Project 3
Guantao Zhao

Language: python
Package: pandas, numpy

1. Some samples have missing features: There are several rows of data containing '?'. Replace the missing feature values for nominal and numeric attributes with the modes and means from the training data. Provide your code in the report (pdf file)

   As a first step, I discovered that the missing data existed in only three categories, "workclass", "occupation" and "native-country", After statistics, I found that the mode are "Private: 2270", "Untied States: 2931" and "Prof-specialty: 410" Therefore, all question marks in the dataset are replaced separately.

In [81]:
```
import pandas as pd
import numpy as np
columns = ['age','workclass','fnlwgt','education','education-num','marital-status',
            'occupation','relationship','race','sex','capital-gain','capital-loss',
            'hours-per-week','native-country','class']
# using pd to read data
dataset = pd.read_csv('census-income_10percentData.csv', names=columns)
dataset['workclass']= dataset['workclass'].replace('?','private')
dataset['native-country']= dataset['native-country'].replace('?','United-States')
dataset['occupation']= dataset['occupation'].replace('?','Prof-specialty')
#drop the header
dataset.drop(dataset.index[0],inplace=True)
dataset
```

Out[81]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	class
1	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
2	25	Self-emp-not-inc	176756	HS-grad	9	Never-married	Farming-fishing	Own-child	White	Male	0	0	35	United-States	<=50K
3	23	Local-gov	190709	Assoc-acdm	12	Never-married	Protective-serv	Not-in-family	White	Male	0	0	52	United-States	<=50K
4	53	Self-emp-not-inc	88506	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	40	United-States	<=50K
5	47	Self-emp-inc	109832	HS-grad	9	Divorced	Exec-managerial	Not-in-family	White	Male	0	0	60	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3253	53	Local-gov	155314	HS-grad	9	Married-civ-spouse	Adm-clerical	Wife	White	Female	0	0	40	United-States	>50K
3254	35	private	320084	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	White	Female	0	0	55	United-States	>50K
3255	45	State-gov	103406	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	1977	60	United-States	>50K
3256	67	private	182378	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	9386	0	60	United-States	>50K
3257	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

3257 rows × 15 columns

2. Dealing with discrete (categorical) features: There are some categories that contain discrete features. For example, marital.status can have different values: "Widowed", "Divorced", "Never-married", and so on. Find a good representation for them so that they can be used to train a support vector machine and explain your methodology

In this case, I found that there are 7 categories that need to be converted to Integer, Workclass, Marital-status, Occupation, Relationship, Race, Sex, and Native country. I counted the frequency of occurrence in each category, and assigned the highest frequency name to the maximum value, and the lowest frequency name to the minimum value.

Fox example,
In workclass:

Private	2460--19
Self-emp-not-inc	243--12
Local-gov	213--8
State-gov	124--5
Self-emp-inc	120--4
Federal-gov	95--3
Without-pay	2 –-1

In marital-status:

Married-civ-spouse	1462--15
Never-married	1087--10
Divorced	466--7
Widowed	104--5
Separated	90--4
Married-spouse-absent	46--3
Married-AF-spouse	2—1

In occupation:

Prof-specialty	600-- 13
Craft-repair	396--12
Sales	385--11
Exec-managerial	383--10
Adm-clerical	363--9
Other-service	340--8
Machine-op-inspct	217--7
Handlers-cleaners	149--6
Transport-moving	148--5
Farming-fishing	101--4

Tech-support	96--3
Protective-serv	63--2
Priv-house-serv	16—1

In relationship:

Husband	1284--8
Not-in-family	837--7
Own-child	519--5
Unmarried	358--4
Wife	154--2
Other-relative	105--1

In sex,

Male	2180--0
Female	1077—1

In race:

White	2794-13
Black	316-7
Asian-Pac-Islander	96-5
Amer-Indian-Eskimo	26-2
Other	25 -1

In native-country, only US is 1 and other country is 0

United-States	2984
Mexico	66
Philippines	21
Germany	14
El-Salvador	13
Guatemala	12
England	11
India	10
Cuba	8
Canada	8
Jamaica	8
Puerto-Rico	8
South	7
Poland	7
China	6
Japan	6
Portugal	6
Dominican-Republic	5

Columbia	5
Thailand	5
Vietnam	5
Ecuador	5
Italy	4
Haiti	4
Ireland	4
Taiwan	4
Iran	3
Hong	3
Outlying-US(Guam-USVI-etc)	2
Nicaragua	2
France	2
Greece	2
Hungary	2
Trinadad&Tobago	1
Cambodia	1
Scotland	1
Holand-Netherlands	1
Peru	1

At the same time, I also found that Education_num had represented Education, so I could delete this option.

