Task1

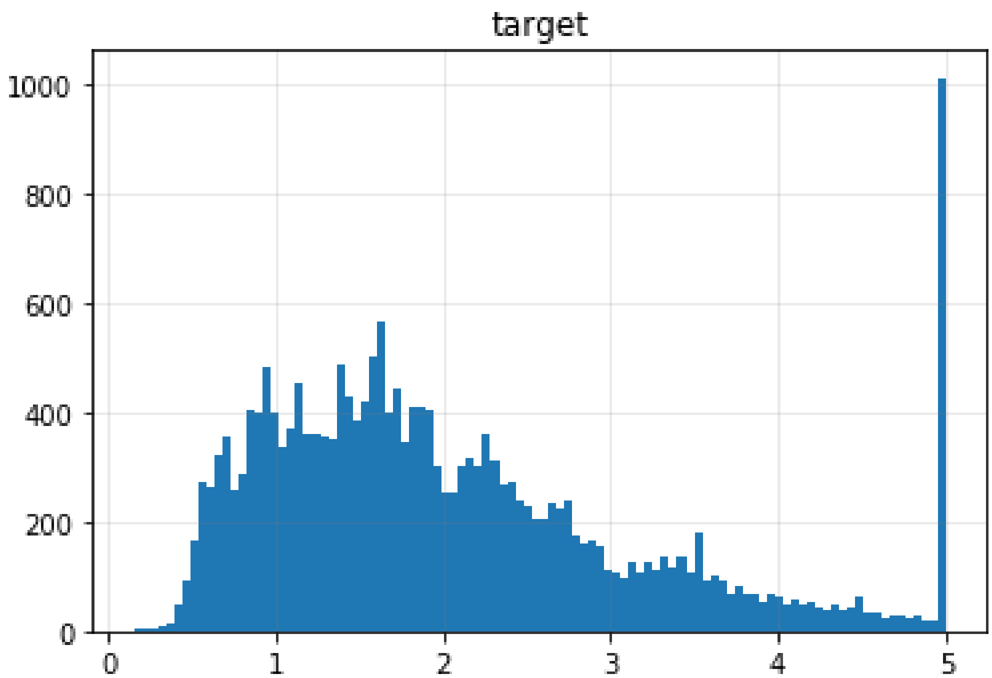
February 7, 2018

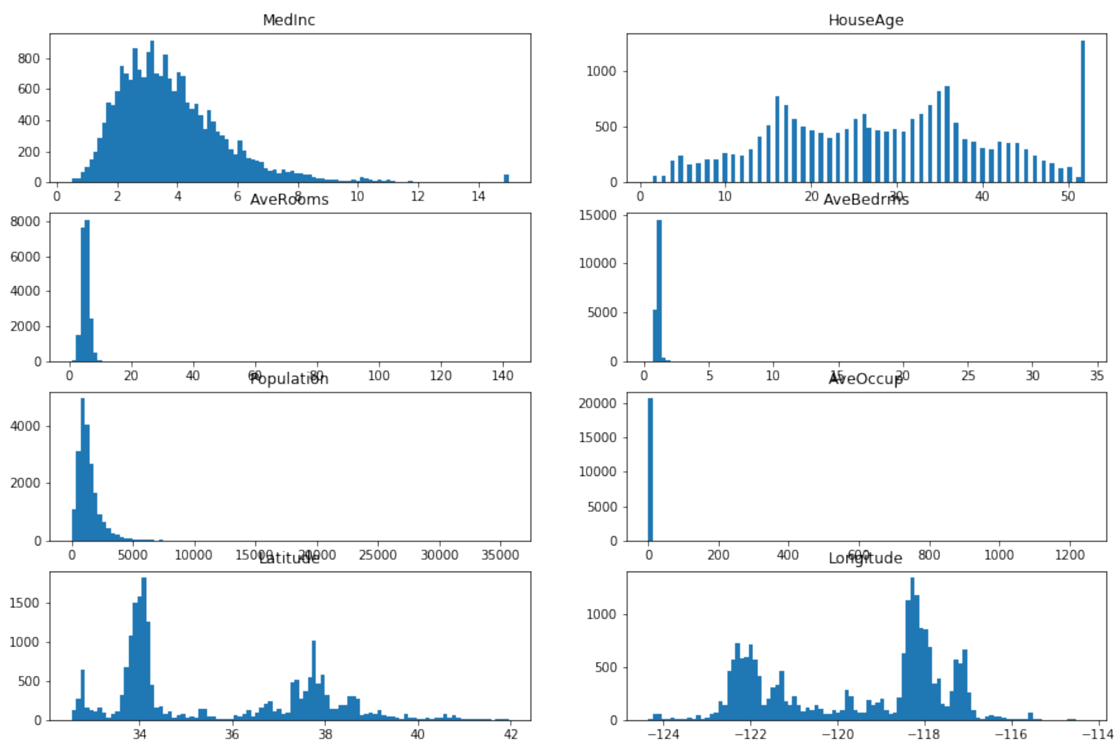
1 Task1

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import sklearn.datasets
        from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
        from sklearn.model_selection import train_test_split, cross_val_score
        from sklearn.preprocessing import StandardScaler
        from sklearn.model_selection import GridSearchCV
```

1.1

```
In [18]: california_dataset = sklearn.datasets.fetch_california_housing()

         plt.hist(california_dataset['target'],bins=100)
         plt.grid(color='gray', linestyle='-', linewidth=0.5, alpha=0.3)
         plt.title('target')
         plt.show()

         fig, axes = plt.subplots(4,2,figsize=(15,10))
         for i in range(0,4):
             for j in range(0,2):
                 axes[i,j].hist(california_dataset['data'][:,i*2+j],bins=100)
                 axes[i,j].set_title(california_dataset['feature_names'][i*2+j])
         plt.show()
```

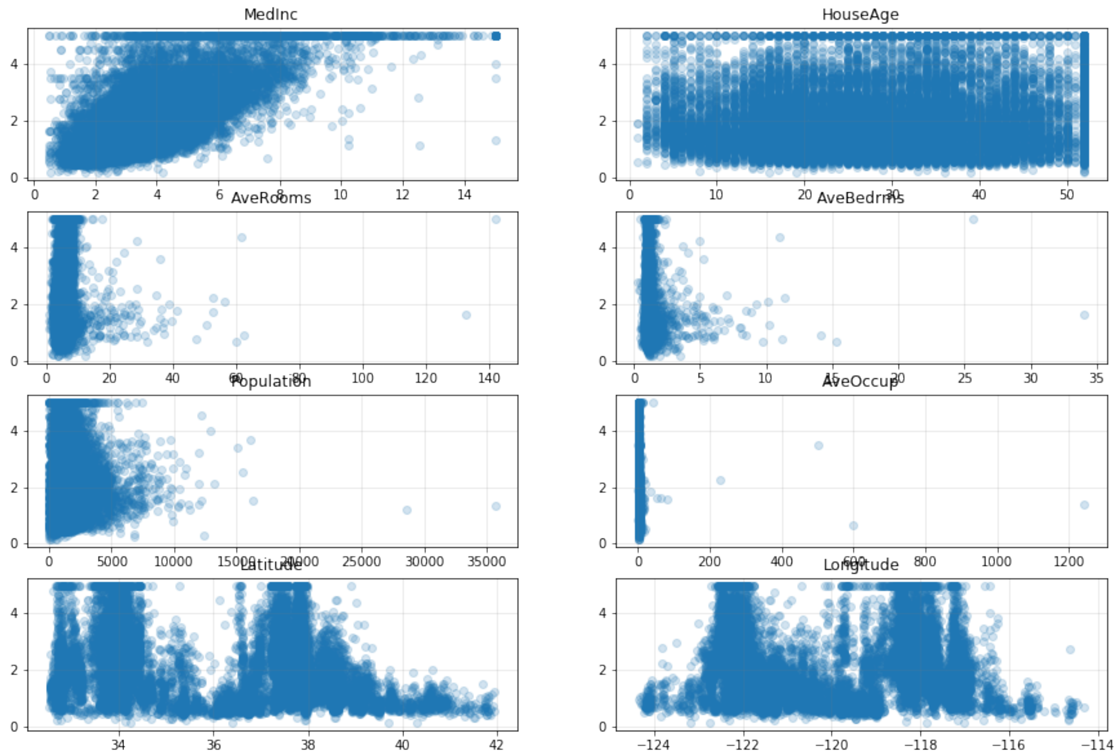

target
0
200
400
600
800
1000
0
1
2
3
4
5


MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude

From the above eight feature plots, 'AveRooms', 'AveBedrms','Population' and 'AveOccup' have inappropriate x scale, which means there are some outliers on these four features. We may need to remove these outliers, than re-scale the x axis.

1.2

```
In [4]: fig, axes = plt.subplots(4,2,figsize=(15,10))
        for i in range(0,4):
            for j in range(0,2):
                axes[i,j].scatter(california_dataset['data'][:,i*2+j],california_dataset['targe
                axes[i,j].set_title(california_dataset['feature_names'][i*2+j])
        plt.show()
```



1.3

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(california_dataset['data'],califo
         LRScore=np.mean(cross_val_score(LinearRegression(), X_train, y_train))
         RidgeScore=np.mean(cross_val_score(Ridge(), X_train, y_train))
         LassoScore=np.mean(cross_val_score(Lasso(), X_train, y_train))
         ElasticNetScore=np.mean(cross_val_score(ElasticNet(), X_train, y_train))
         print('LRScore:{}\nRidgeScore:{}\nLassoScore:{}\nElasticNetScore:{}'.format(LRScore,Ri
```

```
LRScore:0.6102422922538064
RidgeScore:0.6102437749800811
```

```
LassoScore:0.2819718569982819
ElasticNetScore:0.4229514321435757
```

Scaling the features

```
In [13]: scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)
         LRScore_S=np.mean(cross_val_score(LinearRegression(), X_train_scaled, y_train))
         RidgeScore_S=np.mean(cross_val_score(Ridge(), X_train_scaled, y_train))
         LassoScore_S=np.mean(cross_val_score(Lasso(), X_train_scaled, y_train))
         ElasticNetScore_S=np.mean(cross_val_score(ElasticNet(), X_train_scaled, y_train))
         print('LRScore:{}\nRidgeScore:{}\nLassoScore:{}\nElasticNetScore:{}'.format(LRScore_S
```

