Task 1

February 7, 2018

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [2]: from sklearn.datasets import fetch_california_housing
        from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
        from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
        from sklearn.preprocessing import StandardScaler
```

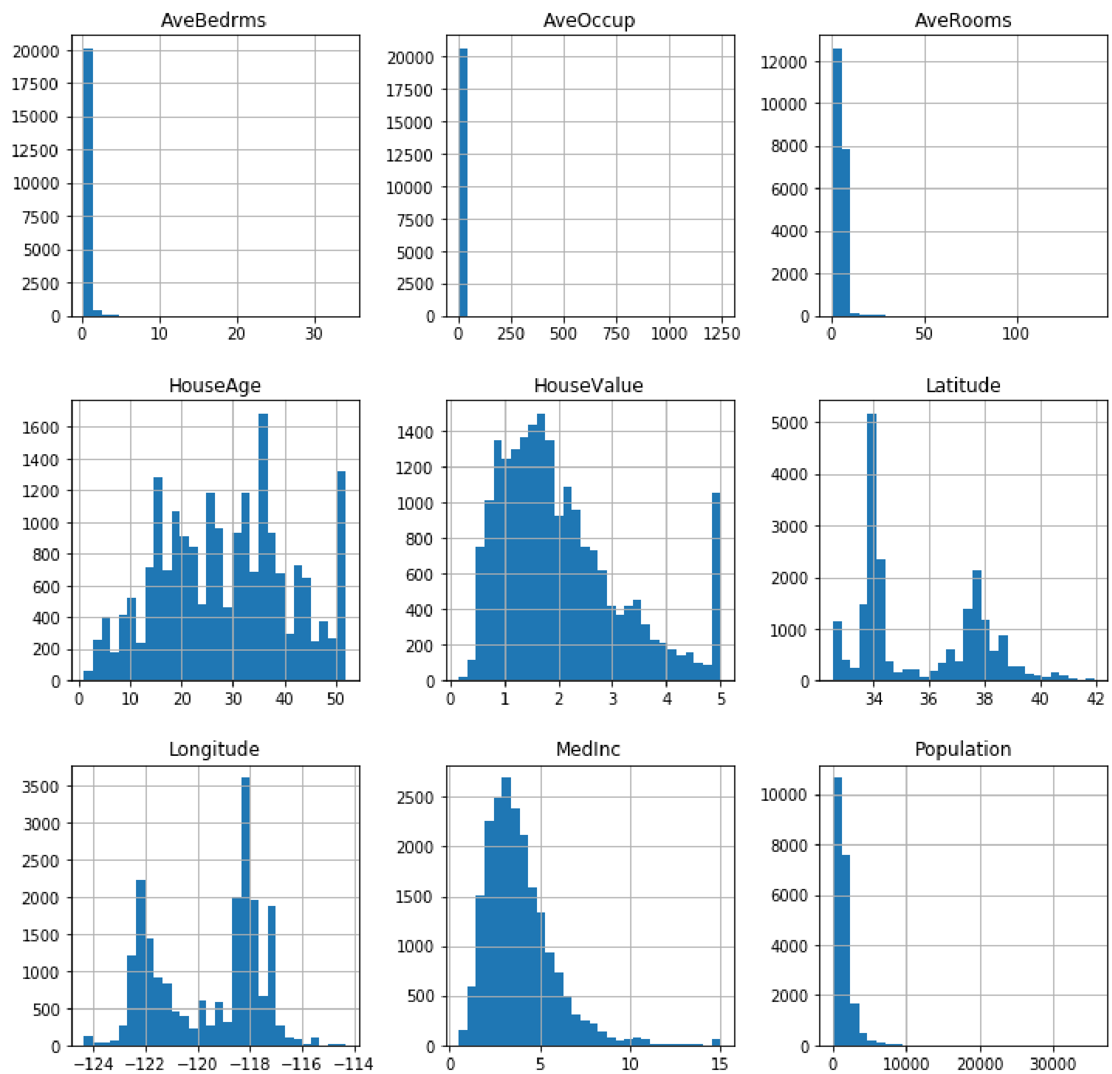
```
In [3]: calhouse = fetch_california_housing(data_home='task1_data')
```

```
In [4]: calhouse_df = pd.DataFrame(calhouse.data, columns=calhouse.feature_names)
```

0.0.1 Task 1.1: Visualize the univariate distribution of each feature, and the distribution of the target

```
In [5]: #Visualize distribution of each feature, including target feature
        calhouse_target_df = calhouse_df
        calhouse_target_df = calhouse_target_df.assign(HouseValue = calhouse.target)
        fig2 = plt.figure(figsize = (12,12))
        _ = calhouse_target_df.hist(bins = 30, ax = plt.gca())
```

```
/home/tungngo/anaconda2/envs/py36/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2
  exec(code_obj, self.user_global_ns, self.user_ns)
```



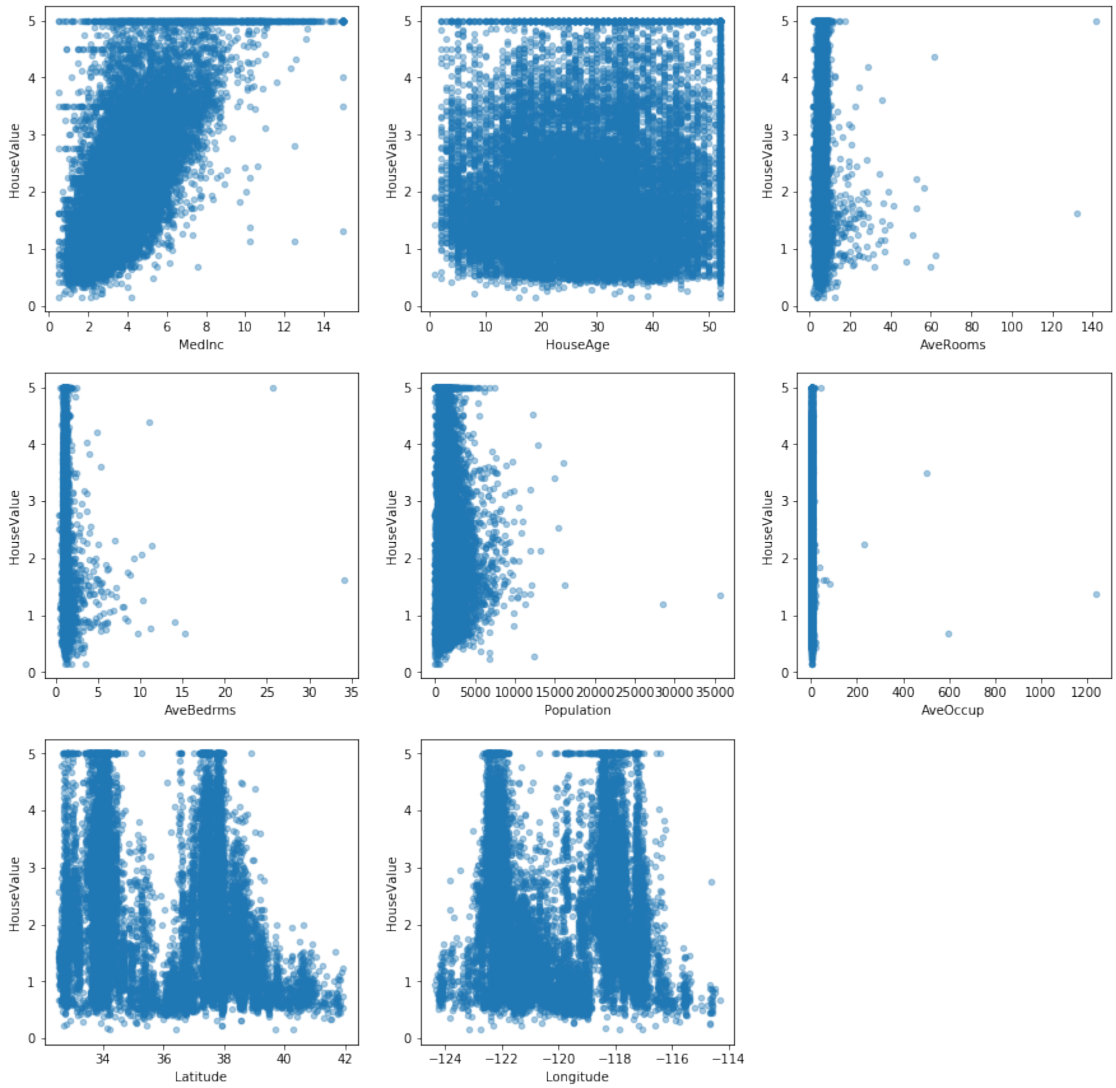
Observations:

- Outliers in `AveRooms`, `AveBedrms`, `Population`, `AveOccup`.
  - Most values of these variables are small, except for very small number of data points having big values
- Target value variable `HouseValue` goes up toward the end. Same observation for `HouseAge`.
  - `HouseValue` has 965 values = 5.00001
  - `HouseAge` 1273 values = 52

0.0.2 Task 1.2: Visualize the dependency of the target on each feature

