- **Programmer: Shaun Pritchard**
- **Date: 10-10-2021**
- **Assignment: 1**
- **Prof: Michael DeGiorgio**

2:33

CAP 52625 COMPUTATIONAL FOUDNATIONS OF AI

Ridge Regression using Gradient Descent using Scikit Learn - Assignment 1

Perform a penalized (regularized) least squares fit of a linear model using ridge regression, with the model parameters obtained by batch gradient descent. The tuning parameter will be chosen using five-fold cross validation, and the best-fit model parameters will be inferred on the training dataset conditional on an optimal tuning parameter

Deliverables

- **Deliverable 6** Implement the assignment using statistical or machine learning libraries in a language of your choice.Compare the results with those obtained above, and provide a discussion as to why you believe your results are different if you found them to be different.

> Note: I implemented 2 variations of the cross validation tuning prama error output charts using using the absolute vlaue on the regression errors to check variation as per the asignment.

Deliverable 1-5 located here:
https://colab.research.google.com/gist/shaungt1/83c9e75f7062e34897957859165f3a0d/spritchard_cap5625_programming_assignment-1_10182021.ipynb

---

Set Up and Import Data for proccessing

> Import the credit balance dataset
>
> - N=400 training observations
> - p=9 features

```
#Project Imports
# Data Science libs
from math import sqrt
from scipy import stats
import numpy as np
import pandas as pd
# Graphics libs
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# labries for stat testing
import statsmodels.api as sm
import statsmodels.formula.api as smf
# sklearn liabries for ridge regression and cross vlaidation
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import make_scorer, mean_squared_error
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
# Mount Google Drive for data access
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

```
# Set up dataframe
df = pd.read_csv('/content/drive/MyDrive/Florida_Atlantic_University/Computati
# Set up datafeme for testing
# df = pd.read_csv('/content/Credit_N400_p9.csv')
```

```
# Check the imported data & features
df.head(3)
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Balance
0	14.891	3606	283	2	34	11	Male	No	Yes	333
1	106.025	6645	483	3	82	15	Female	Yes	Yes	903
2	104.593	7075	514	4	71	11	Male	No	No	580

Pre-Proccess Data

- Re-assign categorical featre attributes with bianary dummy variables
- Split real X and y nx1 features

```
#Assign dummy variables to catigorical features with new dataframe
df1 = df.replace({'Male': 0, 'Female':1, 'No': 0, 'Yes': 1})
```

```
# Validate categorical dummy variables
df1.head(3)
```

