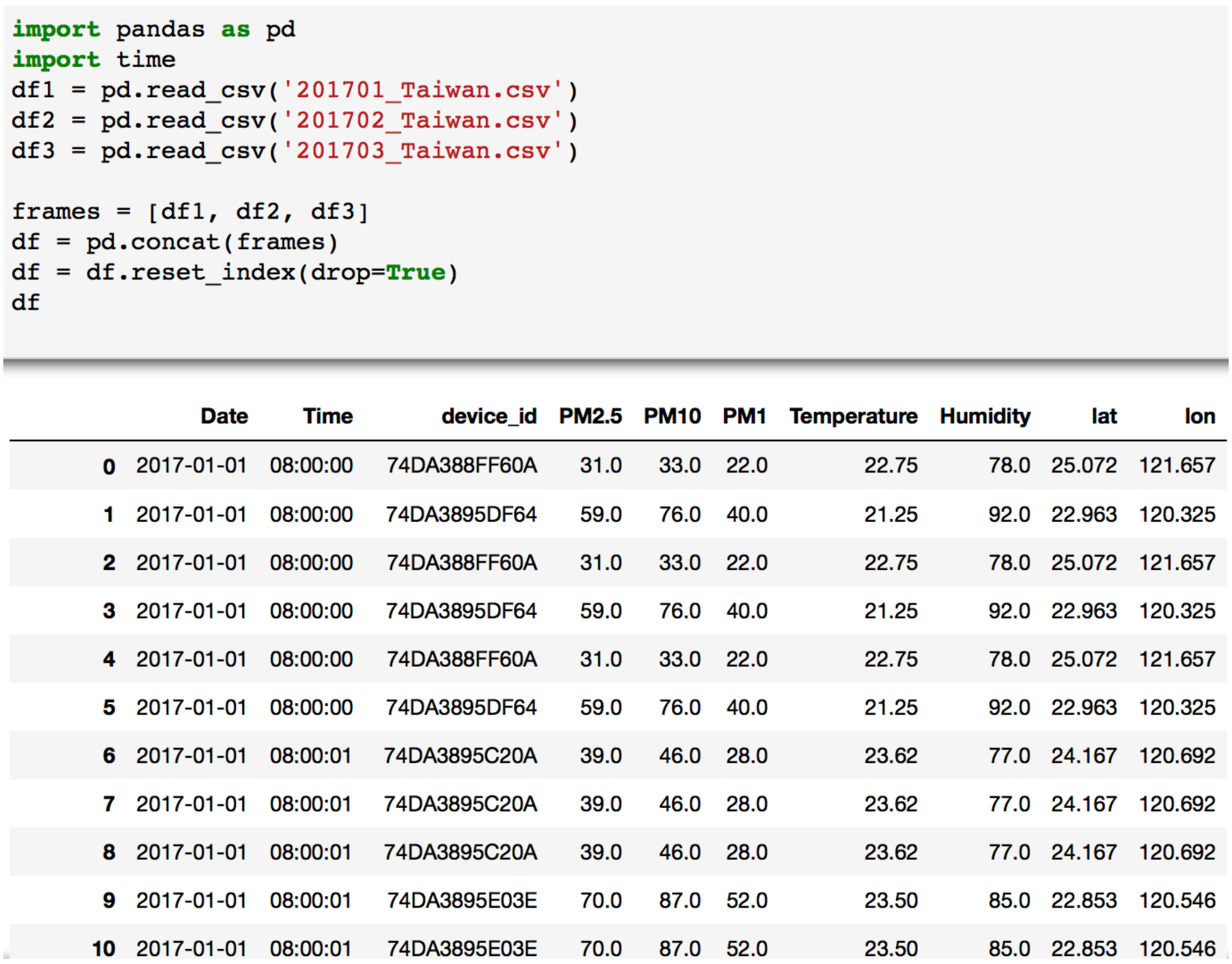
Data Mining HW3

0316323薛世恩

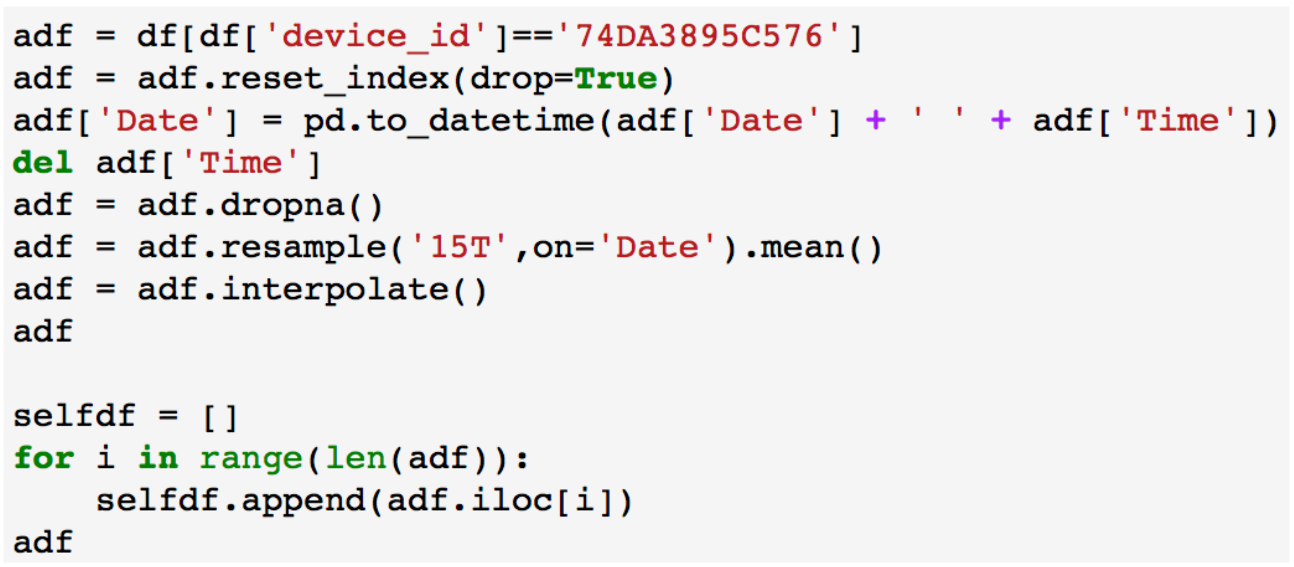
Data preprocessing:



```python
import pandas as pd
import time
df1 = pd.read_csv('201701_Taiwan.csv')
df2 = pd.read_csv('201702_Taiwan.csv')
df3 = pd.read_csv('201703_Taiwan.csv')

frames = [df1, df2, df3]
df = pd.concat(frames)
df = df.reset_index(drop=True)
df
```

