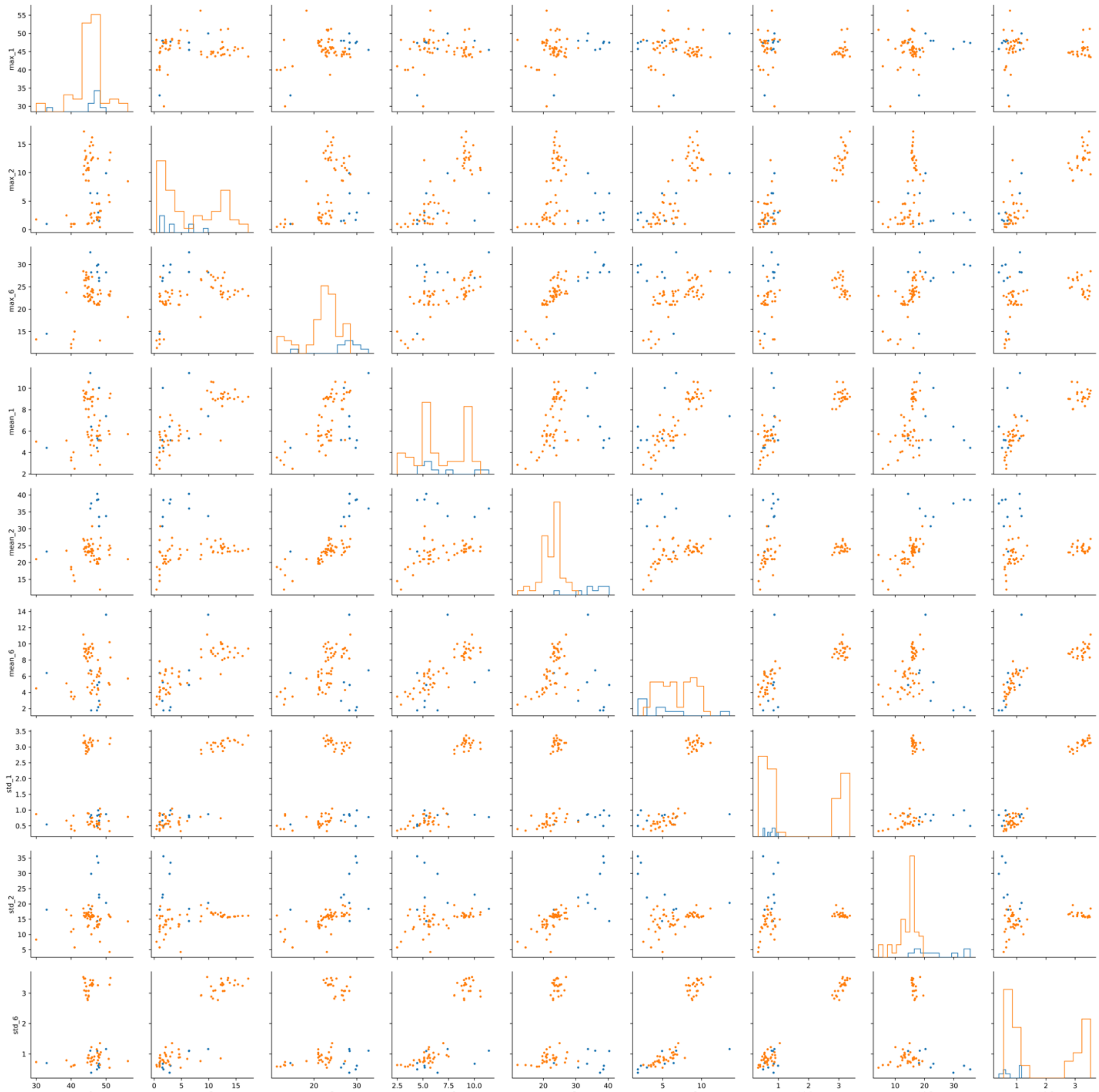
Homework 3 for EE559
Author: Chengyao Wang
USCID: 6961599816
Contact: chengyao@usc.edu
Topic: Logistic Regression

Results and Brief Discussion:
(Courier Fonts are used to get better result alignment.)

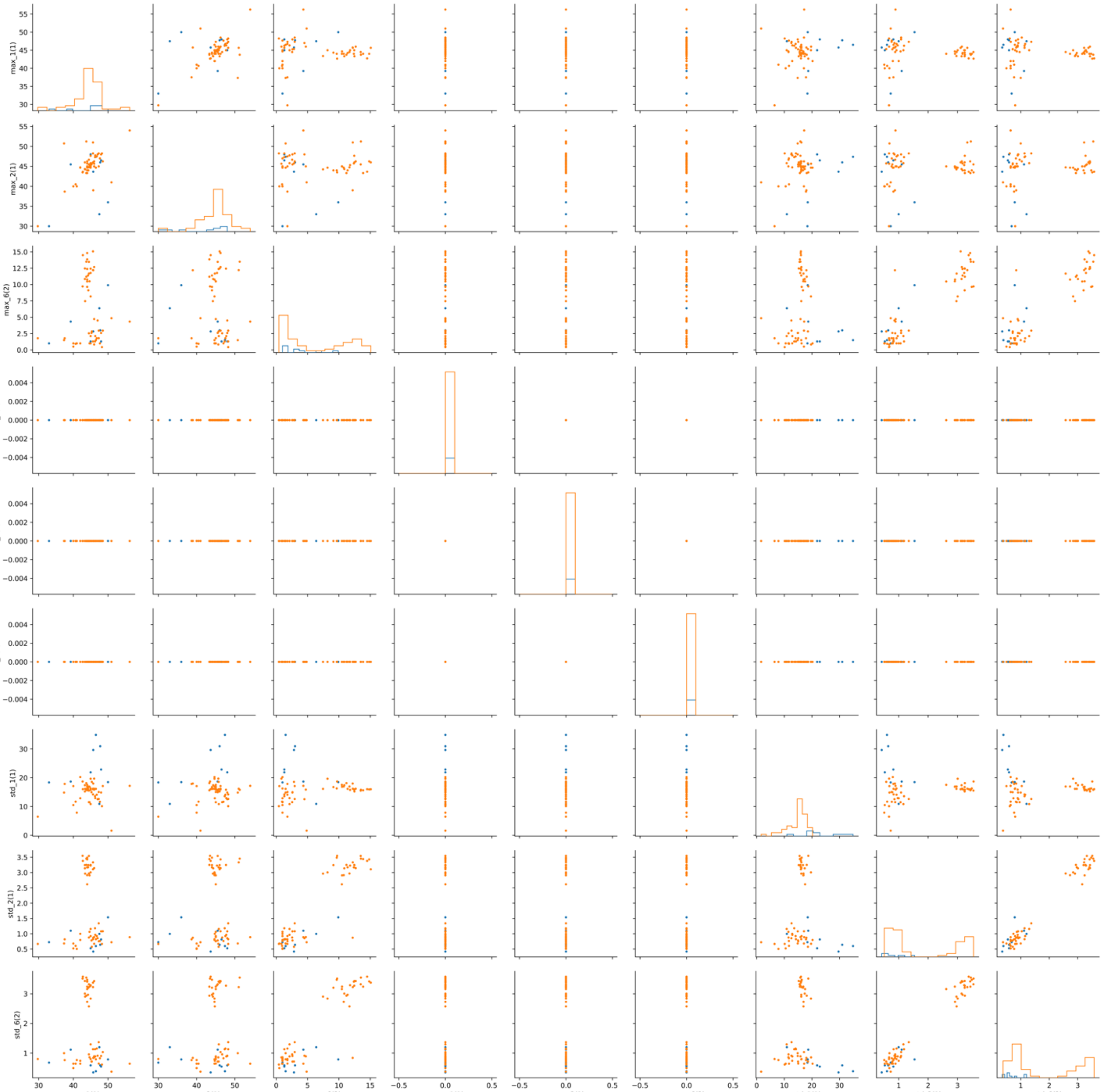
[Pre-processing & Ploting]:

Results:

Scatter plot (time division=1), time series 1,2,6:



Scatter plot (time division=2), time series 1(1), 2(1), 6(2):



Discussion:

Used ‘mean’, ‘std’, ‘max’ for classification.

Commonly used statistical features such as minimum, maximum, mean, median, first quartile, third quartile has a high possibility of having correlation or near-correlation, which is bad for the following statistical analysis, since they all reflects how samples are distributed in its range. Thus, Standard Deviation/Sample variance is included in the flowing exploration. It’s also worth noticing that some time series var_rss12 has a minimum 0, thus minimum is also not used.

[Direct fit, l=1]:

Result:

logModel=sklearn.LogisticRegression(max_iter=10000, C=10000, solver='lbfgs')

```
The Train Result:
[1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
The Test Result:
[1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Coefficient for Logistic Regression:
[[-0.646041 -0.120633  0.587154 -1.207099  3.287315 -2.616488 -1.113481, -
0.215498, -1.558594 -0.311204  1.963192 -0.532586 -1.113481 -0.215498, -
1.558594 -0.311204, 1.963192 -0.532586]]
```

logModel=sklearn.LogisticRegression(max_iter=7, C=10000, solver='lbfgs')

```
The Train Result:
[1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
The Test Result:
[1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Coefficient for Logistic Regression:
[[-0.057973 -0.049863  0.031987 -0.01296   0.117011 -0.032683 -0.072234
  -0.019187  0.011154 -0.01439   0.096655 -0.016627 -0.072234 -0.019187
   0.011154 -0.01439   0.096655 -0.016627]]
```

Discussion:

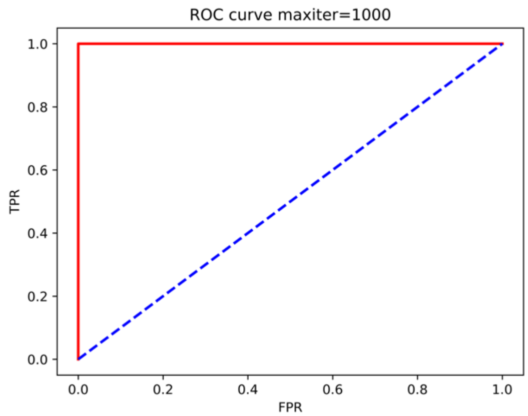
The classification result from sklearn.LogisticRegression is quite good. If not being stopped immaturely, the model will gain full correctness within the scope of current test set and train set. Probably, there will be test errors if more test data is given. The immature model also gains a considerable high correctness, indicating that the data can be easily classified, which is expected to cause problems when are fitted by logistic regression models, one of which is being instability of the algorithm's convergence.