```
In [6]: #Visualize the dependency of the target on each feature (2d scatter plot)
        fig = plt.figure(figsize=(15,15))
        for i in range(1,9):
```

```
    ax = fig.add_subplot(3, 3, i)
    calhouse_target_df.plot.scatter(calhouse_target_df.columns.values[i-1],
                                    calhouse_target_df.columns.values[8], ax=ax, alpha=0
```



0.0.3 Task 1.3: Evaluate Linear Regression (OLS), Ridge, Lasso and ElasticNet using cross-validation with the default parameters

```
In [7]: X_train, X_test, y_train, y_test = train_test_split(calhouse.data,calhouse.target,random
```

```
In [8]: #Linear Regression, Ridge, Lasso
        lr_score = np.mean(cross_val_score(LinearRegression(),X_train,y_train))
        ridge_score = np.mean(cross_val_score(Ridge(random_state=20),X_train,y_train))
        lasso_score = np.mean(cross_val_score(Lasso(random_state=20),X_train,y_train))
        enet_score = np.mean(cross_val_score(ElasticNet(random_state=20),X_train,y_train))
```

```
In [9]: print("linear regression score: {}".format(lr_score))
        print("ridge regression score: {}".format(ridge_score))
        print("lasso regression score: {}".format(lasso_score))
        print("elastic net score: {}".format(enet_score))
```

```
linear regression score: 0.6009222504208382
ridge regression score: 0.6009322158862299
lasso regression score: 0.27846544566928083
elastic net score: 0.41865896236326555
```

```
In [10]: #Use Standard scaler to scale data
         scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)
```

```
In [11]: #Use scaled data for regression
         slr_score = np.mean(cross_val_score(LinearRegression(),X_train_scaled,y_train))
         sridge_score = np.mean(cross_val_score(Ridge(random_state=20),X_train_scaled,y_train))
         slasso_score = np.mean(cross_val_score(Lasso(random_state=20),X_train_scaled,y_train))
         senet_score = np.mean(cross_val_score(ElasticNet(random_state=20),X_train_scaled,y_trai
```

```
In [12]: print("linear regression score (scaled): {}".format(slr_score))
         print("ridge regression score (scaled): {}".format(sridge_score))
         print("lasso regression score (scaled): {}".format(slasso_score))
         print("elastic net score (scaled): {}".format(senet_score))
```

```
linear regression score (scaled): 0.6009222504208398
ridge regression score (scaled): 0.6009303872146475
lasso regression score (scaled): -0.0005254527749588936
elastic net score (scaled): 0.19901416988152856
```

```
In [13]: print("linear regression scaled vs non-scaled: {}".format(slr_score-lr_score))
         print("ridge regression scaled vs non-scaled: {}".format(sridge_score-ridge_score))
         print("lasso regression scaled vs non-scaled: {}".format(slasso_score-lasso_score))
         print("elastic net scaled vs non-scaled: {}".format(senet_score-enet_score))
```

```
linear regression scaled vs non-scaled: 1.5543122344752192e-15
ridge regression scaled vs non-scaled: -1.828671582382313e-06
lasso regression scaled vs non-scaled: -0.27899089844423974
elastic net scaled vs non-scaled: -0.219644792481737
```

Scaling the data does not help improve performance of linear regression or ridge regression. It even worsens the performance of lasso regression and elastic net.

0.0.4 Task 1.4: Tune the parameters of the models using GridSearchCV

In [14]:
```
param_grid_ridge = {'alpha': np.logspace(-3, 1, 20)}
grid_ridge = GridSearchCV(Ridge(random_state=20, normalize=True), param_grid_ridge)
grid_ridge.fit(X_train, y_train)
```

Out[14]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=True, random_state=20, solver='auto', tol=0.001),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': array([  1.00000e-03,   1.62378e-03,   2.63665e-03,   4.281
         6.95193e-03,   1.12884e-02,   1.83298e-02,   2.97635e-02,
         4.83293e-02,   7.84760e-02,   1.27427e-01,   2.06914e-01,
         3.35982e-01,   5.45559e-01,   8.85867e-01,   1.43845e+00,
         2.33572e+00,   3.79269e+00,   6.15848e+00,   1.00000e+01])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

In [15]:
```
print(grid_ridge.best_params_)
print(grid_ridge.best_score_)
```

```
{'alpha': 0.0016237767391887211}
0.600990210481
```

In [16]:
```
param_grid_lasso = {'alpha': np.logspace(-4, 0, 9)}
grid_lasso = GridSearchCV(Lasso(random_state=20, normalize=True), param_grid_lasso)
grid_lasso.fit(X_train, y_train)
```

Out[16]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=True, positive=False, precompute=False, random_state=20,
   selection='cyclic', tol=0.0001, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': array([  1.00000e-04,   3.16228e-04,   1.00000e-03,   3.162
         1.00000e-02,   3.16228e-02,   1.00000e-01,   3.16228e-01,
         1.00000e+00])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

In [17]:
```
print(grid_lasso.best_params_)
print(grid_lasso.best_score_)
```

```
{'alpha': 0.0001}
0.598328713871
```

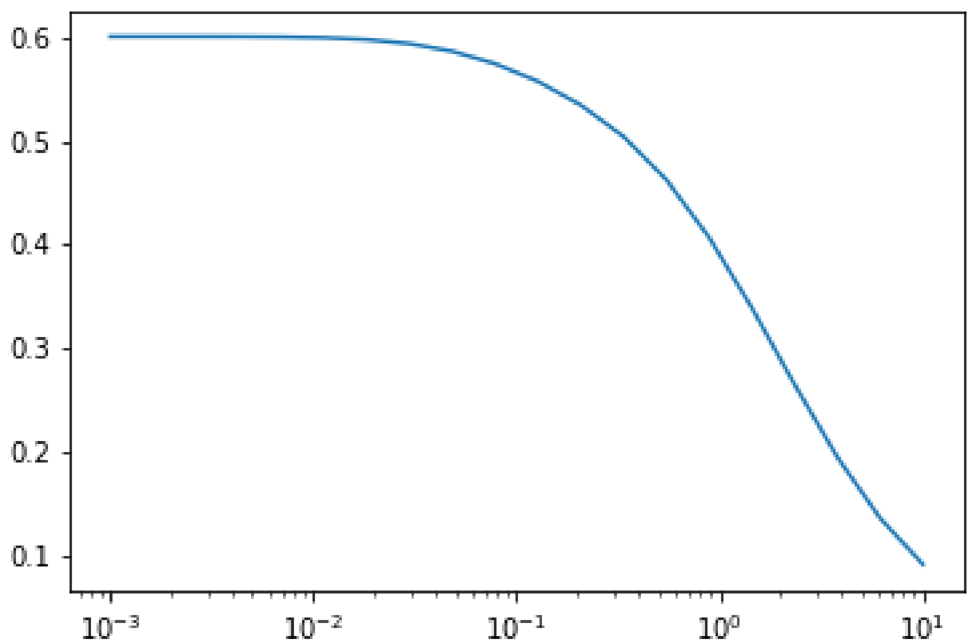
In [18]:
```
param_grid_elastic = {'alpha': np.logspace(-4, -1, 7),
              'l1_ratio': [0.01, .1, .5, .9, .98, 1]}
grid_elastic = GridSearchCV(ElasticNet(random_state=20, normalize=True), param_grid_ela
grid_elastic.fit(X_train, y_train)
```

