Final Project
Regularization & Feature Selection

CS164 - Professor Shekhar, R.

Ash Nguyen

2018/12/06

1 Problem Description

In statistics, whenever we try to perform ordinary least square there is a trade-off between the bias and variance for our model. If our model is highly optimized for the relationships between variables in the training data, it is biased because often we try to optimize certain cost function and therefore must make unverifiable assumptions regarding the relationship between the variables (for example, we assume linearity when using a norm-2 cost function). If our model is highly optimized for the observations in the training data, it has high variance due to sensitivity against noises inherent in the data collection process. This is a trade-off a modeler need to make, and this trade-off is usually the situation worsen with a large number of variable to choose, each of which has different significance to our prediction variable. The question often posed is feature selection: which features should we choose to minimize the cost function without compromising the bias-variance trade-off. This questions is important for at least two reasons:

1. For large dataset, the computational resource required to run sophisticated regressions is huge, therefore we want to identify important variables that contribute the most information to our variable of interest with simple regression first, and this can be done via feature selection.

2. Having a smaller set of features means we can interpret the relationship between the variable of interest and the features easier.

There are multiple ways to do feature selection, but they usually fall into two categories: discrete or continuous. Discrete method uses algorithmic approaches, such as repeatedly using cross-validation to choose a subset of variable (such as forward stepwise selection). Continuous method penalizes different irregularities of the coefficients themselves in the model, and uses optimizers to find the best fit. While discrete method

tends to be susceptible to high variance, therefore might not perform well with new observations, continuous methods is customizable in terms of how much variance we are willing to trade-off. In this project, we will discuss two popular continuous feature selection methods: ridge and lasso, then extend our analysis to regularization.

2 Problem Formulation: Ridge & Lasso

Let us consider a standard regression problem: we have a $n \times k$ matrix A represents the collected n observations on k variables; a $n \times 1$ row vector $\mathbf{y}$ represents the n actual values of the variable of interest. Let $\mathbf{x}$ be a $(k+1) \times 1$ (including the constant) row vector that represents our coefficients and let stack a column vector of n ones onto A for dimensionality consistency (so we have a $n \times (k+1)$ matrix), the regression problem is of following:

Minimize:

$$||A\mathbf{x} - \mathbf{y}||_2$$

This is an unconstrained optimization problem with a closed form solution, but as discussed above is susceptible to overfitting. With the ridge, usually called the shrinkage method, we add a penalization to the norm-2 in terms of coefficients:

Minimize:

$$||A\mathbf{x} - \mathbf{y}||_2 + \lambda \sum_{i=1}^{n} x_i^2 \quad x_i \in \mathbf{x}$$

Notice that we exclude the constant term in the square penalization of coefficients, because including the constant term will make the optimizer mistakenly think that our choice of the variable depends on the coordinate system chosen by the constant. The ridge shrinks the variable toward zero, since the norm-2 prefer very small coefficient to minimize the objective function, therefore we have small coefficients. The ridge penalizes coefficient that is nonzero, therefore effectively retain only the important features (variables) that actually contributes to the inference of our variable of interest. As λ increases, all variable are shrinked toward 0, since we choose to penalize really hard. We can also reformulate the optimization problem to control our penalization more directly by imposing a finite allowed size of the coefficients:

Minimize:

$$||A\mathbf{x} - \mathbf{y}||_2$$

subject to:

$$\sum_{i=1}^{n} x_i^2 \leq t \quad x_i \in \mathbf{x}$$

where t is a chosen parameter.

Similar to the ridge, the lasso penalize the nonzero coefficients by a norm-1:

Minimize:

$$||A\mathbf{x} - \mathbf{y}||_2 + \lambda \sum_{i=1}^{n} |x_i| \quad x_i \in \mathbf{x}$$

and of course, as with the ridge, the lasso can be casted in constrained optimization form similar to the above.

