Guozhen Li | ligz@ucdavis.edu

Table of Contents

STA 208: Homework 1 (Do not distribute)

Due 4/18/2018 in class and online by 10am

Instructions: To make grading easier for the TAs please print out your homework and also submit it on canvas. The canvas should include all of your code either in this notebook file, or a separate python file that is imported and ran in this notebook. We should be able to open this notebook and run everything here by running the cells in sequence. The written portions can be either done in markdown and TeX in new cells or written clearly by hand when you hand it in.

- Code should be well organized and documented
- All math should be clear and make sense sequentially
- When in doubt explain what is going on
- You will be graded on correctness of your math, code efficiency and succinctness, and conclusions and modelling decisions

Exercise 1 (20 pts)

Consider the binary classification setting where the training data is $x_i \in \mathbb{R}^p, y_i \in \{0, 1\}, i = 1, \ldots, n$ and recall the empirical risk (as a function of the predictor $g : \mathbb{R}^p \to \{0, 1\}$),

$$R_n(g) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, g(x_i)),$$

and the true risk is

$$R(g) = \mathbb{E}\ell(Y, g(X))$$

where X, Y are drawn from their joint distribution $f_{X,Y}(x, y)$.

1. Suppose that the loss function is the Hamming loss, $\ell(y, \hat{y}) = 1\{y \neq \hat{y}\}$. What is the predictor that minimizes the True risk? (Hint: derive an expression for this based on the joint distribution - you can use expressions involving conditional probabilities.)
2. The predictor above is called the Bayes rule. For the Hamming loss, write an expression for the true risk of the Bayes rule, this is the irreducible error.
3. Suppose that we know that the Bayes rule can be written in the form,
   $$h(x) = 1\{x_j > 0\},$$
   for some $j = 1, \ldots, p$, but we don't know which one. Now using the empirical risk, what is a reasonable classification algorithm (Hint: you can think of finding a good classifier as a problem of finding j). Describe the fit and predict methods.
4. Suppose that I told you that for any fixed g the following probability bound holds (from Hoeffding's inequality)
   $$\mathbb{P}\left\{|R_n(g) - R(g)| > t\right\} \leq 2\exp\left(-2nt^2\right).$$
   How many samples would I need to get a classifier $\hat{g}$, from the previous question, such that
   $$\mathbb{P}\left\{R(\hat{g}) < R(h) + 0.1\right\} \geq 0.95.$$
   (Hint: use the union bound)

Exercise 1 Answers

Please see `hw1_writeup.pdf`

Exercise 2 (20 pts)

Consider the regression setting in which $x_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$, for $i = 1, \ldots, n$ and $p < n$.

1. For a given regressor, let $\hat{y}_i$ be prediction given x_i, and $\hat{y}$ be the vector form. Show that both linear regression and k-nearest neighbors can be written in the form
$$\hat{y} = Hy,$$
where H is dependent on X (the matrix of where each row is x_i). Give a clear expression for H.
2. By modifying the matrix H, how can we ensure that a given sample is not used as a k nearest neighbor. Derive an expression for the leave-one-out cross validated square error based on this.
3. For linear regression, let $X = UDV^\top$ be the singular value decomposition where U is $n \times p$, and V, D is $p \times p$ (D is diagonal). Derive an expression for the OLS coefficients $\beta = Ab$ such that A is $p \times p$ and depends on V and D, and b is a p vector and does not depend on D. Describe a fit method that precomputes these quantities separately, and describe a predict method.
4. Consider a regressor that performs OLS using the SVD above, but every instance of D will only use the largest r values on the diagonal (all others are set to 0). Call this new matrix D_r ($r < p$). Given that you have computed b already, how could you make a method `change_rank` that recomputes A with D_r instead of D? What is its computational complexity (it should not depend on n)?