```
Out[18]: GridSearchCV(cv=None, error_score='raise',
              estimator=ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
             max_iter=1000, normalize=True, positive=False, precompute=False,
             random_state=20, selection='cyclic', tol=0.0001, warm_start=False),
              fit_params=None, iid=True, n_jobs=1,
              param_grid={'alpha': array([ 0.0001 ,  0.00032,  0.001  ,  0.00316,  0.01   ,  0
              pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
              scoring=None, verbose=0)
```

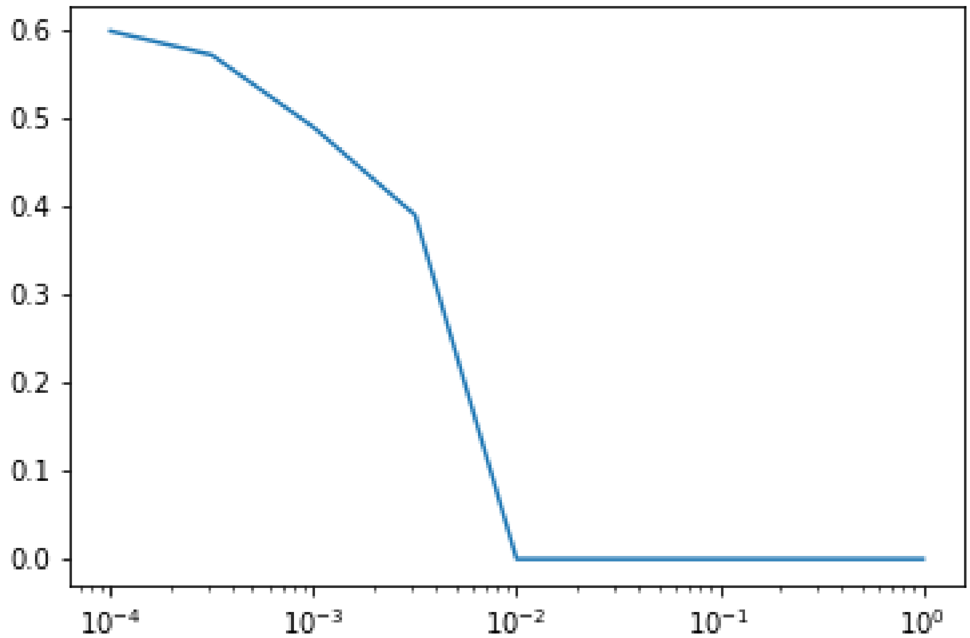
```
In [19]: print(grid_elastic.best_params_)
         print(grid_elastic.best_score_)
```

```
{'alpha': 0.0001, 'l1_ratio': 1}
0.598328713871
```

```
In [20]: #Visualize the dependence of the validation score on the parameters for Ridge, Lasso an
         ridge_line, = plt.semilogx(param_grid_ridge['alpha'],grid_ridge.cv_results_['mean_test_
```




```
In [21]: lasso_line, = plt.semilogx(param_grid_lasso['alpha'],grid_lasso.cv_results_['mean_test_
```

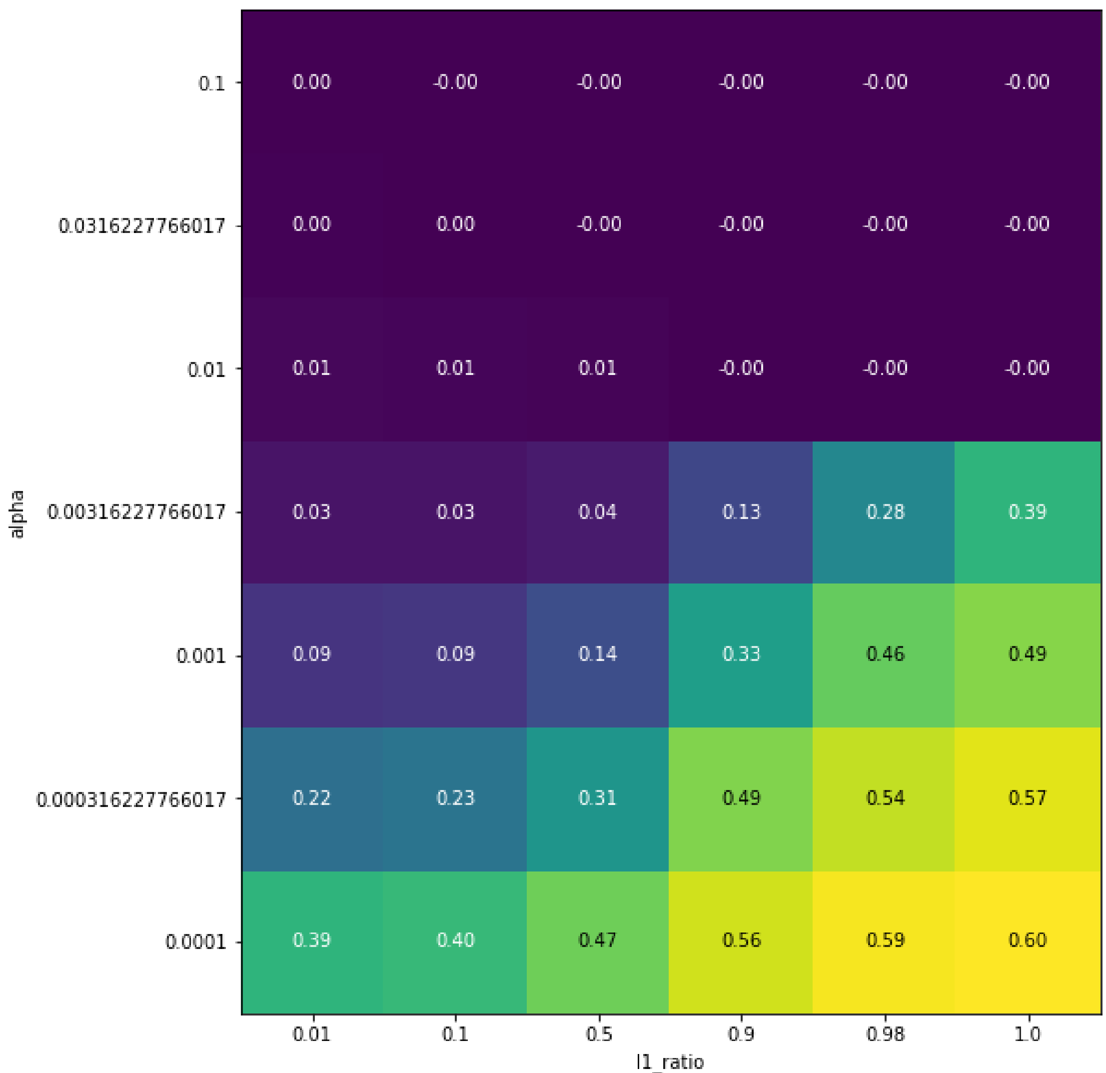



```
In [22]: elastic_net_para = pd.pivot_table(pd.DataFrame(grid_elastic.cv_results_),
             values='mean_test_score', index='param_alpha', columns='param_l1_ratio')

In [23]: def heatmap(values, xlabel, ylabel, xticklabels, yticklabels, cmap=None,
                     vmin=None, vmax=None, ax=None, fmt="%0.2f"):
             if ax is None:
                 ax = plt.gca()
             # plot the mean cross-validation scores
             img = ax.pcolor(values, cmap=cmap, vmin=vmin, vmax=vmax)
             img.update_scalarmappable()
             ax.set_xlabel(xlabel)
             ax.set_ylabel(ylabel)
             ax.set_xticks(np.arange(len(xticklabels)) + .5)
             ax.set_yticks(np.arange(len(yticklabels)) + .5)
             ax.set_xticklabels(xticklabels)
             ax.set_yticklabels(yticklabels)
             ax.set_aspect(1)

             for p, color, value in zip(img.get_paths(), img.get_facecolors(),
                                        img.get_array()):
                 x, y = p.vertices[:-2, :].mean(0)
                 if np.mean(color[:3]) > 0.5:
                     c = 'k'
                 else:
                     c = 'w'
```

```
                ax.text(x, y, fmt % value, color=c, ha="center", va="center")
            return img
```

```
In [24]: plt.figure(figsize=(10,10))
         heatmap(elastic_net_para,'l1_ratio','alpha',xticklabels=elastic_net_para.columns,ytickl
```