	Income	Limit	Rating	Cards	Age	Education	Gender	Student	Married	Balance
0	14.891	3606	283	2	34	11	0	0	1	333
1	106.025	6645	483	3	82	15	1	1	1	903
2	104.593	7075	514	4	71	11	0	0	0	580

```
# Seperate real features to variable X
X = df1.iloc[:, :-1].values
```

```
# Seperate real features to variable X
y = df1.iloc[:, -1].values
```

```
# Check X features
print('Matrix shape X := {X}\nValidate array:(row:col)'  .format(X = X.shape),
```

```
Matrix shape X := (400, 9)
Validate array:(row:col)
 [[1.48910e+01 3.60600e+03 2.83000e+02 ... 0.00000e+00 0.00000e+00
  1.00000e+00]
 [1.06025e+02 6.64500e+03 4.83000e+02 ... 1.00000e+00 1.00000e+00
  1.00000e+00]
 [1.04593e+02 7.07500e+03 5.14000e+02 ... 0.00000e+00 0.00000e+00
  0.00000e+00]
 ...
 [5.78720e+01 4.17100e+03 3.21000e+02 ... 1.00000e+00 0.00000e+00
  1.00000e+00]
 [3.77280e+01 2.52500e+03 1.92000e+02 ... 0.00000e+00 0.00000e+00
  1.00000e+00]
 [1.87010e+01 5.52400e+03 4.15000e+02 ... 1.00000e+00 0.00000e+00
  0.00000e+00]]
```

```
# Check y features
print('Matrix shape y := {y}\nValidate array:(row:col)'  .format(y = y.shape),
```

```
Matrix shape:(400,)
Validate array:(row:col)
 [ 333  903  580  964  331 1151  203  872  279 1350 1407    0  204 1081
  148    0    0  368  891 1048   89  968    0  411    0  671  654  467
 1809  915  863    0  526    0    0  419  762 1093  531  344   50 1155
  385  976 1120  997 1241  797    0  902  654  211  607  957    0    0
  379  133  333  531  631  108    0  133    0  602 1388  889  822 1084
  357 1103  663  601  945   29  532  145  391    0  162   99  503    0
    0 1779  815    0  579 1176 1023  812    0  937    0    0 1380  155
  375 1311  298  431 1587 1050  745  210    0    0  227  297   47    0
 1046  768  271  510    0 1341    0    0    0  454  904    0    0    0
 1404    0 1259  255  868    0  912 1018  835    8   75  187    0 1597
 1425  605  669  710   68  642  805    0    0    0  581  534  156    0
    0    0  429 1020  653    0  836    0 1086    0  548  570    0    0
    0 1099    0  283  108  724 1573    0    0  384  453 1237  423  516
  789    0 1448  450  188    0  930  126  538 1687  336 1426    0  802
  749   69    0  571  829 1048    0 1411  456  638    0 1216  230  732
   95  799  308  637  681  246   52  955  195  653 1246 1230 1549  573
  701 1075 1032  482  156 1058  661  657  689    0 1329  191  489  443
   52  163  148    0   16  856    0    0  199    0    0   98    0  132
 1355  218 1048  118    0    0    0 1092  345 1050  465  133  651  549
   15  942    0  772  136  436  728 1255  967  529  209  531  250  269
  541    0 1298  890    0    0    0    0  863  485  159  309  481 1677
    0    0  293  188    0  711  580  172  295  414  905    0   70    0
  681  885 1036  844  823  843 1140  463 1142  136    0    0    5   81
  265 1999  415  732 1361  984  121  846 1054  474  380  182  594  194
  926    0  606 1107  320  426  204  410  633    0  907 1192    0  503
    0  302  583  425  413 1405  962    0  347  611  712  382  710  578
 1243  790 1264  216  345 1208  992    0  840 1003  588 1000  767    0
  717    0  661  849 1352  382    0  905  371    0 1129  806 1393  721
    0    0  734  560  480  138    0  966]
```

Scale, Center and Standardize Data

```
# Implement standardize used for X with SciKit learn
standardize = StandardScaler()
#Implement Center for y (implemented in scikit learn Ridge() method)
y_center = StandardScaler(with_std=False)
```

```
# Apply standardize to X
X = standardize.fit_transform(X)
X = pd.DataFrame(X)
```

```
# Label column headers for X
X.columns=['Income', 'Limit', 'Rating', 'Cards', 'Age', 'Education', 'Gender',
```

```
# Check X column head
print('X standardized feture vector\n', X)
```

```
X standardized feture vector
        Income     Limit    Rating  ...    Gender   Student   Married
0   -0.861583 -0.489999 -0.465539  ... -1.035635 -0.333333  0.795395
1    1.727437  0.828261  0.828703  ...  0.965592  3.000000  0.795395
2    1.686756  1.014787  1.029311  ... -1.035635 -0.333333 -1.257237
3    2.946152  2.068440  2.110003  ...  0.965592 -0.333333 -1.257237
4    0.302928  0.070012  0.013331  ... -1.035635 -0.333333  0.795395
..        ...       ...       ...  ...       ...       ...       ...
395 -0.940986 -0.275711 -0.310230  ... -1.035635 -0.333333  0.795395
396 -0.904963 -0.389362 -0.381413  ... -1.035635 -0.333333 -1.257237
397  0.359462 -0.244913 -0.219633  ...  0.965592 -0.333333  0.795395
398 -0.212808 -0.958916 -1.054419  ... -1.035635 -0.333333  0.795395
399 -0.753345  0.341993  0.388661  ...  0.965592 -0.333333 -1.257237

[400 rows x 9 columns]
```