Out[1]:

	age	workclass	fnlwgt	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	class
1	39	5	77516	13	10	9	7	13	0	2174	0	40	1	-1
2	25	12	176756	9	10	4	5	13	0	0	0	35	1	-1
3	23	8	190709	12	10	2	7	13	0	0	0	52	1	-1
4	53	12	88506	13	15	13	8	13	0	0	0	40	1	-1
5	47	4	109832	9	7	10	7	13	0	0	0	60	1	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3253	53	8	155314	9	15	9	2	13	1	0	0	40	1	1
3254	35	private	320084	13	15	13	2	13	1	0	0	55	1	1
3255	45	5	103406	13	15	10	8	13	0	0	1977	60	1	1
3256	67	private	182378	13	15	13	8	13	0	9386	0	60	1	1
3257	52	4	287927	9	15	10	2	13	1	15024	0	40	1	1

3. Split the dataset for stratified 10-fold-cross validation. Provide your code in the report (pdffile)
   I spilt 3257 fold to 10 fold base on the income

```python
#10 folds
def Kfolds(data):
    size = len(data)
    step = size // 10
    folds = [data[i: i+step].sample(frac=1) for i in range(0, size, step)]
    return folds
moreThan50k = dataset[dataset['income'] == 1]
lessThan50k = dataset[dataset['income'] == -1]
moreThan50k = Kfolds(moreThan50k)
lessThan50k = Kfolds(lessThan50k)
# fill new data to new fold
def new_dataset(moreThan50k, lessThan50k):
    new_data = pd.DataFrame(columns=['age','workclass','fnlwgt','education-num','marital-status','occupation','relatior
    for i in range(10):
        frames1 = [moreThan50k[i], lessThan50k[i]]
        fold = pd.concat(frames1)
        frames2 = [new_data, fold]
        new_data = pd.concat(frames2)
    new_data = Kfolds(new_data)
    return new_data
new_dataset = new_dataset(moreThan50k, lessThan50k)
new_dataset
```

There have one example:

IN fold #1

```
Out[9]: [      age workclass  fnlwgt education-num marital-status occupation  \
         51     31        19   55849            10             15         12
         133    47        12  162924            13              7         10
         201    41        19  176069             9              4          7
         229    49        19   36032            10             10         10
         68     17        19  368700             7             10         11
         ...    ..       ...     ...           ...            ...        ...
         27     19        19  104112             9             10         11
         107    51        19  246519             6             15          6
         186    26        12  284343            12             10         12
         2477   58         4  113806            13             15         10
         26     58        12  321171             9             15          6

      relationship race sex capital-gain capital-loss hours-per-week  \
 51              8   13   0            0            0             45
 133             7    5   0            0            0             60
 201             4   13   1            0            0             40
 229             7    7   1            0            0             40
 68              5   13   0            0            0             28
 ...           ...  ...  ..          ...          ...            ...
 27              4    7   0            0            0             30
 107             8   13   0         2105            0             45
 186             7   13   0            0            0             40
 2477            8   13   0         7688            0             30
 26              8   13   0            0            0             40


      native-country  income
 51                1      -1
 133               0      -1
 201               1      -1
 229               1      -1
 68                1      -1
 ...             ...     ...
 27                0      -1
 107               1      -1
 186               1      -1
 2477              1       1
 26                1      -1

 [325 rows x 14 columns],
```

4. Analyze the features and make a scatter plot with the two features that have the highest information gain. Which features are these and what is their information gain values?