The lasso does not have a closed form solution due to the norm-1, but it is a quadratic program and can be solved easily by an optimizer such as CVXPY. Much like the ridge, the lasso prefer small coefficient, but one important difference is in addition to shrinking coefficient for improved interpretability, the lasso also prefers zero coefficient because of the norm-1, so it chooses a subset of variable that contributes the most to inferences of the variable of interest. Another interesting difference from an optimization perspective is the feasible region (when casted in the constrained form) of the ridge is convex, while the lasso feasible region might not be convex in high dimensions. In practice, the values of λ or t are usually chosen via cross-validation.

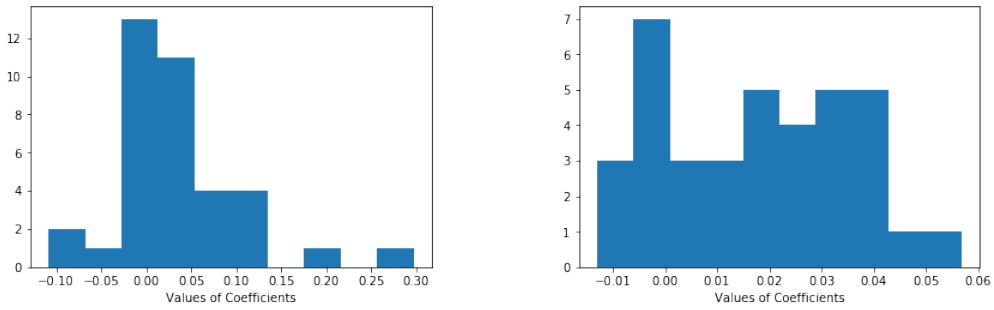
As we discussed in class, the lasso is pretty similar to the $l-1$ norm linear regression: it encourages sparsity of the coefficients. In addition to the feature selection via lasso or ridge, we can add regularization of the data to further reduce variance by imposing an $l-1$ norm or $l-infinity$ norm cost function, as the $l-1$ cost is less sensitive to outliers (and therefore possibly noise) and the $l-infinity$ tends to equalize the residuals of the observations, and can be used to spot important characteristics of the dataset.

3 Problem Analysis: Python Implementation

Implementation of these optimization problem in Python using CVXPY returns interesting results when we play with different values of λ. We used a dataset consisting of 37 numerical variables represents everything from area to basement size of 1460 recorded houses in Ames, Iowa to predict the sale price. Following the standard protocol of analysis, we first impute the missing data with the median of the variable and normalize the values by the mean, then separate the data with the ratio 80/20 for train/test set.

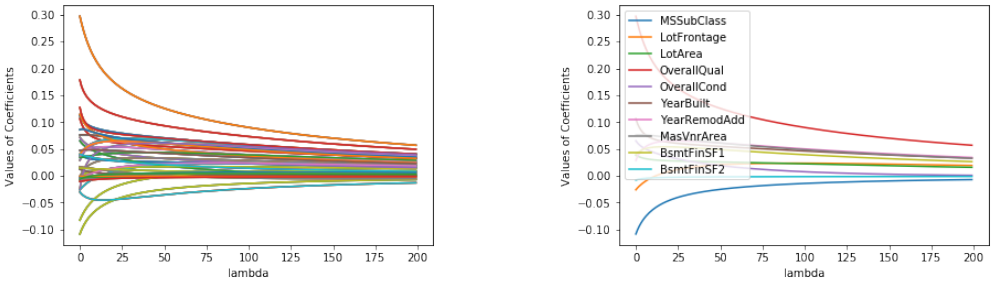
In figure 1, we clearly see the distribution of the coefficients shrinks as λ increases, resulting in a closer centering around 0, which in practice is more interpretable. In figure 2, the values of coefficients tend to shrink toward 0, but don't actually reach 0, as we increase λ, as we expected since the norm-2 encourages shrinkage, not selection.