	Date	Time	device_id	PM2.5	PM10	PM1	Temperature	Humidity	lat	lon
0	2017-01-01	08:00:00	74DA388FF60A	31.0	33.0	22.0	22.75	78.0	25.072	121.657
1	2017-01-01	08:00:00	74DA3895DF64	59.0	76.0	40.0	21.25	92.0	22.963	120.325
2	2017-01-01	08:00:00	74DA388FF60A	31.0	33.0	22.0	22.75	78.0	25.072	121.657
3	2017-01-01	08:00:00	74DA3895DF64	59.0	76.0	40.0	21.25	92.0	22.963	120.325
4	2017-01-01	08:00:00	74DA388FF60A	31.0	33.0	22.0	22.75	78.0	25.072	121.657
5	2017-01-01	08:00:00	74DA3895DF64	59.0	76.0	40.0	21.25	92.0	22.963	120.325
6	2017-01-01	08:00:01	74DA3895C20A	39.0	46.0	28.0	23.62	77.0	24.167	120.692
7	2017-01-01	08:00:01	74DA3895C20A	39.0	46.0	28.0	23.62	77.0	24.167	120.692
8	2017-01-01	08:00:01	74DA3895C20A	39.0	46.0	28.0	23.62	77.0	24.167	120.692
9	2017-01-01	08:00:01	74DA3895E03E	70.0	87.0	52.0	23.50	85.0	22.853	120.546
10	2017-01-01	08:00:01	74DA3895E03E	70.0	87.0	52.0	23.50	85.0	22.853	120.546

11890929 rows × 10 columns

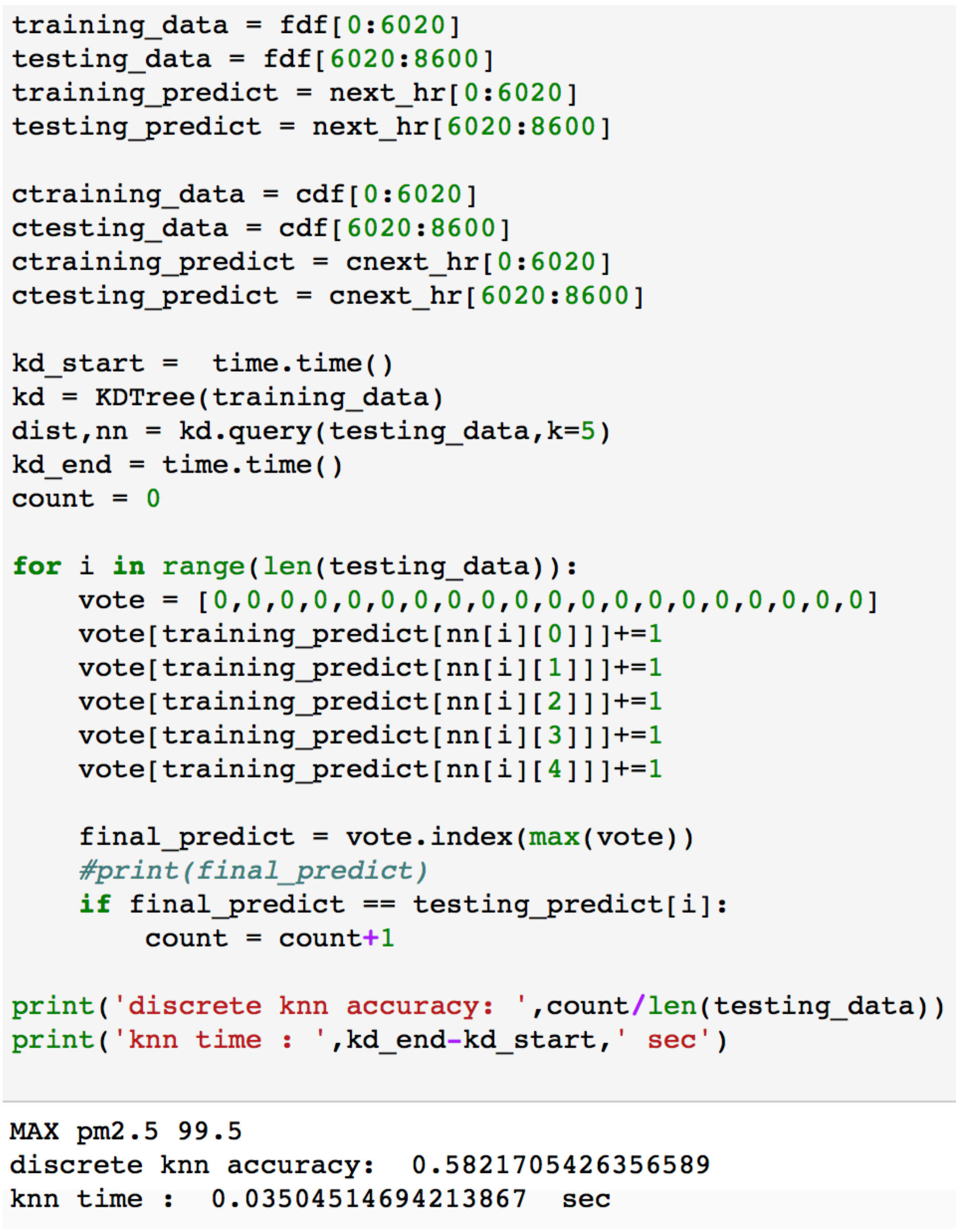


```python
adf = df[df['device_id']=='74DA3895C576']
adf = adf.reset_index(drop=True)
adf['Date'] = pd.to_datetime(adf['Date'] + ' ' + adf['Time'])
del adf['Time']
adf = adf.dropna()
adf = adf.resample('15T',on='Date').mean()
adf = adf.interpolate()
adf

selfdf = []
for i in range(len(adf)):
    selfdf.append(adf.iloc[i])
adf
```