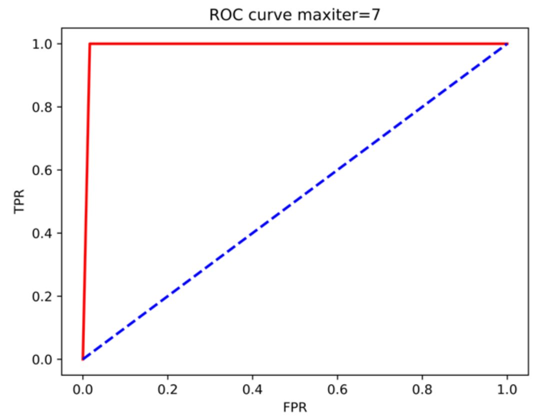
[Feature Selection via CV]

Result:

rfecvModel=LogisticRegression(max_iter=10000, C=10000, solver='lbfgs')
RFECV(estimator=rfecvModel,step=1,cv=StratifiedKFold(5),scoring='accuracy')

```
Best Correct Rate:  1.0
Number of Features:  13
Selected Feature Set:
max
False True  False False True  False False False False False False False
False False False False True  False False False False True  True  True
mean
False False True  False False False False False False False False False
False False False False True  False False False False False False False
std
False False True  False False False False False True  False False False
False False False False True  True  False True  False False False False
Fold which optimal circumstances is in:  4
WHICH is translates into The Name of Features IS:
TIME SERIES 1: (2)max, (3)mean,(3)std                         — 1, 4
TIME SERIES 2: (1)max, (4)max, (4)mean, (4)std                — 16, 21, 22, 23
TIME SERIES 3: (1)std                                         — 26
TIME SERIES 4: (2)mean                                        — 40
TIME SERIES 5: (1)max, (2)max, (1)std, (2)std, (3)std         — 50, 56
TIME SERIES 6: (2)max, (3)max, (4)max                         — 64, 65, 67
Implementing best Classifier Shown Above:
The Train Result:
[1.1.1.1.1.1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
The Test Result:
[1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
Coefficient for Logistic Regression:
[[-0.89306 -0.82921  1.48963 -0.90892 -0.71112 -0.75309 -0.60748  1.01452
  -0.60748 -0.96915  1.01452  0.82083  0.71152]]
Distance from data points to decision hyperplane:
[ 61.98845  60.14579  41.45457  13.10337  10.08737  15.91562  24.33125
  17.8562   26.00907 -36.52966 -39.34569 -40.35577 -37.76007 -34.20199
 -35.95088 -30.64278 -30.47535 -37.14388 -29.52404 -23.41987 -23.33637
 -69.032   -72.34328 -36.0254  -37.76936 -10.91336 -60.85451 -23.62279
 -81.28437 -72.34328 -37.76936 -60.85451 -23.62119 -26.56452 -14.02165
 -10.29081 -12.58768 -29.20141 -26.67615 -28.63305 -26.53325 -40.87919
 -24.93031 -42.99732 -28.79408 -51.58879 -51.89999 -50.74403 -61.44375
 -57.22573 -53.5358  -51.72908 -39.22584 -28.42468 -35.2628  -21.34048
 -46.74376 -36.72588 -28.77572 -37.67548 -34.09418 -33.34843 -35.53191
 -35.37793 -26.46248 -38.15205 -32.16392 -37.42928 -35.27362]
Confusion Matrix:
[[60  0]
 [ 0  9]]
AUC on train set: 1.0
```





rfecvModel=LogisticRegression(max_iter=7, C=10000, solver='lbfgs')
RFECV(estimator=rfecvModel,step=1,cv=StratifiedKFold(5),scoring='accuracy')

```
MANY WARNINGS : FAIL TO CONVERGE
Best Correct Rate:  0.9857142857142858
Number of Features:  6
Selected Feature Set:
max
False False False False False False False False False False True  False
False True  False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False
mean
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False True  True  False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False
std
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False True  True  False False False False False
False False False False False False False False False False False False
False False False False False False False False False False False False
False False False False False False
WHICH is translates into The Name of Features IS:
TIME SERIES 1: (11)max, (14)max                  – 10, 13
TIME SERIES 5: (2)mean, (3)mean, (2)std, (3)std  – 191, 192, 305, 306
```

```
Fold which optimal circumstances is in:  19

The Train Result:
[1.1.1.0.1.1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
The Test Result:
[1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
Coefficient for Logistic Regression:
[[-0.09453 -0.09445  0.08092  0.08956  0.08092  0.08956]]
Distance from data points to decision hyperplane:
[ 3.53902  4.07986  1.73921 -0.31407  0.42129  2.29669  1.04657  1.44479
  4.06264 -3.23789 -2.72847 -2.63113 -2.51171 -2.69883 -2.76329 -2.61878
 -1.70796 -2.41479 -2.64624 -1.70756 -1.81169 -5.12136 -7.18025 -4.1711
 -3.97257 -2.27907 -6.41077 -2.1099  -8.00376 -7.18025 -3.97257 -6.41077
 -2.1099  -2.57638 -1.7931  -0.49757 -2.23413 -2.79375 -3.65395 -3.58144
 -3.24438 -4.21445 -3.80143 -5.86871 -3.45445 -4.06048 -5.36293 -4.49024
 -4.12677 -4.98556 -3.89328 -5.52511 -3.48322 -3.49648 -3.60068 -2.76093
 -4.76177 -2.49901 -3.01028 -2.74506 -2.9643  -2.76936 -2.0948  -2.38102
 -2.31534 -3.20305 -3.01907 -2.95732 -2.70714]
Confusion Matrix:
[[60  1]
 [ 0  8]]
AUC on train set: 0.9918032786885246

A little exploration of the data set and Fisher Information Matrix:
Rank of this 69 * 18 matrix: 12

Eigenvalues of Fisher Information Matrix 3 is:
[ 2.654e+00  1.569e-02  7.101e-03  1.787e-03  9.392e-04  3.113e-04
  6.216e-05  2.378e-06  8.152e-08  5.947e-09  5.229e-11  9.223e-12
 -2.328e-21 -7.609e-24 -3.068e-24  2.005e-24 -8.580e-34  7.232e-28]
```

Discussion:

This is a logistic regression model, and hypothesis test on $\vec{\beta}$ is done using z-test to determine whether a particular β_i is significant. Since $\vec{\beta}$ is the MLE of a likelihood function in binary logistic regression, the theoretical process is the following, also known as Wald Test[1]:

1. Log Likelihood function of logistic regression & its Hessian:

$$\ln\left(\mathcal{L}\left(\vec{\beta}\middle|\mathcal{D}\right)\right) = \sum_{i}^{n}\left[Y_i \ln\left(\frac{1}{1+e^{-(\beta_0+\vec{\beta}x_i)}}\right) + (1-Y_i)\ln\left(1-\frac{1}{1+e^{-(\beta_0+\vec{\beta}x_i)}}\right)\right]$$

$$\mathcal{H}_{pq} = -\frac{\partial^2}{\partial\beta_p\partial\beta_q}\mathcal{L}\left(\vec{\beta}\middle|\mathcal{D}\right) = \sum_{i}^{n} X_{ip}X_{iq}\left[p(\vec{x_i})\left(1-p(\vec{x_i})\right)\right]$$

$$p(\vec{x_i})\left(1-p(\vec{x_i})\right) = \left(\frac{1}{1+e^{-(\beta_0+\vec{\beta}x_i)}}\right)\left(1-\frac{1}{1+e^{-(\beta_0+\vec{\beta}x_i)}}\right) = \frac{1}{\cosh\left((\beta_0+\vec{\beta}x_i)\right)}$$

2. Fisher information matrix:

$$I\left(\hat{\vec{\beta}}\right) = \mathcal{H}\left(\widehat{\vec{\beta}_{MLE}}\right)$$

3. Invert FIM, choose the diagonal elements as standard error:

$$SE\left(\widehat{\beta_{\iota,MLE}}\right) = [I^{-1}\left(\hat{\vec{\beta}}\right)]_{ii}$$

4. Use z-statistics to determine the p-value of $\widehat{\beta_{\iota,MLE}}$:

[1] https://en.wikipedia.org/wiki/Wald_test, Wikipedia, Wald Test

$$p_i = \Phi^{-1}\left(\frac{\widehat{\beta_{\iota,MLE}} - 0}{SE\left(\widehat{\beta_{\iota,MLE}}\right)}\right)$$

5. Sort the features in ascending order using p_i as keyword. Choose the first k features as the result of feature selection.

Note: $(\beta_0 + \vec{\beta}x_i)$ can be achieved using ***logModel.decision_function()***, which returns the Euclidean distance between a data point to the decision hyperplane.

Because the hypothesis test is based on the null hypothesis that $\widehat{\beta_{\iota,MLE}} = 0$, which means that this feature is not related to this classification problem, P value stands for the probability of null hypothesis being true -- the smaller it is, further $\widehat{\beta_{\iota,MLE}}$ is different from 0 and more the $\widehat{\beta_{\iota,MLE}}$'s significance.