Exercise 2 Answers

Please see `hw1_writeup.pdf`

Exercise 3 (30 pts)

We call the method that zeros out all but r singular values in OLS, singular value regression. Implement the singular value regressor as a sklearn style class below. Test it by simulating the training and test data from the `gen_LR_data` function below and calculate the average square error on the test set. Do this for each choice of `r` in the fit function.

In [1]:
```
import numpy as np
from scipy.linalg import svd
# from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler, add_dummy_feature
```

In [2]:
```
class SingularValueRegressor:
    """
    : A regression object that uses the SVD to fit and predict
    : Init: specify rank r
    : .fit: solve the SVD of X and precompute beta
    : .predict: Return y hat for X_test
    """

    def __init__(self, r=None):
        """
        : Store the rank
        """
        self.r = r
        self.b = None
        self.b_r = None
        self.A = None
        self.A_r = None
        self.beta = None
        self.Xscaler = StandardScaler()
        self.design_matrix = None

    def fit(self, X, y):
        """
        : Computes and stores the SVD (use scipy.linalg.svd)
        : Computes beta for rank r singular value regression
        """
        self.design_matrix = add_dummy_feature(self.Xscaler.fit_transfor
m(X))
        if not self.r:
            self.r = self.design_matrix.shape[1]
        U, D, Vh = svd(self.design_matrix)
        self.A = Vh.T.dot(np.diag(np.reciprocal(D)))
        self.b = U.T.dot(y)
        self.change_rank(self.r)
        return self

    def predict(self, X_test):
        """
        : Outputs the predicted y given the predictors X_test
        """
        X_test_scaled = add_dummy_feature(self.Xscaler.transform(X_test
))
        return X_test_scaled.dot(self.beta)

    def change_rank(self, r):
        """
        : Assumes that the SVD has been computed and uses it to change t
he rank; after
        : running the new regressor object should be as if we fit with t
he new rank r
        """
        self.r = r
        self.A_r = self.A[:, :r]  # take first r columns
        self.b_r = self.b[:r]
        self.beta = self.A_r.dot(self.b_r)
        return self
```

In [3]:
```
def gen_LR_data(r = 10, p = 20, n = 100):
    """
    : generate data from an approx low rank regression model
    """
    alpha = np.random.chisquare(1,size = (p,r)) #X coefficients
    Z = np.random.normal(size=(n,r)) #X covariates
    beta = np.random.normal(size = r) #reg covs
    mu = Z @ beta #hidden mean
    Xmean = Z @ alpha.T #random transformation of Z
    X = Xmean + np.random.normal(0,.5,size=(n,p)) #add noise - not exact
ly low rank
    X_test = Xmean + np.random.normal(0,.5,size=(n,p))
    y = mu + np.random.normal(0,1,size=(n))
    y_test = mu + np.random.normal(0,1,size=(n))
    return (X,y,X_test,y_test)
```

In [35]:
```
X,y,X_test,y_test = gen_LR_data()
```

In [36]:
```
for v in ['X', 'y', 'X_test', 'y_test']:
    print(v, 'shape:', eval(v).shape)
```

```
X shape: (100, 20)
y shape: (100,)
X_test shape: (100, 20)
y_test shape: (100,)
```