	PM2.5	PM10	PM1	Temperature	Humidity	lat	lon
Date							
2017-01-01 08:00:00	34.000000	40.000000	25.000000	19.750000	87.000000	24.164	120.715
2017-01-01 08:15:00	35.333333	42.000000	25.000000	20.536667	86.333333	24.164	120.715
2017-01-01 08:30:00	41.500000	52.500000	29.000000	21.245000	84.500000	24.164	120.715
2017-01-01 08:45:00	44.333333	55.666667	31.666667	21.870000	83.333333	24.164	120.715
2017-01-01 09:00:00	47.000000	59.000000	33.000000	22.370000	82.000000	24.164	120.715
2017-01-01 09:15:00	51.666667	67.000000	36.000000	22.663333	80.333333	24.164	120.715
2017-01-01 09:30:00	52.000000	67.000000	36.000000	23.060000	79.000000	24.164	120.715
2017-01-01 09:45:00	53.500000	69.000000	36.500000	23.810000	77.000000	24.164	120.715
2017-01-01 10:00:00	54.500000	71.500000	38.500000	24.500000	75.000000	24.164	120.715
2017-01-01 10:15:00	54.500000	71.500000	38.000000	25.310000	74.000000	24.164	120.715

首先先將三個csv檔的資料concat起來，資料筆數變得很多，高達一千多萬筆，將資料的index稍作整理後，我選出資料筆數最多的device來做resample(有先前group by device_id分析過)，並將有缺失的資料做interpolation。

```python
from sklearn.neighbors import KDTree
from random import shuffle
print('MAX pm2.5',adf['PM2.5'].max())
fdf=[]
cdf=[]
next_hr=[]
cnext_hr=[]
for i in range(len(adf)-7):
    fdf.append([adf.iloc[i][0],adf.iloc[i+1][0],adf.iloc[i+2][0],adf.iloc[i+3][0],adf.iloc[i+7][0]])
    cdf.append([adf.iloc[i][0],adf.iloc[i+1][0],adf.iloc[i+2][0],adf.iloc[i+3][0],adf.iloc[i+7][0]])
    #next_hr.append(adf.iloc[i+7][0])
#for i in range(len(adf)-7):
#    print(fdf[i],end=' ')
#    print(next_hr[i])

shuffle(fdf)
shuffle(cdf)


for i in range(len(fdf)):
    next_hr.append(fdf[i][4])
    cnext_hr.append(cdf[i][4])
    fdf[i] = fdf[i][:-1]
    cdf[i] = cdf[i][:-1]


for i in range(len(fdf)):
    #fdf[i][0] = int(fdf[i][0]/10)
    #fdf[i][1] = int(fdf[i][1]/10)
    #fdf[i][2] = int(fdf[i][2]/10)
    #fdf[i][3] = int(fdf[i][3]/10)
    next_hr[i] = int(next_hr[i]/10)
```



```
training_data = fdf[0:6020]
testing_data = fdf[6020:8600]
training_predict = next_hr[0:6020]
testing_predict = next_hr[6020:8600]

ctraining_data = cdf[0:6020]
ctesting_data = cdf[6020:8600]
ctraining_predict = cnext_hr[0:6020]
ctesting_predict = cnext_hr[6020:8600]

kd_start =  time.time()
kd = KDTree(training_data)
dist,nn = kd.query(testing_data,k=5)
kd_end = time.time()
count = 0

for i in range(len(testing_data)):
    vote = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    vote[training_predict[nn[i][0]]]+=1
    vote[training_predict[nn[i][1]]]+=1
    vote[training_predict[nn[i][2]]]+=1
    vote[training_predict[nn[i][3]]]+=1
    vote[training_predict[nn[i][4]]]+=1

    final_predict = vote.index(max(vote))
    #print(final_predict)
    if final_predict == testing_predict[i]:
        count = count+1

print('discrete knn accuracy: ',count/len(testing_data))
print('knn time : ',kd_end-kd_start,' sec')
```

```
MAX pm2.5 99.5
discrete knn accuracy:  0.5821705426356589
knn time :  0.03504514694213867  sec
```