But with the current given data set, there are mainly two problems:

1. FIM can't be inverted due to rank deficiency/ill-conditioned which inherited from the data matrix X, even if we choose 'max, mean, std' as the target features. The columns of the data matrix are linearly dependent, and the eigenvalue analysis even indicated that FIM is undefinite. The Hessian/FIM is calculated by:

```
denom = (2.0*(1.0+np.cosh(logModel.decision_function(self.trainSetIn))))
denom = np.tile(denom,(self.trainSetIn.shape[1],1)).T
F_ij3 = np.dot(np.divide(self.trainSetIn, denom).T, self.trainSetIn)
```

2. The characteristic of well seperatedness of the data resulted in unclosed contour of the unregularized convex likelihood function[2], leading normalized $\vec{\beta}$ to reach infinity. If there is complete separation of the data points, the maximum likelihood estimate $\hat{\beta}$ does not exist.[3] Also, the separation of the data will cause computational algorithms, like Newton-Raphson, oscillate when searching for $\widehat{\beta_{MLE}}$, which will lead to unstability as well as divergence. Z-statistics are not feasible in these scenarios, because standard errors are 0s or near-zeros. The following is derived by another package, just for numerical reference, and relationship between separation of data points and contour plot of likelihood function in logistic regression[4]:

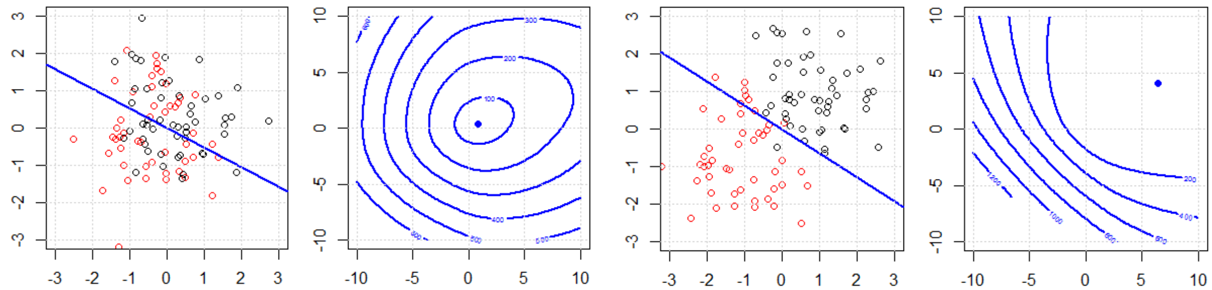
	Coefficients	StandardErrors	z values	Probabilities
0	-0.0795	0.0	-inf	0.0
1	-0.1290	0.0	-inf	0.0
2	-0.1659	0.0	-inf	0.0
3	0.1101	0.0	inf	0.0
4	-0.0589	0.0	-inf	0.0
5	0.6327	0.0	inf	0.0
6	-0.4034	0.0	-inf	0.0
7	-0.2874	-0.0	inf	0.0
8	-0.0495	0.0	-inf	0.0
9	-0.3025	-0.0	inf	0.0
10	-0.0669	-0.0	inf	0.0
11	0.5004	0.0	inf	0.0
12	-0.1073	0.0	-inf	0.0
13	-0.2874	-0.0	inf	0.0
14	-0.0495	0.0	-inf	0.0

[2] Convexity, Maximum Likelihood and all that. Adam Berger, CMU

[3] On the existence of maximum likelihood estimates in logistic regression models. A. Albert, Oxford, 1984

[4] https://stats.stackexchange.com/questions/239928/is-there-any-intuitive-explanation-of-why-logistic-regression-will-not-work-for?noredirect=1&lq=1

15	-0.3025	-0.0	inf	0.0
16	-0.0669	-0.0	inf	0.0
17	0.5004	0.0	inf	0.0
18	-0.1073	0.0	-inf	0.0



Pesudo inverse, inverting after compact SVD or eliminating correlated columns in data matrix X may be a solution to problem 1.

Feature Selection can also be done using recursive feature elimination (either forward or backward). This is done by sklearn.feature_selection.RFECV with CV & Stratified Sampling implemented:

```
rfecvModel=LogisticRegression(max_iter=7, C=10000, solver='lbfgs')
rfecv=RFECV(estimator=rfecvModel,step=1,cv=StratifiedKFold(5),
scoring='accuracy')
rfecv.fit(self.trainSetIn, self.trueTrainLabel)
```

Iterate though l=1~20, to find the highest correct rate in CV.

Choose (l,p) with lowest l and lowest features among those who has the same accuracy. Because lesser the features, the lesser complexity of the model will be. And the Result are shown above. P values of the features selected by RFE are still not avalible due to the afore reasons.

Thoeretically, we cannot used train error to choose models with different complexcities. Because models with more complexity tend to behave better upon training sets, we have to use a validation set/cross validation to choose from different models, or introduce penalty over model complexities, like AIC or BIC.

Since logistic regression asks for instance number to surpass feature numbers, backward stepwise feature elimination is not avalible when time division fold is larger than 8. But sklearn.rfecv & sklearn.logisticregression have already addressed this problem.[5]

```
X, y = check_X_y(X, y, accept_sparse='csr', dtype=np.float64, order="C",
accept_large_sparse=solver != 'liblinear')
```

[5] https://github.com/scikit-learn/scikit-learn/blob/7813f7efb/sklearn/linear_model/logistic.py#L1202, logisticregression Source Code

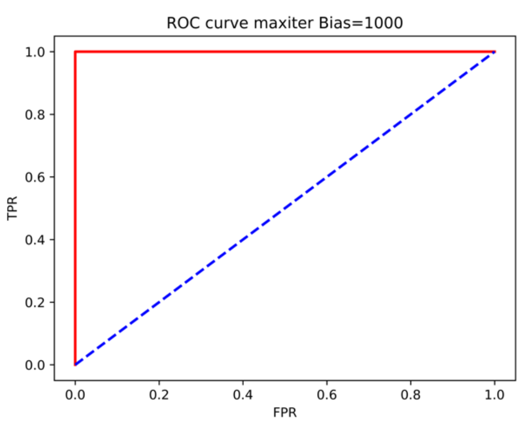
[Intercept Calibration]

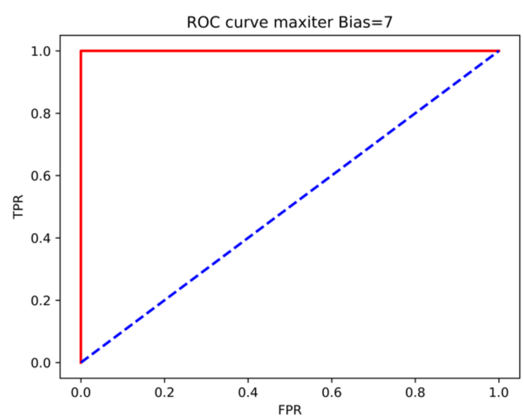
Result:

```
Bias= 0.1446693982
Fit into Original Logistic Regression:
```

MAX_Iter=1000:

```
The Train Result:
[1.1.1.1.1.1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
The Test Result:
[1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
Coefficient for Logistic Regression:
[[-0.58416 -0.6741   1.34176 -0.7865  -0.5998  -0.56334 -0.43941  0.6251
  -0.43941 -0.64014  0.6251   0.34477  0.4963 ]]
Distance from data points to decision hyperplane:
[ 40.96862  40.20967  28.22325  10.56378   7.85438  11.9091   17.06384
  12.45654  22.93463 -27.05667 -29.08389 -30.1709  -28.645   -25.48996
 -27.40678 -23.52702 -22.88581 -27.56845 -21.59324 -17.7054  -18.59672
 -47.41239 -48.30868 -25.46138 -27.19592  -8.50629 -42.25396 -17.47581
 -56.76069 -48.30868 -27.19592 -42.25396 -17.4749  -19.72747  -9.33611
  -7.95783  -9.87565 -20.19425 -18.37875 -17.35244 -18.85719 -28.68227
 -15.96404 -28.80913 -20.15674 -35.85912 -35.53865 -35.61339 -44.22517
 -39.3716  -37.20729 -33.44969 -28.00198 -19.59709 -24.65563 -15.41129
 -31.61057 -27.44385 -21.40438 -27.76141 -25.30552 -24.99847 -27.07516
 -26.42314 -19.29797 -28.37711 -24.20806 -27.95945 -26.26567]
Confusion Matrix:
[[60  0]
 [ 0  9]]
AUC: 1.0
```





MAX_Iter=7:

```
The Train Result:
[1.1.1.1.1.1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
The Test Result:
[1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
Coefficient for Logistic Regression:
[[-0.40969 -0.36853  0.3875   0.35839  0.3875   0.35839]]
Distance from data points to decision hyperplane:
[ 17.72166  20.00851   9.68433   0.90654   3.61733  11.68114   5.88119
   7.83839  19.20707 -12.07973  -9.76133  -9.38727  -8.82304  -9.69943
  -9.94785  -9.24792  -5.3596   -8.44672  -9.39866  -5.48158  -5.73278
```

```
 -20.23209 -29.46598 -16.32942 -15.52698  -7.83996 -25.57489  -7.83115
 -32.78816 -29.46598 -15.52698 -25.57489  -7.83115  -9.18845  -5.67147
  -0.33927  -8.14124  -9.68682 -13.97741 -13.18279 -12.2214  -16.48117
 -14.28658 -23.46804 -13.28281 -15.56843 -21.31989 -17.50248 -15.69074
 -19.5371  -14.78986 -22.0345  -13.03511 -13.09272 -13.52541  -9.98198
 -18.6214   -8.81838 -10.86356  -9.93844 -10.93201  -9.94868  -7.20862
  -8.46418  -8.10528 -11.97077 -11.12994 -10.73193  -9.67761]
Confusion Matrix:
[[60  0]
 [ 0  9]]
AUC: 1.0
```

Discussion:

The classes are imbalanced with specified train/test set division. With regard to the logistic regression model, the controll-case ratio of 5 is optimal with which we will get a low variance at a minimum cost. We can compensate for the bias calculated by the following equation:

$$\widetilde{\beta_0} = \beta_0 + \ln\left(\frac{\pi}{1-\pi}\right) - \ln\left(\frac{\pi^*}{1-\pi^*}\right), \qquad \pi = \frac{13}{88}, \qquad \pi^* = \frac{6}{69}$$

logModel=LogisticRegression(max_iter=1000, C=10000, solver='liblinear', fit_intercept=True, intercept_scaling=0.1446693982)

From the results, we can see that under insufficient iteration condition, the bias calibration successfully improved the train error by being all correct. However, test error still remains the same.