```
Out[24]: <matplotlib.collections.PolyCollection at 0x7f1b741e7b00>
```



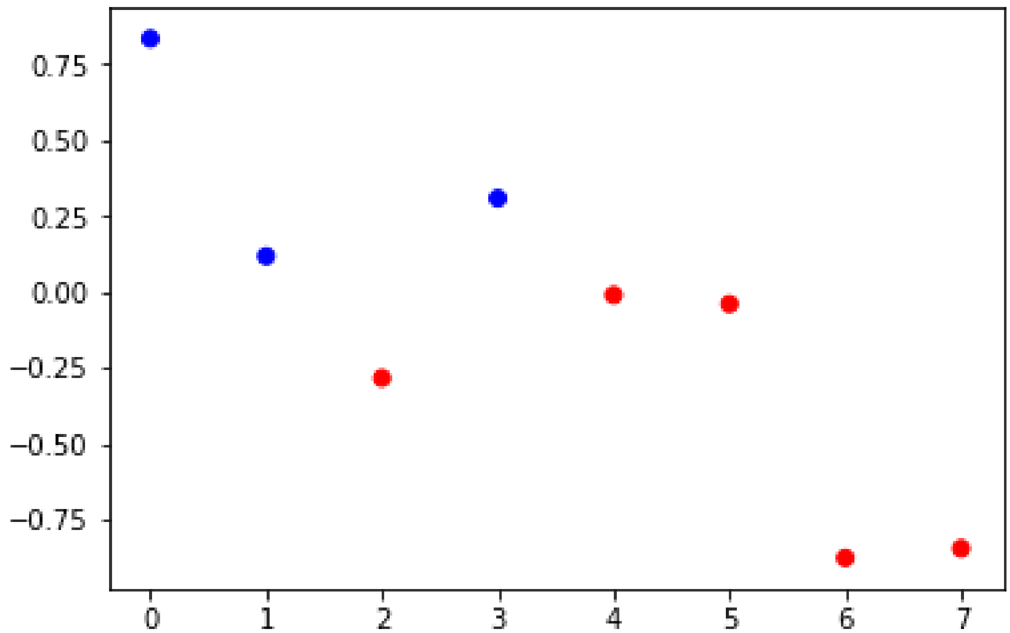
Tuning the parameter helps Lasso regression and Elastic Net approach the accuracy of linear regression and ridge regression, but tuning does not help improve the accuracy of the best model. The best score still stays at 60.1% - belongs to linear regression and ridge regression.

0.0.5 Task 1.5

```
In [25]: #Visualize the coefficients of the resulting models
```

```
In [26]: #Linear regression
         lr_est = LinearRegression().fit(X_train_scaled,y_train)
         plt.scatter(range(X_train_scaled.shape[1]), lr_est.coef_,
                     c=np.sign(lr_est.coef_), cmap="bwr_r")
```

```
Out[26]: <matplotlib.collections.PathCollection at 0x7f1b740d05c0>
```

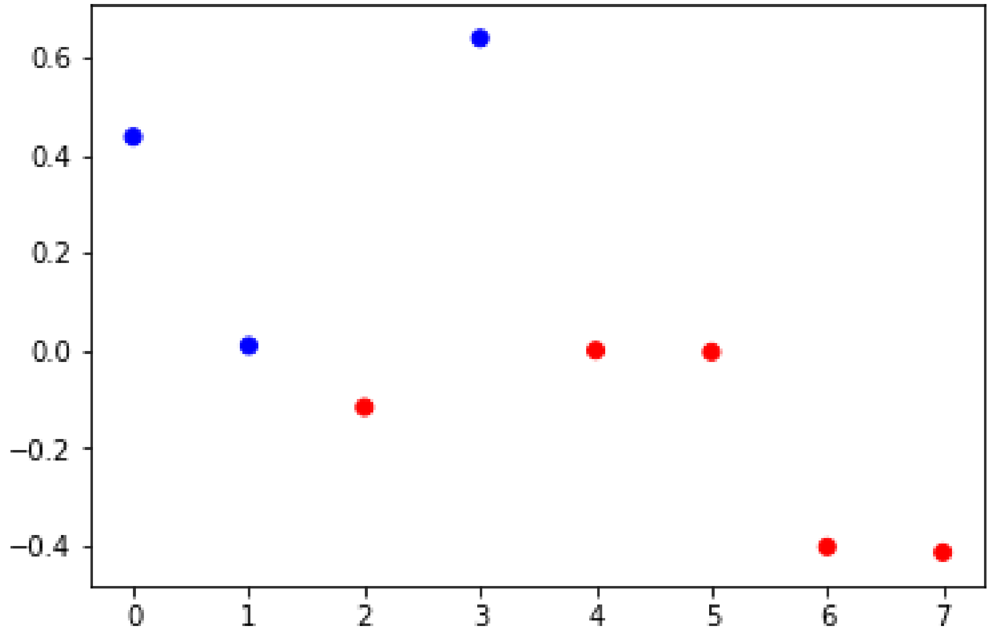



```
In [27]: lr_est.coef_
```

```
Out[27]: array([ 0.83400129,  0.11738854, -0.28411895,  0.30813331, -0.01071326,
                -0.04025842, -0.87663514, -0.84554562])
```

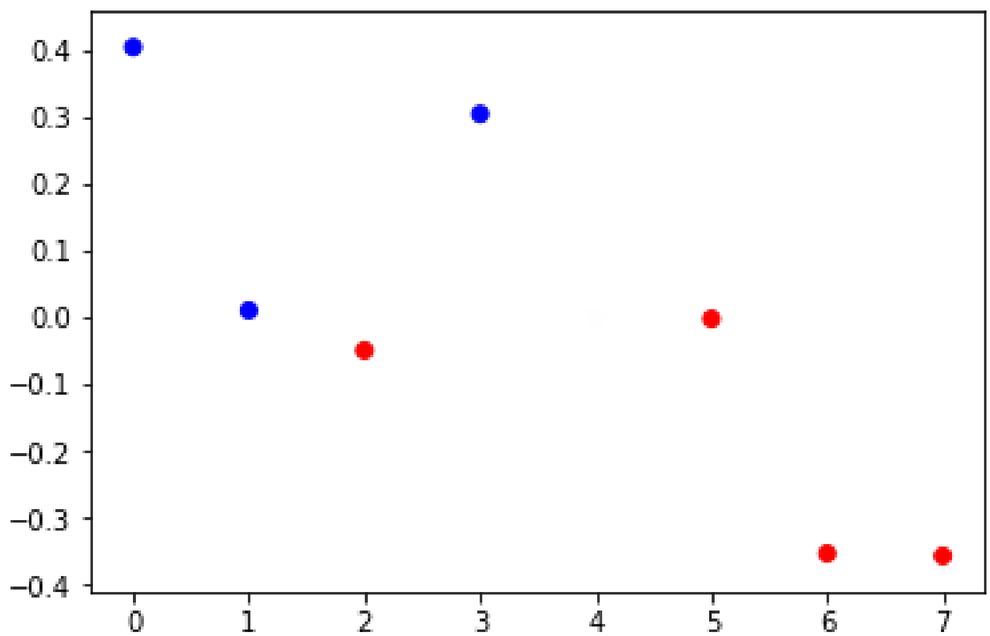
```
In [28]: #Ridge regression
         ridge_est = grid_ridge.best_estimator_
         plt.scatter(range(X_train.shape[1]), ridge_est.coef_,
                     c=np.sign(ridge_est.coef_), cmap="bwr_r")
```

```
Out[28]: <matplotlib.collections.PathCollection at 0x7f1b740a8d30>
```



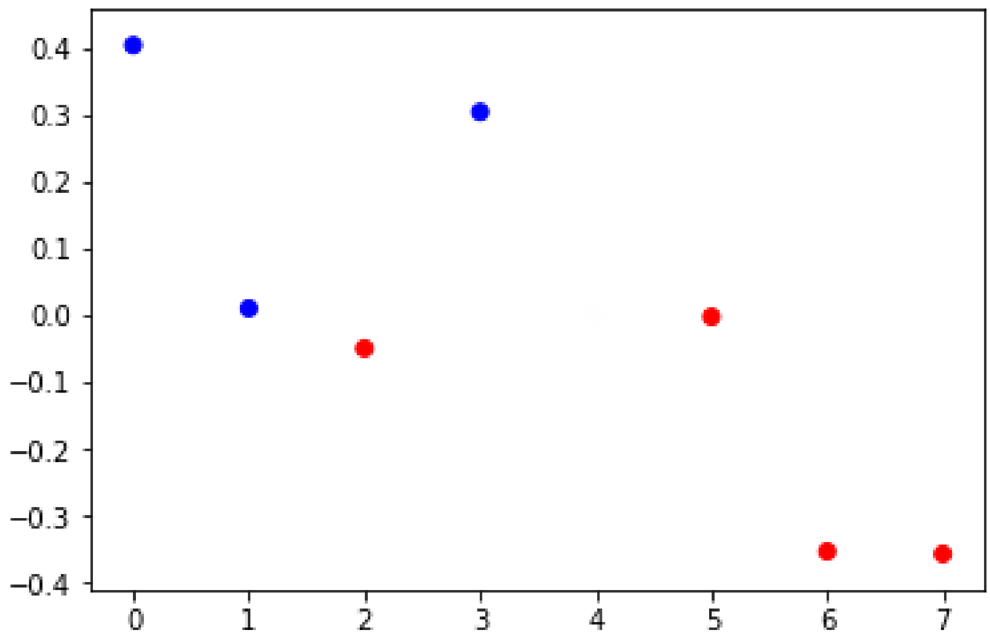

