ps1_problem1

April 20, 2020

0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

0.1.1 PS1, Problem 1: K-NN and Linear Regression for Regression

Name: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```
import numpy as np
import numpy.matlib

def load_data(filename):
    """
    filename - filename to open and load

    Returns file as matrix where last column is
    y_i and columns up to last one is x_i
    """
    res = np.loadtxt(open(filename, "rb"), delimiter=",", skiprows=1)
    return res

def average_error(ys, y_preds):
    """
    ys - vector of real outputs
    y_preds - vector of predicted outputs

    Returns L2 loss between vectors
    """
    return np.mean((ys - y_preds)**2)
```

- Part A

[2]:
```
def knn_regression(K, D, X):
    """
```

```python
    K - number of neighbors
    D - training data consisting of pairs of p-dimensional vectors and outputs
    X - a column p-vector that represents a new input

    Returns the K-NN regression of X using D
    """
    train_x = D[:,:-1]
    train_y = D[:,-1]

    # find distances to X and sort points in D by that
    dists = np.sqrt(np.sum((train_x - np.matlib.repmat(X, len(train_x), 1))**2,␣
↪axis=1))
    inds = dists.argsort()

    # return the mean of the outputs of the first K observations
    return np.mean(train_y[inds][:K])
```

- Part B

[3]:
```python
def linreg_regression(D, X):
    """
    D - training data consisting of pairs of p-dimensional vectors and output
    X - a column p-vector that represents a new input

    Returns the linear regression of X using D
    """

    x = D[:,:-1]
    y = D[:,-1]

    # add bias term to training data
    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    # calculate beta
    intermediate = np.matmul(x.transpose(), x)
    inverse_intermediate = np.linalg.inv(np.array(intermediate))
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    beta = np.matmul(pseudo_x, y)

    # apply beta weight to X
    return np.matmul(np.insert(X, 0, 1), beta)
```

- Part C

[4]:
```python
def knn_vs_linear_reg(train_filename, test_filename, dataset):
    """
```

```python
    train_filename - filename of training data to load
    test_filename - filename of test data to load
    dataset - number of dataset

    Prints Results for KNN vs LinReg
    """
    K = 5
    training_data = load_data(train_filename)
    test_data = load_data(test_filename)

    test_x = test_data[:,:-1]
    test_y = test_data[:,-1]

    # run KNN and Linear Regression on all points in test dataset
    knn = [knn_regression(K, training_data, test_x[i]) for i in␣
↪range(len(test_x))]
    lr = [linreg_regression(training_data, test_x[i]) for i in␣
↪range(len(test_x))]

    # compute the L2 loss of both models
    Err_knn = average_error(test_y, np.array(knn))
    Err_lr = average_error(test_y, np.array(lr))
    R = Err_knn / Err_lr

    print('For dataset {}\n Err_knn is {:1.4f} \n Err_lr is {:1.4f} \n R = {:1.
↪4f}'.format(dataset, Err_knn, Err_lr, R))
    if R > 1:
        print(' Linear regression is better.')
    else:
        print(' k-NN is better.')
```

[5]:

```python
knn_vs_linear_reg('dataset1_train.csv', 'dataset1_test.csv', 1)
print()
knn_vs_linear_reg('dataset2_train.csv', 'dataset2_test.csv', 2)
```

```
For dataset 1
 Err_knn is 0.3416
 Err_lr is 0.0427
 R = 8.0073
 Linear regression is better.