The effect of introducing case-sampling i.e. bias calibration is largely consumed by the excellent performance of vanilla logistic regression, and the goodness of the data. In max iteration=1000, they are already all correct in both train data and test data. But case sampling calibration will play its role when more unseen data are introduced, especially those which are near the P(x)=0.5 hyperplane.

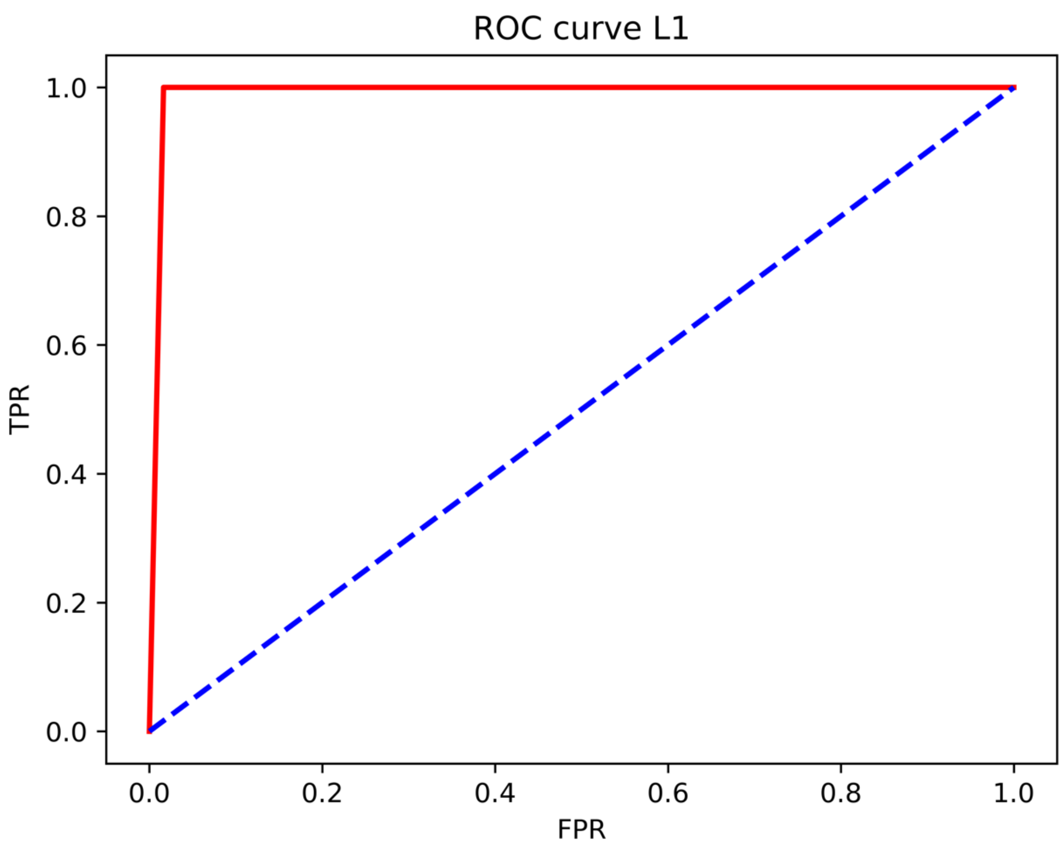
[L1 Penalized]

Result:

```
Best Correct Rate:  1.0
Fold which optimal circumstances is in:  1
Optimal alpha:[0.36787944]

The Train Result:
[1.1.1.1.0.1.1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
The Test Result:
[1.1.1.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.]
Probability estimated for Wrongly Predicted Data:
[5.74892e-01 4.25108e-01]
Coefficient for Logistic Regression:
[[ 0.       -0.31859  0.        0.        0.30558  0.       -0.19079  0.
```

```
  -0.00353  0.        0.20841  0.       -0.19072  0.       -0.04705  0.
   0.16967  0.       ]]
Distance from data points to decision hyperplane:
[  6.79015   6.31      4.98376   1.06496  -0.30184   1.71425   1.82265
   3.37309   3.11558  -5.02639  -5.26446  -5.09325  -5.32202  -4.78265
  -4.98789  -4.07882  -4.14352  -4.98183  -3.54629  -2.6096   -2.54786
  -9.96352  -9.38439  -6.30806  -6.03805  -3.55774  -9.41878  -1.88268
 -12.14933  -9.38439  -6.03805  -9.41878  -1.88255  -1.69058  -3.85058
  -0.15265  -4.01431  -4.47509  -4.91901  -3.05447  -5.2656   -5.05868
  -5.08423  -3.21186  -5.06799  -7.22725  -6.97389  -7.56352  -8.34824
  -8.7078   -8.00909  -7.15553  -5.61269  -5.97503  -6.18744  -3.80658
  -7.87881  -6.10134  -4.11842  -4.31602  -4.5298   -5.68955  -5.90039
  -4.80403  -4.61439  -5.1713   -5.0105   -5.62596  -5.30571]
Confusion Matrix:
[[60  1]
 [ 0  8]]
AUC: 0.9918032786885246
```




```
Since in this case, p-values are not easily computable, thus use RFE to work
on fold=1 and compare its feature selection results with L1 Penalty.
RFE when fold=1:(Specify the same number of features to choose)
RFE Feature Selection Results:
[False  False  False  False  True  True   True  False  True   False  True
 False  True   False  True  False  True   False]
RFE Feature Ranking:
[ 5  2  6  9  1  1  1 10  1  8  1  4  1 11  1  7  1  3]
```

Discussion:

The cross-entropy function:

$$-\ln\left(\mathcal{L}(\vec{\beta}|\mathcal{D})\right)+\lambda\|\vec{\beta}\|_1$$

The set of values of α on which cross validate:

$$\ln\left(\frac{1}{\lambda}\right)=\ln(\alpha)=\{-5,-4,-3,-2,-1,0,1,2,3,4,5\}$$

The result explicitly shows that Lasso Logistic Regression shuts down certain features contributing to the classification at the cost of complexity, which resulted in a worsen the train error. However, in the scope of test data provided in this situation, the test error does not decrease. Thus, L1's impact on correctness in classification in limited, but since the feature is reduced, the model gain advantage over time/computation cost as well as can serve as a feature selection method when datas are correlated & easily separable (that's when p-value's method becomes invalid). The larger λ is, the more features excluded by the algorithm.

```
lassoModel=LogisticRegressionCV(Cs=regStrength, penalty='l1',
solver='liblinear', cv=5, refit=True)
lassoModel.fit(self.trainSetIn, self.trueTrainLabel)
```

List and compare the feature selection results from L1 Penalty & RFE:
RFE: [F F F F T T T F T F T F T F T F T F]
L1: [F T F F T F T F T F T F T F T F T F]
The results are generally the same. The Only difference is LASSO choose feature 2 and RFE choose feature 6. On the basis of current outcome, Lasso is compatible to a wider range of situations where datas comes in good, heavily correlated and easily/linearly separable, compared with feature selection guided by z-statistics/p-values. RFE is easier to control than Lasso, since we can explicitly specify how many features we want and the algorithm also returns the ranking of the features, which plays a perfect role in feature analysis. In Lasso, number of features selected is determined by λ (regularization strength), and is hard to specify how many features we want by λ. Lasso exceeds RFE since the latter is a greedy algorithm which does not guarantees global optimality.

[L1 Penalized MultiClass]

Result:

Result for Direct Fit:
multiModel=LogisticRegression(max_iter=10000, penalty='l1', solver='saga', multi_class='multinomial')
multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)
Train Result:
[0.0.0.0.0.0.0.0.0.1.1.1.1.1.1.1.1.1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.3.3.3.3.3.
4.3.3.3.3.3.3.4.4.4.4.4.4.4.4.4.4.3.4.5.5.5.5.5.5.5.5.5.5.5.5.]
Test Result:
[0.0.0.1.1.1.1.2.2.2.3.4.3.2.4.4.5.5.5.]
5-Fold Cross Validation Scores on the set of α:

multiclass Scores

```
{0.0: array([[0.17647059, 0.17647059, 0.70588235, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.17647059, 0.29411765, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.16666667, 0.33333333, 0.83333333, 0.91666667, 0.83333333, 0.75      , 0.75      , 0.75      , 0.75      , 0.75      , 0.75      ],
             [0.16666667, 0.33333333, 0.58333333, 0.75      , 0.75      , 0.75      , 0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.83333333],
             [0.18181818, 0.36363636, 0.90909091, 0.81818182, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091]]),
 1.0: array([[0.17647059, 0.17647059, 0.70588235, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.17647059, 0.29411765, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.16666667, 0.33333333, 0.83333333, 0.91666667, 0.83333333, 0.75      , 0.75      , 0.75      , 0.75      , 0.75      , 0.75      ],
             [0.16666667, 0.33333333, 0.58333333, 0.75      , 0.75      , 0.75      , 0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.83333333],
             [0.18181818, 0.36363636, 0.90909091, 0.81818182, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091]]),
 2.0: array([[0.17647059, 0.17647059, 0.70588235, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.17647059, 0.29411765, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.16666667, 0.33333333, 0.83333333, 0.91666667, 0.83333333, 0.75      , 0.75      , 0.75      , 0.75      , 0.75      , 0.75      ],
             [0.16666667, 0.33333333, 0.58333333, 0.75      , 0.75      , 0.75      , 0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.83333333],
             [0.18181818, 0.36363636, 0.90909091, 0.81818182, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091]]),
 3.0: array([[0.17647059, 0.17647059, 0.70588235, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.17647059, 0.29411765, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.16666667, 0.33333333, 0.83333333, 0.91666667, 0.83333333, 0.75      , 0.75      , 0.75      , 0.75      , 0.75      , 0.75      ],
             [0.16666667, 0.33333333, 0.58333333, 0.75      , 0.75      , 0.75      , 0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.83333333],
             [0.18181818, 0.36363636, 0.90909091, 0.81818182, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091]]),
 4.0: array([[0.17647059, 0.17647059, 0.70588235, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.17647059, 0.29411765, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.16666667, 0.33333333, 0.83333333, 0.91666667, 0.83333333, 0.75      , 0.75      , 0.75      , 0.75      , 0.75      , 0.75      ],
             [0.16666667, 0.33333333, 0.58333333, 0.75      , 0.75      , 0.75      , 0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.83333333],
             [0.18181818, 0.36363636, 0.90909091, 0.81818182, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091]]),
 5.0: array([[0.17647059, 0.17647059, 0.70588235, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.17647059, 0.29411765, 0.76470588, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294, 0.88235294],
             [0.16666667, 0.33333333, 0.83333333, 0.91666667, 0.83333333, 0.75      , 0.75      , 0.75      , 0.75      , 0.75      , 0.75      ],
             [0.16666667, 0.33333333, 0.58333333, 0.75      , 0.75      , 0.75      , 0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.83333333],
             [0.18181818, 0.36363636, 0.90909091, 0.81818182, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091, 0.90909091]])}
```

