STA 208: Assignment #1

Due on Monday, April 4, 2016

Prof. James Sharpnack MW 12:00 - 2:00 P.M.

Wenhao Wu

Contents

1. (Learning paradyms) **3**

2. (Bayes rule) **3**

3. (Linear Regression) **4**

4. (Simulation and ridge regression.) **5**

5. (Airfoil) **9**

1. (Learning paradyms)

Decribe the issues involved in the following learning problems using the terminology that we learned in the first lecture. Provide a sentence or two for each problem.

(a) A 'smart farm' has distributed sensors that detect moisture levels, and the farmers know what are the ideal moisture levels for each plant. They have many controls that adjust the irrigation system and they would like to know what settings produce the most ideal moisture levels.

Answer: This problem can be studied as a supervised learning problem, more specifically, a regression problem. Given a plant, with p controls to adjust the mirrigation system, the farmers can try N different settings represented by an N-by-p matrix $\mathbf{X}$ (predictors) and use the sensors to record the observed moisture levels $\mathbf{y}$ (response) and use these data to find a relationship between the moisture level and the irrigation system settings $\hat{y}(\mathbf{x})$ and then solve the inverse problem $\hat{y}(\mathbf{x}_{opt}) = y_{opt}$ where y_{opt} is the known optimal moisture level.

(b) Astronomers are trying to map the structure of the universe in terms of how galaxies cluster and form topological structures that they call filaments.

Answer: This problem can be studied as an unsupervised learning problem. The spacial locations of many galaxies are treated as data, and the how galaxies cluster and form topological structures are closely related to the distributions of the data.

(c) An online ad company wants to determine which of many ads to show each user based on their browser cookies.

Answer: This problem can be studied as a semi-supervised learning problem. The predictors are the cookies information for each user and the response is the ads that should be shown to the corresponding user. The response can be collected with a "survey" on a subset of users asking what are the ads they are interested in and how interested they are. However, not all users will answer the survey and no user can provide a comprehensive list of all the ads they are interested in.

(d) NASA is mapping the strength of the gravitational field on the surface of Mars. They want you to help with determining its values in a grid of locations on the surface from remote sensing measurements.

Answer: This problem can be studied as a supervised learning, regression problem. First NASA needs to measure the strength of the gravitational fields (responses) at n locations (predictors) on the mars suface. With this data one can find a relationship between the strength of the gravitational fields and the coordinate of a certain location and make predictions on new locations.

2. (Bayes rule)

Consider the classification setting with features $\mathbf{x} \in \mathbb{R}^p$ and response $y \in \{0, 1\}$. Suppose that we know the joint distribution of $P(x, y)$, and the conditional distributions $P(y|x)$, $P(x|y)$ (an unlikely setting, but bear with me).

(a) Under the Hamming loss, what is the true risk of a classifier $\hat{y} : \mathbb{R}^p \to \{0, 1\}$? Write it in terms of conditional distributions.

continued on next page...

Answer:

$$\begin{aligned} R(\hat{y}) &= \mathbb{E}_{\mathbf{x},y}[1(\hat{y}(\mathbf{x}) \neq y)] \\ &= \mathbb{E}_{\mathbf{x}}[\mathbb{E}_{y|\mathbf{x}}[1(\hat{y}(\mathbf{x}) \neq y)]] \\ &= \mathbb{E}_{\mathbf{x}}[P(\hat{y}(\mathbf{x}) \neq y|\mathbf{x})] \\ &= \int_{\hat{y}(\mathbf{x})=1} P(0|\mathbf{x})p(\mathbf{x})d\mathbf{x} + \int_{\hat{y}(\mathbf{x})=0} P(1|\mathbf{x})p(\mathbf{x})d\mathbf{x} \end{aligned} \tag{1}$$

where $1(\cdot)$ is the indicator function.

(b) What is the Bayes rule, i.e. the classifier that minimizes the true risk?

Answer: We can rewrite the true risk into

$$R(\hat{y}) = \int_{\hat{y}(\mathbf{x})=1} [P(0|\mathbf{x}) - P(1|\mathbf{x})]p(\mathbf{x})d\mathbf{x} + \int P(1|\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{2}$$

Since the second term on the RHS does not depend on $\hat{y}$, to minimize $R(\hat{y})$, the Bayes rule is

$$\hat{y}(\mathbf{x}) = \begin{cases} 1 & \text{if } P(0|\mathbf{x}) < P(1|\mathbf{x}), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

(c) Write in one sentence, what is the Bayes rule, as if you needed to describe what the Bayes risk was to someone in an elevator before you reached the lobby.