Evaluate Model with Ridge Regression BGD w/ Scikit Learn

```
# Set local tunning parameters
λ=[1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3, 1e4]
```

```
# Evaluate tuning parameters with ridge regression L2 penalty
βx=[] # set empty list
for lamb in λ:
    # Solver set to auto to best fit data
    ridge=Ridge(alpha=lamb, max_iter=1000)
    # fir the ridge regression coefficients
    ridge.fit(X, y)
    print('X & y features:={}\n'.format(ridge.fit(X, y)))
    βx.append(ridge.coef_)
```

```
X & y features:=Ridge(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, random_state=None, solver='auto', tol=0.001)

X & y features:=Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, random_state=None, solver='auto', tol=0.001)

X & y features:=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, random_state=None, solver='auto', tol=0.001)

X & y features:=Ridge(alpha=10.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, random_state=None, solver='auto', tol=0.001)

X & y features:=Ridge(alpha=100.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, random_state=None, solver='auto', tol=0.001)

X & y features:=Ridge(alpha=1000.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, random_state=None, solver='auto', tol=0.001)

X & y features:=Ridge(alpha=10000.0, copy_X=True, fit_intercept=True, max_iter=1000,
```

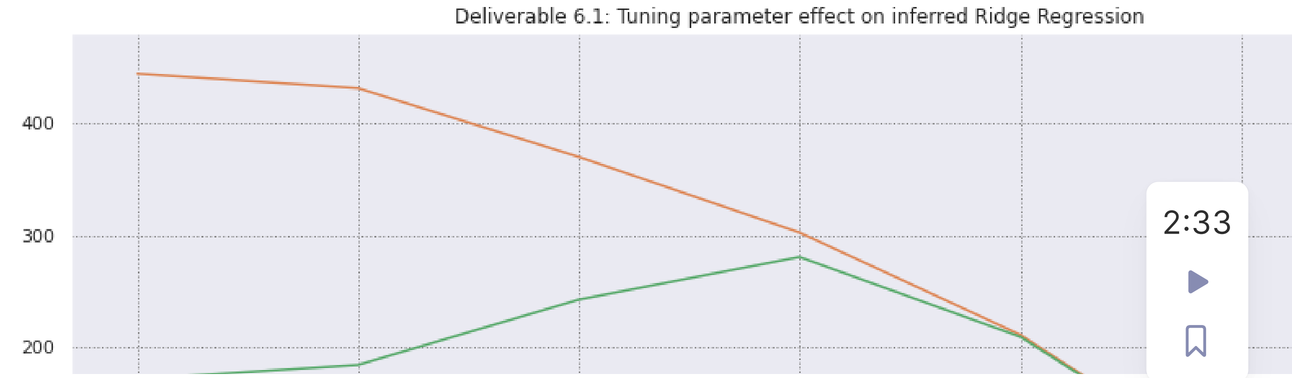
```
normalize=False, random_state=None, solver='auto', tol=0.001)
```

Deliverable 1

- Build graph of dataset N=9 features tuning parameter effect on inferred Ridge regre

```
# Output Deviverable 6.1: inferred tuning parmeters of ridge regresic ng l
sns.set_theme()
sns.set_style("darkgrid", {"grid.color": ".5","image.cmap": "mako", "grid.line
dev1=pd.DataFrame(βx)
dev1.index=λ
dev1.columns=['Income', 'Limit', 'Rating', 'Cards', 'Age', 'Education', 'Gende
plt.rcParams["figure.figsize"] = (16,10)
dev1.plot()
plt.title('Deliverable 6.1: Tuning parameter effect on inferred Ridge Regressi
plt.xlabel('λ Tuning Params')
plt.ylabel('p=9 features')
plt.xscale('log')
plt.legend(loc='lower right', title='Beta_λ')
plt.savefig('SPritchard_CAP5625_Assignment1_Deliverable_6.1_(Using libaries).j
plt.show()