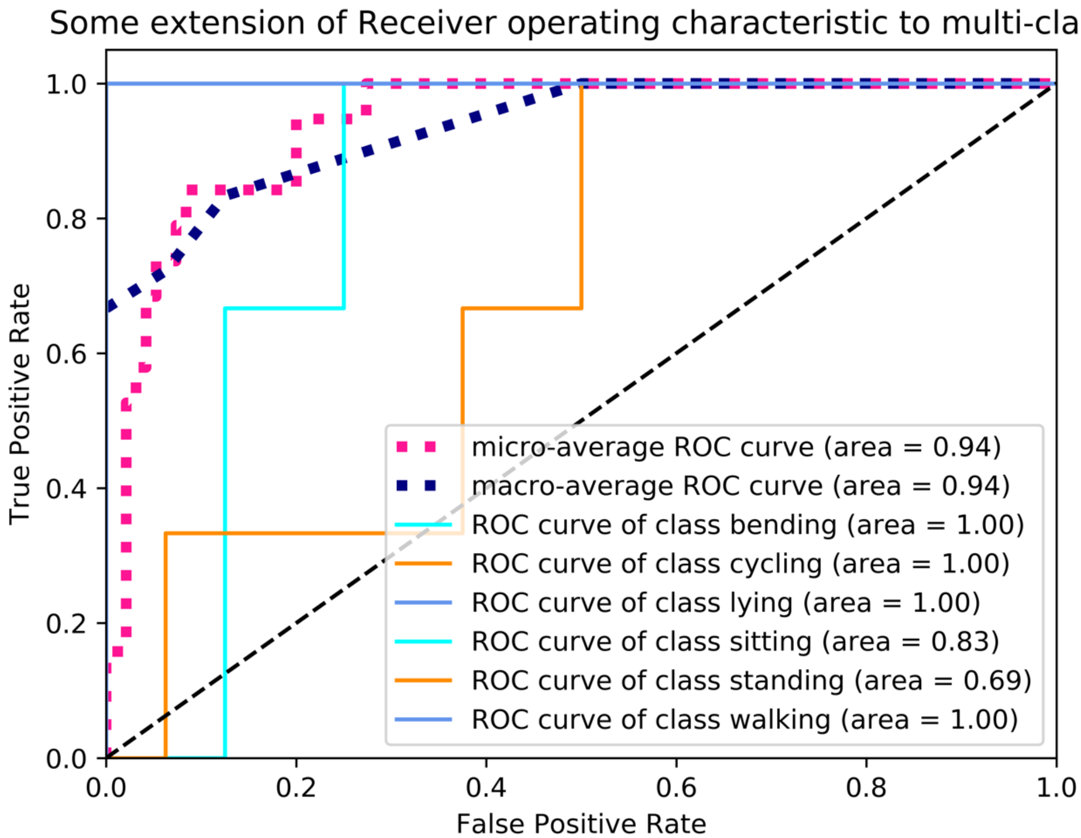
MultiNomial Logistic Regression:
Scores for $l = \{1, 2, 3, \dots\dots, 19, 20\}$ Fold:
0.85159, 0.83982, 0.83982, 0.83341, 0.86826, 0.82654, 0.82851,
0.78679, 0.78679, 0.79169, 0.79169, 0.77351, 0.82164, 0.77992,
0.76174, 0.77351, 0.74507, 0.76325, 0.77351, 0.77992,
Score of each class under optimal α:
[0.86809269 0.86809269 0.86809269 0.86809269 0.86809269 0.86809269]
Average score with prior knowledge:
0.8682663101604278
Fold which optimal circumstances is in: 5
Optimal alpha: 148.4131591025766
The Train Result:
[0.0.0.0.0.0.0.0.0.1.1.1.1.1.1.1.1.1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.3.3.3.3.3.
3.3.3.3.3.3.3.4.4.4.4.4.4.4.4.4.4.4.4.5.5.5.5.5.5.5.5.5.5.5.5.]
The Test Result:
[0. 0. 0. 3. 1. 1. 1. 2. 2. 2. 4. 3. 3. 4. 4. 4. 5. 5. 5.]
Probability estimated for Predict failure cases:
[6.995e-03 3.706e-01 3.997e-20 6.224e-01 3.157e-07 1.461e-09]
[1.095e-06 9.170e-03 3.107e-06 4.630e-01 5.278e-01 3.852e-10]
Confusion Matix for Train Data:

```
 [[ 9  0  0  0  0  0]
 [ 0 12  0  0  0  0]
 [ 0  0 12  0  0  0]
 [ 0  0  0 12  0  0]
 [ 0  0  0  0 12  0]
 [ 0  0  0  0  0 12]]
```

Confusion Matrix for Test Data:

```
[[ 3  0  0  0  0  0]
 [ 0  3  0  0  0  0]
 [ 0  0  3  0  0  0]
 [ 1  0  0  2  0  0]
```

```
 [ 0   0   0   1   3   0]
 [ 0   0   0   0   0   3]]
```



Discussion:

multiModel=LogisticRegressionCV(max_iter=10000, Cs=regStrength, penalty='l1', solver='saga', cv=5, refit=True, multi_class='multinomial')
multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)

The Tested α in cross validation is still from the range:

$$\ln\left(\frac{1}{\lambda}\right) = \ln(\alpha) = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$$

The sklearn.LogisticRegression chooses the best regularization strength and returns scores matrix with respect to each trial of K-Fold & different α, of which the form is shown above.

multiModel.set_params(Cs=opti_strength)
multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)

Implement the optimal alpha and fit again (since optimal value doesn't necessarily takes on value listed). The final criterion of choosing between different l is the weighted sum of the scores of classes with repect to the optimal α associated with l. The weight is the prior probability of each class:

$$\text{weight} = \{0.1477, 0.1705, 0.1705, 0.1705, 0.1705, 0.1705\}$$

The hyperparameters (α,l) chosen by cross validation is shown above. The new model has better performance over both test & train errors, but still cannot achieve the correctness level when it's a binary classification due to the curse of dimensionality.

[Naïve Bayes]

Result:

Result based on MultiNomial Prior:
multinoModel=MultinomialNB().fit(self.trainSetIn, self.trueMultiTrainLabel)

```
Predicted Train Label:
[0.0.0.0.0.0.0.0.0.0.1.1.1.1.1.1.1.1.1.1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.3.3.0.3.3.
 3.3.3.3.3.3.3.3.4.4.4.4.4.4.4.4.4.4.4.4.3.4.5.5.5.5.5.5.5.5.5.5.5.5.5.5.]
Predicted Test Label:
[0.0.0.0.1.1.1.2.2.2.4.3.3.4.4.4.5.5.5.]
Probability of Prediction Failures:
 [9.9853e-001 1.2503e-027 4.0048e-034 1.4715e-003 4.7769e-010 8.1538e-041]
 [7.4972e-009 3.5532e-042 3.6997e-011 9.9027e-001 9.7295e-003 3.9258e-057]
 [2.1562e-019 2.3769e-032 2.1485e-017 6.4809e-003 9.9352e-001 7.2283e-047]
Predict Accuracy on Train Set:
0.9710144927536232
Predict Accuracy on Test Set:
0.9473684210526315
```

Result based on Gaussian Prior:
guassianModel=GaussianNB().fit(self.trainSetIn, self.trueMultiTrainLabel)

```
Predicted Train Label:
[0.0.0.0.0.0.0.2.0.0.1.1.1.1.1.1.1.1.1.1.1.1.1.2.2.2.2.2.2.2.2.2.2.2.2.2.2.3.3.3.3.3.
 3.3.3.3.3.3.3.3.4.4.4.4.4.4.4.4.4.4.4.4.3.4.5.5.5.5.5.5.5.5.5.5.5.5.5.5.]
Predicted Test Label:
[0.0.0.0.1.1.1.2.2.2.4.3.3.3.4.4.5.5.5.]
Probability of Prediction Failures:
 [2.0083e-013 0.0000e+000 1.0000e+000 4.3484e-042 1.2795e-303 0.0000e+000]
 [1.9126e-016 0.0000e+000 4.5680e-029 9.9715e-001 2.8459e-003 0.0000e+000]
 [1.0256e-020 0.0000e+000 1.7168e-070 8.1645e-024 1.0000e+000 0.0000e+000]
 [6.2123e-014 0.0000e+000 3.9461e-033 1.0000e+000 8.6817e-008 0.0000e+000]
Predict Accuracy on Train Set:
0.9710144927536232
Predict Accuracy on Test Set:
0.8947368421052632
```

Discussion:

Naïve Bayes based on Gaussian prior is slightly better than Multinomial based in this case, which is both better than multiclass logistic regression. Part of the reason may be relative data insufficiency, because logistic regression relies more heavily on data numbers. A total amount of 88 instances can hardly be considered as sufficient in machine learning algorithms, but this major weakness is largely held back by well seperatedness of the data. As soon as the problem changed from binary to multiclass, the datas become even more sparse, and errors merge. Of course, Baysian approach has its own limits, mainly due to its assumptions, but seemed to be well suited for this problem.