Answer: Bayes rule is the predictor for a supervised learning problem that minimizes the true risk, which is the risk taken expectation over the joint distribution of the predictors and responses $P(\mathbf{x}, y)$

(d) Prove that the Bayes risk is $1 - P(y = y^*(\mathbf{x})|\mathbf{x})$ where y^* is the Bayes rule.

Answer: The Bayes risk given $\mathbf{x}$ is

$$R(y^*(\mathbf{x})) = \mathbb{E}_{y|\mathbf{x}}[1(y^*(\mathbf{x}) \neq y)]] = P(y^*(\mathbf{x}) \neq y|\mathbf{x}) = 1 - P(y^*(\mathbf{x}) = y|\mathbf{x}) \tag{4}$$

3. (Linear Regression)

Suppose that we are in the regression setting, $\mathbf{x}_i \in \mathbb{R}^p, y_i \in \mathbb{R}$ are n pairs drawn iid, let $\mathbf{y} = (y_1, \ldots, y_n)$ and $\mathbf{X}^T = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, and consider the linear regression estimator

$$\hat{\boldsymbol{\beta}} := \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2. \tag{5}$$

(a) When is the solution to this program unique? In this case, what is the unique minimizer $\hat{\boldsymbol{\beta}}$?

Answer: The stationary condition to minimize the square-error loss function is

$$\frac{\partial l(\hat{\mathbf{y}}, \mathbf{y})}{\partial \boldsymbol{\beta}} = 2\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta} - \mathbf{y}) = \mathbf{0} \tag{6}$$

i.e. $\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$. The solution to this program is unique iff $\mathbf{X}$ has a full columnn rank of p, or positive definite equivalently, when the unique solution t (6) is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{7}$$

continued on next page...

Since the Hessian matrix $2\mathbf{X}^T\mathbf{X}$ is positive definite in this case, $\hat{\boldsymbol{\beta}}$ indeed minnimizes the square loss function. On the other hand, if $\mathbf{X}$ does not have full column rank, then there exists $\Delta\boldsymbol{\beta} \neq \mathbf{0}$ such that $\mathbf{X}\Delta\boldsymbol{\beta} = \mathbf{0}$. Consequently, if $\hat{\boldsymbol{\beta}}$ is a solution that minimizes (6), then $\hat{\boldsymbol{\beta}} + \Delta\boldsymbol{\beta} \neq \hat{\boldsymbol{\beta}}$ results in exactly the same square loss, therefore the solution can not be unique.

(b) Give an equation that the minimizers satisfy regardless of uniqueness?

Answer: (6) is the 1st-order necessary condition that the minimizers satisfy regardless of uniqueness.

(c) Given a solution to (5), give a reasonable prediction rule $\hat{y} : \mathbb{R}^p \to \mathbb{R}$.

Answer: Assuming the solution to (5) $\hat{\boldsymbol{\beta}}$, a reasonable prediction rule is simply

$$\hat{y}(\mathbf{x}) = \mathbf{x}^T\hat{\boldsymbol{\beta}} \tag{8}$$

(d) Suppose that $\mathbb{E}[y_i|\mathbf{x}_i] = \mathbf{x}_i^T\boldsymbol{\beta}$ for $i = 1, \ldots, n+1$. For a new random draw $(\mathbf{x}_{n+1}, y_{n+1})$, then what is the bias of $\hat{y}(\mathbf{x}_{n+1})$, i.e. $\mathbb{E}[\hat{y}(\mathbf{x}_{n+1}) - y_{n+1}]$?

Answer:

$$\begin{aligned}
\mathbb{E}[\hat{y}(\mathbf{x}_{n+1}) - y_{n+1}] &= \mathbb{E}[\hat{y}(\mathbf{x}_{n+1})] - \mathbb{E}[y_{n+1}] && \text{(9a)}\\
&= \mathbb{E}[\mathbf{x}_{n+1}^T\hat{\beta}] - \mathbf{x}_{n+1}^T\boldsymbol{\beta} && \text{(9b)}\\
&= \mathbf{x}_{n+1}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{y}] - \mathbf{x}_{n+1}^T\boldsymbol{\beta} && \text{(9c)}\\
&= \mathbf{x}_{n+1}^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta}) - \mathbf{x}_{n+1}^T\boldsymbol{\beta} && \text{(9d)}\\
&= 0 && \text{(9e)}
\end{aligned}$$

4. (Simulation and ridge regression.)