For the first step: It returns the information gain after selecting one feature with functions: E = -sum(p * log(p)) and Gain = E - sum(featureP * feature)

For the second part: I use the class in first part to calculate the information gain for each feature and rank it

There is the result:

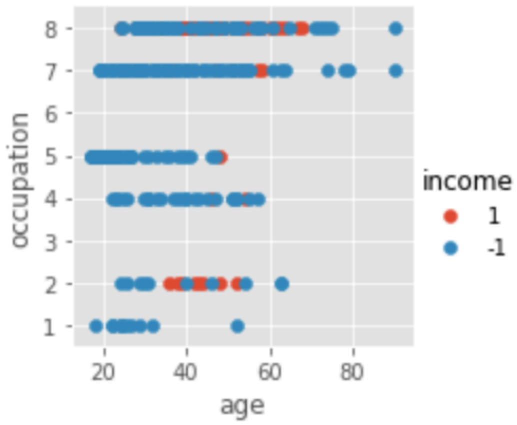
```
--
Information Gain: ['fnlwgt', '0.2496']
Information Gain: ['relationship', '0.1722']
Information Gain: ['marital-status', '0.1679']
Information Gain: ['age', '0.1151']
Information Gain: ['occupation', '0.0975']
Information Gain: ['educationNum', '0.0922']
Information Gain: ['education', '0.0922']
Information Gain: ['capital-gain', '0.0758']
Information Gain: ['hours-per-week', '0.0725']
Information Gain: ['sex', '0.0375']
Information Gain: ['workclass', '0.0201']
Information Gain: ['native-country', '0.0196']
Information Gain: ['race', '0.0085']
Information Gain: ['capital-loss', '0.0033']
```

Therefore, we can found the highest information gain are fnlwgt and relationship. However, we give up the fnlwgt, because each of fnlwgt is different and not desirable. One advantage of using age and occupation is that it is more intuitionistic than other features, at the same time; these two factors are very logical. I do not selection the discrete feature 'relationship' and "marital", it is because the relationship and marital are not too much correlation inside of the classes of the feature and is not appropriate to SVM model. Therefore, I chose highest information gain is age and occupation.

The scatter plot:

```
#######income '1' is > 50k, '-1' is <= 50k ########

<Figure size 57600x28800 with 0 Axes>
```



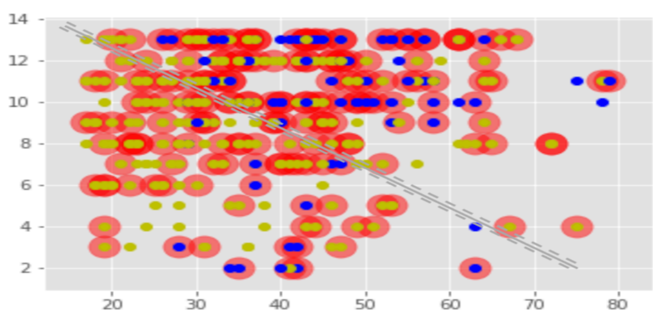
3.1 Train your SVM with stratified 10-fold-cross-validationon the 2 features with the highest information gain and visualize your boundary. i.e. plot the support vectors (list which data points they are), the margin, and draw the decision boundary.

In this case, I still choose the two highest ig, age and education_num.
When implementing linear soft SVM, I followed the following steps:
Load data-> Pick a random integer that is not equal to i-> write a function to make sure A in L and H-> kernelTrans-> find Ek-> randomly pick aj and return E->check ai in KKT-> use SMO mode to find alpha quickly-> test SVM.

In this case, I got 188 support vector and the accuracy is 76%

```
svm with 10-cross-validation
C=0.0001

####### Test #######
there are 188 Support Vectors
the test accuracy rate is: 0.760000
```



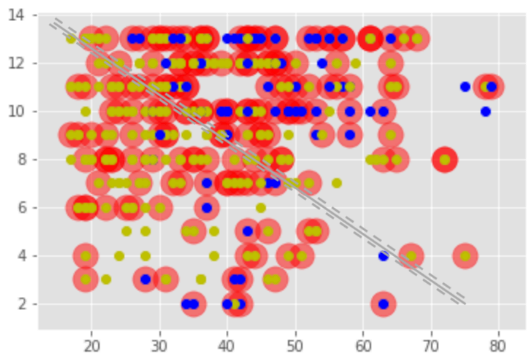
For scatter plots, red circles are representing support vectors and the blue and yellow dots represent age and occupation

3.2 Change the hyper-parameter C from small to larger values. Report your observations on how the value of C would affect SVM's performance. Draw the decision boundaries and margins with smaller and larger values of C to explain its effect in two separate figures.