In figure 3, we observe the selection properties of the lasso: with even small value of λ, we can see the variable quickly converge to 0, leaving a subset of variables. By this graph we can actually get a sense of the importances of each of the variable: there are



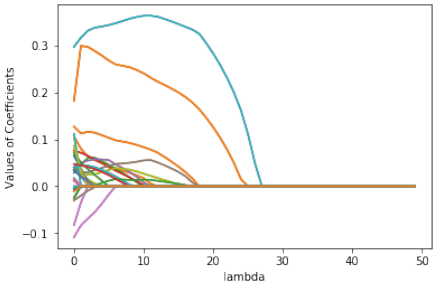
(a) $\lambda = 1$ (b) $\lambda = 200$

Figure 1: Histograms of coefficients values for different λ (Ridge)

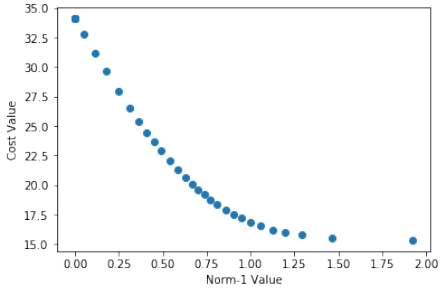


(a) 37 variables (b) 10 variables

Figure 2: Values of different coefficients as λ increases (Ridge)

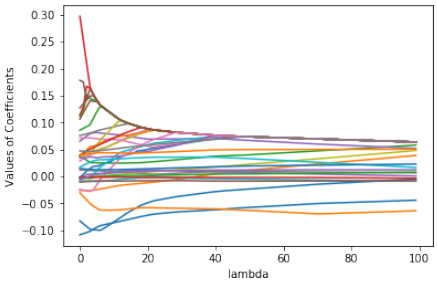


(a) Selection Of Variable By λ

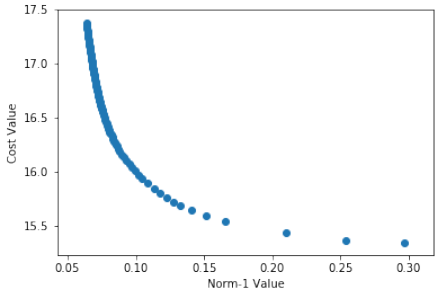


(b) Norm-1 & Actual Cost Trade-off For $\lambda = 0 - 50$

Figure 3: Lasso Descriptive Graphs



(a) Selection Of Variable By λ



(b) Norm-1 & Actual Cost Trade-off For $\lambda = 0 - 100$

Figure 4: Infinity Lasso Descriptive Graphs

certain variables that the optimizer hesitated to omit because it will increase the actual cost of the norm-2 term in the objective function a lot. Variables that slow to be reduced to 0 as λ increases are the ones with the most predictive power. On the right, the actual cost tends to decreases as the norm-1 value increase, which is of course true because as the norm-1 increase, we are allowing the optimizer to uses more variables and therefore can overfit the data: the best value of λ needs to be chosen via cross-validation on the test set, but we can roughly see the spot where the scatterplot is concentrated in the middle, which signifies much changes in the norm-1 without compromising the actual cost, and this is probably the point we need to stop λ from penalizing the coefficients further since it does not bring us much decreases in the actual cost.

An interesting case of the lasso is when we used the infinity norm: penalizing only the maximum coefficient. In figure 4, we can see that the coefficients did not converge to 0 but rather stabilize at a specific range, since we are penalizing the largest coefficient. The trade-off graph also shows tight concentration at the beginning, indicating that this is probably not the best method to do feature selection but might be useful when we want to restrict the coefficient in a specific upper limit.

References

[1] Kaggle. (n.d.). *Ames, Iowa Housing Dataset*. Retrieved from:
https://www.kaggle.com/c/house-prices-advanced-regression-techniques

[2] Stanford University (2006). *Regularization: Ridge Regression and the LASSO*. Retrieved from:
http://statweb.stanford.edu/~tibs/sta305files/Rudyregularization.pdf

[3] Hastie, T., Tibshirani, R., Friedman, J. (2004). *Elements Of Statistical Learning, Chapter 3*.