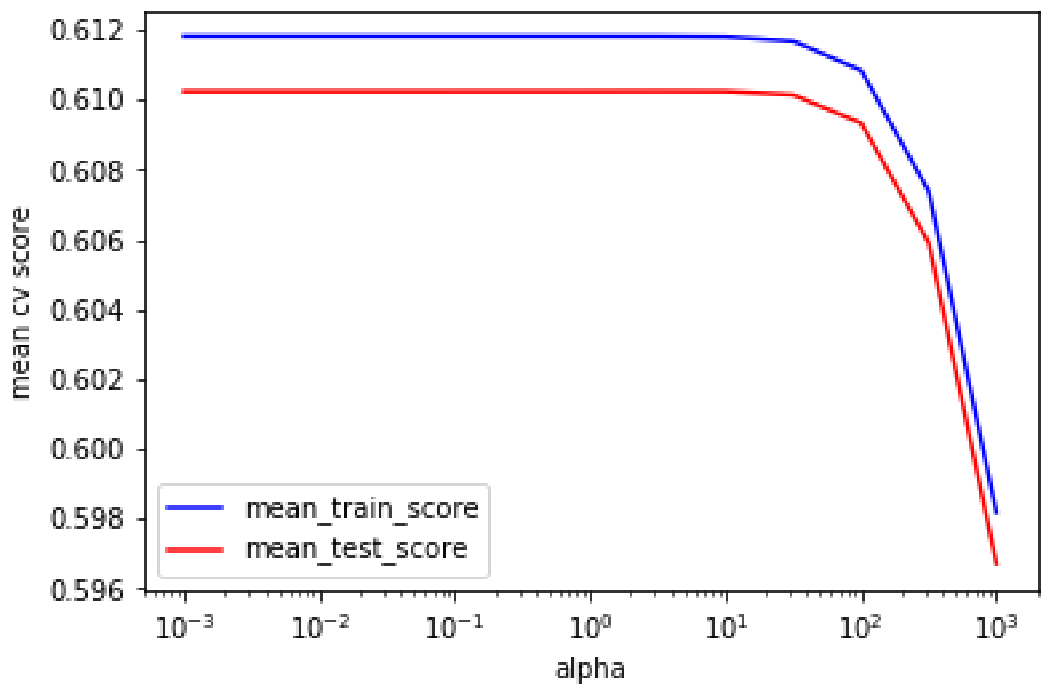
```
LRScore:0.6102422922538069
RidgeScore:0.6102441875598729
LassoScore:-0.00015307833209199373
ElasticNetScore:0.20448945641200397
```

According to the results, Scaling doesn't help the OLR and Ridge. Besides, it even has a negative influence on the Lasso and ElasticNet.

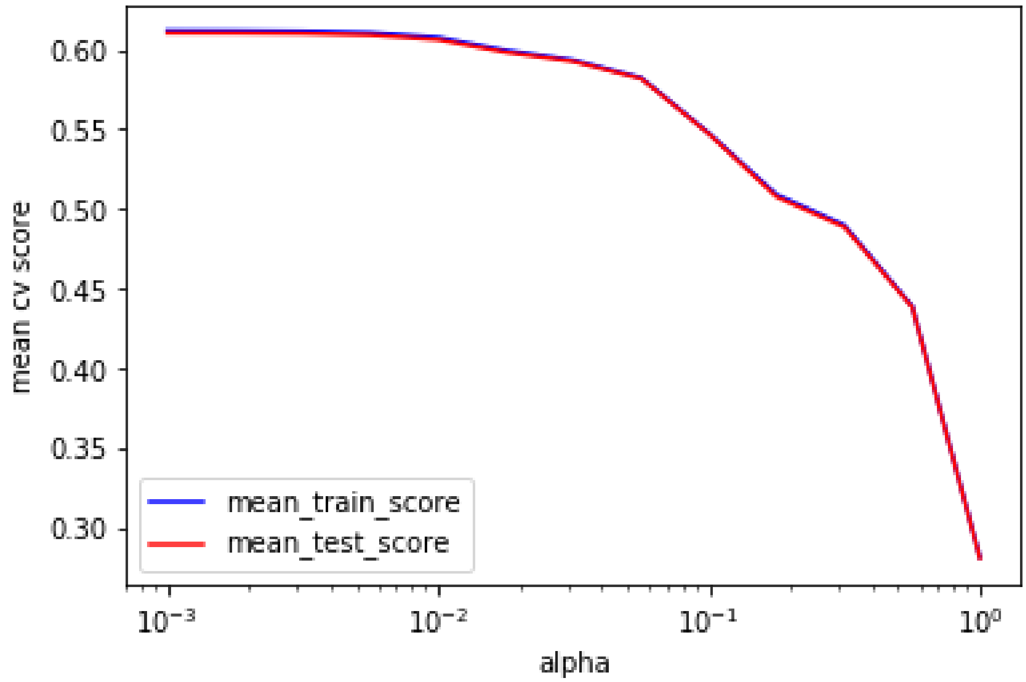
1.4

```
In [20]: # Ridge
         param_grid = {'alpha': np.logspace(-3, 3, 13)}
         grid = GridSearchCV(Ridge(), param_grid,return_train_score=True)
         grid.fit(X_train, y_train)
         plt.plot(param_grid['alpha'], grid.cv_results_['mean_train_score'],c='blue',label='mea
         plt.plot(param_grid['alpha'], grid.cv_results_['mean_test_score'],c='red',label='mean_
         plt.xlabel('alpha')
         plt.ylabel('mean cv score')
         plt.xscale('log')
         plt.legend()
         plt.show()
         ridge = grid.best_estimator_
         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))

         #Lasso
         param_grid = {'alpha': np.logspace(-3, 0, 13)}
         grid = GridSearchCV(Lasso(), param_grid,return_train_score=True)
         grid.fit(X_train, y_train)
         plt.plot(param_grid['alpha'], grid.cv_results_['mean_train_score'],c='blue',label='mea
         plt.plot(param_grid['alpha'], grid.cv_results_['mean_test_score'],c='red',label='mean_
         plt.xlabel('alpha')
         plt.ylabel('mean cv score')
```

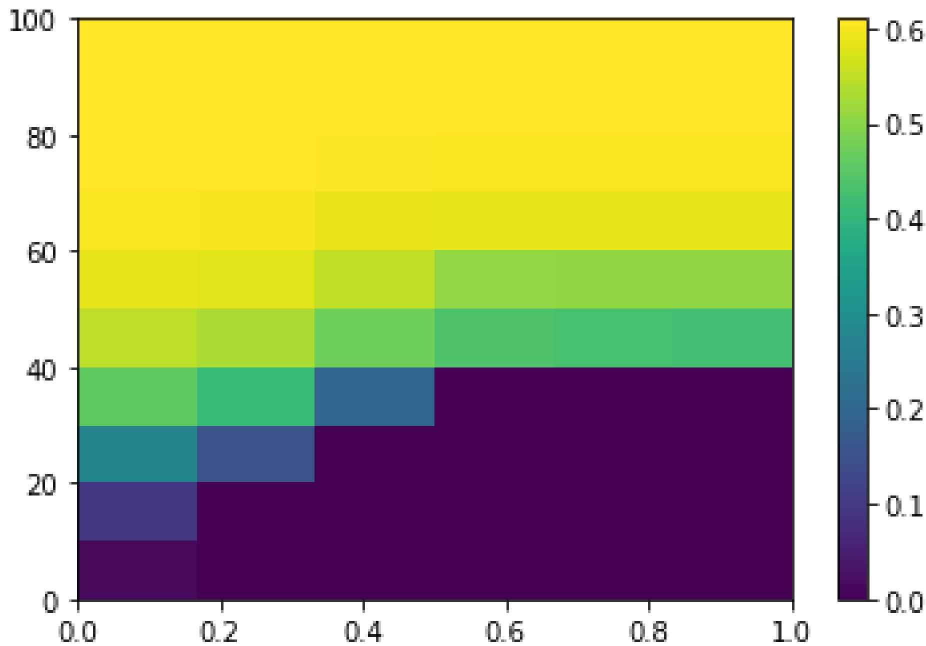
```
plt.xscale('log')
plt.legend()
plt.show()
lasso = grid.best_estimator_
print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))

# ElasticNet
param_grid = {'alpha': np.logspace(-3, 2, 10),'l1_ratio': [0.01, .1, .5, .9, .98, 1]}
grid = GridSearchCV(ElasticNet(), param_grid,return_train_score=True)
grid.fit(X_train, y_train)
res = pd.pivot_table(pd.DataFrame(grid.cv_results_),values='mean_test_score', index='
plt.imshow(res,extent=[0,1,0,100], aspect="auto")
plt.colorbar()
plt.show()
en = grid.best_estimator_
print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```




```
best score:0.6102457318944084
best parameters:{'alpha': 3.1622776601683795}
```