(a) Simulate $\{\mathbf{x}_i\}_{i=1}^n \subset \mathbb{R}^p$ with $p = 12$ and $n = 200$, iid normal with mean $\mathbf{0}$ and variance $\boldsymbol{\Sigma}$ such that

$$\Sigma_{j,k} = \rho^{|j-k|},\ j, k = 1, \ldots, p. \tag{10}$$

for $0 < \rho < 1$. Draw $\boldsymbol{\beta} \in \mathbb{R}^p$ such that β_j are iid normal with mean 0 and variance 1, and y_i independently normal with mean $\mathbf{x}_i^T\boldsymbol{\beta}$ and variance 1. Print out your code (not the output), which should consist of functions for generating these objects.

Answer:

```
import numpy as np
from scipy.linalg import toeplitz

p = 12
n = 200
rho = 0.5

X = np.random.multivariate_normal(np.zeros(p, dtype="float64"),\
                                  toeplitz(rho ** np.arange(p)),\
                                  n)
beta = np.random.normal(0, 1, p)
y = np.dot(X, beta) + np.random.normal(0, 1, n)
```

continued on next page...

(b) Derive an analytical expression for the solution to ridge regression,

$$\hat{\boldsymbol{\beta}} := \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \tag{11}$$

as a function of $\mathbf{X}, \mathbf{y}, .$

Answer: The stationary condition for (11) is

$$\frac{\partial(\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2)}{\partial\boldsymbol{\beta}} = 2(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{y} = \mathbf{0} \tag{12}$$

When $\lambda > 0$, $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})$ is always positive-definite. Therefore the solution to ridge regression is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{13}$$

(c) Provide code for solving ridge regression. Use any linear solver you like.

Answer:

```
alpha = 0.5
C = np.dot(X.T, X) + alpha * np.eye(p)
beta_est = np.linalg.solve(C, np.dot(X.T, y))
```

(d) Set $\rho = 0.5$. Simulate the bias of $\hat{\boldsymbol{\beta}}$, $\|\mathbb{E}\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2$, the variance, $\mathbb{E}\|\hat{\boldsymbol{\beta}} - \mathbb{E}\hat{\boldsymbol{\beta}}\|_2^2$, and the mean square error, $\mathbb{E}\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2$, for many values of λ. Be sure that you see instances of overfitting and underfitting and can clearly see the point where λ is optimal. Plot these curves as functions of λ and include your code.

Answer: Since $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{v}$ where $\mathbf{v} \sim \mathcal{N}(0, 1)$. Given a set of predictors $\mathbf{X}$, we have

$$\begin{aligned}
\text{bias}_{\mathbf{X}} &= \mathbb{E}_{\hat{\boldsymbol{\beta}}}\left[\|\mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta}\|_2^2\right] \\
&= \mathbb{E}_{\boldsymbol{\beta}}\left[\|\mathbb{E}_{\mathbf{v}}[(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta} + \mathbf{v})] - \boldsymbol{\beta}\|_2^2\right] \\
&= \mathbb{E}_{\boldsymbol{\beta}}\left[\boldsymbol{\beta}^T\mathbf{A}^T\mathbf{A}\boldsymbol{\beta}\right] \\
&= \|\mathbf{A}\|_F^2
\end{aligned} \tag{14}$$

where $\mathbf{A} = -\lambda(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}$. Similarly, we have

$$\begin{aligned}
\text{var}_{\mathbf{X}} &= \mathbb{E}_{\boldsymbol{\beta}}\mathbb{E}_{\mathbf{v}}\left[\|\hat{\boldsymbol{\beta}} - \mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{\beta}}]\|_2^2\right] \\
&= \mathbb{E}_{\mathbf{v}}\left[\|(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{v}\|_2^2\right] \\
&= \mathbb{E}_{\mathbf{v}}\left[\mathbf{v}^T\mathbf{B}^T\mathbf{B}\mathbf{v}\right] \\
&= \|\mathbf{B}\|_F^2
\end{aligned} \tag{15}$$

