tr == 0)]
        y_tr1 = y_tr[np.where(y_tr == 1)]
        y_tr = np.concatenate((y_tr0, y_tr1, y_tr1, y_tr1))
        break
    elif yorn == 'n':
        break
    else:
        print("Please enter y or n only")
while(True):
    yorn1 = input('Do you want to standardize data? Enter y or n: ')
    if yorn1 == 'y':
        # data standardization, uncomment if you want to
        X_tr = standardize(X_tr)
        X_te = standardize(X_te)
        break
    elif yorn1 == 'n':
        break
    else:
        print("Please enter y or n only")
while(True):
    yorn = input('Do you want to use PCA? Enter y or n: ')
    if yorn == 'y':
        dim = input('How many dimensions do you want to reduce to? Enter an positive
integer less than 105: ')
        PCA_r(X_tr, int(dim))
        PCA_r(X_te, int(dim))
        break
    elif yorn == 'n':
        break
    else:
        print("Please enter y or n only")
input('Press any key to launch RANDOM ASSIGNMENT WITHOUT PRIOR...')
conf_mat(y_te, RA(X_tr, y_tr, X_te))
print('==========================================')
```

```
input('Press any key to launch RANDOM ASSIGNMENT WITH PRIOR...')
conf_mat(y_te, RA1(X_tr, y_tr, X_te))
print('================================================')
input('Press any key to launch RANDOM ASSIGNMENT WITH MAXIMUM PRIOR...')
conf_mat(y_te, RA2(X_tr, y_tr, X_te))
print('================================================')
input('Press any key to launch MINIMUM TO CLASS MEAN CLASSIFIER (my
baseline)...')
conf_mat(y_te, baseline(X_tr, y_tr, X_te))
print('================================================')
input('Press any key to launch PSEUDOINVERSE LEARNING...')
conf_mat(y_te, pinv(X_tr, y_tr, X_te))
print('================================================')
input('Press any key to launch PERCEPTRON LEARNING...')
conf_mat(y_te, percep(X_tr, y_tr, X_te))
print('================================================')
input('Press any key to launch K NEAREST NEIGHBORS...')
conf_mat(y_te, knn(X_tr, y_tr, X_te))
print('================================================')
input('Press any key to launch PARZEN WINDOW...')
if yorn1 == 'n':
   conf_mat(y_te, par(X_tr, y_tr, X_te, 40000))
else:
   conf_mat(y_te, par(X_tr, y_tr, X_te, 27))
print('================================================')
while(True):
   yorn = input('Do you want to launch SUPPORT VECTOR MACHINE? Takes more
than one hour, will not stop if using SMOTE. Enter y or n: ')
   if yorn == 'y':
      conf_mat(y_te, svm(X_tr, y_tr, X_te))
      break
   elif yorn == 'n':
      break
   else:
      print("Please enter y or n only")
```