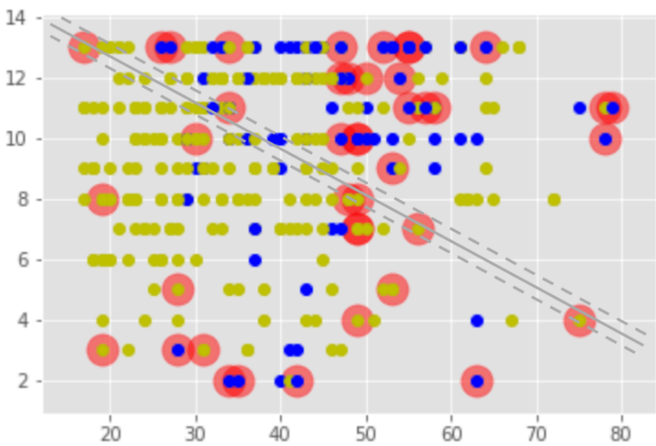
I chose the small C values is c=0.0001, and the large C values is c=0.6

C=0.0001, the accuracy rate is 76.0% and I got 188 support vector

```
svm with 10-cross-validation
C=0.0001
____________________
####### Test #######
there are 188 Support Vectors
the test accuracy rate is: 0.760000
```



C=0.06 , The accuracy rate is 78.8% and I got 42 support vector

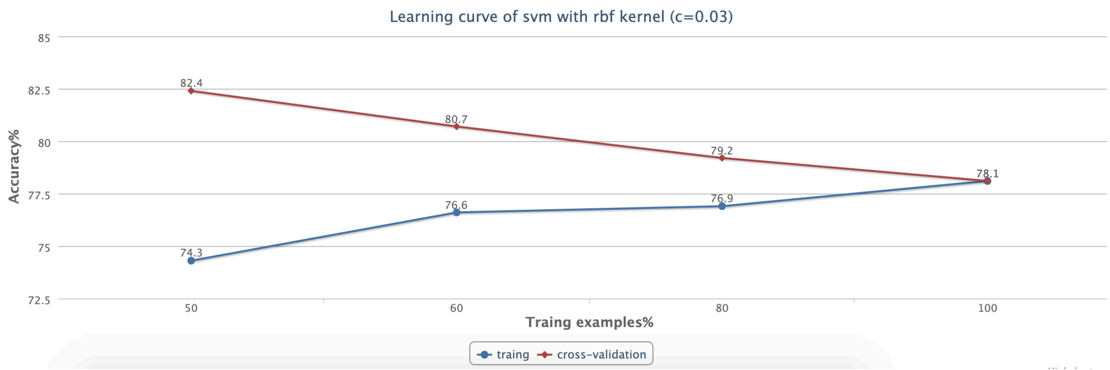


Therefore, we can find from the figure that the size of c will affect the sv number, margin and accuracy. In conclusion, when c increases, accuracy increases, sv quantity decreases and margin decreases. Conversely, when c decreases, accuracy wills decreases, the number of SV increases, and the margin decreases.

3.3 Train the SVM using all the features. Find a way to determine the optimal value of C. Report your methodology and accuracy from stratified 10-fold-cross-validation by using learning curves.

Generally, learning curve of kfold needs to make changes for training examples, but according to the requirements of question, I have trained all the factors and data in this question.
First of all, I split my data in a 10 fold cross validation way as before, and I use a serious of increasing C to test the performance of all features linear svm model. And will finally choose the C with highest performance.

```
C:  0.01 accuracy:0.781
C:  0.03 accuracy:0.781
C:  0.05 accuracy:0.778
C:  0.1 accuracy:0.769
C:  0.2 accuracy:0.769
C:  0.5 accuracy:0.769
C:  0.9 accuracy:0.769
C:  3.0 accuracy:0.769
C:  10.0 accuracy:0.780
```