Appendix:

```
# -*- coding: utf-8 -*-
"""
Created on Jun Sat 13:52:59 2019
Homework3 for EE559
Author:Chengyao Wang
USCID:6961599816
Contact Email:chengyao@usc.edu
"""
import os, csv
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.feature_selection import RFECV, RFE
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.datasets import make_classification
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, auc
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn import datasets
from scipy import interp
import scipy.stats as stat
import statsmodels.formula.api as sm
from itertools import cycle
#Important note: Dataset8 of Sitting missing value of t=13500
#Filled with arithmic average of t=13250 and t=13750
class hw3(object):
    #Some Information about data sets, and Configurations of the Projects
    raw_dataset=np.zeros((88,480,6), dtype=float)
    maxList=list(['min', 'max', 'mean', 'median', 'std', '25percentile', '75percentile'])
    testList=[1, 2, 8, 9, 14, 15, 16, 29, 30, 31, 44, 45, 46, 59, 60, 61, 74, 75, 76]
    dirList=list(['bending1', 'bending2', 'cycling', 'lying', 'sitting', 'standing', 'walking'])
    selectedList=list(['mean', 'std', 'max'])
    NumTrainClass=[9, 12, 12, 12, 12, 12]
    NumTestClass=[4, 3, 3, 3, 3, 3]
    #Some Optimal Values found
    bestL_1000=4
    bestL_7=19
    bestl1=1
    bestAlpha=0.36787944
    bestMultiAlpha=148.4131591025766
    bestfeatureList_1000=[1, 4, 16, 21, 22, 23, 26, 40, 50, 56, 64, 65, 67]
    bestfeatureList_7=[10, 13, 191, 192, 305, 306]
    #Data Set After data preprocessing
    trueTrainLabel=np.zeros(69)
    trueTestLabel=np.zeros(19)
    trueMultiTrainLabel=np.zeros(69)
    trueMultiTestLabel=np.zeros(19)
    def __init__(self, l):
        self.l=l
        self.readData()
        self.timeSeriesDivision()
        self.featureSelection()
        self.truelabelInit()
        self.dataResize()
        os.chdir("/Users/Gaara/Desktop/USC/EE559/Homework/Homework34/")
        print "Initialization Complete."
    def truelabelInit(self):
        for i in range(69):
            if i<9:
                self.trueTrainLabel[i]=1
        for i in range(19):
            if i<4:
                self.trueTestLabel[i]=1
        testCnt, trainCnt=0, 0
        for i in range(6):
            for pnt in range(self.NumTestClass[i]):
```

```
                self.trueMultiTestLabel[testCnt]=i
                testCnt+=1
            for pnt in range(self.NumTrainClass[i]):
                self.trueMultiTrainLabel[trainCnt]=i
                trainCnt+=1
    def readOneCsv(self, dir, fileName, numInstance):
        os.chdir("/Users/Gaara/Desktop/USC/EE559/Homework/Homework34/AReM/"+dir)
        read_latch, row_cnt=0, 0
        with open(fileName+".csv") as csv_file:
            csv_reader=csv.reader(csv_file, delimiter=',')
            for rowPointer in csv_reader:
                if (rowPointer[0]=='0') & (~read_latch):
                    read_latch=1
                if read_latch:
                    temp=np.array(rowPointer, dtype=float)
                    self.raw_dataset[numInstance, row_cnt, :]=temp[-6:]
                    row_cnt+=1
    def readData(self):
        fileName_base="dataset"
        instance_cnt=0
        for dir_iter in self.dirList:
            for file_iter in range(1,16):
                try:
                    self.readOneCsv(dir_iter, fileName_base+str(file_iter), instance_cnt)
                    #print dir_iter,"Dataset",file_iter,"Input Completed"
                    instance_cnt+=1
                except:
                    pass
        print "Data Reading Complete"
    def featureExtract(self, arrayIn):
        out=np.zeros((7), dtype=float)
        out[0]=min(arrayIn)
        out[1]=max(arrayIn)
        out[2]=np.mean(arrayIn)
        out[3]=np.median(arrayIn)
        out[4]=np.std(arrayIn)
        out[5]=np.percentile(arrayIn,25)
        out[6]=np.percentile(arrayIn,75)
        return out
    def featureSelection(self):
        self.divTrainSet_Sorted=np.zeros((6*self.l, 69, len(self.selectedList)), dtype=float)
        self.divTestSet_Sorted=np.zeros((6*self.l, 19, len(self.selectedList)), dtype=float)
        selectedFeature=[]
        iter=0
        for maxList_iter in self.maxList:
            if maxList_iter in self.selectedList:
                selectedFeature.append(iter)
            iter+=1
        for iter in range(len(selectedFeature)):
            for featureIndex in range(6*self.l):
                self.divTrainSet_Sorted[featureIndex, :,
iter]=self.divTrainSet[featureIndex,:,selectedFeature[iter]]
                self.divTestSet_Sorted[featureIndex, :,
iter]=self.divTestSet[featureIndex,:,selectedFeature[iter]]
        print "Feature Selection Completed, Selected Features:",self.selectedList
#Break Time Series into approximately equal length l parts, l takes values from 1 to 20
#Little Note: MAX(480 % i)==12, others < 7.
    def division_op(self, target_sequence, l=1):
        avg=len(target_sequence)/float(l)
        out=[]
        last=0.0
        while last<len(target_sequence):
            out.append(target_sequence[int(last):int(last+avg)])
            last+=avg
        return out
#Directly derive the statistical matrix
    def timeSeriesDivision(self):
        self.divTrainSet=np.zeros((6*self.l, 69, 7), dtype=float)
        self.divTestSet=np.zeros((6*self.l, 19, 7), dtype=float)
        testCnt, trainCnt= 0, 0
        for instancePnt in range(88):
            testFeaturePnt, trainFeaturePnt = 0, 0
            for tSeriesPnt in range(6):
```

```
                divOutput=self.division_op(self.raw_dataset[instancePnt, :, tSeriesPnt], self.l)
                if instancePnt+1 in self.testList:
                    for subSeries in range(self.l):
                        self.divTestSet[testFeaturePnt,
testCnt, :]=self.featureExtract(divOutput[subSeries])
                        testFeaturePnt+=1
                    flag=0
                else:
                    for subSeries in range(self.l):
                        self.divTrainSet[trainFeaturePnt,
trainCnt, :]=self.featureExtract(divOutput[subSeries])
                        trainFeaturePnt+=1
                    flag=1
            testCnt+=(1-flag)
            trainCnt+=flag
        print "Feature Extraction Complete, with l =", self.l
#Resize data for Logistic Regression & RFE
    def dataResize(self):
        self.testSetIn=np.zeros((19, 6 * self.l * len(self.selectedList)), dtype=float)
        self.trainSetIn=np.zeros((69, 6 * self.l * len(self.selectedList)), dtype=float)
        for testIter in range(19):
            for i in range(len(self.selectedList)-1):
                self.testSetIn[testIter][6*self.l*i:6*self.l*(i+1)]=self.divTestSet_Sorted[:,
testIter, i]
            self.testSetIn[testIter][-6*self.l:]=self.divTestSet_Sorted[:, testIter, i]
        for trainIter in range(69):
            for j in range(len(self.selectedList)-1):
                self.trainSetIn[trainIter][6*self.l*j:6*self.l*(j+1)]=self.divTrainSet_Sorted[:,
trainIter, j]
            self.trainSetIn[trainIter][-6*self.l:]=self.divTrainSet_Sorted[:, trainIter, j]
        print "Data ready for Logistic Regression."
#Logistic Regression
    def logisticRegression_perform(self):
        print "Starting Logistic Regression.\n"
        logModel=LogisticRegression(max_iter=10000, C=10000, solver='lbfgs').fit(self.trainSetIn,
self.trueTrainLabel)
        resultTrain, resultTest = logModel.predict(self.trainSetIn),
logModel.predict(self.testSetIn)
        probTrain, probTest = logModel.predict_proba(self.trainSetIn),
logModel.predict_proba(self.testSetIn)
        np.set_printoptions(precision=3)
        print 'Rank of this 69 * 18 matrix:',np.linalg.matrix_rank(self.trainSetIn)
        print 'The Train Result:\n', resultTrain
        print 'The Test Result:\n', resultTest
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Pribability estimated for TestSet:\n', probTest
        print 'Coefficient for Logistic Regression:\n', logModel.coef_
        denom = (2.0*(1.0+np.cosh(logModel.decision_function(self.trainSetIn))))
        F_ij=np.zeros((18, 18), dtype=float)
        for i in range(18):
            for j in range(18):
                for ggg in range(69):
                    F_ij[i][j]+=(self.trainSetIn[ggg, i]*self.trainSetIn[ggg, j])/denom[ggg]
        F_ij2=np.dot(np.dot(self.trainSetIn.T, np.diag(denom)), self.trainSetIn)
        denom = np.tile(denom,(self.trainSetIn.shape[1],1)).T
        F_ij3 = np.dot(np.divide(self.trainSetIn, denom).T, self.trainSetIn) ## Fisher
Information Matrix
        Cramer_Rao = np.linalg.inv(F_ij3) ## Inverse Information Matrix
        #print np.diag(Cramer_Rao)
        print 'Inverse of Fisher Information Matrix 3 is:\n',np.linalg.eigvals(Cramer_Rao)
        print 'Fisher Information Matrix 1:\n', np.linalg.eigvals(F_ij)
        print 'Fisher Information Matrix 2 is:\n', np.linalg.eigvals(F_ij2)
        print 'Fisher Information Matrix 3 is:\n', np.linalg.eigvals(F_ij3)
        sigma_estimates = np.sqrt(np.diagonal(Cramer_Rao))
        #print sigma_estimates
        z_scores = logModel.coef_[0]/sigma_estimates # z-score for eaach model coefficient
        p_values = [stat.norm.sf(abs(x))*2 for x in z_scores] ### two tailed test for p-values
#StatsModel Logistic Regression
    def smModel(self):
        best_Model=np.zeros((69, 13), dtype=float)
        for i in range(13):
            best_Model[:, i] = self.trainSetIn[:, self.bestfeatureList_1000[i]]
        model=sm.Logit(self.trueTrainLabel, best_Model)
```

```
        #result=model.fit()
        #print result.summary2()
#Recursive Feature Selection & Cross Validation
    def rfe_perform(self):
        rfecvModel=LogisticRegression(max_iter=7, C=10000, solver='lbfgs')
        rfecv = RFECV(estimator=rfecvModel, step=1, cv=StratifiedKFold(5), scoring='accuracy')