```
best score:0.6102299335181556
best parameters:{'alpha': 0.001}
```

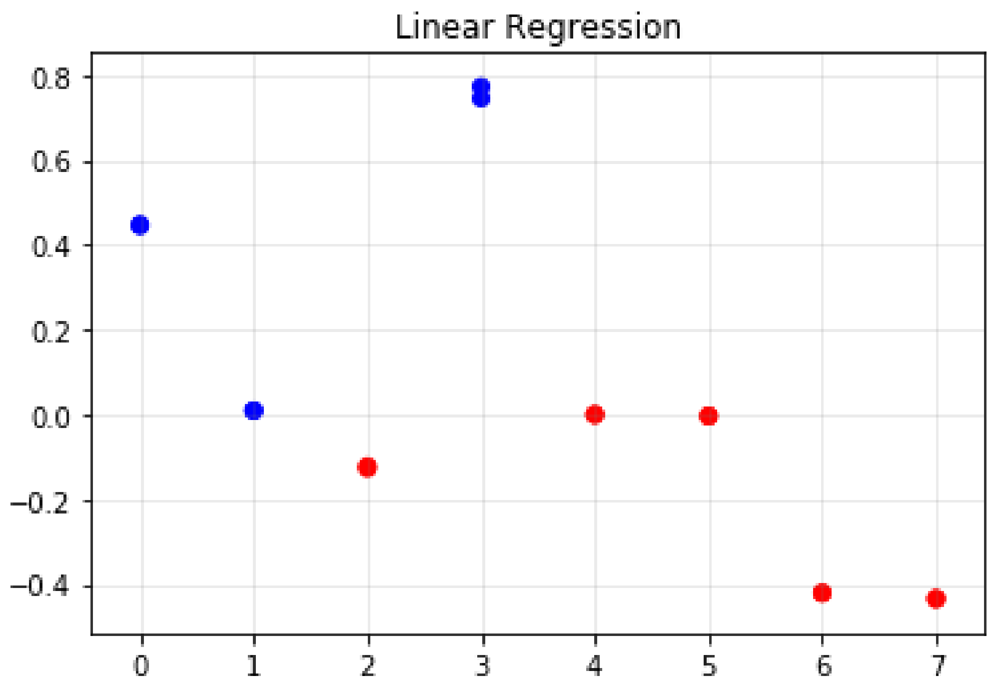


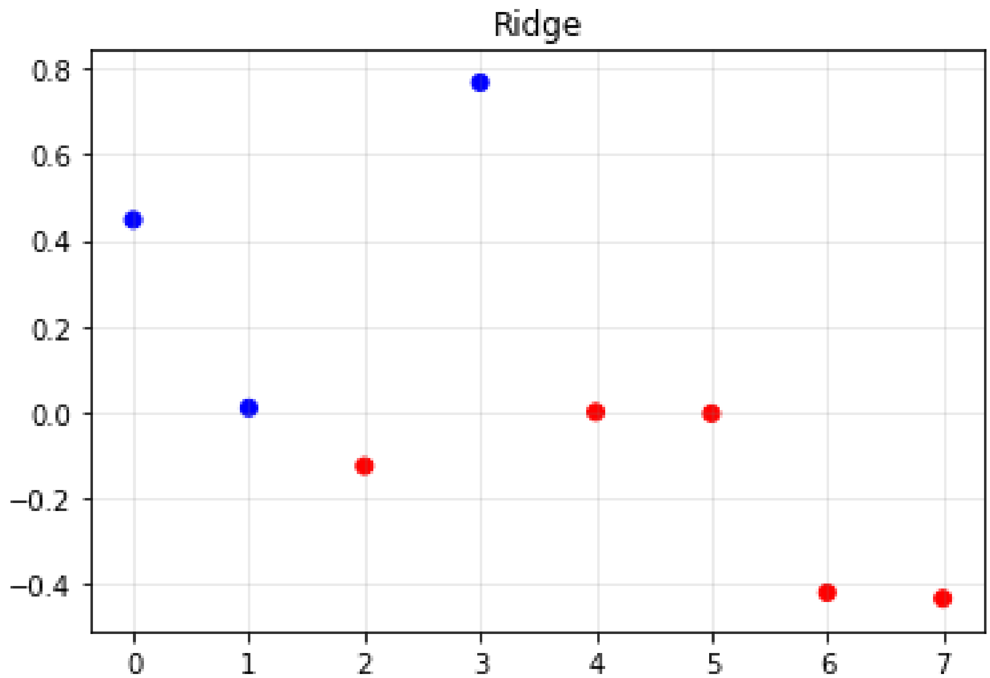
```
best score:0.61024079283954 85
best parameters:{'alpha': 0.001, 'l1_ratio': 0.01}
```

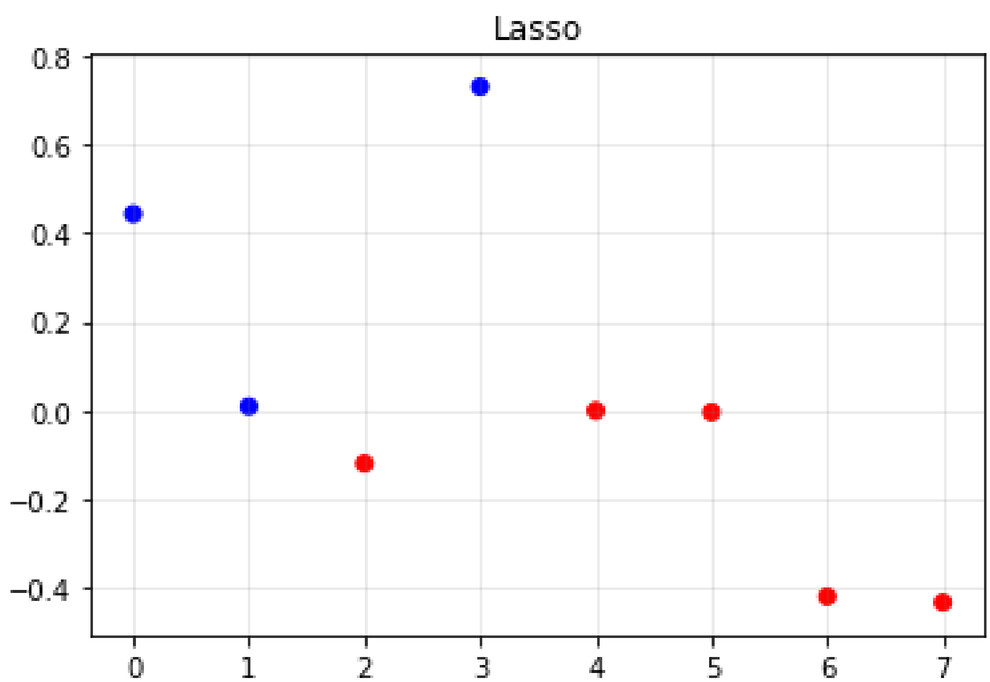
According to the results, GridSearch only can improve Lasso and ElasticNet.

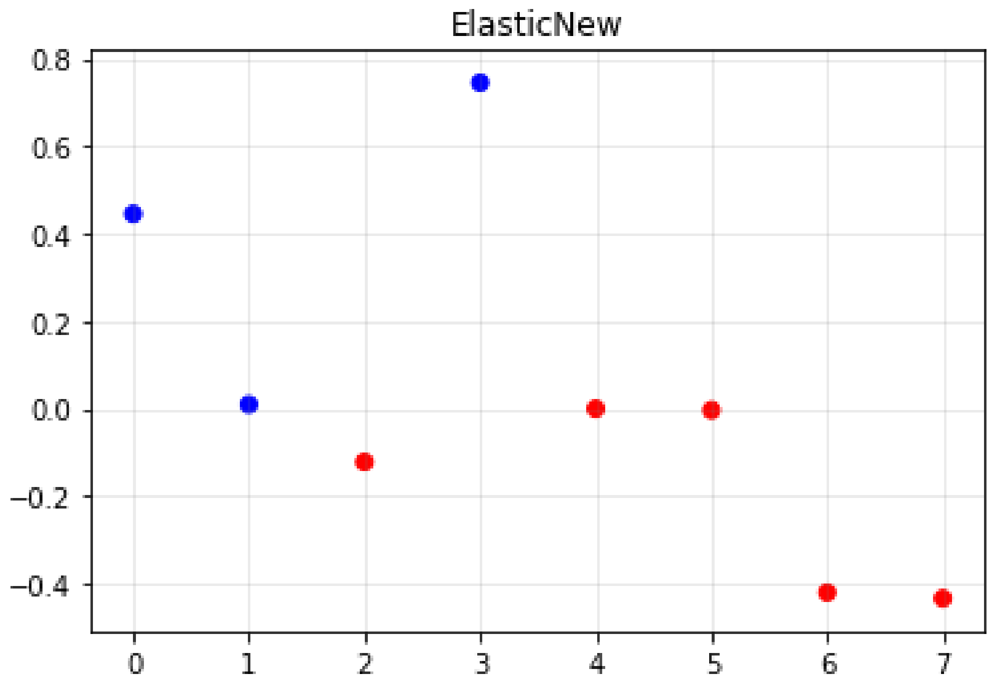
1.5

```
In [17]: lr = LinearRegression().fit(X_train, y_train)
         plt.scatter(range(X_train.shape[1]), lr.coef_,c=np.sign(lr.coef_), cmap="bwr_r")
         plt.title('Linear Regression')
         plt.show()

         plt.scatter(range(X_train.shape[1]), ridge.coef_,c=np.sign(ridge.coef_), cmap="bwr_r"
         plt.title('Ridge')
         plt.show()

         plt.scatter(range(X_train.shape[1]),lasso.coef_, c=np.sign(lasso.coef_), cmap="bwr_r"
         plt.title('Lasso')
         plt.show()