```



(5) K-fold Cross Validation Algorithm w/ Scikit Learn

```
# Evaluate Ridge with CV gridsearch 5 k-folds 1000 iterations
ridge=Ridge(max_iter=1000)
parameters={'alpha': λ}
# Calculate MSE
MSE=make_scorer(mean_squared_error, greater_is_better=False)
# use grid search with 5-k-folds
ridge_regressor=GridSearchCV(ridge, parameters, scoring=MSE, cv=5, refit=False
# Fit ridge regression to cross validation folds
ridge_regressor.fit(X, y)
```

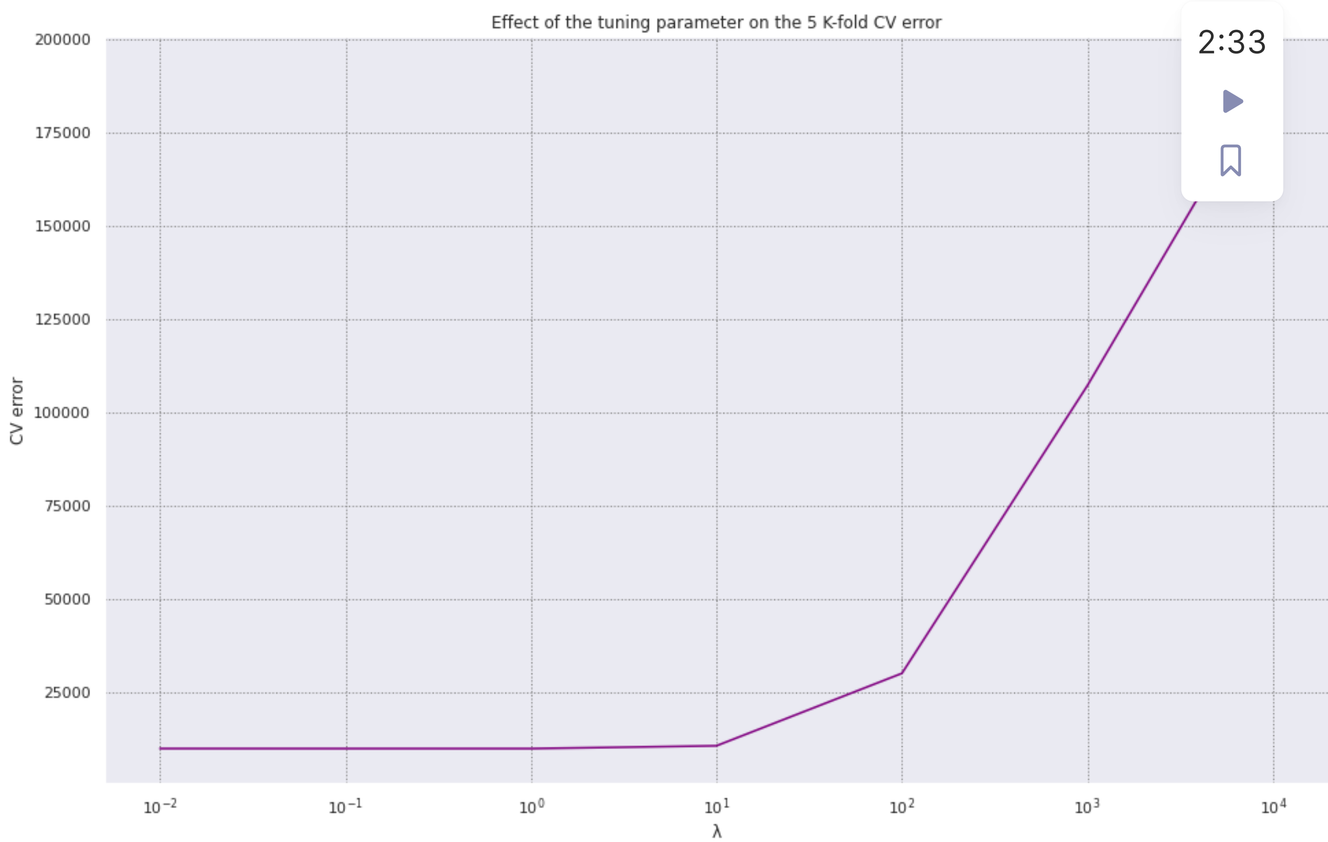
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0,
                                   10000.0]},
             pre_dispatch='2*n_jobs', refit=False, return_train_score=False,
             scoring=make_scorer(mean_squared_error, greater_is_better=False),
             verbose=0)
```

Deliverable 2

Illustrate the effect of the tuning parameter on the cross-validation error