        rfecv.fit(self.trainSetIn, self.trueTrainLabel)
        plt.figure()
        plt.xlabel("Number of features selected")
        plt.ylim(0.5, 1.1)
        plt.ylabel("Cross validation score (nb of correct classifications)")
        plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
        #Returns the best score + the numbers of feature selected + Selection of Features
        return  max(rfecv.grid_scores_), rfecv.n_features_, rfecv.support_
#Scatter Plot Matrix
    def scattorPlot(self):
        plotFeatures=np.zeros((69, 9))
        if self.l == 1:
            plotFeatures[:, 0:3]=self.trainSetIn[:, 0:3]
            plotFeatures[:, 3:6]=self.trainSetIn[:, 3:6]
            plotFeatures[:, -3:]=self.trainSetIn[:, -3:]
        elif self.l == 2:
            plotFeatures[:, 0:3]=self.trainSetIn[:, 0:3]
            plotFeatures[:, 6:9]=self.trainSetIn[:, 6:9]
            plotFeatures[:, -3:]=self.trainSetIn[:, -3:]
        df = pd.DataFrame(plotFeatures)
        labelList=list()
        for i in range(9):
            labelList.append("bending")
        for i in range(60):
            labelList.append("not bending")
        df['label'] = labelList
        if self.l == 1:
            df.rename(columns={0:'max_1', 1:'max_2', 2:'max_6', 3:'mean_1', 4:'mean_2',
5:'mean_6', 6:'std_1', 7:'std_2', 8:'std_6'}, inplace=True)
        elif self.l == 2:
            df.rename(columns={0:'max_1(1)', 1:'max_2(1)', 2:'max_6(2)', 3:'mean_1(1)',
4:'mean_2(1)', 5:'mean_6(2)', 6:'std_1(1)', 7:'std_2(1)', 8:'std_6(2)'}, inplace=True)
        g=sns.PairGrid(df, hue='label')
        g=g.map_diag(plt.hist, histtype="step", linewidth=1)
        g=g.map_offdiag(plt.scatter, s=5)
        if self.l == 1:
            g.savefig("scatterPlot(l=1).png", dpi=800)
        elif self.l == 2:
            g.savefig("scatterPlot(l=2).png", dpi=800)
#Best Classifier
    def bestClassifier_1000(self):
        if self.l != 19:
            return 0
        opti_testSetIn=np.zeros((19, 13), dtype=float)
        opti_trainSetIn=np.zeros((69, 13), dtype=float)
        for iter in range(13):
            opti_testSetIn[:, iter]=self.testSetIn[:, self.bestfeatureList_1000[iter]]
            opti_trainSetIn[:, iter]=self.trainSetIn[:, self.bestfeatureList_1000[iter]]
        logModel=LogisticRegression(max_iter=1000, C=10000, solver='lbfgs').fit(opti_trainSetIn,
self.trueTrainLabel)
        resultTrain, resultTest = logModel.predict(opti_trainSetIn),
logModel.predict(opti_testSetIn)
        probTrain, probTest = logModel.predict_proba(opti_trainSetIn),
logModel.predict_proba(opti_testSetIn)
        np.set_printoptions(precision=5)
        print 'The Train Result:\n', resultTrain
        print 'The Test Result:\n', resultTest
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Probability estimated for TestSet:\n', probTest
        print 'Coefficient for Logistic Regression:\n', logModel.coef_
        print 'Distance from data points to decision hyperplane:\n',
logModel.decision_function(opti_trainSetIn)
        print confusion_matrix(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        fpr, tpr,_=roc_curve(logModel.predict(opti_trainSetIn),self.trueTrainLabel,
drop_intermediate=False)
        print roc_auc_score(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        plt.figure()
```

```
        plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve')
        plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('ROC curve maxiter=1000')
        plt.savefig('ROCAUC-max_iter=1000', dpi=800)
        plt.show()
        #Case-Control Sampling, introducing Bias Calibration: 0.1446693982
        #Update train Data to introduce Bias
        logModel=LogisticRegression(max_iter=1000, C=10000, solver='liblinear',
fit_intercept=True, intercept_scaling=0.1446693982).fit(opti_trainSetIn, self.trueTrainLabel)
        resultTrain, resultTest = logModel.predict(opti_trainSetIn),
logModel.predict(opti_testSetIn)
        probTrain, probTest = logModel.predict_proba(opti_trainSetIn),
logModel.predict_proba(opti_testSetIn)
        np.set_printoptions(precision=5)
        print 'The Train Result:\n', resultTrain
        print 'The Test Result:\n', resultTest
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Probability estimated for TestSet:\n', probTest
        print 'Coefficient for Logistic Regression:\n', logModel.coef_
        print 'Distance from data points to decision hyperplane:\n',
logModel.decision_function(opti_trainSetIn)
        #DRAW ROC & AOC & Confusion Matrix
        ##Computing false and true positive rates
        print confusion_matrix(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        fpr, tpr,_=roc_curve(logModel.predict(opti_trainSetIn),self.trueTrainLabel,
drop_intermediate=False)
        print roc_auc_score(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        plt.figure()
        plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve')
        plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('ROC curve maxiter Bias=1000')
        plt.savefig('ROCAUC-max_iter=1000 Bias', dpi=800)
        plt.show()
    def bestClassifier_7(self):
        if self.l != 4:
            return 0
        opti_testSetIn=np.zeros((19, 6), dtype=float)
        opti_trainSetIn=np.zeros((69, 6), dtype=float)
        for iter in range(6):
            opti_testSetIn[:, iter]=self.testSetIn[:, self.bestfeatureList_7[iter]]
            opti_trainSetIn[:, iter]=self.trainSetIn[:, self.bestfeatureList_7[iter]]
        logModel=LogisticRegression(max_iter=7, C=10000, solver='lbfgs').fit(opti_trainSetIn,
self.trueTrainLabel)
        resultTrain, resultTest = logModel.predict(opti_trainSetIn),
logModel.predict(opti_testSetIn)
        probTrain, probTest = logModel.predict_proba(opti_trainSetIn),
logModel.predict_proba(opti_testSetIn)
        np.set_printoptions(precision=5)
        print 'The Train Result:\n', resultTrain
        print 'The Test Result:\n', resultTest
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Probability estimated for TestSet:\n', probTest
        print 'Coefficient for Logistic Regression:\n', logModel.coef_
        print 'Distance from data points to dicision hyperplane:\n',
logModel.decision_function(opti_trainSetIn)
        #DRAW ROC & AOC & Confusion Matrix
        ##Computing false and true positive rates
        print confusion_matrix(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        fpr, tpr,_=roc_curve(logModel.predict(opti_trainSetIn),self.trueTrainLabel,
drop_intermediate=False)
        print roc_auc_score(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        plt.figure()
        plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve')
        plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('ROC curve maxiter=7')
        plt.savefig('ROCAUC-max_iter=7', dpi=800)
        plt.show()
```

```
        #Case-Control Sampling, introducing Bias Calibration: 0.1446693982
        #Update train Data to introduce Bias
        logModel=LogisticRegression(max_iter=7, C=10000, solver='liblinear', fit_intercept=True, 
intercept_scaling=0.1446693982).fit(opti_trainSetIn, self.trueTrainLabel)
        resultTrain, resultTest = logModel.predict(opti_trainSetIn), 
logModel.predict(opti_testSetIn)
        probTrain, probTest = logModel.predict_proba(opti_trainSetIn), 
logModel.predict_proba(opti_testSetIn)
        np.set_printoptions(precision=5)
        print 'The Train Result:\n', resultTrain
        print 'The Test Result:\n', resultTest
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Probability estimated for TestSet:\n', probTest
        print 'Coefficient for Logistic Regression:\n', logModel.coef_
        print 'Distance from data points to dicision hyperplane:\n', 
logModel.decision_function(opti_trainSetIn)
        #DRAW ROC & AOC & Confusion Matrix
        ##Computing false and true positive rates
        print confusion_matrix(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        fpr, tpr,_=roc_curve(logModel.predict(opti_trainSetIn),self.trueTrainLabel, 
drop_intermediate=False)
        print roc_auc_score(logModel.predict(opti_trainSetIn),self.trueTrainLabel)
        plt.figure()
        plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve')
        plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('ROC curve maxiter Bias=7')
        plt.savefig('ROCAUC-max_iter=7 Bias', dpi=800)
        plt.show()
#L1 Penalized Logistic Regression
    def l1_perform(self):
        regStrength=[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
        regStrength=np.exp(regStrength)
        lassoModel=LogisticRegressionCV(Cs=regStrength, penalty='l1', solver='liblinear', cv=5, 
refit=True)
        lassoModel.fit(self.trainSetIn, self.trueTrainLabel)
        #Returns the best score + Best C
        return  max(lassoModel.scores_), lassoModel.C_
#Best Classifier with L1 Penalty
    def bestClassifierL1(self):
        if self.l != self.bestl1:
            return 0
        logModel=LogisticRegression(max_iter=10000, C=self.bestAlpha, solver='liblinear', 
penalty='l1')
        logModel.fit(self.trainSetIn, self.trueTrainLabel)
        resultTrain, resultTest = logModel.predict(self.trainSetIn), 
logModel.predict(self.testSetIn)
        probTrain, probTest = logModel.predict_proba(self.trainSetIn), 
logModel.predict_proba(self.testSetIn)
        np.set_printoptions(precision=5)
        print 'The Train Result:\n', resultTrain
        print 'The Test Result:\n', resultTest
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Probability estimated for TestSet:\n', probTest
        print 'Coefficient for Logistic Regression:\n', logModel.coef_
        print 'Distance from data points to dicision hyperplane:\n', 
logModel.decision_function(self.trainSetIn)