         plt.scatter(range(X_train.shape[1]),en.coef_, c=np.sign(en.coef_), cmap="bwr_r")
         plt.title('ElasticNew')
         plt.show()
```




Ridge
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6
7


Lasso
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6
7



Yes, it agrees on the features which are important.

Task2

February 7, 2018

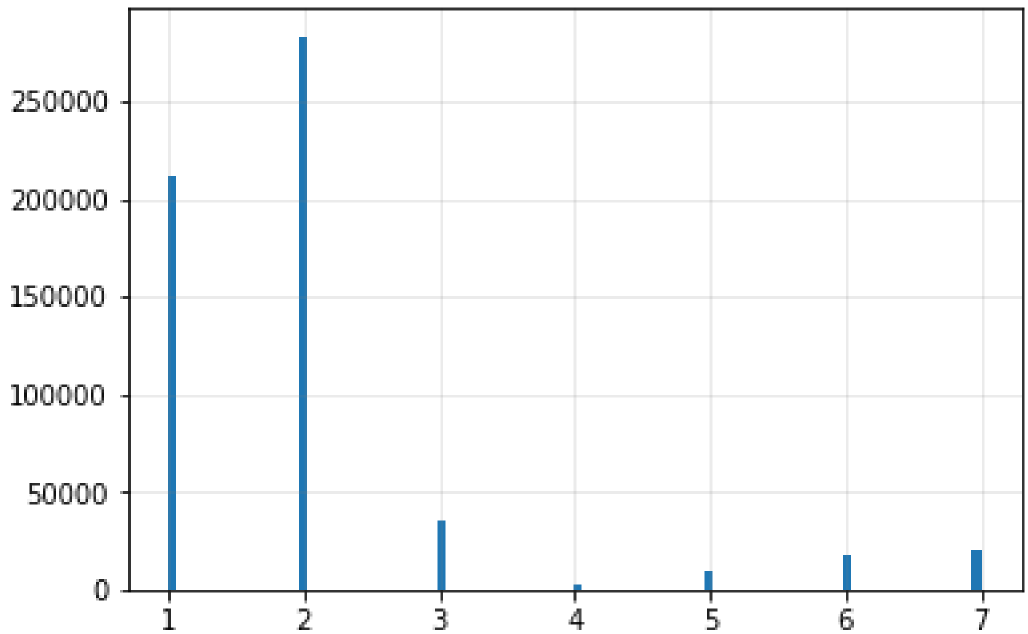
1 Task2

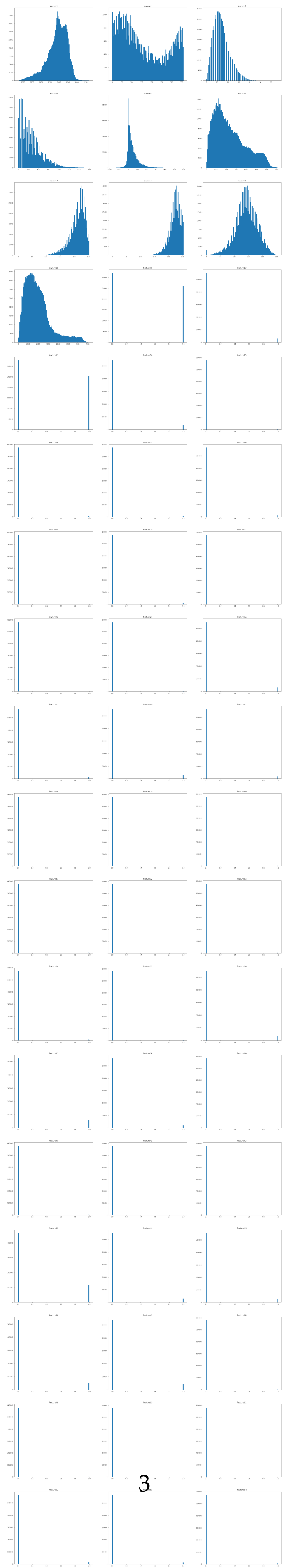
```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import pandas as pd
        import sklearn.datasets
        from sklearn.svm import LinearSVC
        from sklearn.neighbors import NearestCentroid
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import train_test_split, cross_val_score, KFold
        from sklearn.preprocessing import StandardScaler
        from sklearn.model_selection import GridSearchCV
```

```
In [2]: covtype_dataset = sklearn.datasets.fetch_covtype()
```

2.1

```
In [7]: plt.hist(covtype_dataset['target'],bins=100)
        plt.grid(color='gray', linestyle='-', linewidth=0.5, alpha=0.3)
        plt.show()

        fig, axes = plt.subplots(18,3,figsize=(30,180))
        for i in range(0,18):
            for j in range(0,3):
                axes[i,j].hist(covtype_dataset['data'][:,i*3+j],bins=100)
                axes[i,j].set_title('feature{}'.format(i*3+j+1))
        plt.show()
```


250000
200000
150000
100000
50000
0
1
2
3
4
5
6
7



2.2

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(covtype_dataset['data'],covtype_da
         LRScore=np.mean(cross_val_score(LogisticRegression(tol=0.1,dual=False,solver='sag'), X
         LinearSVCScore=np.mean(cross_val_score(LinearSVC(tol=0.1,dual=False), X_train, y_trai
         NearestCentroidScore=np.mean(cross_val_score(NearestCentroid(), X_train, y_train))

         print('LRScore:{}\nLinearSVCScore:{}\nNearestCentroidScore:{}\n'.format(LRScore,Linea
```

```
LRScore:0.6331321130115436
LinearSVCScore:0.528099269612307
NearestCentroidScore:0.19453872731809618
```

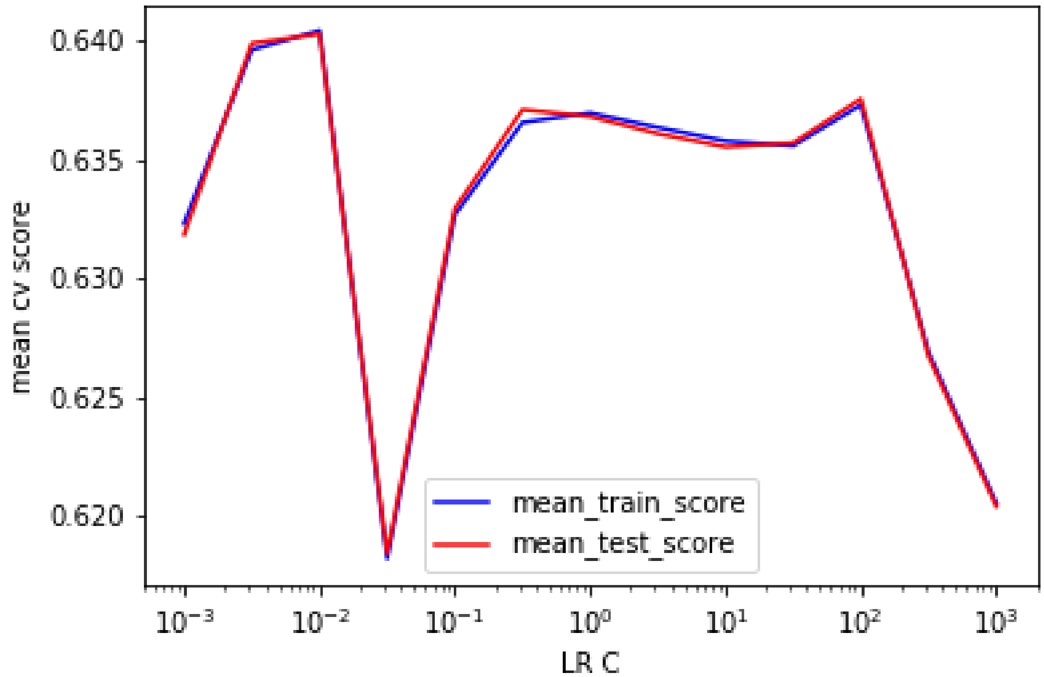
Scaling the Features:

```
In [11]: scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)
         LRScore_S=np.mean(cross_val_score(LogisticRegression(tol=0.1,dual=False,solver='sag')
         LinearSVCScore_S=np.mean(cross_val_score(LinearSVC(tol=0.1,dual=False), X_train_scaled
         NearestCentroidScore_S=np.mean(cross_val_score(NearestCentroid(), X_train_scaled, y_t

         print('LRScore:{}\nLinearSVCScore:{}\nNearestCentroidScore:{}\n'.format(LRScore_S,Line
```

```
LRScore:0.7147207572598478
LinearSVCScore:0.712627852387348
NearestCentroidScore:0.5508319986457934
```

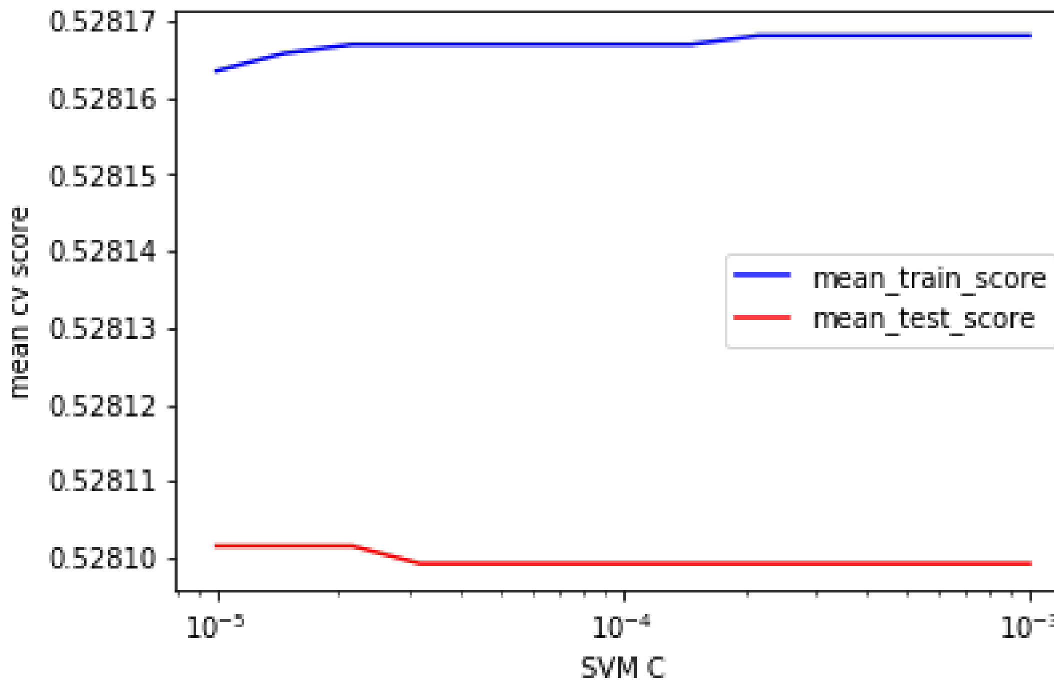
After scaling, the scores get highly promoted. Scaling works.
2.3

```
In [24]: # LR
         param_grid = {'C': np.logspace(-3, 3, 13)}
         grid = GridSearchCV(LogisticRegression(tol=0.1,dual=False,solver='sag'), param_grid,re
         grid.fit(X_train, y_train)
         lr = grid.best_estimator_
         plt.plot(param_grid['C'], grid.cv_results_['mean_train_score'],c='blue',label='mean_t
         plt.plot(param_grid['C'], grid.cv_results_['mean_test_score'],c='red',label='mean_tes
         plt.xlabel('LR C')
         plt.ylabel('mean cv score')
         plt.xscale('log')
         plt.legend()
         plt.show()
         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```



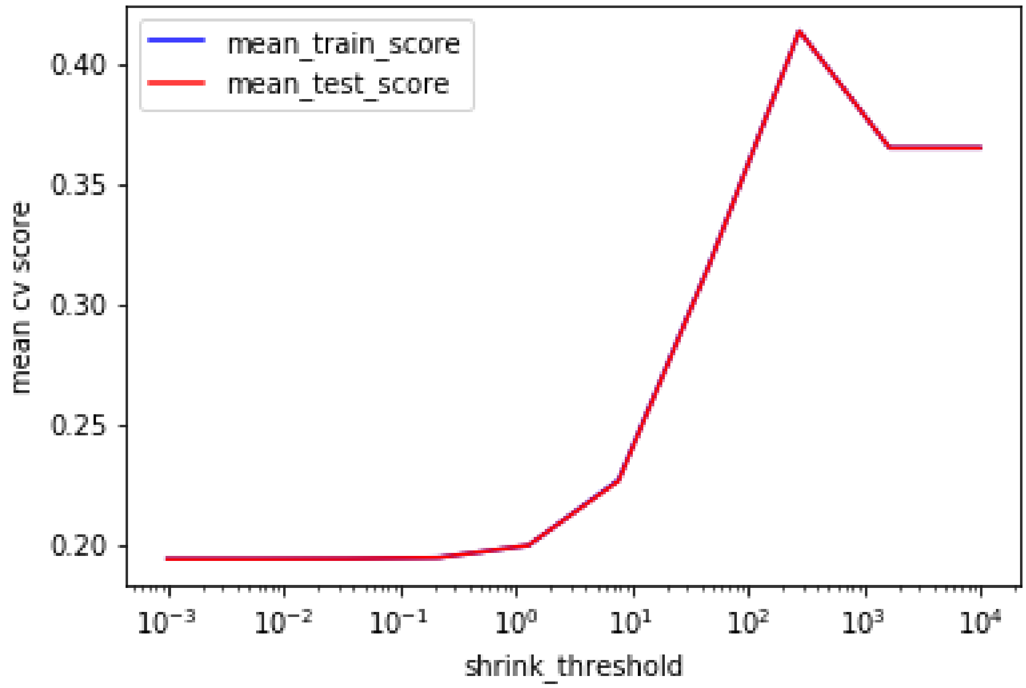