where $\mathbf{B} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T$. The mean square error given $\mathbf{X}$ can be simply evaluated as

$$\begin{aligned}
\text{mse}_{\mathbf{X}} &= \mathbb{E}_{\boldsymbol{\beta}}\left[\mathbb{E}_{\mathbf{v}}\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|_2^2\right] \\
&= \mathbb{E}_{\boldsymbol{\beta}}\left[\mathbb{E}_{\mathbf{v}}\|(\hat{\boldsymbol{\beta}} - \mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{\beta}}]) + (\mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta})\|_2^2\right] \\
&= \text{bias}_{\mathbf{X}} + \text{var}_{\mathbf{X}} + 2\mathbb{E}_{\boldsymbol{\beta}}\left[\mathbb{E}_{\mathbf{v}}\left[(\hat{\boldsymbol{\beta}} - \mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{\beta}}])^T(\mathbb{E}_{\mathbf{v}}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta})\right]\right] \\
&= \text{bias}_{\mathbf{X}} + \text{var}_{\mathbf{X}}
\end{aligned} \tag{16}$$

continued on next page...

As a result, the total bias, variance and mean-square error averaged over $\mathbf{X}$ can be evaluated with a Monte-Carlo simulation by generating $n_{\mathbf{X}}$ instances of $\mathbf{X}$ and take the empirical means:

$$\text{bias}_{emp} = \frac{1}{n_{\mathbf{X}}} \sum_{i=1}^{n_{\mathbf{X}}} \text{bias}_{\mathbf{X}_i}, \ \text{var}_{emp} = \frac{1}{n_{\mathbf{X}}} \sum_{i=1}^{n_{\mathbf{X}}} \text{var}_{\mathbf{X}_i} \tag{17}$$

and $\text{mse}_{emp} = \text{bias}_{emp} + \text{var}_{emp}$. The results are shown in Fig. 1. As λ grows, the bias increases while the variance decreases. The minimum MSE is achieved at $\lambda = 1$, which corresponds to the maximization of the posterior distribution of $\boldsymbol{\beta}$ [1, Sec.3.3.1]. The code is listed as follows:

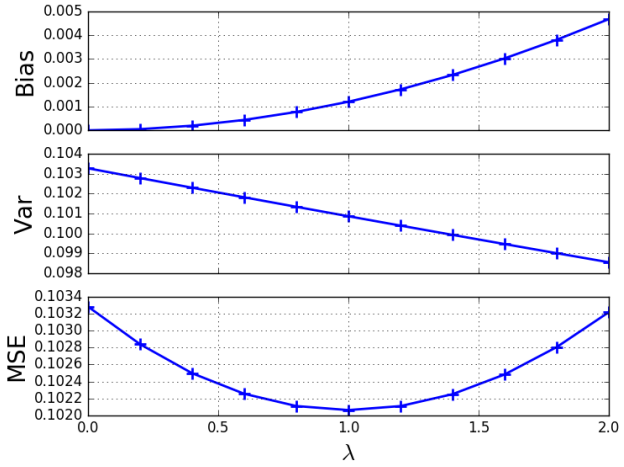

Figure 1: Bias, variance ane MSE for ridge regression.

```
def get_error_cond_X_beta_montecarlo(X, beta, alpha, nrun):

    n, p = X.shape
    y_mean = np.dot(X, beta)
    noise = np.random.normal(0, 1, (nrun, n))

    C = np.dot(X.T, X) + alpha * np.eye(p)
    v = np.dot(X.T, y_mean)
    beta_est = np.empty((nrun, p), dtype="float64")
    for irun in range(nrun):
        beta_est[irun] = np.linalg.solve(C, v + np.dot(X.T, noise[irun]))

    bias = np.linalg.norm(beta_est.mean(axis=0) - beta) ** 2
    var = (np.linalg.norm(beta_est - beta_est.mean(axis=0)) ** 2) / nrun
    mse = bias + var

    return np.array((bias, var, mse))

def get_error_cond_X_montecarlo(X, alpha, n_beta=100, nrun=1000):

    n, p = X.shape
    beta = np.random.normal(0, 1, (n_beta, p))
    error = np.empty([n_beta, 3], dtype="float64")
```

```
    for i_beta in range(n_beta):
        error[i_beta] = get_error_cond_X_beta_montecarlo(X, beta[i_beta],
            ↪ alpha, nrun)

    return error.mean(axis=0)

def get_error_cond_X(X, alpha):
    n, p = X.shape

    XX = np.dot(X.T, X)
    C = XX + alpha * np.eye(p)
    bias = np.linalg.norm(np.linalg.solve(C, XX) - np.eye(p)) ** 2
    var = np.linalg.norm(np.linalg.solve(C, X.T)) ** 2
    mse = bias + var

    return np.array((bias, var, mse))

def get_error(alpha, p = 12, n = 200, rho = 0.5, n_X = 1000, n_montecarlo
    ↪ = None):

    n_alpha = len(alpha)
    error = np.empty([n_alpha, n_X, 3], dtype="float64")
    X = np.random.multivariate_normal(np.zeros(p, dtype="float64"),\
                                      toeplitz(rho ** np.arange(p)),\
                                      (n_X, n))
    if n_montecarlo is None:
        for i_alpha in range(n_alpha):
            for i_X in range(n_X):
                error[i_alpha, i_X] = get_error_cond_X(X[i_X], alpha[
                    ↪ i_alpha])
    else:
        for i_alpha in range(n_alpha):
            for i_X in range(n_X):
                n_beta, nrun = n_montecarlo
                error[i_alpha, i_X] = get_error_cond_X_montecarlo(X[i_X],\
                alpha[i_alpha], n_beta, nrun)

    return error.mean(axis=1)

alpha = np.linspace(0.0, 2.0, 11)
errors = get_error(alpha)
# errors = get_error(alpha, n_X = 100, n_montecarlo=(20, 100))

import matplotlib.pyplot as plt
%matplotlib qt

axis_font = {'size':'20'}
```

```
fig, axs = plt.subplots(3, 1, sharex=True)

axs[0].plot(alpha, errors[:, 0], "+-", lw=2, markersize=10,
    ↪ markeredgewidth=2)
axs[0].set_ylabel("Bias", **axis_font)
axs[0].grid(True)
axs[1].plot(alpha, errors[:, 1], "+-", lw=2, markersize=10,
    ↪ markeredgewidth=2)
axs[1].set_ylabel("Var", **axis_font)
axs[1].grid(True)
axs[2].plot(alpha, errors[:, 2], "+-", lw=2, markersize=10,
    ↪ markeredgewidth=2)
axs[2].set_ylabel("MSE", **axis_font)
axs[2].set_xlabel("$\lambda$", **axis_font)
axs[2].grid(True)
```

5. (Airfoil)

Download the airfoil dataset, which is linked in the homework section of the course site. We will focus on predicting the scaled sound pressure, which is the 6th row.

(a) Set aside a test set at random.

Answer: We randomly set aside $\tilde{2}0\%$ data as the test set. Since some of the algorithm to implement depends on the scale of each predictor (e.g. KNN and kernel smoother), the features are also normalized. The code that imports and preprocesses the data are listed as follows

```
import pandas as pd
import numpy as np
import random
import math

from sklearn import preprocessing
from sklearn import linear_model
from sklearn import neighbors

df = pd.read_csv('airfoil_self_noise.dat', header=None, delimiter=r"\s+")

frac = 0.2 # The portion of data to set aside

n, p = df.shape

row_test= random.sample(range(n), math.floor(frac * n))
row_train = list(set(range(n)) - set(row_test))

X = preprocessing.normalize(df.values[:, :5], axis=0)
y = df.values[:, 5]
X_train, X_test = X[row_train], X[row_test]
y_train, y_test = y[row_train], y[row_test]
```

continued on next page...

(b) Form the coefficients for ordinary least squares with the training set. Write a function with a new $\mathbf{x}$ and $\hat{\boldsymbol{\beta}}$ as arguments and returns the prediction. Use any linear solver/Cholesky decomposition you like.

Answer:

```
regr_lin = linear_model.LinearRegression()
regr_lin.fit(X_train, y_train)

error_lin = ((regr_lin.predict(X_test) - y_test) ** 2).mean()
```

(c) Write a function that takes a new $\mathbf{x}, k$, and the training data, and outputs the k-nearest neighbor prediction with Euclidean distance.