```
In [29]: #Lasso regression
         lasso_est = grid_lasso.best_estimator_
         plt.scatter(range(X_train.shape[1]), lasso_est.coef_,
                     c=np.sign(lasso_est.coef_), cmap="bwr_r")
```

```
Out[29]: <matplotlib.collections.PathCollection at 0x7f1b74088160>
```



In [30]:
```
#Elastic Net
elastic_est = grid_elastic.best_estimator_
plt.scatter(range(X_train.shape[1]), elastic_est.coef_,
            c=np.sign(elastic_est.coef_), cmap="bwr_r")
```

Out[30]: <matplotlib.collections.PathCollection at 0x7f1b6f7dc198>



All models agree on which features have positive or negative coefficients. However, the resulting models differ in term of scale of features. While linear regression seems to take negative and positive-coefficient features to the extreme, ridge regression scales down these coefficients - which is expected from the effect of regularization. We can say the same about lasso regression and elastic net when compared with linear regression (for example, features 0,6,7 are lower in absolute values in the 3 models - around 0.4 - compared to linear regression around 0.8). The resulting models of lasso regression and elastic net are very similar as the l1 ratio of elastic net is 1. Final observation is that feature 3 of ridge regression have higher importance - positive coefficient of around 0.6 - compared to other 3 models - coefficient of around 0.3

Task 2

February 7, 2018

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set()

        %matplotlib inline

In [2]: from sklearn.linear_model import LogisticRegression
        from sklearn.svm import LinearSVC
        from sklearn.neighbors import NearestCentroid
        from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFo
        from sklearn.preprocessing import StandardScaler

In [3]: from sklearn.datasets import fetch_covtype

In [4]: data = fetch_covtype(data_home='task2_data')

In [5]: data

Out[5]: {'DESCR': 'Forest covertype dataset.\n\nA classic dataset for classification benchmarks,
         'data': array([[  2.59600000e+03,   5.10000000e+01,   3.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
               [  2.59000000e+03,   5.60000000e+01,   2.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
               [  2.80400000e+03,   1.39000000e+02,   9.00000000e+00, ...,
                  0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
               ...,
               [  2.38600000e+03,   1.59000000e+02,   1.70000000e+01, ...,
                  0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
               [  2.38400000e+03,   1.70000000e+02,   1.50000000e+01, ...,
                  0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
               [  2.38300000e+03,   1.65000000e+02,   1.30000000e+01, ...,
                  0.00000000e+00,   0.00000000e+00,   0.00000000e+00]]),
         'target': array([5, 5, 2, ..., 3, 3, 3], dtype=int32)}

In [6]: cov = pd.DataFrame(data['data'])
        cov['Cover_Type'] = data['target']
```

```
In [7]: # Get column names
        feature_file = open('feature_names','r')
        features = []
        for line in feature_file:
            features.append(line.split()[0])

In [8]: wildareas = [features[10] + str(i) for i in range(1,5)]
        soiltypes = [features[11] + str(i) for i in range(1,41)]
        features = features[:10] + wildareas + soiltypes + [features[12]]

In [9]: cov.columns = features

In [10]: cov.head()
```

```
Out[10]:    Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  \
         0     2596.0    51.0    3.0                             258.0
         1     2590.0    56.0    2.0                             212.0
         2     2804.0   139.0    9.0                             268.0
         3     2785.0   155.0   18.0                             242.0
         4     2595.0    45.0    2.0                             153.0

            Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  \
         0                             0.0                            510.0
         1                            -6.0                            390.0
         2                            65.0                           3180.0
         3                           118.0                           3090.0
         4                            -1.0                            391.0

            Hillshade_9am  Hillshade_Noon  Hillshade_3pm  \
         0          221.0           232.0          148.0
         1          220.0           235.0          151.0
         2          234.0           238.0          135.0
         3          238.0           238.0          122.0
         4          220.0           234.0          150.0

            Horizontal_Distance_To_Fire_Points     ...       Soil_Type32  Soil_Type33  \
         0                              6279.0     ...               0.0          0.0
         1                              6225.0     ...               0.0          0.0
         2                              6121.0     ...               0.0          0.0
         3                              6211.0     ...               0.0          0.0
         4                              6172.0     ...               0.0          0.0

            Soil_Type34  Soil_Type35  Soil_Type36  Soil_Type37  Soil_Type38  \
         0          0.0          0.0          0.0          0.0          0.0
         1          0.0          0.0          0.0          0.0          0.0
         2          0.0          0.0          0.0          0.0          0.0
         3          0.0          0.0          0.0          0.0          0.0
         4          0.0          0.0          0.0          0.0          0.0
```

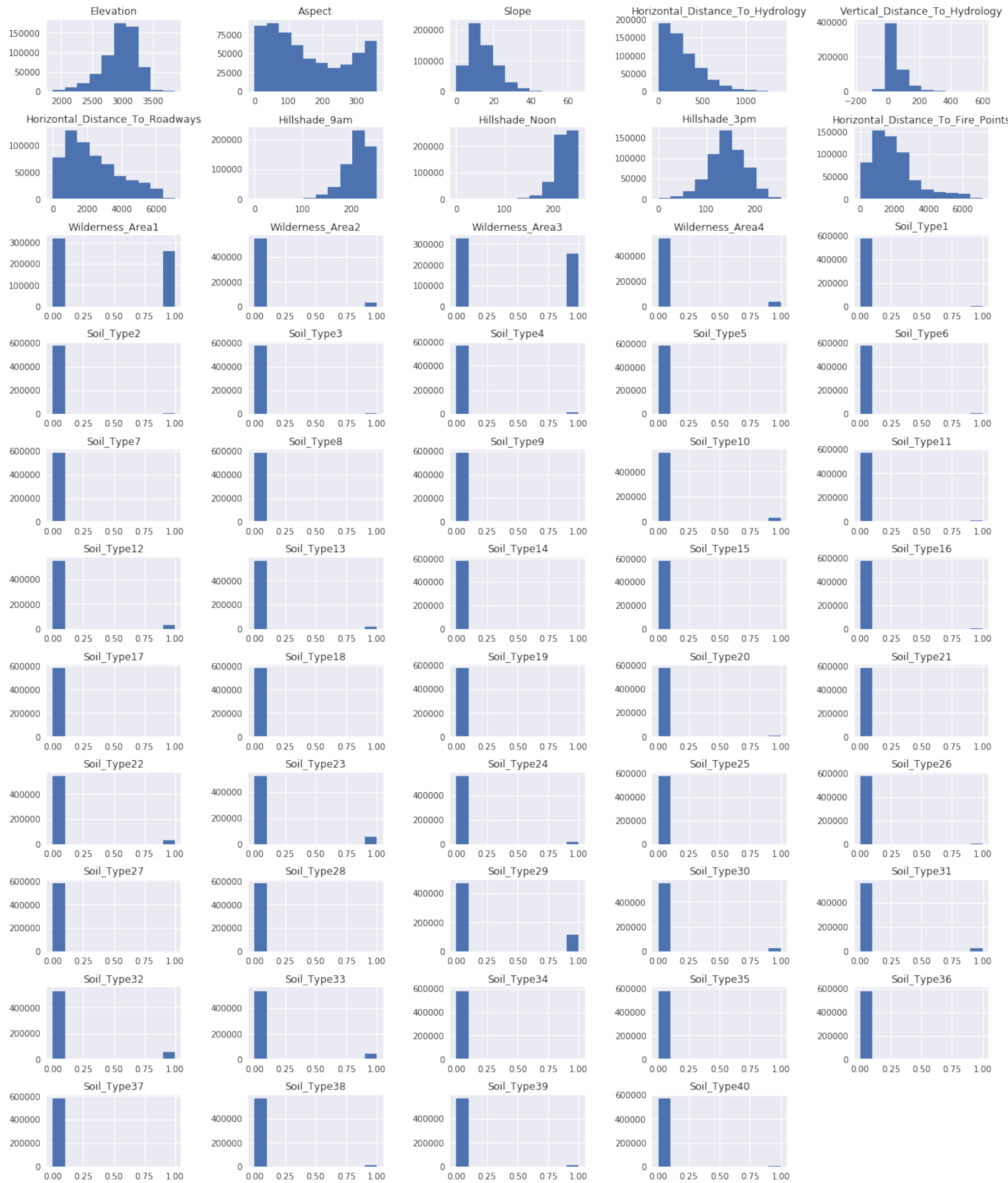
```
   Soil_Type39  Soil_Type40  Cover_Type
0          0.0          0.0           5
1          0.0          0.0           5
2          0.0          0.0           2
3          0.0          0.0           2
4          0.0          0.0           5

[5 rows x 55 columns]
```