TASK1.
首先先將每小時(4筆pm2.5資料)跟下一小時的pm2.5資料放入list，建出一個二維list，然後做shuffle，因為現在時間資料已經不存在了，所以可以做shuffle，讓資料分佈隨機，之後training的效果會比較好，接下來就是把training data、testing data跟結果分開，然後classification要做離散化，最簡單但是效果不錯的方法就是除以10，0~9為0，10~19為1，20~29為2，以此類推。其中70%作為training data剩餘30%為testing data。

1.K-Nearest-Neighbor
首先輸入為剛剛切好的training data，training data就是一小時內的pm2.5資料(每15分鐘一筆共四筆)。我們先用training data建出一個KDTree，並使用KNN會去尋找最近的k個點，我設定k=5，也就是每一筆testing data會找到最近的五個點，並回傳他們的距離跟index，接下來依照他們回傳的index去找出他們分類，並作投票，由票數最多的作為最後的預測，然後再跟testing data的分類做比較，算出accuracy大概在60%左右，算是相當的高，耗費時間僅有0.035sec，花費時間也算頗短。

```python
from sklearn.naive_bayes import MultinomialNB
nb_start = time.time()
mnb = MultinomialNB()

predictor = mnb.fit(training_data,training_predict)
ans = predictor.predict(testing_data)
nb_end = time.time()

count1 = 0
for i in range(len(testing_data)):
    if ans[i] == testing_predict[i]:
        count1+=1

print('discrete Naive Bayes accuracy : ',count1/len(testing_data))
print('nb time : ',nb_end-nb_start,' sec')
```

```
discrete Naive Bayes accuracy :  0.29534883720930233
nb time :  0.020452976226806641  sec
```

2.Naive Bayes
Naive Bayes假設在features之間都是獨立並利用條件機率來計算的方法。相較其他複雜的分類器，他的假設是非常簡單且運作快速的，且不需要太大的training data就可以預測。並且在面對許多現實問題的時候運作的還不錯。然而在現實中，我們也常常不能確定所給的feature間是完全獨立的，我認為這就有可能會影響到準確度。
我所採用的是Multinomial Naive Bayes，他default alpha =1，也就是有Laplace smoothing，Laplace smooth的重點在於，並不是在資料中沒有出現過的機率就是0，例如某一球隊連續四場比賽都戰敗，但並不代表第五場戰勝的機率就是為0，藉此更貼近現實。
MultinomialNB classifier需要training data跟training data的分類結果。在predict實則跟前面一樣輸入testing data，並跟testing data的種類做比較算出準確度。
至於Multinomial bayes是對於離散features合適的(因為採word counts)所以我也嘗試過把feature跟label一樣做離散化，後來發現準確度較低(accuracy)大概在30%左右。花費時間相當的少，只有0.02sec。我推測準確度低是跟分布還有feature並不是全部獨立有關係。如果feature沒有離散化，就單純以數據當作word counts輸入，準確度也還是有在30%左右。

```python
#fdf[i][0] = int(fdf[i][0]/10)
#fdf[i][1] = int(fdf[i][1]/10)
#fdf[i][2] = int(fdf[i][2]/10)
#fdf[i][3] = int(fdf[i][3]/10)
```

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

```
from sklearn.ensemble import RandomForestClassifier
rf_start = time.time()
clf = RandomForestClassifier(max_depth =2,random_state=0)
clf.fit(training_data,training_predict)
answer = clf.predict(testing_data)
rf_end = time.time()
count2 = 0

for i in range(len(testing_data)):
    if answer[i] == testing_predict[i]:
        count2+=1
print('discrete random forest accuracy : ',count2/len(testing_data))
print('rf time : ',rf_end-rf_start,' sec')
```

```
discrete random forest accuracy :  0.5887596899224806
rf time :  0.03965997695922851 6  sec
```

```
from sklearn.ensemble import RandomForestClassifier
rf_start = time.time()
clf = RandomForestClassifier(max_depth =5,random_state=0)
clf.fit(training_data,training_predict)
answer = clf.predict(testing_data)
rf_end = time.time()
count2 = 0

for i in range(len(testing_data)):
    if answer[i] == testing_predict[i]:
        count2+=1
print('discrete random forest accuracy : ',count2/len(testing_data))
print('rf time : ',rf_end-rf_start,' sec')
```

```
discrete random forest accuracy :  0.6337209302325582
rf time :  0.0563490390777587 9  sec
```

3.Random Forest
Random Forest Classifier 的 number of tree in forest default是10棵，而我也是使用他的default值，至於我有去改動max_depth，可以看出max_depth會去影響其準確度跟時間，depth上升時，有機會讓accuracy上升，但時間也會相對增加。建樹時也是需要跟前面一樣的training data跟training data的label，其內部所使用的是decision tree classifier，只是將很多decision tree的結果選出最好(最多樹投票)的答案，來增加其準度。random_state就是random number generator 所使用的seed。accuracy大概在60%上下，算是非常的高，時間則是相當快速，只需要0.056sec。

```
from sklearn.svm import SVC
svc_start = time.time()
classifier = SVC()
classifier.fit(training_data,training_predict)
svc_answer = classifier.predict(testing_data)
svc_end = time.time()
count3=0

for i in range(len(testing_data)):
    if svc_answer[i] == testing_predict[i]:
        count3+=1
print('discrete svc accuracy : ',count3/len(testing_data))
print('svc time : ',svc_end-svc_start,' sec')
```

```
discrete svc accuracy :  0.5864341085271317
svc time :  2.245457887649536  sec
```