Appendix

Preparing & Cleaning Data

```
import cvxpy as cvx
import numpy as np
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer, LabelBinarizer, StandardScaler

data=pd.read_csv("/Users/ash/Downloads/train.csv")


def preprocess_num(data):
    num_cova=data.select_dtypes(exclude='object')
    pipe_num=Pipeline([
        ('imputer', Imputer(strategy='mean')),
        ('fscaling',StandardScaler()),
    ])
    cleaned=pd.DataFrame(pipe_num.fit_transform(data[list(num_cova)]),
    columns=list(num_cova))
    pd.DataFrame(cleaned,columns=list(num_cova))
    return pd.DataFrame(cleaned)

data=preprocess_num(data)

def split_train_test(data, test_ratio=0.2):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]

data_train,data_test=split_train_test(data)

Predict_train=data_train['SalePrice']
Var_train=data_train.drop(['SalePrice','Id'],axis=1)
Predict_test=data_test['SalePrice']
Var_test=data_test.drop(['SalePrice','Id'],axis=1)

colnames=Var_train.columns.values
```

Lasso & Ridge Optimization

```
#Pure Ridge
import matplotlib.pyplot as plt
import math

Var_train=np.matrix(Var_train)
Predict_train=np.matrix(Predict_train)
K=Var_train.shape[1]
N=Var_train.shape[0]
ones=np.ones(N)
Q=np.column_stack((Var_train,ones))
```

```
coef=cvx.Variable(K+1,1)
r=cvx.Parameter(sign='positive')
obj=cvx.Minimize(cvx.norm(Q*coef-Predict_train.transpose())+r*cvx.
    sum_entries(coef[0:K]**2,axis=0))
constraints=[]
prob=cvx.Problem(obj,constraints)

def test(coefs,Var_test=Var_test, Predict_test=Predict_test):
    N=Var_test.shape[0]
    Var_test=np.matrix(Var_test)
    ones=np.ones(N)
    Var_test=np.column_stack((Var_test,ones))
    Predict_test=np.matrix(Predict_test)
    cost=math.sqrt((Var_test*coefs-Predict_test.transpose()).transpose()*(
    Var_test*coefs-Predict_test.transpose()))
    return cost

lam=np.arange(0,200,1)
coefs=[]
for i in lam:
    r.value=i
    prob.solve()
    coefs.append(coef.value)

for _ in range(10):
    a=plt.figure(1)
    plt.plot(np.arange(0,200,1),[float(coefs[j][_]) for j in range(200)],
    label=colnames[_])
    plt.xlabel('lambda')
    plt.ylabel('Values of Coefficients')
    plt.legend(loc=0)

x=plt.figure(2)
plt.hist(coefs[0])
plt.xlabel('Values of Coefficients')

y=plt.figure(3)
plt.hist(coefs[199])
plt.xlabel('Values of Coefficients')
plt.show()
```

```
#Pure Lasso
import matplotlib.pyplot as plt
import math


Var_train=np.matrix(Var_train)
Predict_train=np.matrix(Predict_train)
K=Var_train.shape[1]
N=Var_train.shape[0]
ones=np.ones(N)
Q=np.column_stack((Var_train,ones))
coef=cvx.Variable(K+1,1)
r=cvx.Parameter(sign='positive')
```

```
obj=cvx.Minimize(cvx.norm(Q*coef-Predict_train.transpose())+r*cvx.norm(coef[0:K],1))
constraints=[]
prob=cvx.Problem(obj,constraints)

lam=np.arange(0,50,1)
coefs=[]
error=[]
norm1=[]
for i in lam:
    r.value=i
    prob.solve()
    coefs.append(coef.value)
    error.append(cvx.norm(Q*coef-Predict_train.transpose()).value)
    norm1.append(cvx.norm(coef[0:K],1).value)

for _ in range(36):
    a=plt.figure(1)
    plt.plot(np.arange(0,50,1),[float(coefs[j][_]) for j in range(50)],label=colnames[_])
    plt.xlabel('lambda')
    plt.ylabel('Values of Coefficients')

g=plt.figure(2)
plt.scatter(norm1,error)
plt.xlabel('Norm-1 Value')
plt.ylabel('Cost Value')
plt.show()
```

```
#Infinity lasso
import matplotlib.pyplot as plt
import math


Var_train=np.matrix(Var_train)
Predict_train=np.matrix(Predict_train)
K=Var_train.shape[1]
N=Var_train.shape[0]
ones=np.ones(N)
Q=np.column_stack((Var_train,ones))
coef=cvx.Variable(K+1,1)
r=cvx.Parameter(sign='positive')
obj=cvx.Minimize(cvx.norm(Q*coef-Predict_train.transpose())+r*cvx.norm(coef[0:K],'inf'))
constraints=[]
prob=cvx.Problem(obj,constraints)

lam=np.arange(0,100,1)
coefs=[]
error=[]
norm1=[]
for i in lam:
    r.value=i
    prob.solve()
```

```
    coefs.append(coef.value)
    error.append(cvx.norm(Q*coef-Predict_train.transpose()).value)
    norm1.append(cvx.norm(coef[0:K],'inf').value)

for _ in range(36):
    a=plt.figure(1)
    plt.plot(np.arange(0,100,1),[float(coefs[j][_]) for j in range(100)],
    label=colnames[_])
    plt.xlabel('lambda')
    plt.ylabel('Values of Coefficients')

g=plt.figure(2)
plt.scatter(norm1,error)
plt.xlabel('Norm-1 Value')
plt.ylabel('Cost Value')
plt.show()
```