In [11]:
```
cov.shape
```

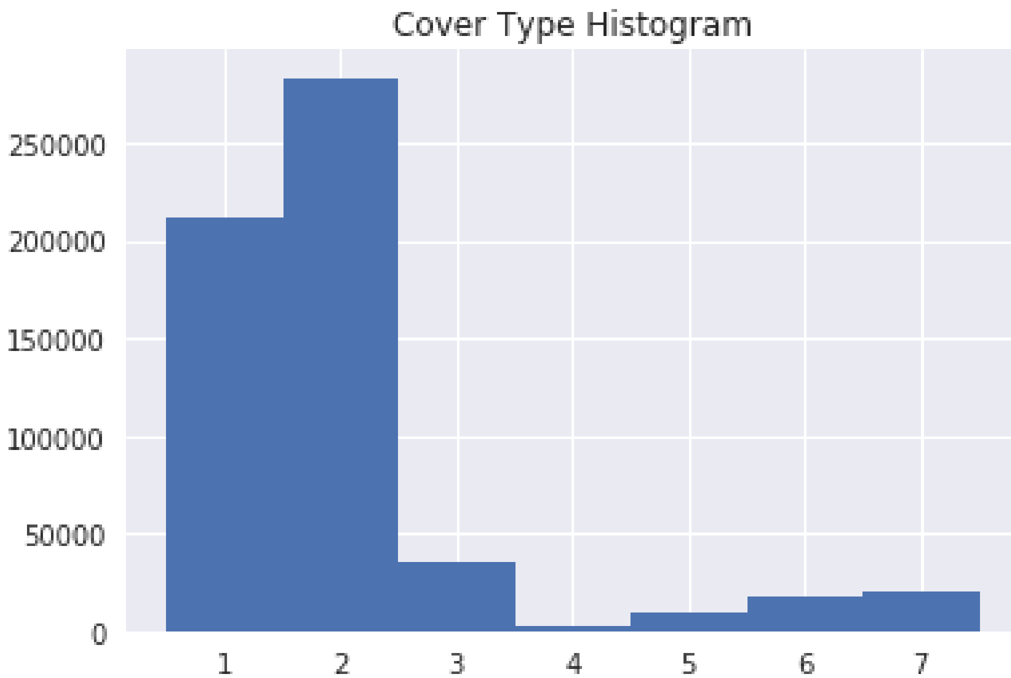
Out[11]:
```
(581012, 55)
```

0.0.1 Task 2.1: Visualize the univariate distribution of each feature, and the distribution of the target

In [12]:
```
fig = plt.figure(figsize=(20,25))
fig.subplots_adjust(hspace=0.5, wspace=0.5)
for i in range(1, 55):
    ax = fig.add_subplot(11, 5, i)
    cov[cov.columns[i-1]].hist(ax=ax)
    ax.set(title=cov.columns[i-1])
```




```
In [13]: #Distribution of target
         plt.hist(cov['Cover_Type'],range=(0.5,7.5),bins=7)
         _ = plt.title('Cover Type Histogram')
```



0.0.2 Task 2.2: Split data into training and test set. Evaluate Logistic Regression, linear support vector machines and nearest centroids using cross-validation

```
In [14]: X_train, X_test, y_train, y_test = train_test_split(data['data'],data['target'],random_
```

X_train = X_train[:50000] y_train = y_train[:50000]

```
In [15]: #Evaluate Logistic Regression
         logit_score = np.mean(cross_val_score(LogisticRegression(tol=0.0001), X_train, y_train)

In [16]: #Linear SVM
         linearSVM_score = np.mean(cross_val_score(LinearSVC(tol=0.0001, dual=False), X_train, y

In [17]: #Nearest Centroid
         NC_score = np.mean(cross_val_score(NearestCentroid(), X_train, y_train))

In [18]: print(logit_score)
         print(linearSVM_score)
         print(NC_score)
```

```
0.708256150789
0.705947542451
0.194639694306
```

```
In [19]: #Use standardScaler
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
```

```
In [20]: slogit_score = np.mean(cross_val_score(LogisticRegression(tol=0.0001), X_train_scaled,
         slinearSVM_score = np.mean(cross_val_score(LinearSVC(tol=0.0001, dual=False), X_train_s
         sNC_score = np.mean(cross_val_score(NearestCentroid(), X_train_scaled, y_train))
```

```
In [21]: print(slogit_score)
         print(slinearSVM_score)
         print(sNC_score)
```

```
0.715418375449
0.712604890841
0.548706947943
```

Logistic regression and Linear SVM have close scores - Logistic regression is slightly better, while Nearest Centroid has very bad score compared to other 2 algorithms. When applying StandardScaler, all 3 algorithms improve in score: logistic regression from 70.8% to 71.5%, linear SVM from 70.6% to 71.3%. Nearest Centroid improves the greatest from 19.5% to 54.9%, althought final score is still far behind other 2 algorithms.

0.0.3 Task 2.3: Tune the parameters using GridSearchCV

```
In [22]: #Tune logistic regression
         param_grid_logit = {'C':np.logspace(-2,2,6)}
         grid_logit = GridSearchCV(LogisticRegression(tol=0.0001,random_state=20), param_grid_lo
         grid_logit.fit(X_train_scaled, y_train)
         print(grid_logit.best_params_)
         print(grid_logit.best_score_)
```

```
{'C': 100.0}
0.715496409713
```

```
In [23]: #Tune Linear SVM
         param_grid_svm = {'C':np.logspace(-2,2,6)}
         grid_svm = GridSearchCV(LinearSVC(tol=0.0001, dual=False, random_state=20), param_grid_
         grid_svm.fit(X_train_scaled, y_train)
         print(grid_svm.best_params_)
         print(grid_svm.best_score_)
```

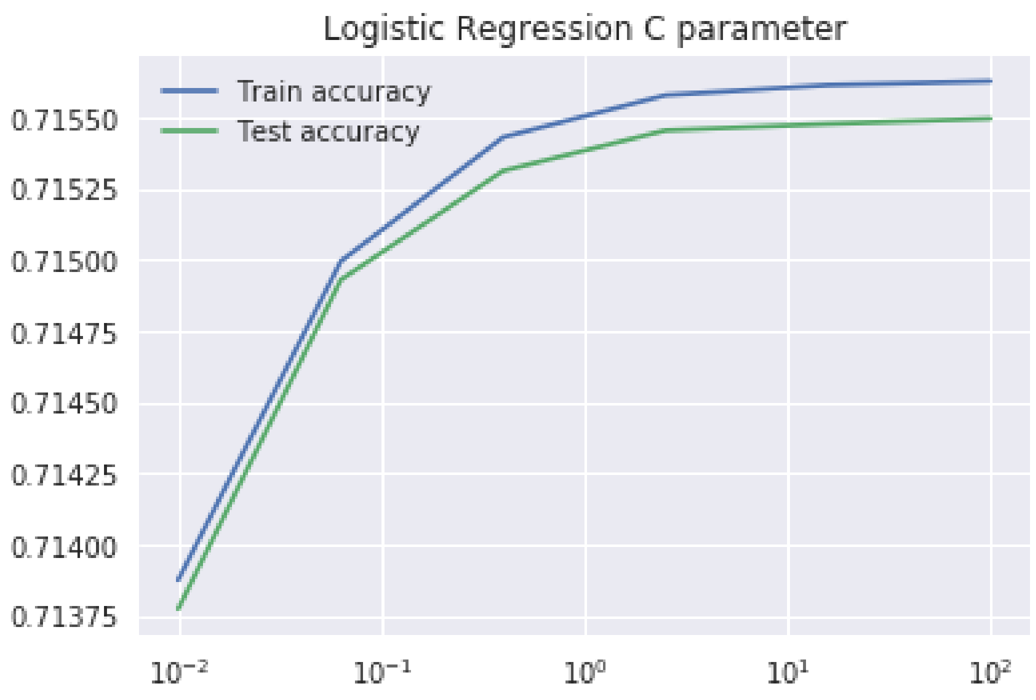
```
{'C': 2.5118864315095824}
0.712616377401
```

```
In [24]: #Tune Nearest Centroid
         param_grid_NC = {'shrink_threshold':np.linspace(0,20,11)}
```

