Andrew Floyd

CS3001: Intro to Data Science

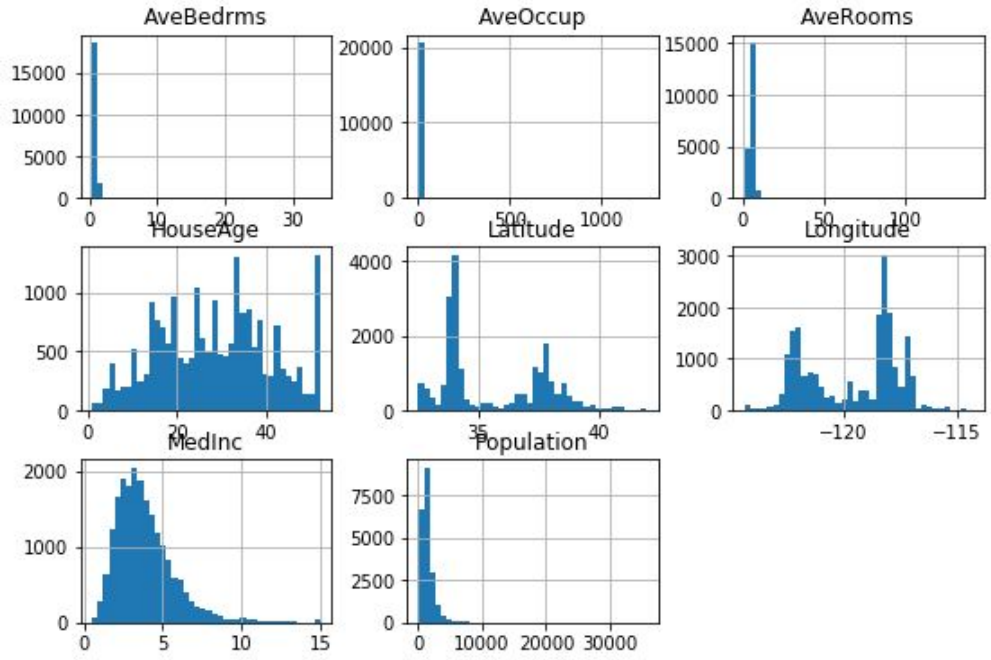
Dr. Fu

November 11th, 2018

HW6: Regression

Github repository: https://github.com/afzm4/cs3001hw6

1. Below is the combined visualization of the univariate distribution of each feature (see the code link for all individual graphs.)



(Here is some code for this)

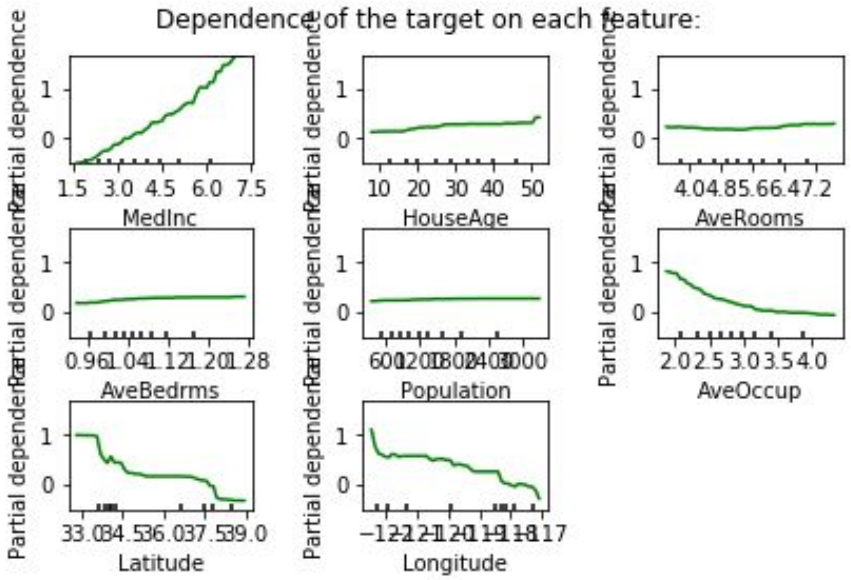
```
housingDF = pd.DataFrame(data=data, columns=names)
#All 8 DistPlots together
fig1 = housingDF.hist(bins=40, figsize=(9, 6))
```

(Here is the code for the individual plots)

```
#Q1
#DistPlots for all 8 features, individually
sns.distplot(data[:,0], axlabel=names[0])
sns.distplot(data[:,1], axlabel=names[1])
sns.distplot(data[:,2], axlabel=names[2])
sns.distplot(data[:,3], axlabel=names[3])
sns.distplot(data[:,4], axlabel=names[4])
sns.distplot(data[:,5], axlabel=names[5])
sns.distplot(data[:,6], axlabel=names[6])
sns.distplot(data[:,7], axlabel=names[7])
```

It appears that there may be a couple of outliers on the high end of the some of the features like avg Bedrooms and avg Occu that are causing those distributions to look strange.