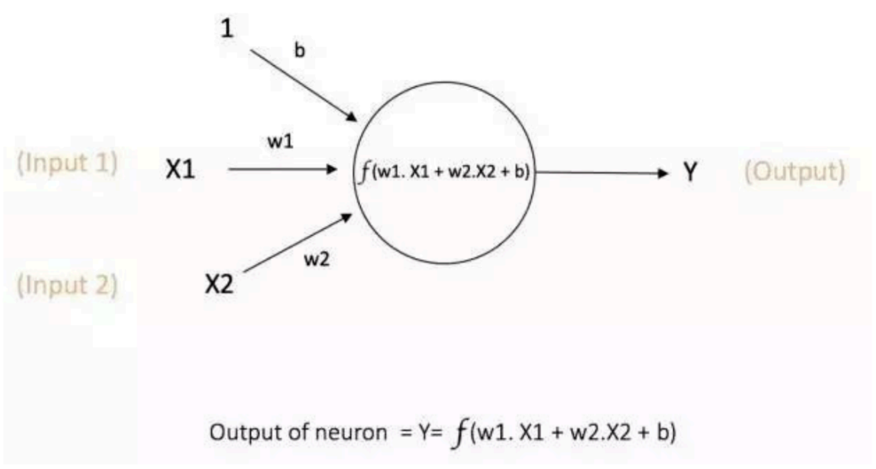
4.Support vector machine (SVC)
SVM試圖找出一個超平面(hyper-plane)將兩個不同的集合分開，且我們希望這個超平面距離集合邊界的距離越遠越好，如此才能清楚辨別點是屬於哪個集合。他針對小樣本有許多優勢，但是樣本數大的時候就不適合，他的時間複雜度較大，可以從其計時看出來2.245秒比前面方法都久上許多。(sklearn網站上說最好不要超過10000 samples)雖然時間花費較長，但是精確度也是在60%左右，有著相當不錯的表現。

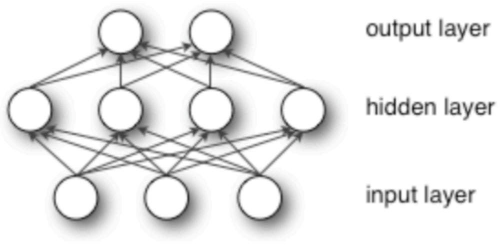
```
from sklearn.neural_network import MLPClassifier
mlp_start = time.time()
mlp = MLPClassifier()
mlp.fit(training_data,training_predict)
mlp_ans = mlp.predict(testing_data)
mlp_end = time.time()
count4 = 0

for i in range(len(testing_data)):
    if mlp_ans[i] == testing_predict[i]:
        count4+=1
print('discrete MLP(Neural Network) accuracy : ',count4/len(testing_data))
print('mlp time : ',mlp_end-mlp_start,' sec')
```

```
discrete MLP(Neural Network) accuracy :  0.5829457364341085
mlp time :  1.4077088832855225  sec
```

5.Neural Network (MLPClassifier)
分成input layer, hidden layer, output layer，上一層的任何一個神經元都跟下一層的所有神經元連結，輸入層如果輸入一個n維向量，就有n個神經元。雖然在時間花費上較長，需要1.4秒，但是在準確度的表現上也在60%左右，有著不錯的表現。MLPClassifier 的defualt hidden_layers_size 為100。

神經網絡中計算的基本單元是神經元，一般稱作「節點」（node）或者「單元」（unit）。節點從其他節點接收輸入，或者從外部源接收輸入，然後計算輸出。每個輸入都輔有「權重」（weight，即 w），權重取決於其他輸入的相對重要性。節點將函數 f（定義如下）應用到加權後的輸入總和，如圖 1 所示：





输入层没什么好说，你输入什么就是什么，比如输入是一个n维向量，就有n个神经元。
隐藏层的神经元怎么得来？首先它与输入层是全连接的，假设输入层用向量X表示，则隐藏层的输出就是
f(W1X+b1)，W1是权重（也叫连接系数），b1是偏置，函数f 可以是常用的sigmoid函数或者tanh函数：

$$sigmoid(a) = 1/(1+e^{-a}) \qquad tanh(a) = (e^a - e^{-a})/(e^a + e^{-a})$$

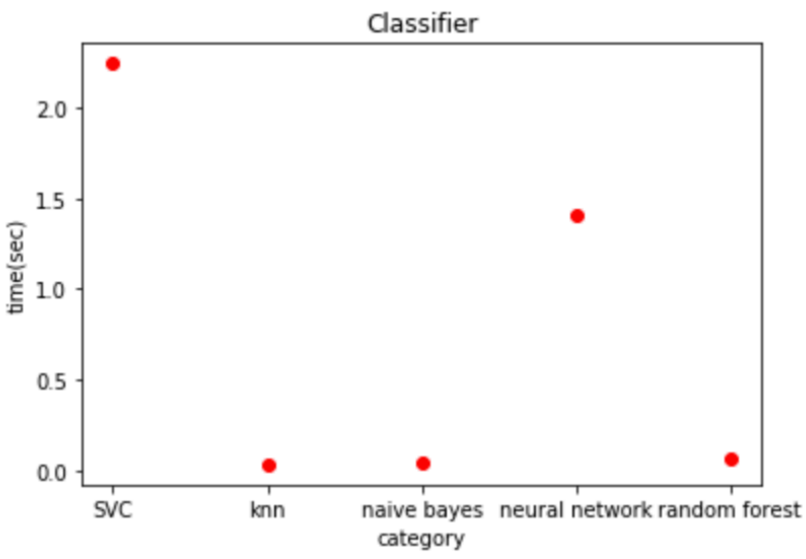
最后就是输出层，输出层与隐藏层是什么关系？其实隐藏层到输出层可以看成是一个多类别的逻辑回归，也即softmax回归，所以输出层的输出就是softmax(W2X1+b2)，X1表示隐藏层的输出f(W1X+b1)。