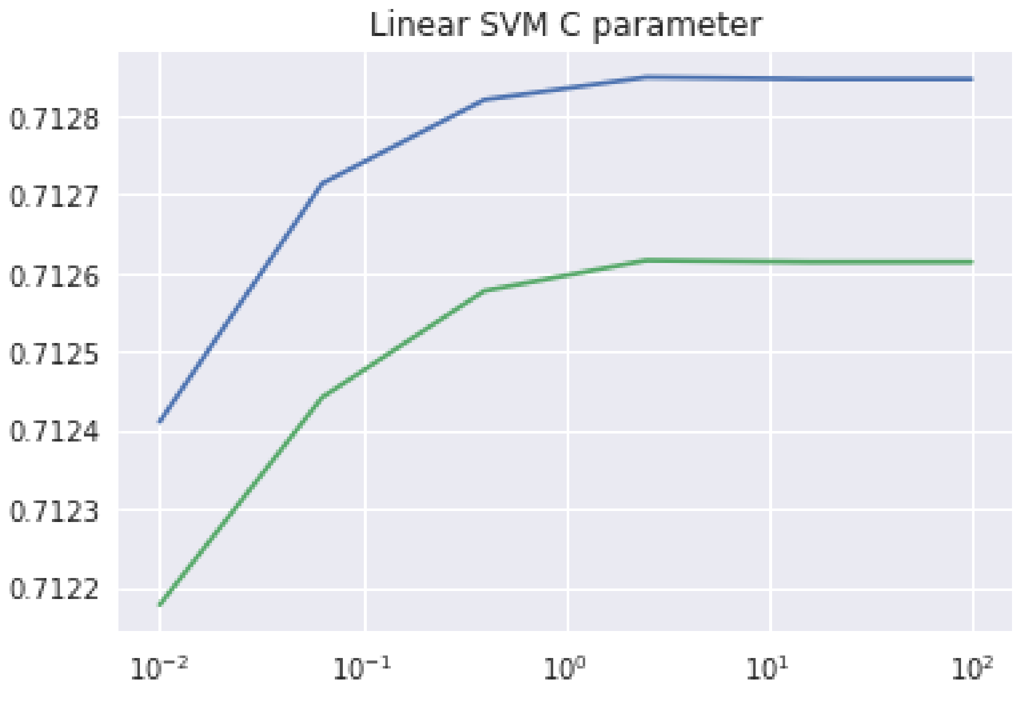
```
grid_NC = GridSearchCV(NearestCentroid(), param_grid_NC)
grid_NC.fit(X_train_scaled, y_train)
print(grid_NC.best_params_)
print(grid_NC.best_score_)
```

```
{'shrink_threshold': 0.0}
0.54870696876
```

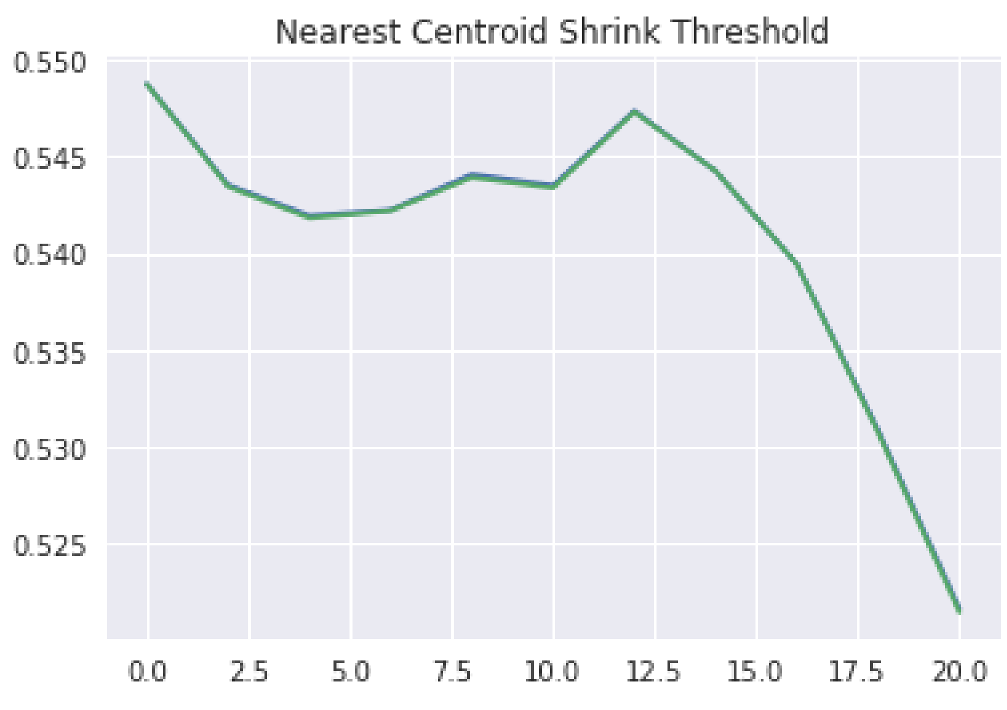
In [25]:
```
#Visualize GridSearch results
plt.semilogx(param_grid_logit['C'],grid_logit.cv_results_['mean_train_score'],label='Tr
plt.semilogx(param_grid_logit['C'],grid_logit.cv_results_['mean_test_score'],label='Tes
_ = plt.title('Logistic Regression C parameter')
_ = plt.legend()
```



In [26]:
```
plt.semilogx(param_grid_svm['C'],grid_svm.cv_results_['mean_train_score'],label='Train
plt.semilogx(param_grid_svm['C'],grid_svm.cv_results_['mean_test_score'],label='Test ac
_ = plt.title('Linear SVM C parameter')
_ = plt.legend()
```



In [27]:
```
plt.plot(param_grid_NC['shrink_threshold'],grid_NC.cv_results_['mean_train_score'],labe
plt.plot(param_grid_NC['shrink_threshold'],grid_NC.cv_results_['mean_test_score'],label
_ = plt.title('Nearest Centroid Shrink Threshold')
_ = plt.legend()
```



After GridSearch, the scores for all 3 algorithms do not improve by too much.