For dataset 2
 Err_knn is 0.4928
 Err_lr is 2.3068
 R = 0.2136
 k-NN is better.
```

2. a) $\mathbb{P}(D \leq d) = 1 - \mathbb{P}(D > d)$

$= 1 - \mathbb{P}(\|x_1\| > d, \ldots, \|x_N\| > d)$

$= 1 - (\mathbb{P}(\|x\| > d))^N$

$= 1 - (1 - d^p)^N$

from lecture 2 pg 8

$\mathbb{P}(\|x\| \leq d) = d^p$

CDF of D is $1-(1-d^p)^N$

PDF of D is $\begin{cases} 0 & , d \leq 0 \text{ and } d > 1 \\ Npd^{(p-1)}(1-d^p)^{N-1} & , 0 < d \leq 1 \end{cases}$

$$\boxed{\mathbb{E}[D] = \int_0^1 Npd^p(1-d^p)^{N-1}}$$ with respect to d

b) Finding $\mathbb{E}[\|pr_a X_i\|^2]$

$\mathbb{E}[\|pr_a X_i\|^2] = \mathbb{E}[\mathbb{E}[\|(a^T X_i)a\|^2 | a]]$

$= \mathbb{E}[\|a\|^2 \mathbb{E}[(a_1X_1 + \ldots + a_pX_p)^2 | a]]$

we know each $X_i \sim N(0,1)$

$= \mathbb{E}[\|a\|^2 \mathbb{E}[a_1^2x_1^2 + \ldots + a_p^2x_p^2 + a_1a_2x_1x_2 + \ldots + a_ia_jx_ix_j + \ldots | a]]$

$\mathbb{V}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2$

$1 = \mathbb{E}[x^2]$

$\mathbb{E}[\mathbb{E}[a^2x^2 | a]]$

$\mathbb{E}[a^2 \mathbb{E}[x^2]]$

$\mathbb{E}[a^2] = a^2$

$\mathbb{E}[\mathbb{E}[a_ia_jx_ix_j | a]]$

$\mathbb{E}[a_ia_j \mathbb{E}[x_i]\mathbb{E}[x_j]]$

$\mathbb{E}[a_ia_j 0]$

0

$= \mathbb{E}[\|a\|^2(a_1^2 + \ldots a_p^2)]$

$= \mathbb{E}[\|a\|^4]$

$= \mathbb{E}[1^4]$

$= 1$

$\|a\| = \left\|\frac{x}{\|x\|}\right\| = \frac{\|x\|}{\|x\|} = 1$

$$\boxed{\mathbb{E}[\|pr_a X_i\|^2] = 1}$$

Scanned with CamScanner

Finding $\mathbb{E}[\|x\|^2]$

$$\mathbb{E}[\|x\|^2] = \mathbb{E}[x_1^2 + x_2^2 + \ldots x_p^2]$$ where $x \sim N(0,1)$

$$= \mathbb{E}[x_1^2] + \ldots + \mathbb{E}[x_p^2]$$ $\mathbb{V}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2$

$$= p\,\mathbb{E}[x^2]$$ $1 = \mathbb{E}[x^2]$

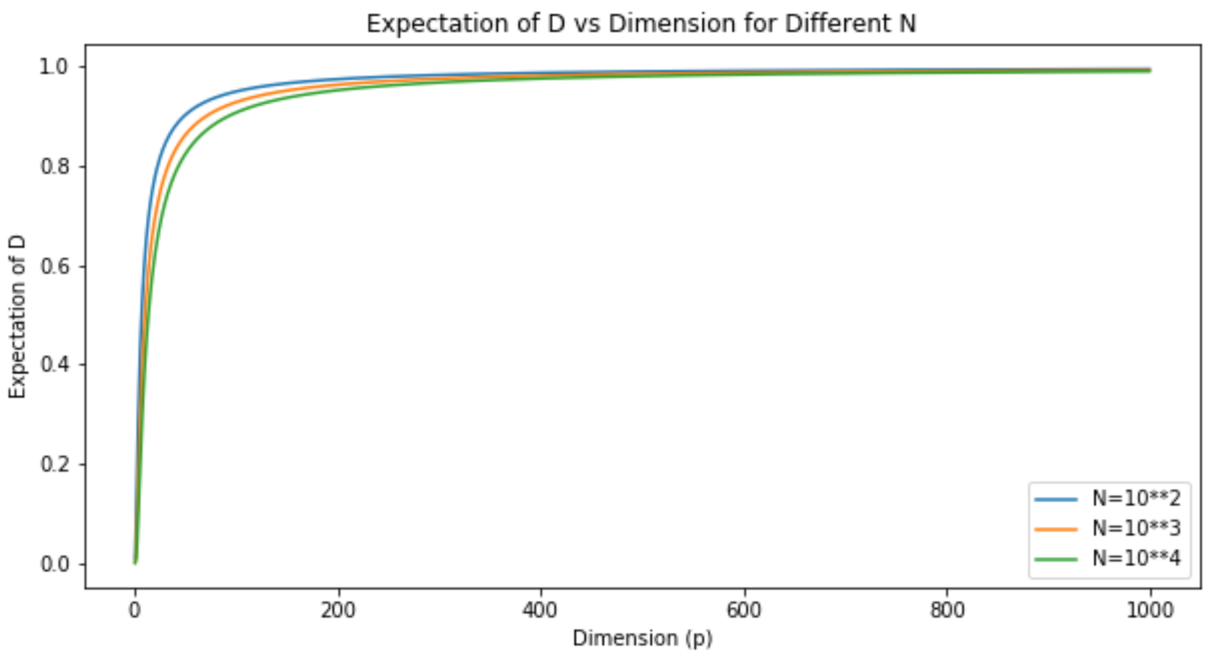
$$= p$$

$$\boxed{\mathbb{E}[\|x\|^2] = p}$$

$\mathbb{E}[\|pr_a X_i\|^2] << \mathbb{E}[\|x\|^2]$ as p is large

$1 << p$

Scanned with CamScanner


Expectation of D vs Dimension for Different N
Expectation of D
Dimension (p)
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000
N=10**2
N=10**3
N=10**4

ps1_problem3

April 20, 2020

0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

0.1.1 PS1, Problem 3: The Curse of Dimensionality: Simulation

Name: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt

%matplotlib inline

def generate_data(p):
    """
    p - dimension of x

    Returns dataset of dimension p according to problem statement
    """
    N = 10**3
    x = np.random.normal(size=(N, p))
    y = np.array([sum(i)+np.random.normal() for i in x])[...,None]
    return np.concatenate((x,y), axis=1)

def average_error(ys, y_preds):
    """
    ys - vector of real outputs
    y_preds - vector of predicted outputs

    Returns L2 loss between vectors
    """
    return np.mean((ys - y_preds)**2)
```

```python
def knn_regression(K, D, X):
    """
    K - number of neighbors
    D - training data consisting of pairs of p-dimensional vectors and outputs
    X - a column p-vector that represents a new input

    Returns the K-NN regression of X using D
    """
    train_x = D[:,:-1]
    train_y = D[:,-1]

    # find distances to X and sort points in D by that
    dists = np.sqrt(np.sum((train_x - np.matlib.repmat(X, len(train_x), 1))**2,␣
 ↪axis=1))
    inds = dists.argsort()

    # return the mean of the outputs of the first K observations
    return np.mean(train_y[inds][:K])

def linreg_regression(D, X):
    """
    D - training data consisting of pairs of p-dimensional vectors and output
    X - a column p-vector that represents a new input

    Returns the linear regression of X using D
    """

    x = D[:,:-1]
    y = D[:,-1]

    # add bias term to training data
    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    # calculate beta
    intermediate = np.matmul(x.transpose(), x)
    inverse_intermediate = np.linalg.inv(np.array(intermediate))
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    beta = np.matmul(pseudo_x, y)

    # apply beta weight to X
    return np.matmul(np.insert(X, 0, 1), beta)

def knn_vs_linear_reg(train_data, test_data):
    """
    train_filename - filename of training data to load
```