MLP整个模型就是这样子的，上面说的这个三层的MLP用公式总结起来就是，函数G是softmax

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))),$$

所有classifier的時間比較

```python
import matplotlib.pyplot as plt
time_list = []
time_list.append(kd_end-kd_start)
time_list.append(nb_end-nb_start)
time_list.append(rf_end-rf_start)
time_list.append(svc_end-svc_start)
time_list.append(mlp_end-mlp_start)
name_list = ['knn','naive bayes','random forest','SVC','neural network']
plt.plot(name_list,time_list,'ro')
plt.title('Classifier')
plt.xlabel('category')
plt.ylabel('time(sec)')
plt.show()
```



TASK2.
所有資料的前處理都跟TASK1相同，但是label的部分就沒有做離散化，對所有預測也是採regressor去預測值，並用square error，(預測值-實際值)^2來計算loss。

```
from sklearn.linear_model import BayesianRidge,LinearRegression
import math
br_start = time.time()
bayes_regressor = BayesianRidge()
bayes_regressor.fit(ctraining_data,ctraining_predict)
bayes_ans = bayes_regressor.predict(ctesting_data)
br_end = time.time()
loss = 0

for i in range(len(ctesting_data)):
    loss += math.pow((bayes_ans[i]-ctesting_predict[i]),2)
    #print(math.pow(math.fabs(bayes_ans[i]-ctesting_predict[i]),2))
print('continuous bayes loss : ',loss)
print('br time : ',br_end-br_start,' sec')
```

```
continuous bayes loss :  94066.40142770344
br time :  0.015254974365234375  sec
```

1.Bayesian Regression
Advantage:
It can be used to include regularization parameters in the estimation procedure.
Disadvantage:
Inference of the model can be time consuming.

大部分預測結果相當準，因此loss值非常低。在時間方面，所耗費的時間也相當的少。

```
from sklearn.tree import DecisionTreeRegressor
dr_start = time.time()
dt = DecisionTreeRegressor()
dt.fit(ctraining_data,ctraining_predict)
dt_ans = dt.predict(ctesting_data)
dr_end = time.time()
dt_loss=0

for i in range(len(ctesting_data)):
    dt_loss += math.pow((dt_ans[i]-ctesting_predict[i]),2)
    #print(math.pow(math.fabs(bayes_ans[i]-ctesting_predict[i]),2))
print('continuous DT loss : ',dt_loss)
print('dr time : ',dr_end-dr_start,' sec')
```

```
continuous DT loss :  192520.42191607822
dr time :  0.03093409538269043  sec
```

2.Decision Tree Regression
如果決策樹的最大深度太深，則會受到雜訊影響，也會造成over fitting。
預測值計算出的loss較大一些，但是還在可以接受的範圍，時間也比bayesian regression久。

```python
from sklearn.svm import SVR
svrr_start = time.time()
svr_regressor = SVR()
svr_regressor.fit(ctraining_data,ctraining_predict)
svr_ans = svr_regressor.predict(ctesting_data)
svrr_end = time.time()
svr_loss=0

for i in range(len(ctesting_data)):
    svr_loss += math.pow((svr_ans[i]-ctesting_predict[i]),2)
    #print(math.pow(math.fabs(bayes_ans[i]-ctesting_predict[i]),2))
print('continuous SVR loss : ',svr_loss)
print('svrr time : ',svrr_end-svrr_start,' sec')
```

```
continuous SVR loss :  230044.6603495053
svrr time :  1.7831511497497559  sec
```

3.SVR
SVR本身不適合大量數據，他的時間清楚地反映出在執行數量較大的資料時會花費較久的時間。且他的loss也比前兩個方法更大。因此我覺得在這個data之下，不太適合使用SVM。

```python
from sklearn.neural_network import MLPRegressor

mlpr_start = time.time()
mlpr = MLPRegressor()
mlp = mlpr.fit(ctraining_data,ctraining_predict)
mlp_ans = mlp.predict(ctesting_data)
mlpr_end = time.time()

#for i in range(len(gpr_ans)):
#    print(gpr_ans[i])

mlp_loss=0

for i in range(len(ctesting_data)):
    mlp_loss += math.pow((mlp_ans[i]-ctesting_predict[i]),2)
    #print(math.pow(math.fabs(gpr_ans[i]-ctesting_predict[i]),2))
print('continuous MLPR loss : ',mlp_loss)
print('MLPR time : ',mlpr_end-mlpr_start,' sec')
```

```
continuous MLPR loss :  90452.78879264994
MLPR time :  0.7429032325744629  sec
```