2. Below is the visualization of the dependency of each target on each feature along with a code snippet from this section:




```
print("")
print("Dependency on Targets: ")
clf = GradientBoostingRegressor(n_estimators=100, max_depth=4, learning_rate=0.1, loss='huber',random_state=1)
clf.fit(data,target)
feat = [0,1,2,3,4,5,6,7]
fig, axs = plot_partial_dependence(clf, data, feat, feature_names=names,n_jobs=3, grid_resolution=50)
fig.suptitle('Dependence of the target on each feature: ')
plt.subplots_adjust(top=0.9, wspace=0.6, hspace=0.6)
plt.show()
```

3. Below is the results of OLS, Ridge, Lasso and ElasticNet using cross-validation w/ default parameters (before and after standardization):

```
Linear Score:  0.6191401072834057
Ridge Score:  0.6190385649168604
Lasso Score:  0.601369228637936
ElasticNet Score:  0.5974164143675413

With Standardization:
Linear Score:  0.6020418270838188
Ridge Score:  0.601969027334209
Lasso Score:  0.6015726438083303
ElasticNet Score:  0.6015459536348887
```

It doesn't appear that standardization helped very much as it actually seemed to have lower scores then before.

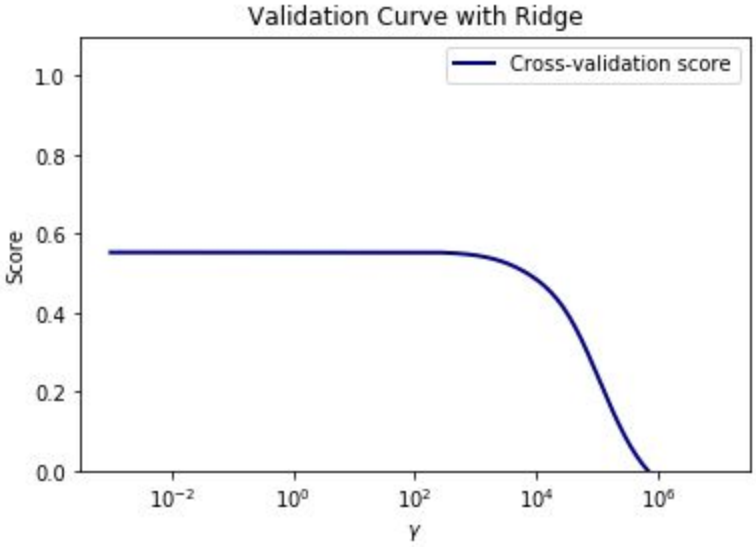
Here is some code from this question:

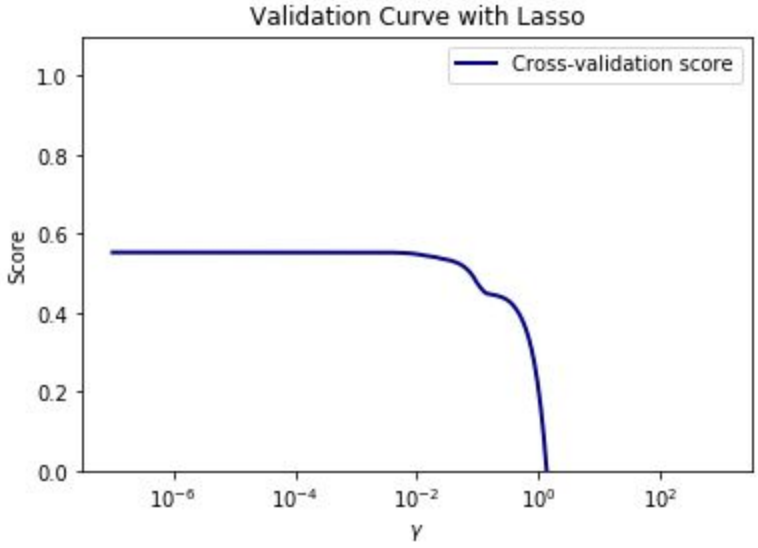
```
#Linear regression
lin = LinearRegression().fit(X_train, y_train)
print("Linear Score: ", lin.score(X_test, y_test))

#Ridge regression w/ CV
rid = RidgeCV().fit(X_train, y_train)
print("Ridge Score: ", rid.score(X_test, y_test))

#Lasso regression w/ CV
lasso = LassoCV().fit(X_train, y_train)
print("Lasso Score: ", lasso.score(X_test, y_test))

#Elastic Net regression w/ CV
ela = ElasticNetCV().fit(X_train, y_train)
print("ElasticNet Score: ", ela.score(X_test, y_test))

#Using StandardScaler
scaler = StandardScaler()
dataSTD = scaler.fit_transform(data, target)
X_train2, X_test2, y_train2, y_test2 = train_test_split(dataSTD, target)
print("")
print("With Standardization:")

#linear regression STD
lin = LinearRegression().fit(X_train2, y_train2)
print("Linear Score: ", lin.score(X_test2, y_test2))

#Ridge regression w/ CV STD
rid = RidgeCV().fit(X_train2, y_train2)
print("Ridge Score: ", rid.score(X_test2, y_test2))

#Lasso regression w/ CV STD
lasso = LassoCV().fit(X_train2, y_train2)
print("Lasso Score: ", lasso.score(X_test2, y_test2))

#Elastic Net regression w/ CV STD
ela = ElasticNetCV().fit(X_train2, y_train2)
print("ElasticNet Score: ", ela.score(X_test2, y_test2))
```