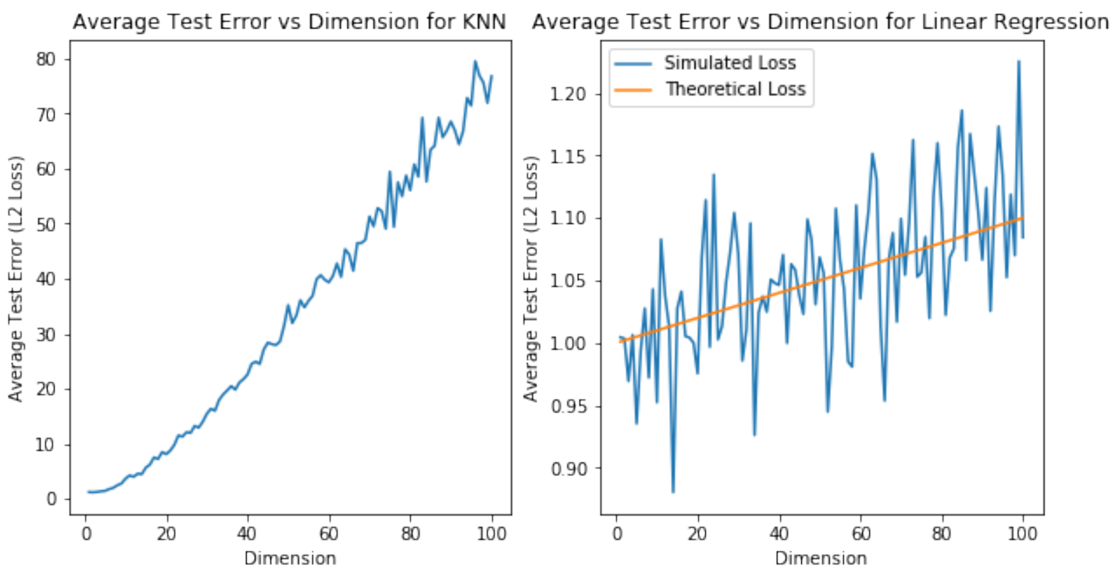
```
    test_filename - filename of test data to load
    dataset - number of dataset

    Prints Results for KNN vs LinReg
    """
    K = 5
    test_x = test_data[:,:-1]
    test_y = test_data[:,-1]

    # run KNN and Linear Regression on all points in test dataset
    knn = [knn_regression(K, train_data, test_x[i]) for i in range(len(test_x))]
    lr = [linreg_regression(train_data, test_x[i]) for i in range(len(test_x))]

    # compute the L2 loss of both models and return
    Err_knn = average_error(test_y, np.array(knn))
    Err_lr = average_error(test_y, np.array(lr))

    return Err_knn, Err_lr
```