In [39]:
```
model = SingularValueRegressor().fit(X, y)
```

In [40]:
```
from sklearn.metrics import mean_squared_error
```

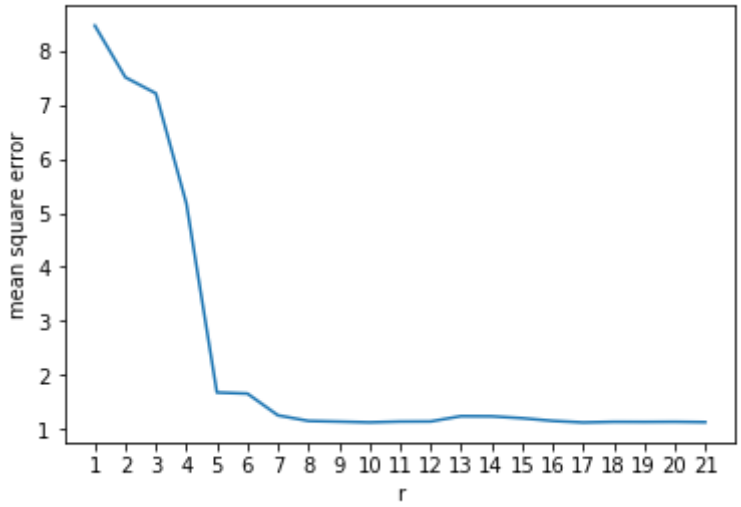
In [41]:
```
r_range = range(1, X.shape[1]+2)
MSE = []

for r in r_range:
    model.change_rank(r)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y, y_pred)
    MSE.append(mse)
```

In [42]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [43]:
```
plt.plot(r_range, MSE, label='MSE')
plt.xticks(r_range)
plt.xlabel('r')
plt.ylabel('mean square error')
```

Out[43]: Text(0,0.5,'mean square error')



Exercise 4 (30 pts)

The dataset in the hw1 directory has a Y variable, 7 predictor variables (X1 - X7). Using sklearn and the class that you constructed above,compare ridge regression, OLS, kNN, and SVRegression.

- Compare and tune the methods using appropriate cross validation.
- Comment on the tuning of each parameter in ridge, kNN, and SVRegression in markdown.
  Be sure to standardize the X variables first and decide how to deal with NAs. Feel free to use the LOO method from lab 1.

In [12]:
```
import pandas as pd
```

In [13]:
```
data_mat = pd.read_csv('hw1_data.csv')
data_mat.head()
```

Out[13]:

	Y	X1	X2	X3	X4	X5	X6	X7
0	18.0	8.0	307.0	130.0	3504.0	12.0	70.0	1.0
1	15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0
2	18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0
3	16.0	8.0	304.0	150.0	3433.0	12.0	70.0	1.0
4	17.0	8.0	302.0	140.0	3449.0	10.5	70.0	1.0

Exercise 4 Answers

Preprocess data

In [14]:
```
data_mat.shape
```

Out[14]:
```
(398, 8)
```

Check missing values

In [15]:
```
data_mat.isnull().sum()
```

Out[15]:
```
Y     0
X1    0
X2    0
X3    6
X4    0
X5    0
X6    0
X7    0
dtype: int64
```

In [16]:
```
data_mat[data_mat.X3.isnull()]
```

Out[16]:

	Y	X1	X2	X3	X4	X5	X6	X7
32	25.0	4.0	98.0	NaN	2046.0	19.0	71.0	1.0
126	21.0	6.0	200.0	NaN	2875.0	17.0	74.0	1.0
330	40.9	4.0	85.0	NaN	1835.0	17.3	80.0	2.0
336	23.6	4.0	140.0	NaN	2905.0	14.3	80.0	1.0
354	34.5	4.0	100.0	NaN	2320.0	15.8	81.0	2.0
374	23.0	4.0	151.0	NaN	3035.0	20.5	82.0	1.0

Drop missing observations

In [17]:
```
data_mat.dropna(inplace=True)
data_mat.isnull().sum()
```

Out[17]:
```
Y     0
X1    0
X2    0
X3    0
X4    0
X5    0
X6    0
X7    0
dtype: int64
```

Dataframe → matrix

In [18]:
```
from sklearn.preprocessing import scale
y = data_mat['Y'].values
X = scale(data_mat.loc[:, 'X1':'X7'].as_matrix())
```

In [19]:
```
display(X.shape)
display(y.shape)
```

```
(392, 7)
```

```
(392,)
```

In [20]:
```
from helper import loo_risk, emp_risk
```

OLS regression

OLS model has no parameter to tune.