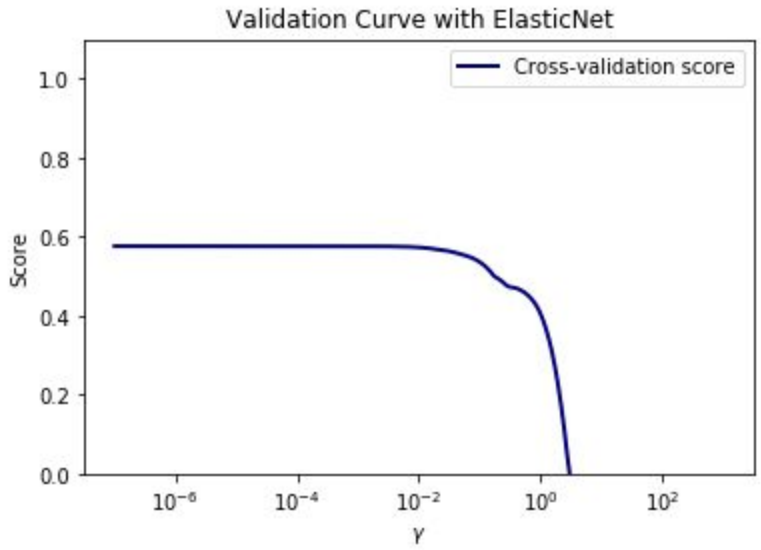
4. Below are the results of tuning the parameters of the models using GridSearchCV. It appears that the tuning only helped slightly as the default parameters seem to be fairly good by themselves.

```
Linear Score:  0.602645758457617
Ridge Score:  0.6026256092718968
Lasso Score:  0.5859636967061526
ElasticNet Score:  0.582975116506645

With Standardization:
Linear Score:  0.5967245738637399
Ridge Score:  0.5966566626670899
Lasso Score:  0.5958451021464822
ElasticNet Score:  0.5960568603497061

Ridge Score(w/ best parameters):  0.6025778930039402
Lasso Score(w/ best parameters):  0.6018449212835475
ElasticNet Score(w/ best parameters):  0.582975116506645
```

Here are the visualizations of the dependence of the validation score with varying levels of alpha (main parameter to change):