- ps1problem3

[8]:
```
lr_errs = []
knn_errs = []

# generate datasets for p dimensions
train_data = [generate_data(p) for p in range(1,101)]
test_data = [generate_data(p) for p in range(1,101)]

# get the L2 loss of both models for every p
for i in range(len(test_data)):
    Err_knn, Err_lr = knn_vs_linear_reg(train_data[i], test_data[i])
    lr_errs.append(Err_lr)
    knn_errs.append(Err_knn)
```

[9]:
```
dims = np.arange(1, 101)
theoretical_linreg_err = [1+(1/10**3)*p for p in dims]

plt.rcParams['figure.figsize'] = [10, 5]
plt.subplot(121)
plt.plot(dims, knn_errs)
plt.xlabel('Dimension')
plt.ylabel('Average Test Error (L2 Loss)')
plt.title('Average Test Error vs Dimension for KNN')

plt.subplot(122)
plt.plot(dims, lr_errs, label='Simulated Loss')
plt.plot(dims, theoretical_linreg_err, label='Theoretical Loss')
```

```
plt.xlabel('Dimension')
plt.ylabel('Average Test Error (L2 Loss)')
plt.title('Average Test Error vs Dimension for Linear Regression')
plt.legend()
plt.show()
```



ps1_problem4

April 20, 2020

0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

0.1.1 PS1, Problem 4: K-NN and Linear Regression for Classification

Name: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```
import numpy as np
import numpy.matlib

def load_data(filename):
    """
    filename - filename to open and load

    Returns file as matrix where last column is
    y_i and columns up to last one is x_i
    """
    res = np.loadtxt(open(filename, "rb"), delimiter=",", skiprows=1)
    return res

def linreg_regression(D, X):
    """
    D - training data consisting of pairs of p-dimensional vectors and output
    X - a column p-vector that represents a new input

    Returns the linear regression of X using D
    """

    x = D[:,:-1]
    y = D[:,-1]

    # add bias term to training data
    bias = np.matlib.repmat(1, len(x), 1)
```

```
    x = np.concatenate((bias, x), axis=1)

    # calculate beta
    intermediate = np.matmul(x.transpose(), x)
    inverse_intermediate = np.linalg.inv(np.array(intermediate))
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    beta = np.matmul(pseudo_x, y)

    # apply beta weight to X
    return np.matmul(np.insert(X, 0, 1), beta)
```

- Part A

[2]:
```
def knn_classification(K, D, X):
    """
    K - number of neighbors
    D - training data consisting of pairs of p-dimensional vectors and outputs
    X - a column p-vector that represents a new input

    Returns the K-NN classification of X using D
    """

    train_x = D[:,:-1]
    train_y = D[:,-1]

    # find distances to X and sort points in D by that
    dists = np.sqrt(np.sum((train_x - np.matlib.repmat(X, len(train_x), 1))**2,␣
↪axis=1))
    inds = dists.argsort()

    # count the occurences of a class within first K and choose the most common␣
↪one
    nearest_neighbors = train_y[inds][:K].tolist()
    return max(nearest_neighbors, key=nearest_neighbors.count)
```

- Part B

[3]:
```
def linreg_classification(D, X):
    """
    D - training data consisting of pairs of p-dimensional vectors and output
    X - a column p-vector that represents a new input