```
# Plot the effect of the tuning parameter on the cross-validation error
sns.set_theme()
sns.set_style("darkgrid", {"grid.color": ".5","image.cmap": "mako", "grid.line
plt.plot(λ, np.absolute(ridge_regressor.cv_results_['mean_test_score']), color
plt.title('Effect of the tuning parameter on the 5 K-fold CV error')
plt.rcParams["figure.figsize"] = (16,10)
plt.xscale('log')
plt.xlabel('λ')
```

```
plt.ylabel('CV error')
plt.savefig('SPritchard_CAP5625_Assignment1_Deliverable_6.2_(Using libaries).j
plt.show()
plt.figure(figsize=(10, 16), dpi=300)
```




```
<Figure size 3000x4800 with 0 Axes>
<Figure size 3000x4800 with 0 Axes>
```

Deliverable 3

Indicate the value of λ that generated the smallest CV(5)error

```
# Output final results of lowest λ tuning param and best score
print("Best CV error of λ", ridge_regressor.best_score_)
print("Best tuning params of λ",ridge_regressor.best_params_)
```

```
Best CV error of λ -10080.822276681745
Best tuning params of λ {'alpha': 1.0}
```

Deliverable 4

- Given the optimal λ, retrain your model on the entire dataset of N = 400 observation provide the estimates of the p = 9 best-fit model parameters.

```
# Retrain model based on λ = 1.0
ridge=Ridge(alpha=1, max_iter=1000)
# Fit data
ridge.fit(X, y)
# Output coefficients based on retrianed model at  λ = 1.0
ridge.coef_
```