In [21]:
```
from sklearn.linear_model import LinearRegression
ols = LinearRegression()
```

In [22]:
```
print('OLS model emperical risk: {}, LOO risk: {}'.format(emp_risk(X, y,
 ols), loo_risk(X, y, ols)))
```

```
OLS model emperical risk: 10.847480945000452, LOO risk: 11.371126332686
616
```

Ridge regression

Ridge regression has 1 parameter, `alpah`, controling the strength of regularization.

`alpha` can be automatically tuned with the `RidgeCV` class provided by Scikit-Learn.

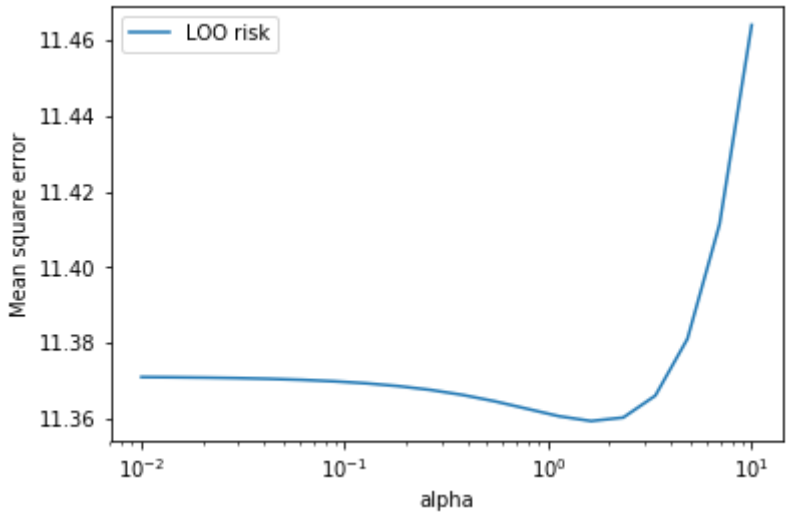
In [23]:
```
alphas = np.logspace(-2, 1, 20)
```

In [24]:
```
from sklearn.linear_model import RidgeCV
ridge = RidgeCV(alphas=alphas, store_cv_values=True)
ridge.fit(X, y)
```

Out[24]:
```
RidgeCV(alphas=array([  0.01   ,   0.01438,   0.02069,   0.02976,   0.04281,   0.06158,
         0.08859,   0.12743,   0.1833 ,   0.26367,   0.37927,   0.54556,
         0.78476,   1.12884,   1.62378,   2.33572,   3.35982,   4.83293,
         6.95193,  10.     ]),
    cv=None, fit_intercept=True, gcv_mode=None, normalize=False,
    scoring=None, store_cv_values=True)
```

In [25]:
```
plt.plot(alphas, np.mean(ridge.cv_values_, axis=0), label='LOO risk')
plt.xscale('log')
plt.legend()
plt.xlabel('alpha')
plt.ylabel('Mean square error')
```

Out[25]:
```
Text(0,0.5,'Mean square error')
```



In [26]:
```
print('Ridge regression best alpha: {}, with LOO risk: {}'.format(ridge.alpha_, np.min(np.mean(ridge.cv_values_, axis=0))))
```

```
Ridge regression best alpha: 1.623776739188721, with LOO risk: 11.359340419579908
```