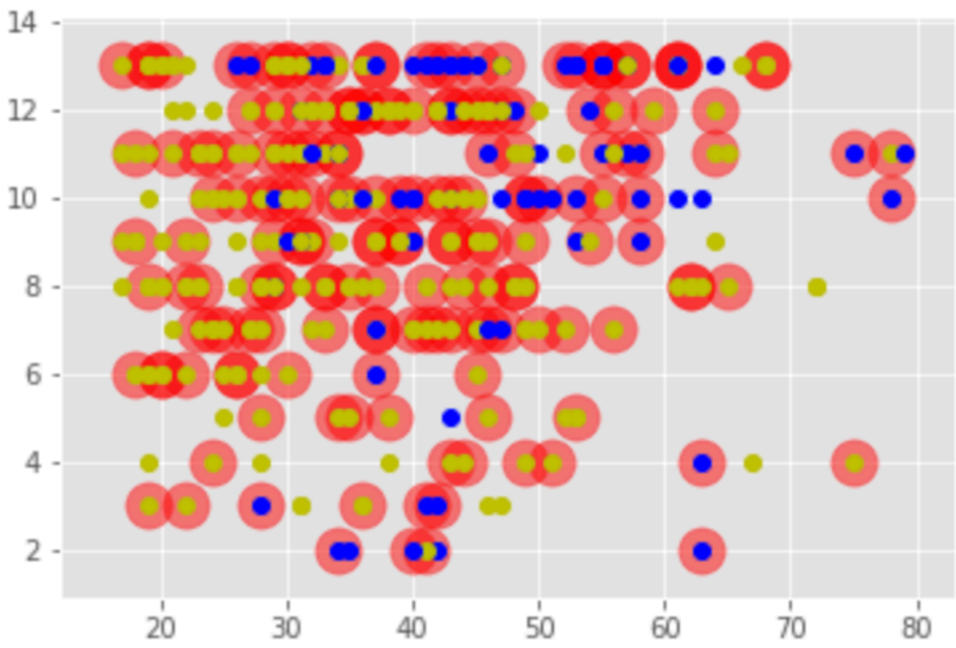


According to the results shown above, we find that the accuracy rate does not vary greatly. We found that performance was highest in 0.01 and 0.03. It performs well at c=10, but overfitting is more likely to occur as c increases. Finally, after comparing the running time, my optimal C value is 0.03 in this linear soft-margin model.

4.1 Compare the performance (precision, recall, f1-score, and variance) of different kernels: Linear, RBF, and polynomial.

I applied each of the three methods and printed out the relevant data (precision, recall, f1-score, and variance). After comparison, we can find that all three methods can achieve more than 70 percent accuracy.

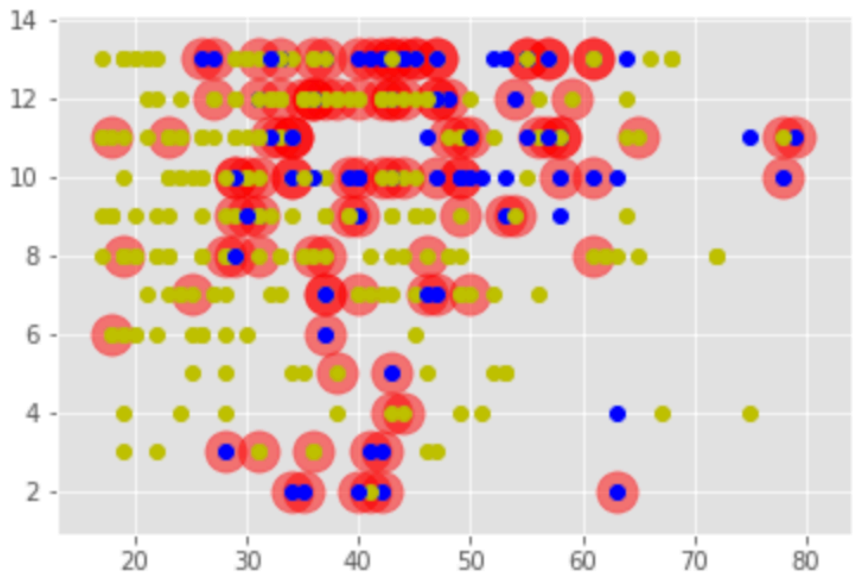
Linear: The red point is test dataset. The total is 178.