```
best score:0.6402919962639899
best parameters:{'C': 0.01}
```

In [25]:
```
#SVM
param_grid = {'C': np.logspace(-5, -3, 13)}
grid = GridSearchCV(LinearSVC(tol=0.1,dual=False), param_grid,return_train_score=True
grid.fit(X_train, y_train)
svm = grid.best_estimator_
plt.plot(param_grid['C'], grid.cv_results_['mean_train_score'],c='blue',label='mean_t
plt.plot(param_grid['C'], grid.cv_results_['mean_test_score'],c='red',label='mean_tes
plt.xlabel('SVM C')
plt.ylabel('mean cv score')
plt.xscale('log')
plt.legend()
plt.show()
print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```




```
best score:0.528101542366308
best parameters:{'C': 1e-05}
```

In [22]:
```
# NC
param_grid = {'shrink_threshold': np.logspace(-3, 4, 10)}
grid = GridSearchCV(NearestCentroid(), param_grid,return_train_score=True)
grid.fit(X_train, y_train)
svm = grid.best_estimator_
plt.plot(param_grid['shrink_threshold'], grid.cv_results_['mean_train_score'],c='blue
plt.plot(param_grid['shrink_threshold'], grid.cv_results_['mean_test_score'],c='red',l
plt.xlabel('shrink_threshold')
plt.ylabel('mean cv score')
plt.xscale('log')
plt.legend()
plt.show()
print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```




```
best score:0.4137172152497137
best parameters:{'shrink_threshold': 278.2559402207126}
```

According to the results, the GridSearch only works on the NearestCentroid model while it nearly has no influence on other two models.

2.4

Change the cross-validation strategy from 'stratified k-fold' to 'kfold' with shuffling

```
In [13]: kf=KFold(shuffle=True)
         param_grid = {'C': np.logspace(-3, 3, 13)}
         grid = GridSearchCV(LogisticRegression(tol=0.1,dual=False,solver='sag'), param_grid, c
         grid.fit(X_train, y_train)
         lr = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
         #SVM
         param_grid = {'C': np.logspace(-5, -3, 13)}
         grid = GridSearchCV(LinearSVC(tol=0.1,dual=False), param_grid, cv=kf)
         grid.fit(X_train, y_train)
         svm = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
         # NC
```

```
param_grid = {'shrink_threshold': np.logspace(-3, 4, 10)}
grid = GridSearchCV(NearestCentroid(), param_grid, cv=kf)
grid.fit(X_train, y_train)
svm = grid.best_estimator_

print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```

```
best score:0.640523775756783
best parameters:{'C': 0.0031622776601683794}
best score:0.5331043076562962
best parameters:{'C': 1e-05}
best score:0.4137860606436126
best parameters:{'shrink_threshold': 278.2559402207126}
```

After changing the cross-validation strategy from 'stratified k-fold' to 'kfold' with shuffling, the parameter of Logistic Regression and inear support vector machines have changed while that of nearest centroids doesn't change at all.

Change the random seed of the shuffling

```
In [14]: kf=KFold(random_state=8)
         param_grid = {'C': np.logspace(-3, 3, 13)}
         grid = GridSearchCV(LogisticRegression(tol=0.1,dual=False,solver='sag'), param_grid, c
         grid.fit(X_train, y_train)
         lr = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
         #SVM
         param_grid = {'C': np.logspace(-5, -3, 13)}
         grid = GridSearchCV(LinearSVC(tol=0.1,dual=False), param_grid, cv=kf)
         grid.fit(X_train, y_train)
         svm = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
         # NC
         param_grid = {'shrink_threshold': np.logspace(-3, 4, 10)}
         grid = GridSearchCV(NearestCentroid(), param_grid, cv=kf)
         grid.fit(X_train, y_train)
         svm = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```

```
best score:0.6431031831815293
best parameters:{'C': 3.1622776601683795}
best score:0.5316493749985657
best parameters:{'C': 0.001}
best score:0.41334086042973295
best parameters:{'shrink_threshold': 278.2559402207126}
```

After changing random seed of the shuffling, the parameter of Logistic Regression and inear support vector machines have changed while that of nearest centroids doesn't change at all.

Change the random state of the split into training and test data

```
In [15]: kf=KFold(n_splits=2)

         param_grid = {'C': np.logspace(-3, 3, 13)}
         grid = GridSearchCV(LogisticRegression(tol=0.1,dual=False,solver='sag'), param_grid, c
         grid.fit(X_train, y_train)
         lr = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
         #SVM
         param_grid = {'C': np.logspace(-5, -3, 13)}
         grid = GridSearchCV(LinearSVC(tol=0.1,dual=False), param_grid, cv=kf)
         grid.fit(X_train, y_train)
         svm = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
         # NC
         param_grid = {'shrink_threshold': np.logspace(-3, 4, 10)}
         grid = GridSearchCV(NearestCentroid(), param_grid, cv=kf)
         grid.fit(X_train, y_train)
         nc = grid.best_estimator_

         print('best score:{}\nbest parameters:{}'.format(grid.best_score_,grid.best_params_))
```