KNN

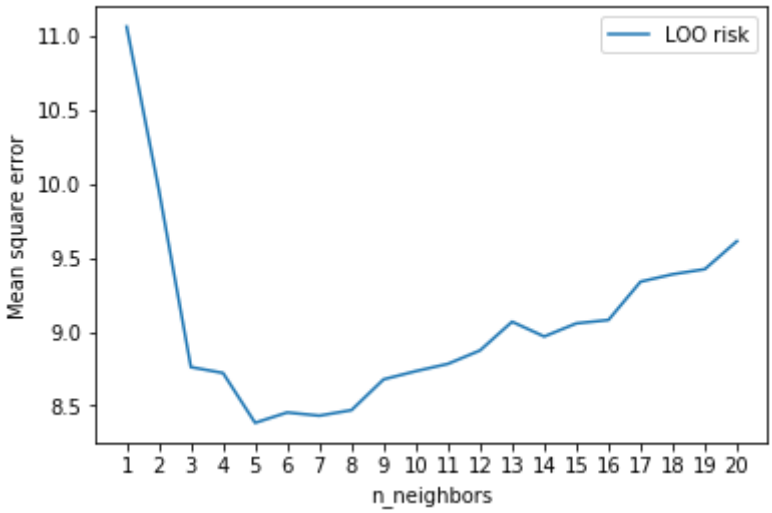
KNN model has 1 parameter, `n_neighbors`

In [27]:
```
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
```

In [28]:
```
ks = range(1, 21)
knn_loo = [loo_risk(X, y, knn.set_params(n_neighbors=k)) for k in ks]
```

In [29]:
```
plt.plot(ks, knn_loo, label='LOO risk')
plt.legend()
plt.xticks(ks)
plt.xlabel('n_neighbors')
plt.ylabel('Mean square error')
```

Out[29]: Text(0,0.5,'Mean square error')



In [30]:
```
best_k = ks[np.argmin(knn_loo)]
knn.set_params(n_neighbors=best_k)
print('KNN model best k: {}, with LOO risk: {}'.format(best_k, np.min(kn
n_loo)))
```

```
KNN model best k: 5, with LOO risk: 8.384319387755102
```

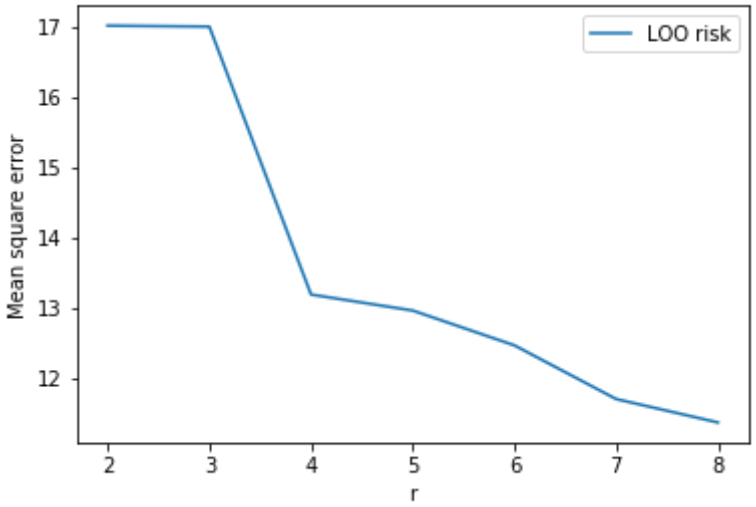
SVRegression

In [31]:
```
svr = SingularValueRegressor().fit(X, y)
svr_loo = []
r_range = list(range(2, X.shape[1]+2))
```

In [32]:
```
for r in r_range:
    svr.change_rank(r)
    svr_loo.append(loo_risk(X, y, svr))
```

In [33]:
```
plt.plot(r_range, svr_loo, label='LOO risk')
plt.legend()
plt.xticks(r_range)
plt.xlabel('r')
plt.ylabel('Mean square error')
```

Out[33]: Text(0,0.5,'Mean square error')



In [34]:
```
best_r = r_range[np.argmin(svr_loo)]
print('SVRegression best r: {}, with LOO risk: {}'.format(best_r, np.min
(svr_loo)))
```

```
SVRegression best r: 8, with LOO risk: 11.371126332686616
```

Conclusions

- OLS, ridge regression, and SVRegression did relatively bad in terms of LOO corss-validation risk. All have LOO risks above 11.
- Ridge regression is slightly better than OLS and SVRegression.
- KNN can reduce LOO risk to below 9, and is the best among all models examined here.