I also printed TP,FP,TN,FN to help me find the variance.

```
Linear Kernel
Test#178
C value: 0.01
True Positive: 51
FalsePositive: 28
TrueNegative: 82
FalseNegative: 17
Accuracy: 0.76
Precision: 0.64
Recall: 0.43
F1-Score: 0.392
Variance: 0.00402
```

RBF:




```
RBF
Test#178
C value: 0.5
True Positive: 51
FalsePositive: 35
TrueNegative: 67
FalseNegative: 25
Accuracy: 0.69
Precision: 0.72
Recall: 0.43
F1-Score: 0.55
Variance: 0.00202
```

Polynomial Kernel:

```
Polynomial Kernel
Test#178
C value: 0.0000005
True Positive: 62
FalsePositive: 23
TrueNegative: 78
FalseNegative: 15
Accuracy: 0.79
Precision: 0.66
Recall: 0.43
F1-Score: 0.5
Variance: 0.00301
```

4.2 Try your best to get higher performance! You can design your own kernel, use baggingor boosting methods, logistic regression, decision trees, Na¨ıve Bayes, or whichever method you prefer.Provide your code and your evaluation method, then explain why the performance is better with yourmethod of choice by using learning curves.

In this problem, I will manually implement the KNN kernel and compare it with the previous rbf SVM. In the process of learning, I used all the factors in two kernel, and recording the training set with 50%, 60%, 80%, 90% accuracy in the learning curve.

There is my Knn code

```
mport csv
import random
import math
import operator
import time

start = time.time()

def loadingData(filename, split, trainingSet=[], testSet=[]):
    with open(filename, 'rb') as csvfile:
        lines = csv.reader(csvfile)
        data = list(lines)
        for x in range(len(data)-1):
            for y in range(15):
                data[x][y] = float(data[x][y])
            if random.random() < split:
                trainingSet.append(data[x])
```

```
            else:
                testSet.append(data[x])

#euclidean
def euclideanDistance(ed1, ed2, length):
    distance = 0
    for x in range(length):
        distance += pow((ed1[x] - ed2[x]), 2)
    return math.sqrt(distance)

#Manhattan
def ManhattanDistance(data1, data2, length):
    distance = 0
    for x in range(length):
        for y in range(x + 1, length):
            distance += (abs(data1[x] - data1[y]) + abs(data2[x] - data2[y]))
    return distance

#"compute cosine similarity of v1 to v2: (v1 dot v2)/{||v1||*||v2||)"
def cosine_similarity(data1, data2, length):
    sumxdouble, sumxy, sumydouble = 0, 0, 0
    for i in range(len(data1)):
        x = data1[i]; y = data2[i]
        sumxdouble += x*x
        sumydouble += y*y
        sumxy += x*y
    return sumxy/math.sqrt(sumxdouble*sumydouble)

def findNeighbors(trainingSet, tested, k):
    distances = []
    length = len(tested)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(tested, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors



def getResponse(neighbors):
    Votes = {}
    for x in range(len(neighbors)):
```

```
            response = neighbors[x][-1]
            if response in Votes:
                  Votes[response] += 1
            else:
                  Votes[response] = 1
      sortedVotes = sorted(Votes.iteritems(), key=operator.itemgetter(1), reverse=True)
      return sortedVotes[0][0]



def calAccuracy(testSet, predictions):
      True = 0
      for x in range(len(testSet)):
            if testSet[x][-1] == predictions[x]:
                  True += 1
      return ( True / float(len(testSet))) * 100.0


def main():


      trainingSet = []
      testSet = []
      splitdata = 0.90
      loadingData('121212.csv', splitdata, trainingSet, testSet)
      print 'Train set is : ' + repr(len(trainingSet))
      print 'Test set is : ' + repr(len(testSet))

      predictions= []
      k = 7
      for x in range(len(testSet)):
            neighbors = findNeighbors(trainingSet, testSet[x], k)
            result = getResponse(neighbors)
            predictions.append(result)
            print('--> Predicted=' + repr(result) + ', Actual=' + repr(testSet[x][-1]))
      accuracy = calAccuracy(testSet, predictions)
      print('--------------------------------------------')
      print('--> Accuracy: ' + repr(accuracy) + '%')
      print('--------------------------------------------')
      print '--> Train set is: ' + repr(len(trainingSet))
      print '--> Test set is: ' + repr(len(testSet))
main()


end = time.time()
print('--------------------------------------------')
print ('--> Run time: ' +repr(end - start))
```