Answer:

```
k_range = np.arange(1, 40, 1, dtype="int32")
n_k = len(k_range)
error_knn = np.empty(n_k, dtype="float64")

regr_knn = neighbors.KNeighborsRegressor()
for i_k, k in enumerate(k_range):
    regr_knn.set_params(n_neighbors=k)
    regr_knn.fit(X_train, y_train)

    error_knn[i_k] = ((regr_knn.predict(X_test) - y_test) ** 2).mean()
```

(d) Write a function that takes a new $\mathbf{x}$, a bandwidth parameter, and the training data, and outputs the kernel prediction with boxcar kernel and Euclidean distance.

Answer: Note that when the bandwidth parameter is so small that there is no samples within the neighborhood, the prediction is made using 1-nearest approach.

```
class KernelSmootherBoxcar:
    def __init__(self, bandwidth = 1.0):
        self.bandwidth = bandwidth

    def set_params(self, **kargs):
        for key in kargs.keys():
            self.key = kargs[key]

    def fit(self, X, y):
        self.X = X
        self.y = y

    def predict(self, X_new):

        m = len(X_new)
        y_new = np.empty(m, dtype="float64")
        for i, x_new in enumerate(X_new):
            distances = np.linalg.norm(x_new - self.X, axis=1)
            neighbors = (distances < bandwidth)
```

continued on next page...

```
            if (sum(neighbors) == 0): # There is no neighbor within the
               ↪ bandwidth, use the nearest neighbor
                y_new[i] = self.y[np.argmin(distances)]
            else: # Average among the neighbors
                y_new[i] = self.y[neighbors].mean(axis=0)

        return y_new

bandwidth_range = np.arange(0.0, 0.02, 0.0004)
n_bandwidth = len(bandwidth_range)
error_ksb = np.empty(n_bandwidth, dtype="float64")

regr_ksb = KernelSmootherBoxcar()
for i_bandwidth, bandwidth in enumerate(bandwidth_range):
    regr_ksb.set_params(bandwidth=bandwidth)
    regr_ksb.fit(X_train, y_train)

    error_ksb[i_bandwidth] = ((regr_ksb.predict(X_test) - y_test) ** 2).
       ↪ mean()
```

(e) Evaluate your methods on the test set, calculating the test error (empirical risk on the test set). Vary the tuning parameters and plot the test error as a function of the tuning parameters.

Answer: The results are shown in Fig 2, where Lin. Reg. represents ordinary least squares (linear regression), KNN represents k-nearest neighbor, and KSB represents kernel smoother using boxcar kernel.

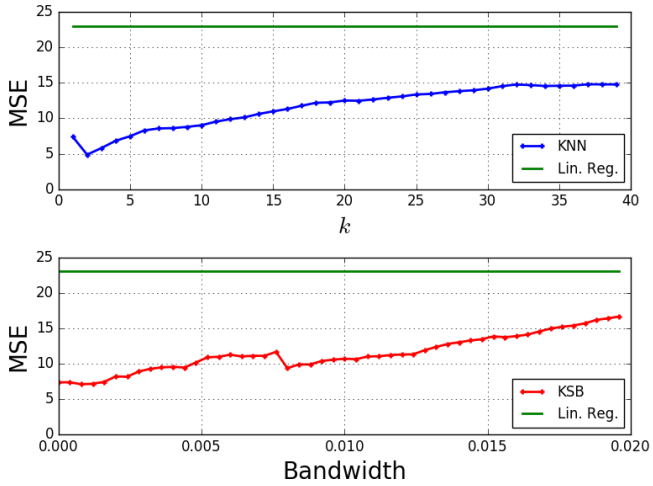

Figure 2: The comparison of MSE among least square regression, k-nearest neighbor and kernel smoother with boxcar kernel.

(f) Is the best test error a good estimate of the true risk for these methods? Why/why not? What can be done to estimate the true risk?

Answer: The best test error may not be a good estimate of the true risk since the data are not made

continued on next page...

full use of. A better approach to estimate the true risk is the K-fold cross-validation estimate or bootstrap method as described in [2, Sec. 7.10, 7.11].

References

[1] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. *The elements of statistical learning: data mining, inference and prediction*. Springer, second edition, 2008.