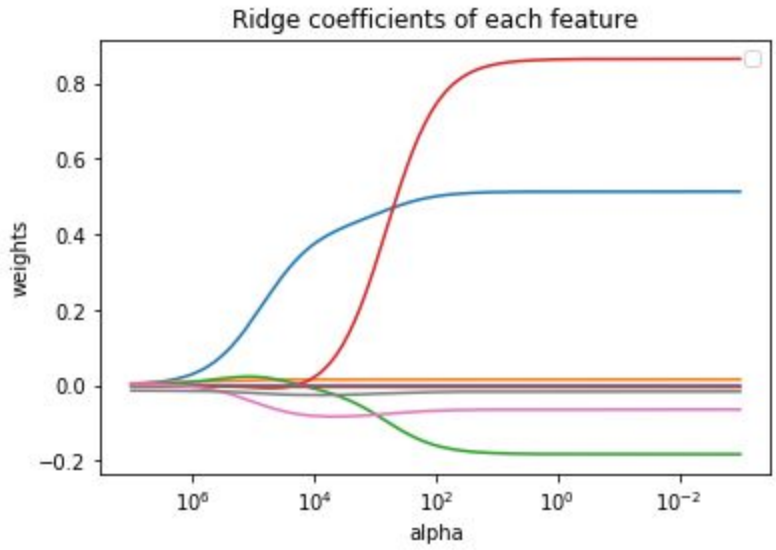


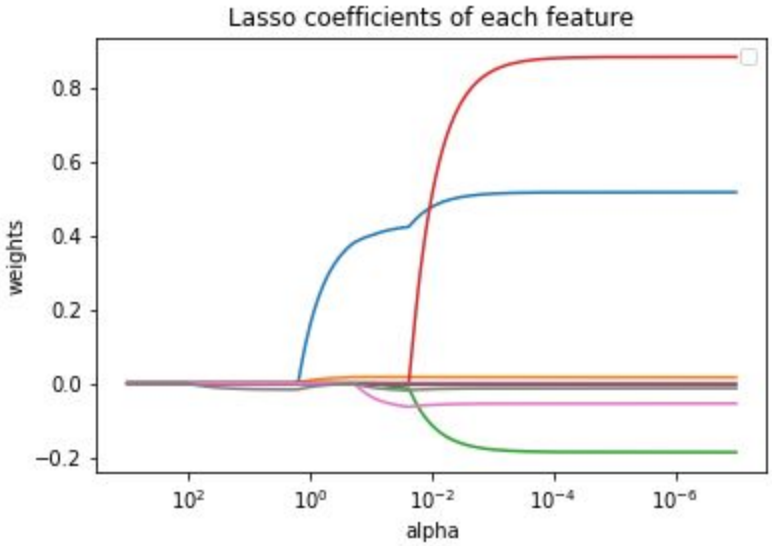


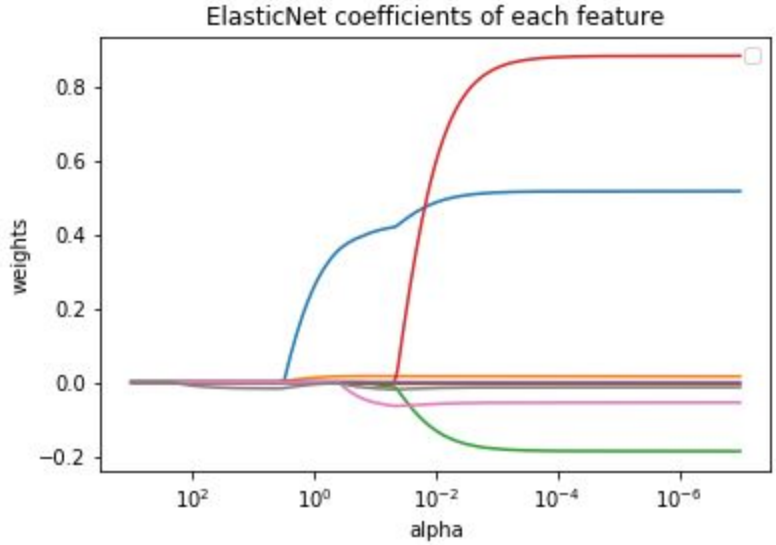

```
gsCVR =  GridSearchCV(estimator, paramsR)
param_range = np.logspace(-7,3,3)
train_scores, test_scores = validation_curve(Ridge(), data, target, "alpha", param_range=param_range, cv=5)
test_scores_mean = np.mean(test_scores, axis=1)

plt.title("Validation Curve with Ridge")
plt.xlabel("$\gamma$")
plt.ylabel("Score")
plt.ylim(0.0, 1.1)
lw = 2
plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.legend(loc="best")
plt.show()
gsCVR.fit(X_train, y_train)
```

5. Legend:
    - Blue = MedInc
    - Orange = HouseAge
    - Green = AveRooms
    - Red = AveBedrms
    - Purple = Population
    - Brown = AveOccup
    - Pink = Latitude
    - Gray = Longitude

    Based on the charts below, it appears that red and blue seem to be the most important. These two are average bedrooms and median income, which seems to make sense based on my knowledge of the housing market. All three methods seem to agree on this.


Ridge coefficients of each feature
weights
alpha
0.8
0.6
0.4
0.2
0.0
−0.2
10^{6}
10^{4}
10^{2}
10^{0}
10^{-2}


Lasso coefficients of each feature
weights
alpha
0.8
0.6
0.4
0.2
0.0
−0.2
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}



Here is the code from one of the visualizations above:

```
alphas = np.logspace(-3, 7, 200)

coefs = []
for a in alphas:
    ridge = Ridge(alpha=a, fit_intercept=False)
    ridge.fit(data, target)
    coefs.append(ridge.coef_)

ax = plt.gca()

ax.plot(alphas, coefs)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients of each feature')
plt.axis('tight')
plt.legend()
plt.show()
```