```
best score:0.6352112061942495
best parameters:{'C': 0.01}
best score:0.4979013629093145
best parameters:{'C': 1e-05}
best score:0.47876922794480437
best parameters:{'shrink_threshold': 278.2559402207126}
```

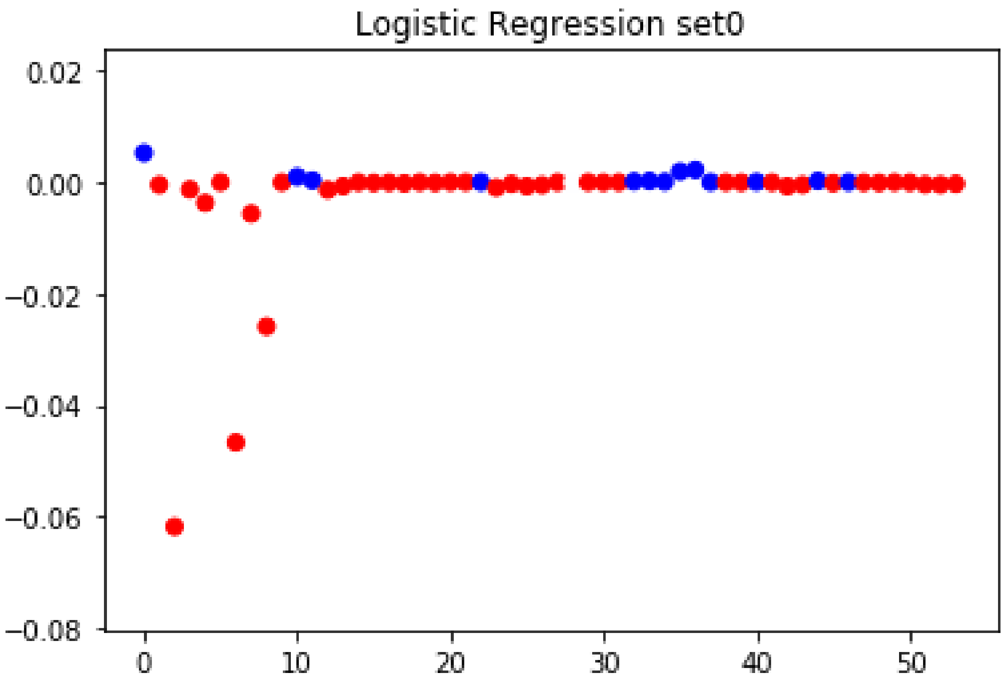
After changing random state of the split into training and test data, the parameter of Logistic Regression and inear support vector machines have changed while that of nearest centroids doesn't change at all.

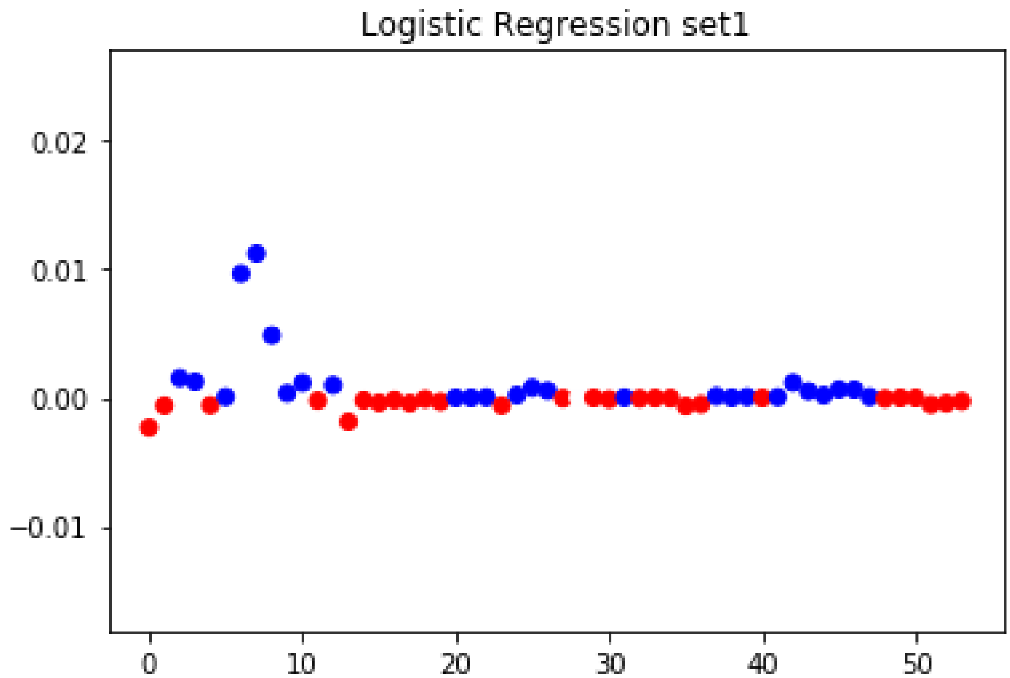
2.5

Choose the model with the best performance, the model after scaling the data with StandardScaler.

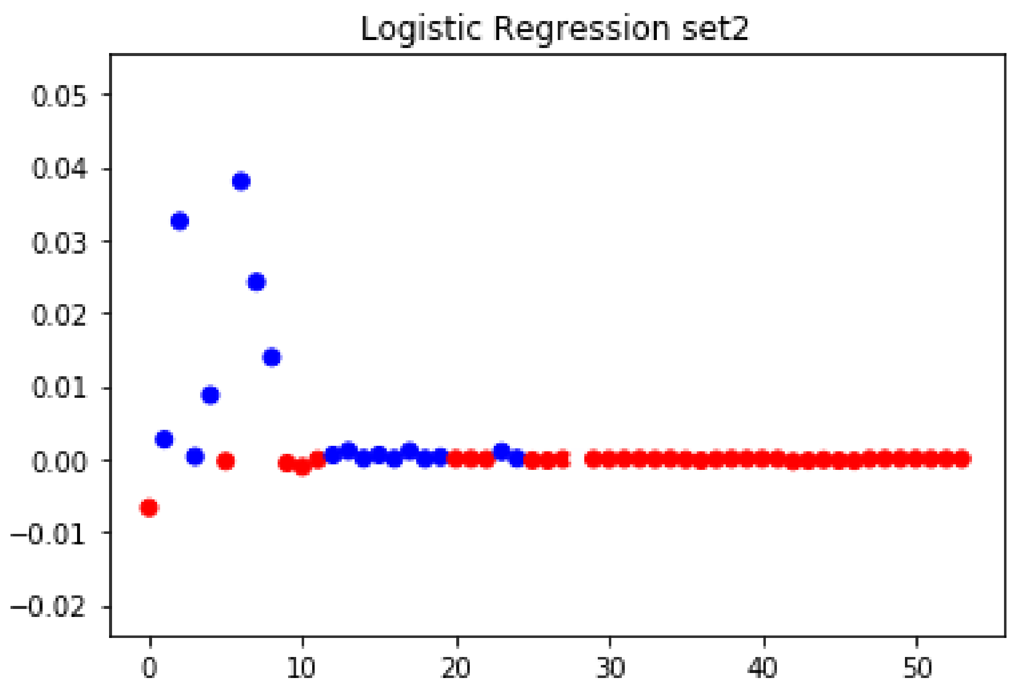
```
In [18]: scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)

         lr = LogisticRegression(tol=0.1,dual=False,solver='sag').fit(X_train, y_train)
         for i in range(0,7):
             plt.scatter(range(X_train.shape[1]), lr.coef_[i],c=np.sign(lr.coef_[i]), cmap="bw
```