    Returns the linear regression classification of X using D
    """

    if linreg_regression(D, X) < .5:
        return 0
```

```
    else:
        return 1
```

- Part C

I would expect linear regression classification to work better on dataset 3 since it looks mostly linearly separable with some noise. I would expect k-NN with K=1 to work better on dataset 4 since it does not appear to be linearly separable. It would be better in this case to just take the closest point in the training set since it doesn't seem like any line could split the data well.

[4]:
```
def knn_vs_linear_reg(train_filename, test_filename, dataset):
    """
    train_filename - filename of training data to load
    test_filename - filename of test data to load
    dataset - number of dataset

    Prints Results for KNN vs LinReg
    """
    K = 1
    training_data = load_data(train_filename)
    test_data = load_data(test_filename)

    test_x = test_data[:,:-1]
    test_y = test_data[:,-1]

    # run KNN and Linear Regression on all points in test dataset
    knn = [knn_classification(K, training_data, test_x[i]) for i in␣
 ↪range(len(test_x))]
    lr = [linreg_classification(training_data, test_x[i]) for i in␣
 ↪range(len(test_x))]

    # compute the zero-one loss of both models
    Err_knn = np.mean(np.not_equal(test_y, knn))
    Err_lr = np.mean(np.not_equal(test_y, lr))
    R = Err_knn / Err_lr

    print('For dataset {}\n Err_knn is {:1.4f} \n Err_lr is {:1.4f} \n R = {:1.
 ↪4f}'.format(dataset, Err_knn, Err_lr, R))
    if R > 1:
        print(' Linear regression is better.')
    else:
        print(' k-NN is better.')
```

[5]:
```
knn_vs_linear_reg('dataset3_train.csv', 'dataset3_test.csv', 3)
print()
knn_vs_linear_reg('dataset4_train.csv', 'dataset4_test.csv', 4)
```

```
For dataset 3
```

```
 Err_knn is 0.3410
 Err_lr is 0.2470
 R = 1.3806
 Linear regression is better.

For dataset 4
 Err_knn is 0.2040
 Err_lr is 0.2960
 R = 0.6892
 k-NN is better.
```

ps1_problem5

April 20, 2020

0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

0.1.1 PS1, Problem 5: Training vs Testing Error and the Bias-Variance Trade-Off

Name: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt

%matplotlib inline

def generate_data(N):
    """
    N - number of observations to generate

    Returns dataset of size N according to problem statement
    """
    x = np.random.normal(size=(N, 3))
    y = np.array([np.sin(i[0]) + np.exp(i[1])+np.log(np.abs(i[2]))+np.random.
↪normal() for i in x])[...,None]
    return np.concatenate((x,y), axis=1)

def average_error(ys, y_preds):
    """
    ys - vector of real outputs
    y_preds - vector of predicted outputs

    Returns L2 loss between vectors
    """
    return np.mean((ys - y_preds)**2)
```