L-1 and L-inf Regularization + Lasso Feature Selection

```
#L-1 objective and lasso

import matplotlib.pyplot as plt
import math


Var_train=np.matrix(Var_train)
Predict_train=np.matrix(Predict_train)
K=Var_train.shape[1]
N=Var_train.shape[0]
ones=np.ones(N)
Q=np.column_stack((Var_train,ones))
coef=cvx.Variable(K+1,1)
r=cvx.Parameter(sign='positive')
obj=cvx.Minimize(cvx.norm(Q*coef-Predict_train.transpose(),1)+r*cvx.norm(
    coef[0:K],1))
constraints=[]
prob=cvx.Problem(obj,constraints)

lam=np.arange(0,100,1)
coefs=[]
error=[]
norm1=[]
for i in lam:
    r.value=i
    prob.solve()
    coefs.append(coef.value)
    error.append(cvx.norm(Q*coef-Predict_train.transpose(),1).value)
    norm1.append(cvx.norm(coef[0:K],1).value)
```

```
#L-inf objective and lasso
import matplotlib.pyplot as plt
import math


```

```
Var_train=np.matrix(Var_train)
Predict_train=np.matrix(Predict_train)
K=Var_train.shape[1]
N=Var_train.shape[0]
ones=np.ones(N)
Q=np.column_stack((Var_train,ones))
coef=cvx.Variable(K+1,1)
r=cvx.Parameter(sign='positive')
obj=cvx.Minimize(cvx.norm(Q*coef-Predict_train.transpose(),'inf')+r*cvx.
    norm(coef[0:K],1))
constraints=[]
prob=cvx.Problem(obj,constraints)

lam=np.arange(0,100,1)
coefs=[]
error=[]
norm1=[]
for i in lam:
    r.value=i
    prob.solve()
    coefs.append(coef.value)
    error.append(cvx.norm(Q*coef-Predict_train.transpose(),'inf').value)
    norm1.append(cvx.norm(coef[0:K],1).value)
```