        #DRAW ROC & AOC & Confusion Matrix
        ##Computing false and true positive rates
        print confusion_matrix(logModel.predict(self.trainSetIn),self.trueTrainLabel)
        fpr, tpr,_=roc_curve(logModel.predict(self.trainSetIn),self.trueTrainLabel, 
drop_intermediate=False)
        print roc_auc_score(logModel.predict(self.trainSetIn),self.trueTrainLabel)
        plt.figure()
        plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve')
        plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
        plt.xlabel('FPR')
        plt.ylabel('TPR')
        plt.title('ROC curve L1')
        plt.savefig('ROCAUC L1', dpi=800)
        plt.show()
        #RFE Comparison
```

```
        rfeModel=LogisticRegression()
        rfe=RFE(rfeModel, 8)
        rfe=rfe.fit(self.trainSetIn, self.trueTrainLabel)
        print 'RFE Feature Selection Results:\n', rfe.support_
        print 'RFE Feature Ranking:\n', rfe.ranking_
#MultiNomial Classification
    def multiToyTest(self):
        multiModel=LogisticRegression(max_iter=10000, penalty='l1', solver='saga',
multi_class='multinomial')
        multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)
        print 'Result for Direct Fit:'
        print 'Train Result:\n', multiModel.predict(self.trainSetIn)
        print 'True Train Label:\n', self.trueMultiTrainLabel
        print 'Test Result:\n', multiModel.predict(self.testSetIn)
        print 'True Test Label:\n', self.trueMultiTestLabel
    def multi_perform(self):
        regStrength=np.exp([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5])
        multiModel=LogisticRegressionCV(max_iter=10000, Cs=regStrength, penalty='l1',
solver='saga', cv=5, refit=True, multi_class='multinomial')
        multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)
        print 'Cross Validation Complete'
        #Refit the Model with Optimal C_
        opti_strength=[np.mean(multiModel.C_)]
        multiModel.set_params(Cs=opti_strength)
        multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)
        scores=np.zeros((6), dtype=float)
        for i in range(6):
            scores[i]=np.mean(multiModel.scores_[i])
        #Returns the best score array + Best C
        return  scores, multiModel.C_
    def bestClassifier_multi(self):
        if self.l != 5:
            return 0
        multiModel=LogisticRegressionCV(max_iter=10000, Cs=[self.bestMultiAlpha], penalty='l1',
solver='saga', cv=5, refit=True, multi_class='multinomial')
        multiModel.fit(self.trainSetIn, self.trueMultiTrainLabel)
        resultTrain, resultTest = multiModel.predict(self.trainSetIn),
multiModel.predict(self.testSetIn)
        probTrain, probTest = multiModel.predict_proba(self.trainSetIn),
multiModel.predict_proba(self.testSetIn)
        np.set_printoptions(precision=3)
        print 'The Train Result:\n', resultTrain
        print 'True Label of train Set:\n', self.trueMultiTrainLabel
        print 'The Test Result:\n', resultTest
        print 'True Label of test Set:\n', self.trueMultiTestLabel
        print 'Probability estimated for TrainSet:\n', probTrain
        print 'Probability estimated for TestSet:\n', probTest
        #DRAW ROC & AOC & Confusion Matrix
        ##Computing false and true positive rates
        print confusion_matrix(multiModel.predict(self.trainSetIn),self.trueMultiTrainLabel)
        print confusion_matrix(multiModel.predict(self.testSetIn), self.trueMultiTestLabel)
        self.drawRocAucCurve()
    def drawRocAucCurve(self):
        classList=list(['bending', 'cycling', 'lying', 'sitting', 'standing', 'walking'])
        X_train = self.trainSetIn
        X_test = self.testSetIn
        y_train = label_binarize(self.trueMultiTrainLabel, classes=[0, 1, 2, 3, 4, 5])
        y_test = label_binarize(self.trueMultiTestLabel, classes=[0, 1, 2, 3, 4, 5])
        multiModel = OneVsRestClassifier(LogisticRegression(penalty='l1', C=self.bestMultiAlpha,
max_iter=10000, solver='liblinear', multi_class='ovr'))
        y_score = multiModel.fit(X_train, y_train).decision_function(X_test)
        fpr, tpr, roc_auc= dict(), dict(), dict()
        for i in range(6):
            fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
            roc_auc[i] = auc(fpr[i], tpr[i])
        fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
        roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
        all_fpr = np.unique(np.concatenate([fpr[i] for i in range(6)]))
        mean_tpr = np.zeros_like(all_fpr)
        for i in range(6):
            mean_tpr += interp(all_fpr, fpr[i], tpr[i])
        mean_tpr /= 6
        fpr["macro"] = all_fpr
```

```
        tpr["macro"] = mean_tpr
        roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
        plt.figure()
        plt.plot(fpr["micro"], tpr["micro"], label='micro-average ROC curve (area =
{0:0.2f})'''.format(roc_auc["micro"]), color='deeppink', linestyle=':', linewidth=4)
        plt.plot(fpr["macro"], tpr["macro"], label='macro-average ROC curve (area =
{0:0.2f})'''.format(roc_auc["macro"]), color='navy', linestyle=':', linewidth=4)
        colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
        for i, color in zip(range(6), colors):
            plt.plot(fpr[i], tpr[i], color=color, label='ROC curve of class {0} (area =
{1:0.2f})'''.format(classList[i], roc_auc[i]))
        plt.plot([0, 1], [0, 1], 'k--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Some extension of Receiver operating characteristic to multi-class')
        plt.legend(loc="lower right")
        plt.savefig("MultiClass ROC-AUC Curve", dpi=800)
        plt.show()
#Naive Bayes Classification
    def multiNomialBayes(self):
        np.set_printoptions(precision=4)
        print 'Result based on MultiNomial Prior:'
        guassianModel=MultinomialNB().fit(self.trainSetIn, self.trueMultiTrainLabel)
        print 'True Train Label:\n', self.trueMultiTrainLabel
        print 'Predicted Train Label:\n', guassianModel.predict(self.trainSetIn)
        print 'True Test Label:\n', self.trueMultiTestLabel
        print 'Predicted Test Label:\n', guassianModel.predict(self.testSetIn)
        print 'Predicted Train Probability:\n', guassianModel.predict_proba(self.trainSetIn)
        print 'Predicted Test Probability:\n',  guassianModel.predict_proba(self.testSetIn)
        print 'Predict Accuracy on Train Set:\n', guassianModel.score(self.trainSetIn,
self.trueMultiTrainLabel)
        print 'Predict Accuracy on Test Set:\n', guassianModel.score(self.testSetIn,
self.trueMultiTestLabel)
    def gaussianBayes(self):
        np.set_printoptions(precision=4)
        print 'Result based on Gaussian Prior:'
        guassianModel=GaussianNB().fit(self.trainSetIn, self.trueMultiTrainLabel)
        print 'True Train Label:\n', self.trueMultiTrainLabel
        print 'Predicted Train Label:\n', guassianModel.predict(self.trainSetIn)
        print 'True Test Label:\n', self.trueMultiTestLabel
        print 'Predicted Test Label:\n', guassianModel.predict(self.testSetIn)
        print 'Predicted Train Probability:\n', guassianModel.predict_proba(self.trainSetIn)
        print 'Predicted Test Probability:\n',  guassianModel.predict_proba(self.testSetIn)
        print 'Predict Accuracy on Train Set:\n', guassianModel.score(self.trainSetIn,
self.trueMultiTrainLabel)
        print 'Predict Accuracy on Test Set:\n', guassianModel.score(self.testSetIn,
self.trueMultiTestLabel)
def timeDivision_RFECV():
    bestScore=0
    bestNumFeature=1000
    for fold in range(1, 21):
        rua=hw3(fold)
        best_temp, bestNumFeature_temp, featureSet = rua.rfe_perform()
        if (best_temp>bestScore)|((best_temp==bestScore)&(bestNumFeature_temp < bestNumFeature)):
            bestScore = best_temp
            bestNumFeature = bestNumFeature_temp
            bestFeatureSet = featureSet
            bestFold = fold
    print "Best Correct Rate: ", bestScore
    print "Number of Features: ", bestNumFeature
    print "Selected Feature Set: ", bestFeatureSet
    print "Fold which optimal circumstances is in: ", bestFold
def timeDivision_l1LassoCV():
    bestScore=0
    for fold in range(1, 21):
        rua=hw3(fold)
        best_temp, alpha = rua.l1_perform()
        if (best_temp>bestScore):
            bestScore = best_temp
            bestFold = fold
            bestAlpha = alpha
```

```
    print "Best Correct Rate: ", bestScore
    print "Fold which optimal circumstances is in: ", bestFold
    print "Optimal alpha: ", bestAlpha
    rua=hw3(bestFold)
def timeDivision_multiCV():
    bestScore=0
    prior=[0.1477, 0.1705, 0.1705, 0.1705, 0.1705, 0.1705]
    for fold in range(1, 21):
        rua=hw3(fold)
        score_array, alpha = rua.multi_perform()
        print np.dot(prior, score_array)
        if ( np.dot(prior, score_array) > bestScore):
            best_scorrArray = score_array
            bestScore = np.dot(prior, score_array)
            bestFold = fold
            bestAlpha = np.mean(alpha)
    print "Optimal Score of each class:\n", best_scorrArray
    print 'Average score with prior knowledge:\n', np.dot(prior, best_scorrArray)
    print "Fold which optimal circumstances is in: ", bestFold
    print "Optimal alpha: ", bestAlpha
#timeDivision_RFECV()
testModel=hw3(2)
#testModel.rfe_perform()
testModel.scattorPlot()
#testModel.logisticRegression_perform()
#testModel.smModel()
#testModel.multiToyTest()

#BestModel_1000=hw3(4)
#BestModel_1000.bestClassifier_1000()
#BestModel_7=hw3(19)
#BestModel_7.bestClassifier_7()
#BestModel_7.smModel()

#timeDivision_l1LassoCV()
#BestModel_l1=hw3(1)
#BestModel_l1.bestClassifierL1()

#timeDivision_multiCV()
#BestModel_multi=hw3(5)
#BestModel_multi.bestClassifier_multi()
#BestModel_multi.drawRocAucCurve()

#bayesModel=hw3(5)
#bayesModel.multiNomialBayes()
#bayesModel.gaussianBayes()
```