4.MLPRegressor

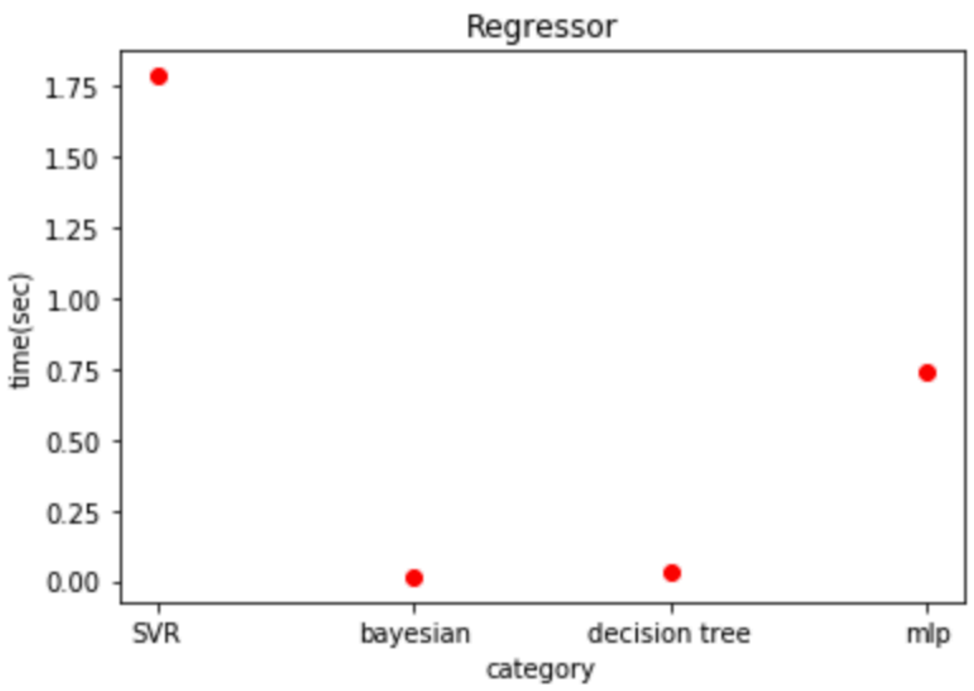
MLPRegressor trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting.

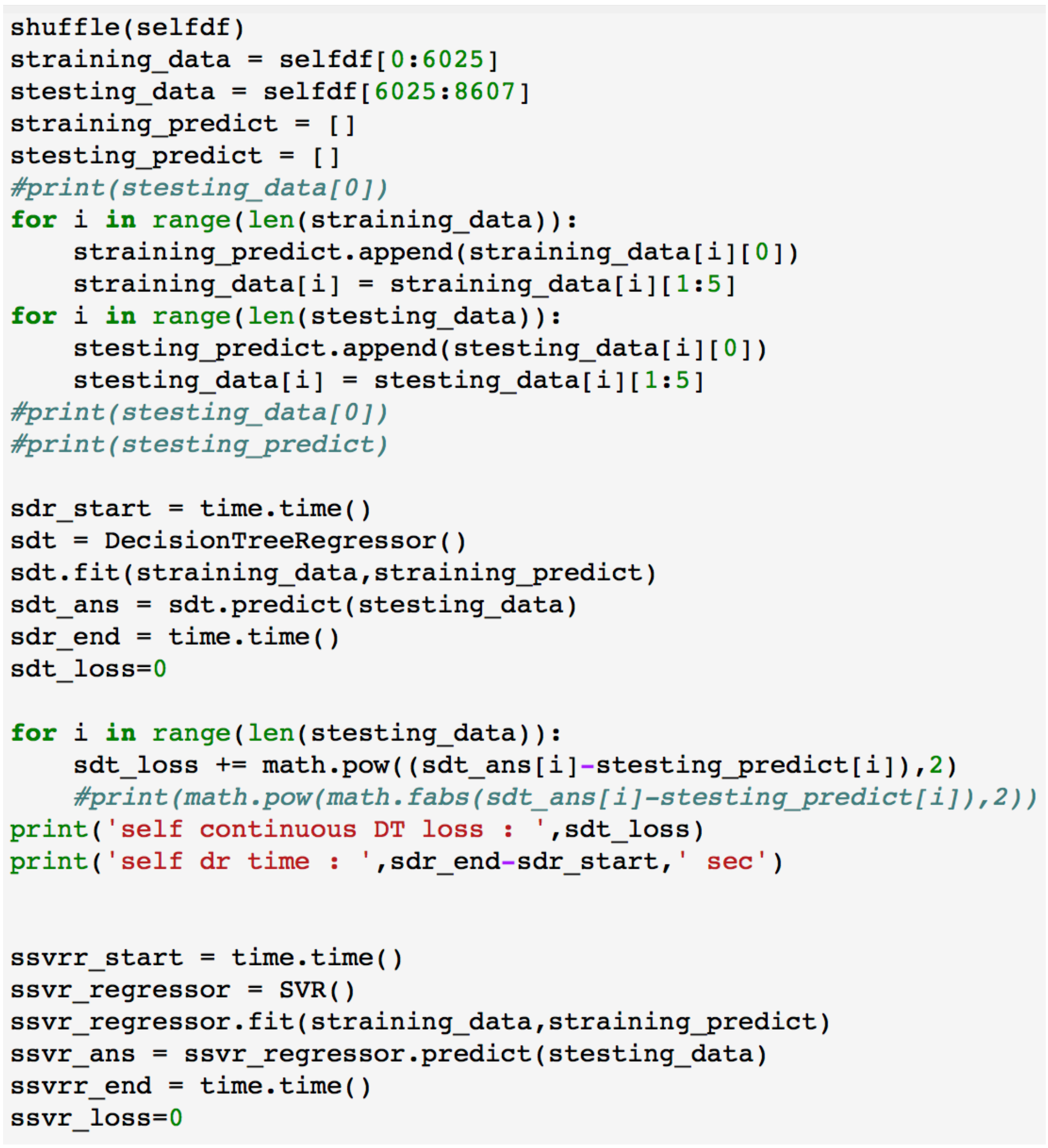
此方法耗時也相較偏高，但是在loss上有良好的表現，是所有方法裡面loss最低的，代表其預測值相當準確。