```
    plt.title('Logistic Regression set{}'.format(i))
    plt.show()
svm = LinearSVC(tol=0.1,dual=False).fit(X_train, y_train)
for i in range(0,7):
    plt.scatter(range(X_train.shape[1]),svm.coef_[i], c=np.sign(svm.coef_[i]), cmap="
    plt.ylim(-0.005,0.005)
    plt.title('SVM set{}'.format(i))
    plt.show()
```

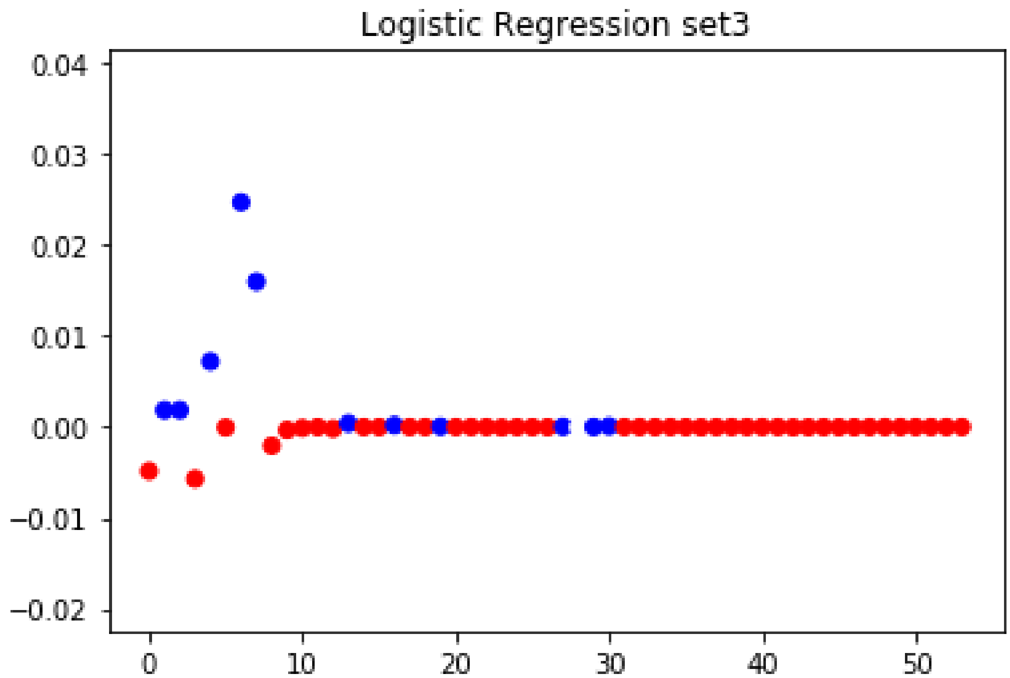


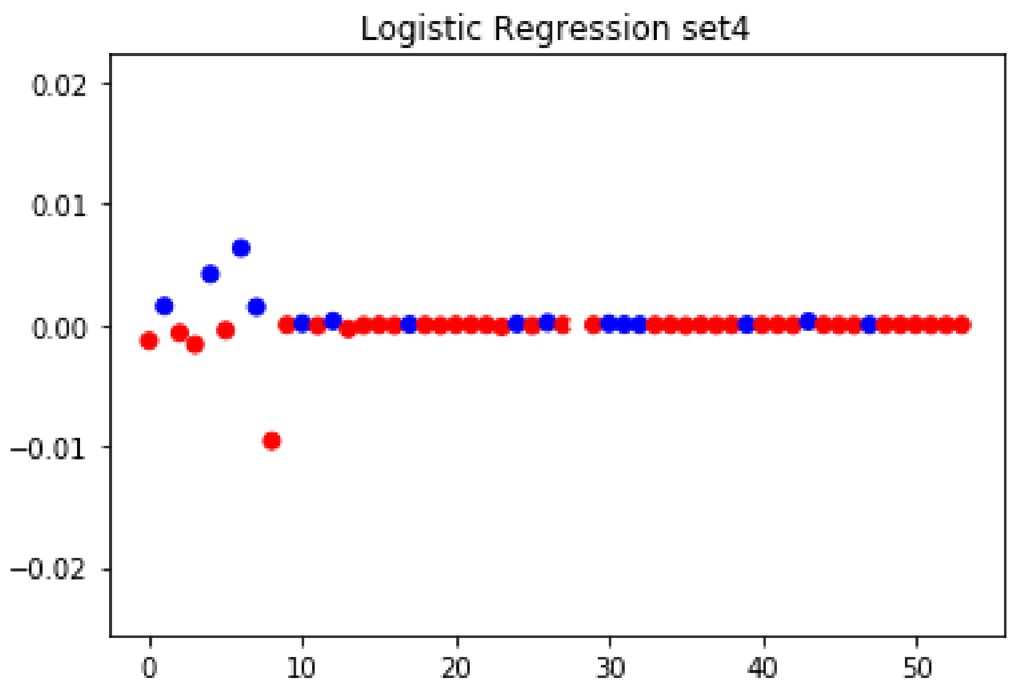


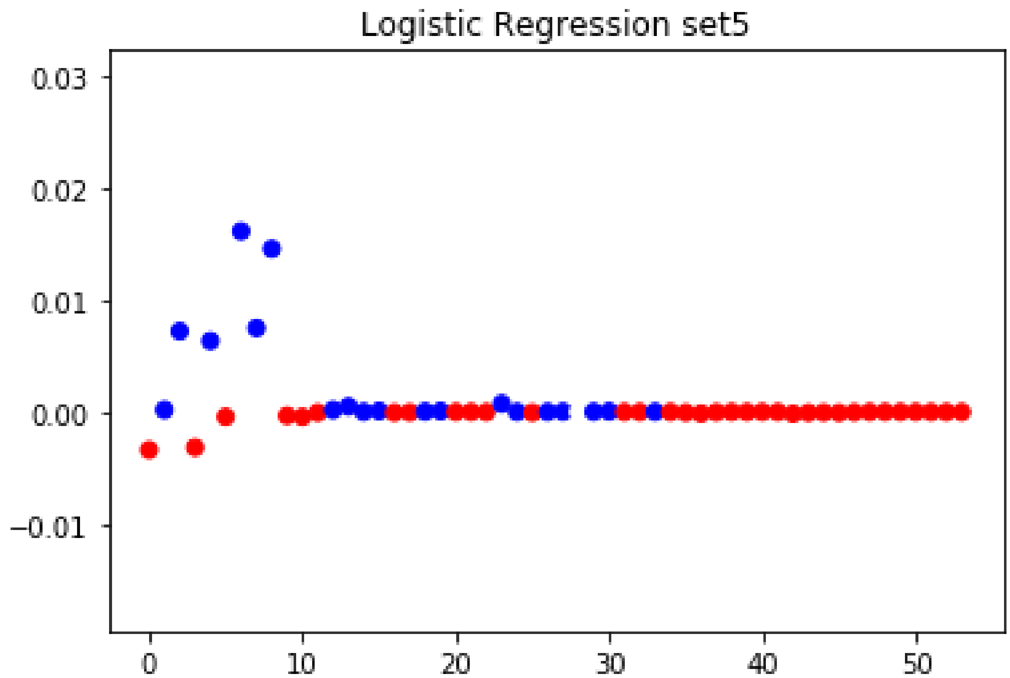
Logistic Regression set1
0.02
0.01
0.00
−0.01
0
10
20
30
40
50

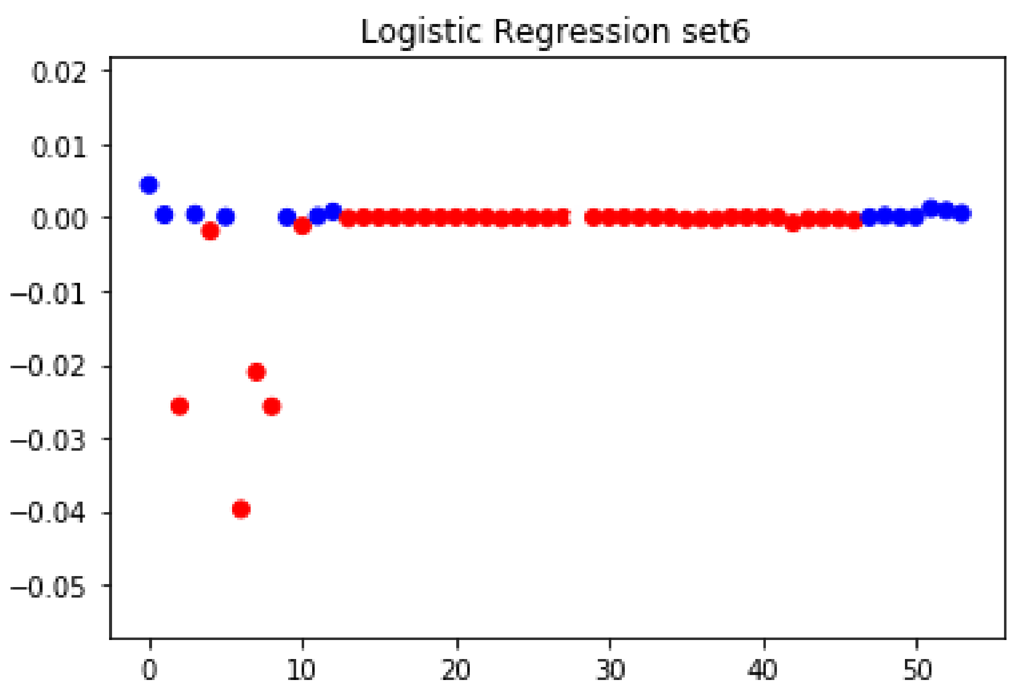


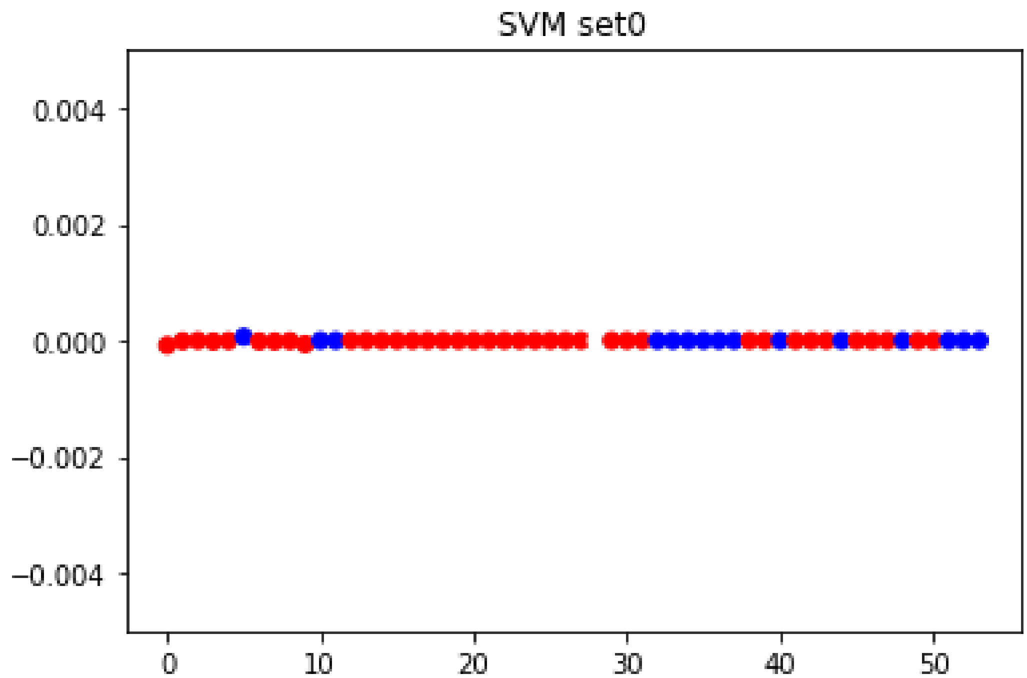
Logistic Regression set2
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
0
10
20
30
40
50

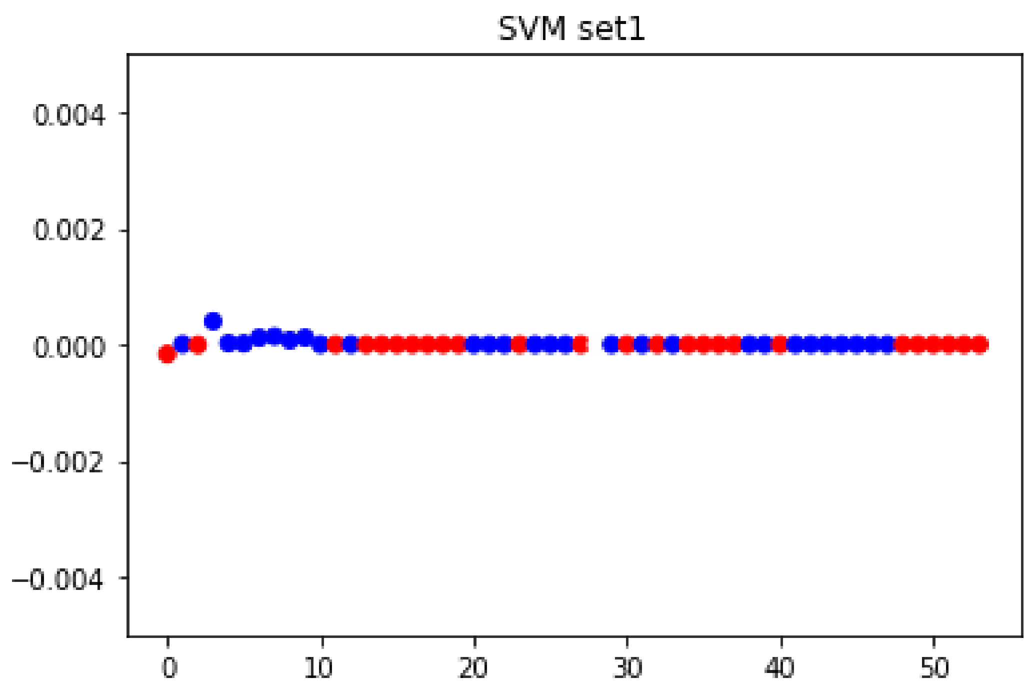

Logistic Regression set3
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
0
10
20
30
40
50

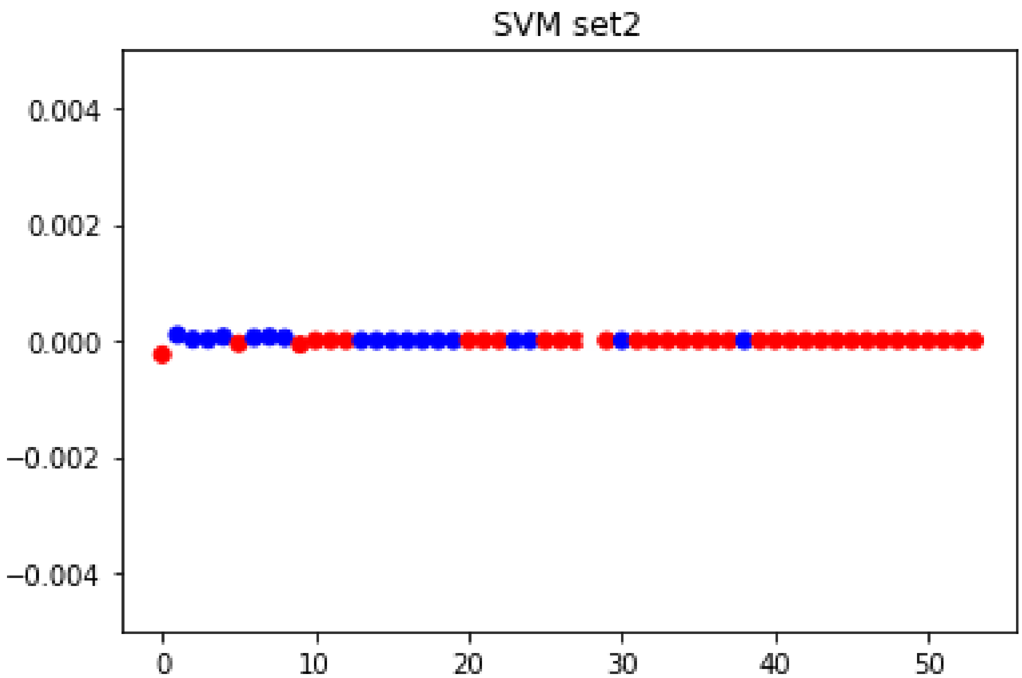