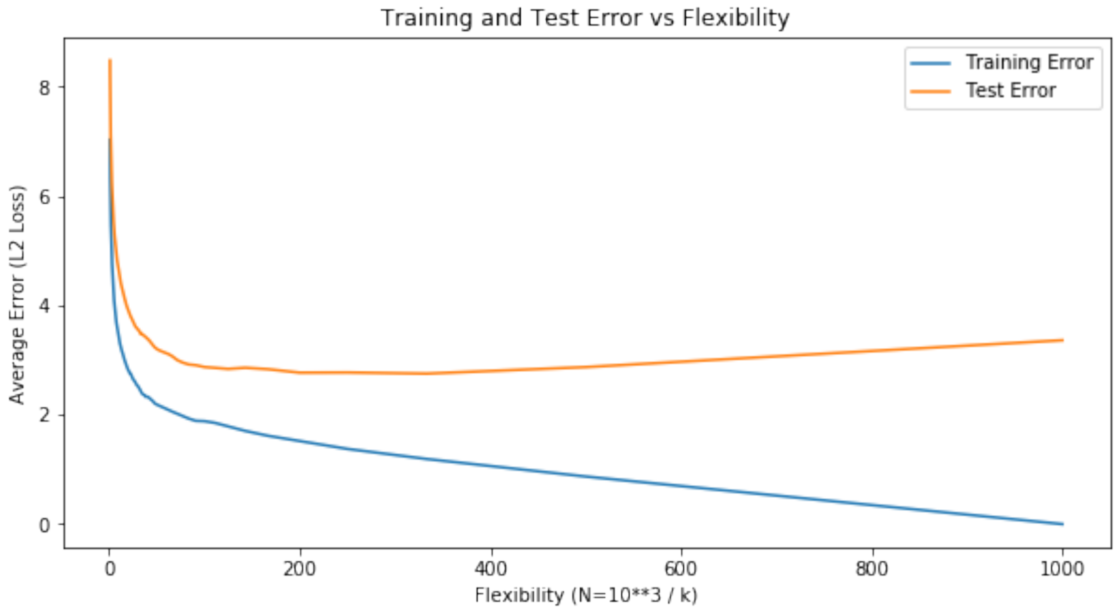
```python
def knn_regression(K, D, X):
    """
    K - number of neighbors
    D - training data consisting of pairs of p-dimensional vectors and outputs
    X - a column p-vector that represents a new input

    Returns the K-NN regression of X using D
    """
    train_x = D[:,:-1]
    train_y = D[:,-1]

    # find distances to X and sort points in D by that
    dists = np.sqrt(np.sum((train_x - np.matlib.repmat(X, len(train_x), 1))**2,␣
 ↪axis=1))
    inds = dists.argsort()

    # return the mean of the outputs of the first K observations
    return np.mean(train_y[inds][:K])
```

- Part A

[8]:
```python
N = K = 10**3
train_data = generate_data(N)
test_data = generate_data(N)

train_err = []
test_err = []

for k in range(1, K+1):
    # run knn on all training and test inputs for specified k
    train_knn = [knn_regression(k, train_data, train_data[i][:-1]) for i in␣
 ↪range(len(train_data))]
    test_knn = [knn_regression(k, train_data, test_data[i][:-1]) for i in␣
 ↪range(len(test_data))]

    # calculate L2 Loss of training and testing predictions
    train_err.append(average_error(train_data[:,-1], train_knn))
    test_err.append(average_error(test_data[:,-1], test_knn))
```

[9]:
```python
dims = np.arange(1, K+1)
x = [N/k for k in dims]

plt.rcParams['figure.figsize'] = [10, 5]
plt.plot(x, train_err, label='Training Error')
plt.plot(x, test_err, label='Test Error')
plt.xlabel('Flexibility (N=10**3 / k)')
plt.ylabel('Average Error (L2 Loss)')
```

```
plt.title('Training and Test Error vs Flexibility')
plt.legend()
plt.show()
```



- Part B

[40]:
```
T = 10**3
K = N = 10**2
tot_err = []
tot_bias = []
tot_var = []

# generate training data
train_data = [generate_data(N) for _ in range(T)]

# generate new input
test_data = np.random.normal(size=3)
f_x = np.sin(test_data[0]) + np.exp(test_data[1])+np.log(np.abs(test_data[2]))

for k in range(1, K+1):
    # run KNN on new input using all datasets
    knn = [knn_regression(k, d, test_data) for d in train_data]

    # create noise and evaluate error
    noise = np.random.normal(size=T)
    errs = (np.tile(f_x, len(knn)) + noise - knn)**2
```

```
    # find average prediction amongst training datasets
    avg_pred = np.mean(knn)

    # calculate squared bias and variance
    bias = (f_x - avg_pred)**2
    var = (knn - np.tile(avg_pred, len(knn)))**2

    tot_err.append(np.mean(errs))
    tot_bias.append(bias)
    tot_var.append(np.mean(var))
```

[41]:
```
dims = np.arange(1, K+1)
x = [N/k for k in dims]
calc_err = [1+tot_bias[i]+tot_var[i] for i in range(len(tot_bias))]

plt.rcParams['figure.figsize'] = [10, 5]
plt.plot(x, tot_err, label='Error')
plt.plot(x, tot_bias, label='Squared Bias')
plt.plot(x, tot_var, label='Variance')
plt.plot(x, calc_err, label='Calculated Error')
plt.xlabel('Flexibility (N=10**2 / k)')
plt.ylabel('Errors')
plt.title('Test Error, Squared Bias, and Variance vs Flexibility')
plt.legend()
plt.show()
```