```
array([-271.23122766,  369.52441617,  242.60689635,   21.49206153,
        -11.27075014,   -3.04525924,   -5.09410044,  127.07827162,
         -3.97552834])
```

Alternate 2nd CV error Algorthm |X|

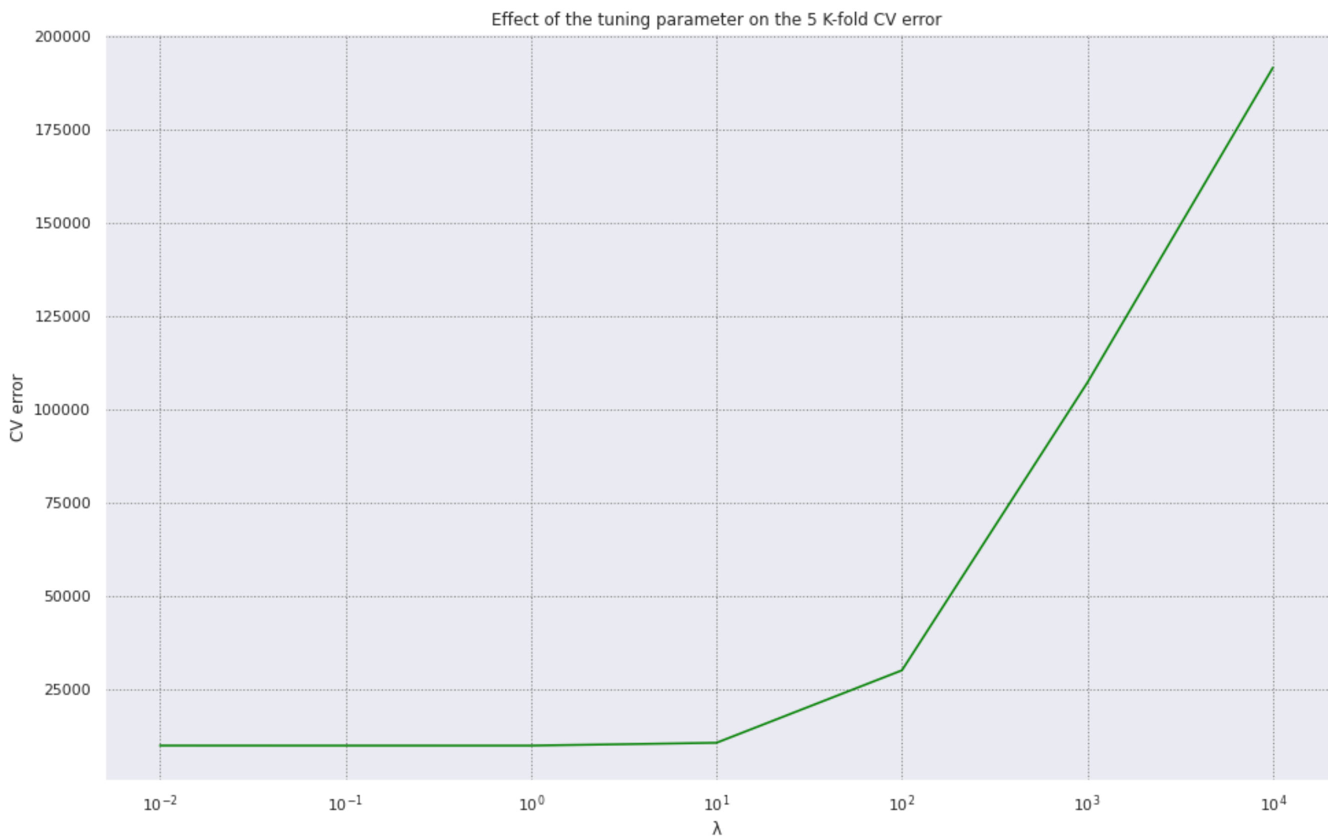
```
#Evaluate Ridge with CV gridsearch 5 k-folds 1000 iterations
# using grid search hyperparameters for ridge regression
# define model find best tuning param errors
model = Ridge()
# define model evaluation method
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
# define grid
grid = dict()
grid['alpha'] = λ # set parementer to 1e-5
# define search
search = GridSearchCV(model, grid, scoring='neg_mean_absolute_error', cv=cv,
# perform the search
results = search.fit(X, y)
# summarize
print('λ that generated the smallest CV(5)error')
print('CV Error: %.3f' % results.best_score_)
print('Best tuning params of λ  : %s' % results.best_params_)
```

```
λ that generated the smallest CV(5)error
CV Error: -80.761
```

```
Best tuning params of λ  : {'alpha': 0.01}
```

```
# Illustrate the effect of the tuning parameter on the cross-validation error
plt.plot(λ, np.absolute(ridge_regressor.cv_results_['mean_test_score']), color
plt.title('Effect of the tuning parameter on the 5 K-fold CV error')
plt.xscale('log')
plt.xlabel('λ')
plt.ylabel('CV error')
plt.savefig('SPritchard_CAP5625_Assignment1_Deliverable_6.3_(Using li[illegible]s).j
plt.show()
plt.figure(figsize=(10, 16), dpi=90)
```

2:33




```
<Figure size 900x1440 with 0 Axes>
<Figure size 900x1440 with 0 Axes>
```

2:33

0s completed at 4:43 PM