0.0.4 Task 2.4: Change the cross-validation strategy from 'stratified k-fold' to 'kfold' with shuffling

```
In [28]: #Change cross-validation strategy to KFold with shuffling and run GridSearch
         kfoldshuffle = KFold(shuffle=True, random_state=1)

In [29]: #Tune logistic regression with KFold with shuffling
         grid_logit_kfold = GridSearchCV(LogisticRegression(tol=0.0001,random_state=20), param_g
         grid_logit_kfold.fit(X_train_scaled, y_train)
         print(grid_logit_kfold.best_params_)
         print(grid_logit_kfold.best_score_)

{'C': 100.0}
0.715503294252


In [30]: #Tune Linear SVM with KFold with shuffling
         grid_svm_kfold = GridSearchCV(LinearSVC(tol=0.0001, dual= False, random_state=20), para
         grid_svm_kfold.fit(X_train_scaled, y_train)
         print(grid_svm_kfold.best_params_)
         print(grid_svm_kfold.best_score_)

{'C': 2.5118864315095824}
0.712731119724


In [31]: #Tune Nearest Centroid with KFold with shuffling
         grid_NC_kfold = GridSearchCV(NearestCentroid(), param_grid_NC,cv=kfoldshuffle)
         grid_NC_kfold.fit(X_train_scaled, y_train)
         print(grid_NC_kfold.best_params_)
         print(grid_NC_kfold.best_score_)

{'shrink_threshold': 0.0}
0.548782698694


In [32]: #Change random seed of the shuffling
         kfoldshuffle2 = KFold(shuffle=True, random_state=2)

In [33]: #Tune logistic regression with KFold with shuffling
         grid_logit_kfold2 = GridSearchCV(LogisticRegression(tol=0.0001,random_state=20), param_
         grid_logit_kfold2.fit(X_train_scaled, y_train)
         print(grid_logit_kfold2.best_params_)
         print(grid_logit_kfold2.best_score_)
```

```
{'C': 15.848931924611142}
0.715455102476
```

```
In [34]: #Tune Linear SVM with KFold with shuffling
         grid_svm_kfold2 = GridSearchCV(LinearSVC(tol=0.0001,dual=False, random_state=20), param
         grid_svm_kfold2.fit(X_train_scaled, y_train)
         print(grid_svm_kfold2.best_params_)
         print(grid_svm_kfold2.best_score_)
```

```
{'C': 15.848931924611142}
0.71274718365
```

```
In [35]: #Tune Nearest Centroid with KFold with shuffling
         grid_NC_kfold2 = GridSearchCV(NearestCentroid(), param_grid_NC,cv=kfoldshuffle2)
         grid_NC_kfold2.fit(X_train_scaled, y_train)
         print(grid_NC_kfold2.best_params_)
         print(grid_NC_kfold2.best_score_)
```

```
{'shrink_threshold': 0.0}
0.548617469748
```

```
In [52]: #Change random state of split and run GridSearch
         X_train3, X_test3, y_train3, y_test3 = train_test_split(data['data'],data['target'],ran
         X_train3_scaled = scaler.fit_transform(X_train3)
```

```
In [56]: #Tune logistic regression with different random split
         grid_logit_split2 = GridSearchCV(LogisticRegression(tol=0.0001,random_state=3), param_g
         grid_logit_split2.fit(X_train3_scaled, y_train3)
         print(grid_logit_split2.best_params_)
         print(grid_logit_split2.best_score_)
```

```
{'C': 100.0}
0.714791891848
```

```
In [62]: #Tune Linear SVM with with different random split
         grid_svm_split2 = GridSearchCV(LinearSVC(tol=0.0001,dual=False,random_state=3), param_g
         grid_svm_split2.fit(X_train3_scaled, y_train3)
         print(grid_svm_split2.best_params_)
         print(grid_svm_split2.best_score_)
```

```
{'C': 2.5118864315095824}
0.712142942554
```

In [58]:
```
#Tune Nearest Centroid with different random split
grid_NC_split2 = GridSearchCV(NearestCentroid(), param_grid_NC)
grid_NC_split2.fit(X_train3_scaled, y_train3)
print(grid_NC_split2.best_params_)
print(grid_NC_split2.best_score_)
```

```
{'shrink_threshold': 0.0}
0.548819416237
```

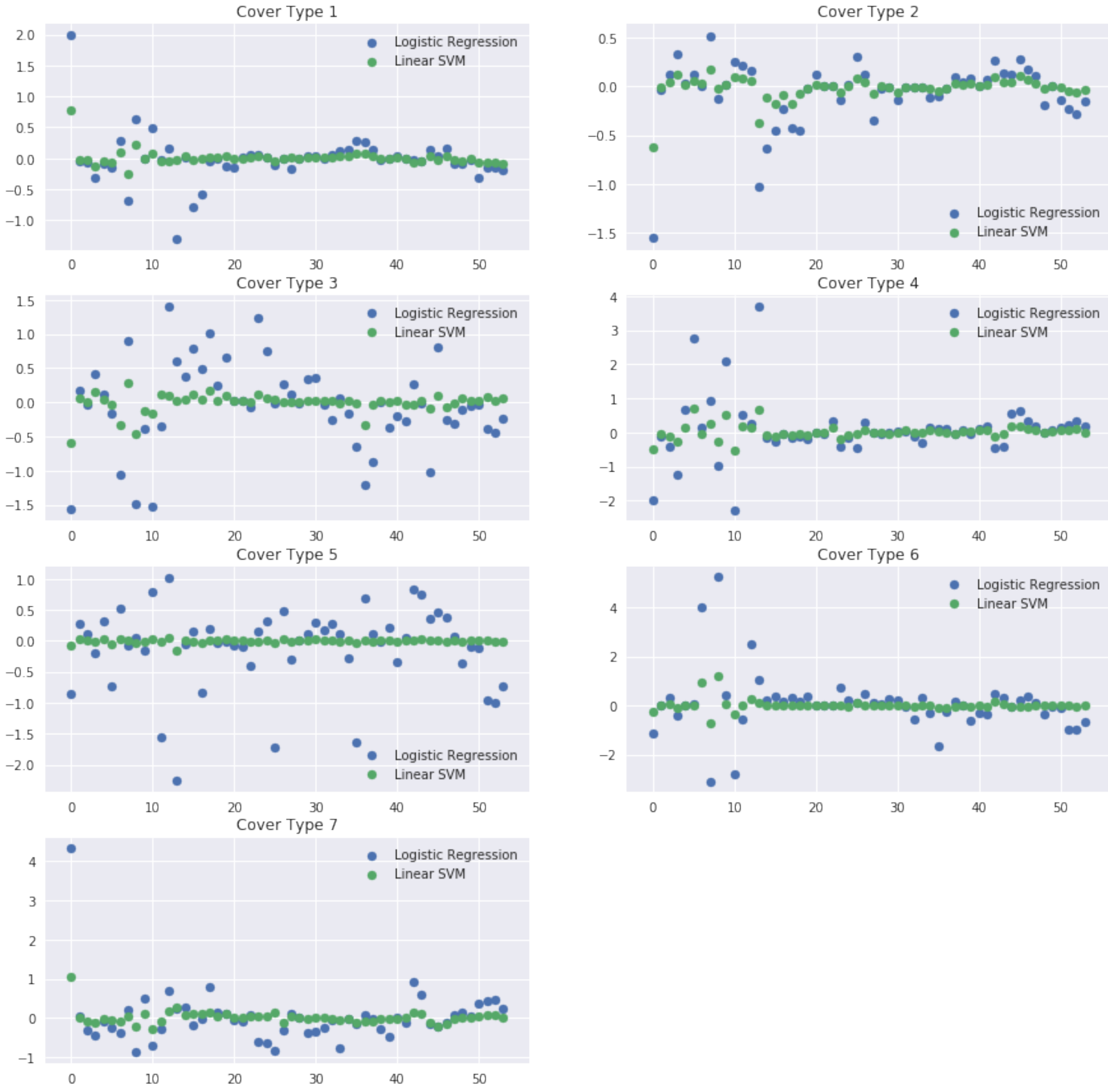
The parameters after GridSearch do not change after changing from StratifiedKFold to KFold with shuffling. However the parameters do change when we change the random state of the shuffling, from (100,2.5) for Logistic Regression and Linear SVM to (15.8,15.8). When changing the random state of the split into training and test data, the parameters also do not change compared to original set of parameters. In the 3 cases, parameters of Nearest Centroid do not change.

0.0.5 Task 2.5: Visualize the coefficients for LogisticRegression and Linear Support Vector Machines

In [42]:
```
#Logistic regression model
logitmodel = LogisticRegression(tol=0.0001,C=grid_logit.best_params_['C'],random_state=
logitmodel.fit(X_train_scaled,y_train)
#Linear SVM model
linearSVMmodel = LinearSVC(tol=0.0001, dual=False, C=grid_svm.best_params_['C'], random
linearSVMmodel.fit(X_train_scaled, y_train)
```

Out[42]:
```
LinearSVC(C=2.5118864315095824, class_weight=None, dual=False,
     fit_intercept=True, intercept_scaling=1, loss='squared_hinge',
     max_iter=1000, multi_class='ovr', penalty='l2', random_state=20,
     tol=0.0001, verbose=0)
```

In [43]:
```
fig = plt.figure(figsize=(15,15))
for i in range(1,8):
    ax = fig.add_subplot(4,2,i)
    ax.scatter(range(X_train.shape[1]), logitmodel.coef_[i-1],label='Logistic Regressio
    ax.scatter(range(X_train.shape[1]), linearSVMmodel.coef_[i-1],label='Linear SVM')
    ax.set(title='Cover Type '+str(i))
    plt.legend()
```


Cover Type 1
Cover Type 2
Cover Type 3
Cover Type 4
Cover Type 5
Cover Type 6
Cover Type 7
Logistic Regression
Linear SVM
0
10
20
30
40
50