training set 50%:

```
----------------------------------------------
--> Accuracy: 84.37306979617047%
----------------------------------------------
--> Train set is: 1637
--> Test set is: 1619
----------------------------------------------
--> Run time: 11.523267030715942
```

training set 60%:

```
----------------------------------------------
--> Accuracy: 83.82022471910112%
----------------------------------------------
--> Train set is: 1921
--> Test set is: 1335
----------------------------------------------
--> Run time: 11.297828197479248
```

training set 80%:

```
----------------------------------------------
--> Accuracy: 86.93009118541033%
----------------------------------------------
--> Train set is: 2598
--> Test set is: 658
----------------------------------------------
--> Run time: 7.338129043579102
```

training set 90%:

```
----------------------------------------------
--> Accuracy: 88.82521489971347%
----------------------------------------------
--> Train set is: 2907
--> Test set is: 349
----------------------------------------------
--> Run time: 4.347449064254761
```

In RBF SVM:

training set 50%:

```
svm with 10-cross-validation
C=0.001
______________________
####### Test #######
there are 67 Support Vectors
the test accuracy rate is: 0.500000
```

training set 60%:

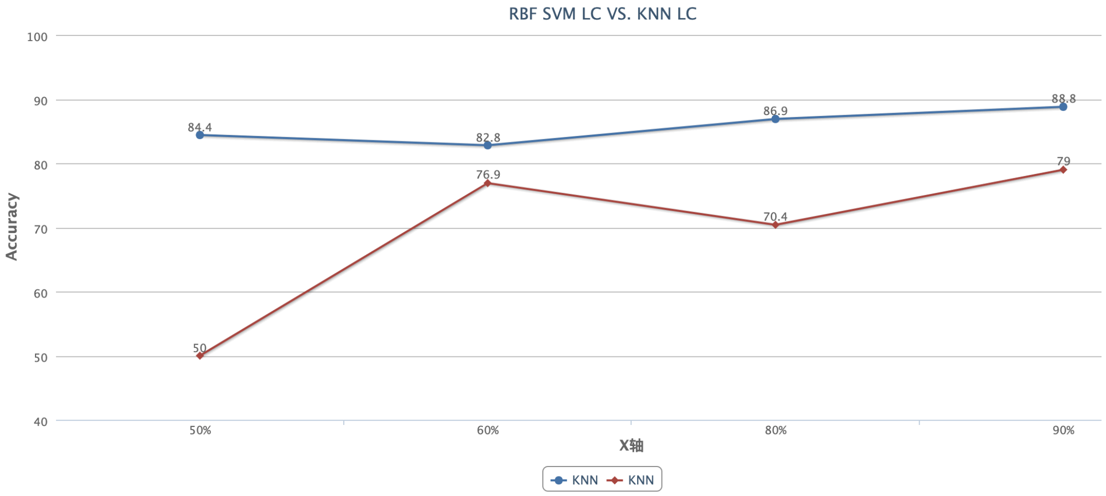
```
svm with 10-cross-validation
C=0.001
______________________
####### Test #######
there are 62 Support Vectors
the test accuracy rate is: 0.769231
```

training set 80%:

```
svm with 10-cross-validation
C=0.001
______________________
####### Test #######
there are 54 Support Vectors
the test accuracy rate is: 0.703846
```

training set 90%:

```
svm with 10-cross-validation
C=0.001
______________________
####### Test #######
there are 88 Support Vectors
the test accuracy rate is: 0.791096
```



From the above learning curve(blue: knn, red: svm), we can find that my KNN method is better than the previous SVM method at each point. I think the most important KNN performs well in such linearly inseparable data. In addition, I also used mode to fill in the missing data in the previous data processing. The values of these data are also very high, so they will have a better performance in the calculation of Euclidean distance.

Reference:

https://blog.csdn.net/csqazwsxedc/article/details/71513197