Logistic Regression set4
0.02
0.01
0.00
−0.01
−0.02
0
10
20
30
40
50

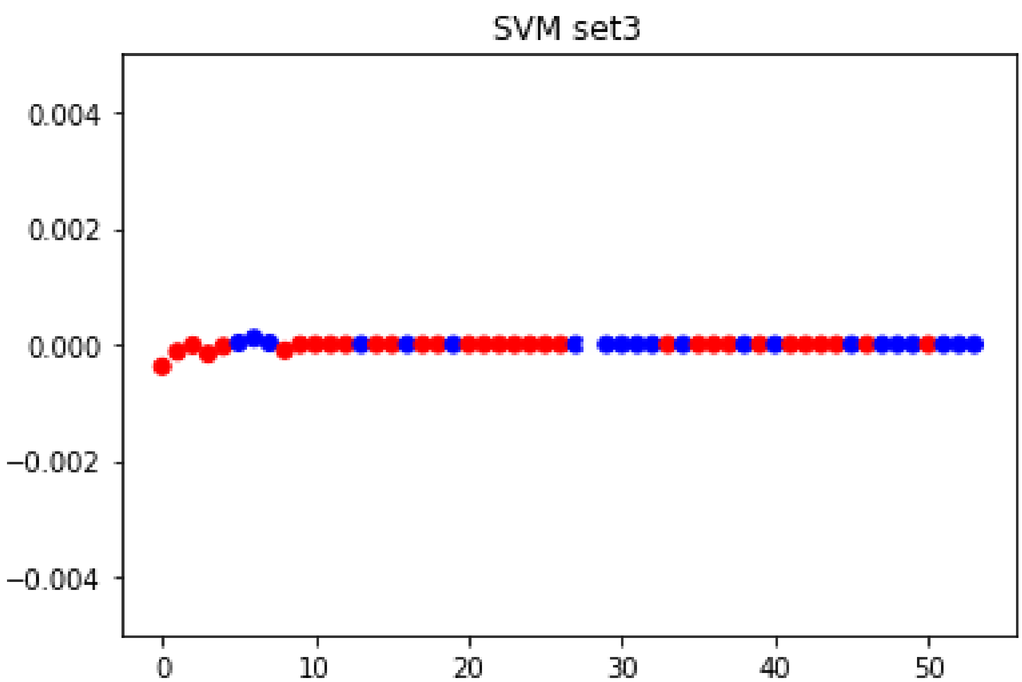

Logistic Regression set5
0.03
0.02
0.01
0.00
−0.01
0
10
20
30
40
50

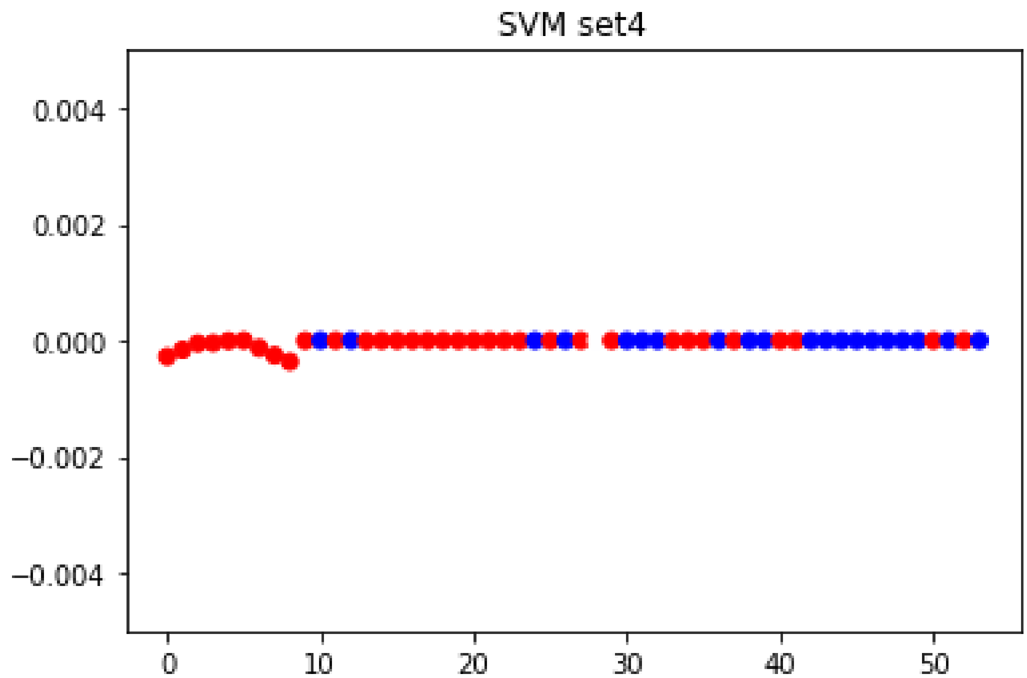

Logistic Regression set6
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
0
10
20
30
40
50

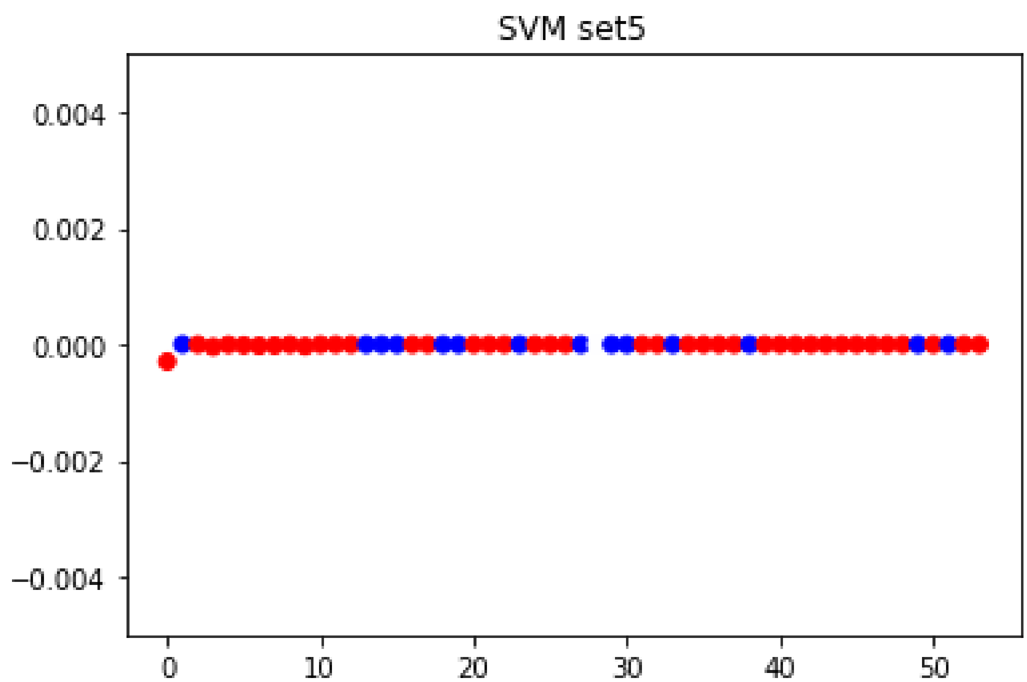

SVM set0
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50

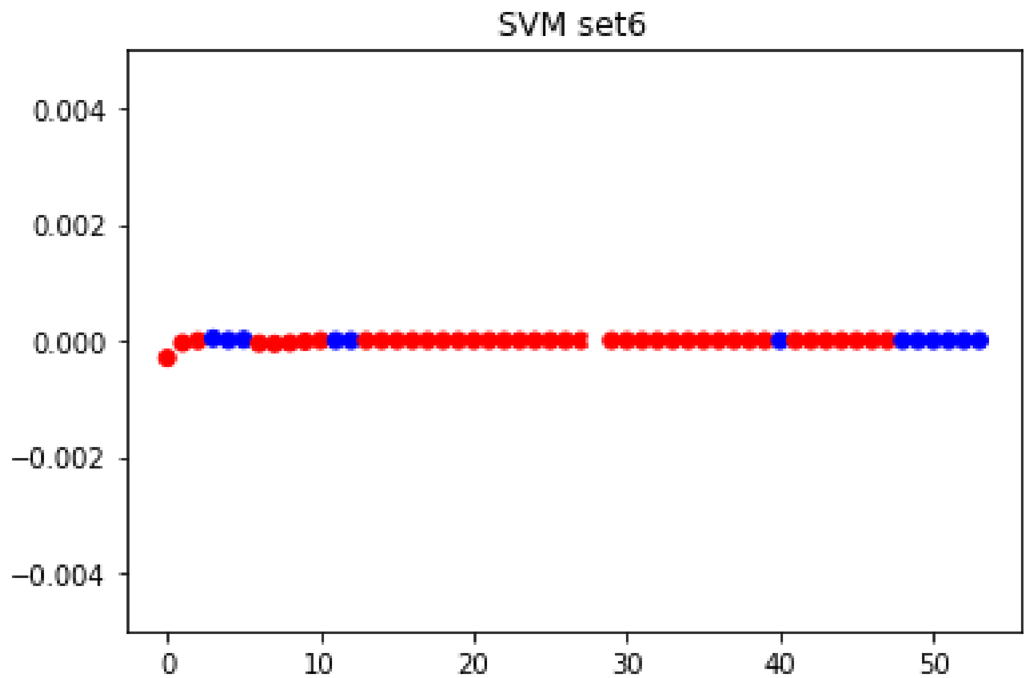

SVM set1
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50


SVM set2
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50


SVM set3
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50


SVM set4
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50


SVM set5
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50


SVM set6
0.004
0.002
0.000
−0.002
−0.004
0
10
20
30
40
50