Regressor時間表現比較

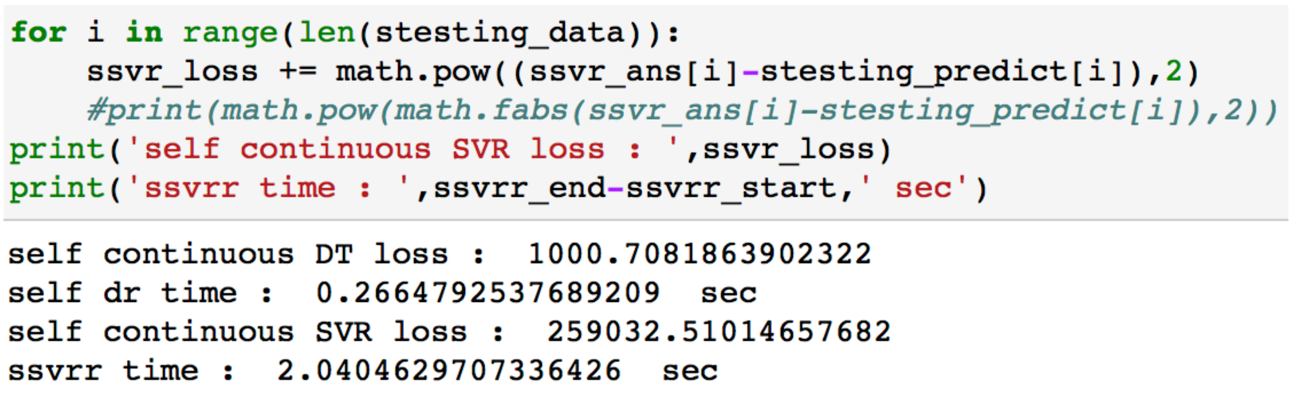
```
ctime_list = []
ctime_list.append(br_end-br_start)
ctime_list.append(dr_end-dr_start)
ctime_list.append(svrr_end-svrr_start)
ctime_list.append(mlpr_end-mlpr_start)
cname_list = ['bayesian','decision tree','SVR','mlp']
plt.plot(cname_list,ctime_list,'ro')
plt.title('Regressor')
plt.xlabel('category')
plt.ylabel('time(sec)')
plt.show()
```



TASK3.
a.problem define
用整理過的資料預測(每十五分鐘一筆)，用PM10、PM1、Humidity、Temperature四個feature來預估pm2.5的值。
b.



```python
shuffle(selfdf)
straining_data = selfdf[0:6025]
stesting_data = selfdf[6025:8607]
straining_predict = []
stesting_predict = []
#print(stesting_data[0])
for i in range(len(straining_data)):
    straining_predict.append(straining_data[i][0])
    straining_data[i] = straining_data[i][1:5]
for i in range(len(stesting_data)):
    stesting_predict.append(stesting_data[i][0])
    stesting_data[i] = stesting_data[i][1:5]
#print(stesting_data[0])
#print(stesting_predict)

sdr_start = time.time()
sdt = DecisionTreeRegressor()
sdt.fit(straining_data,straining_predict)
sdt_ans = sdt.predict(stesting_data)
sdr_end = time.time()
sdt_loss=0

for i in range(len(stesting_data)):
    sdt_loss += math.pow((sdt_ans[i]-stesting_predict[i]),2)
    #print(math.pow(math.fabs(sdt_ans[i]-stesting_predict[i]),2))
print('self continuous DT loss : ',sdt_loss)
print('self dr time : ',sdr_end-sdr_start,' sec')


ssvrr_start = time.time()
ssvr_regressor = SVR()
ssvr_regressor.fit(straining_data,straining_predict)
ssvr_ans = ssvr_regressor.predict(stesting_data)
ssvrr_end = time.time()
ssvr_loss=0
```



```python
for i in range(len(stesting_data)):
    ssvr_loss += math.pow((ssvr_ans[i]-stesting_predict[i]),2)
    #print(math.pow(math.fabs(ssvr_ans[i]-stesting_predict[i]),2))
print('self continuous SVR loss : ',ssvr_loss)
print('ssvrr time : ',ssvrr_end-ssvrr_start,' sec')
```

```
self continuous DT loss :  1000.7081863902322
self dr time :  0.2664792537689209  sec
self continuous SVR loss :  259032.51014657682
ssvrr time :  2.0404629707336426  sec
```

Training data為 PM10 PM1 humidity temperature(70%)
Training predict 為pm2.5的值(70%)
Testing data 為 PM10 PM1 humidity temperature(30%)
Testing product 為pm2.5的值(30%)

1.Decision Tree
Decision Tree非常非常的準確，每一筆誤差幾乎小到1之下，我認為這可能是overfitting，於是我將max_depth改到3(原本為default None)，則loss上升到21000左右。在耗費時間的表現上，Decision Tree所花的時間還是算非常迅速。
2.SVR
SVR的表現就不是那麼出色，不僅loss相當的大，每一筆時常誤差很大，耗費時間也是Decision tree的好幾倍，是因為資料量大的關係。但是準度也是不太準的，可能是跟資料分布有關係。