ISYE 7406 – Homework 1

Shehzad Anwar

August 31, 2025

Introduction

The purpose of this homework assignment is to perform analysis on and compare two methods of analysis, Linear Regression and KNN classification, using the *zipcode* data set. The data set contains 16 x 16 grayscale images of handwritten digits, ranging from 1-9.

The problem that was in this assignment was comparing the performances of the two methods, the error variables, and focusing on two specific digits, ‘2’ and ‘7’, for evaluating the data set.

Exploratory Data Analysis

Filtering the data set to include only the response value digits, 2 and 7, we were able to find that the data set consists of 257 columns, the first one representing the response variable and the rest of the 256 representing their own pixel value. There is a total of 1376 observations in the data set. The total number of response variables that are ‘2’ is 731, 53%, with ‘7’ having 645.

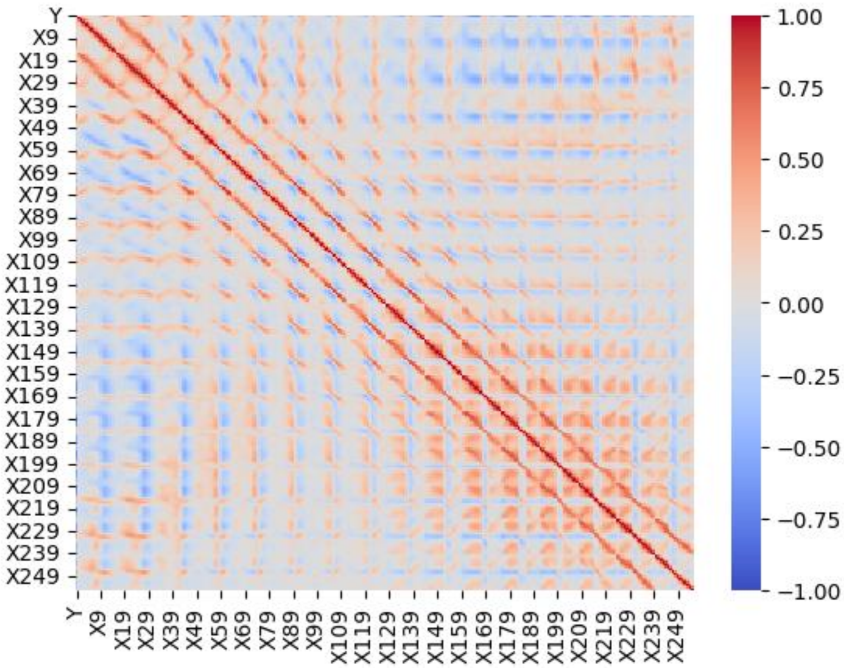


Image 1

Image 1 in the previous page shows a graph, more specifically, a correlation heatmap with the colors inverted to allow for easier differentiation and viewing of the pixels. Red indicates positive correlation, white indicates no correlation and blue indicates negative correlation. Overall there does seem to be a overall high level of correlation with the response variable, both positive and negative. There is high negative correlation in between the first couple of rows and pixels towards the end. It is likely that if the pixels are positively correlated, they are within the vicinity of other pixels like them.

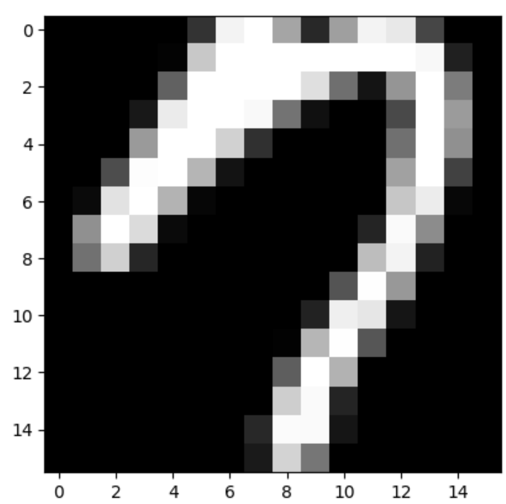
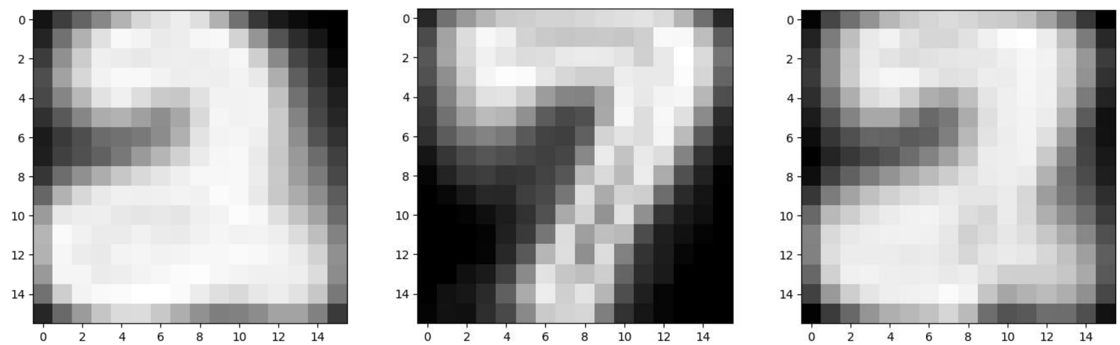


Image 2

Image 2 is what is output when each row of pixels is represented as an image. The grayscale value for each is a number will be between black, -1, and white, +1. Image 2 is representative of Y=7.



The above images represent the average images for each response type. The first image is the average of “2” as the response variable, the second is the average of “7” as the response variable, and the third is the average of the all the observations.

Methodology

There were two types of models that were utilized in this analysis. The first was a linear regression model that was built based on the data. The second was the K Nearest Neigbors (KNN) algorithm. The linear regression model was developed using 2 or 7 as the response variables. The midpoint between the two numbers, 4.5, was set as the boundary value. I was able to calculate the error measurements, i.e. training error, testing error and cross-validation. As for the KNN model, the k values used were (1, 3, 5, 7, 9, 11, 13, 15). The error measurements for this model were also found. The cross-validation process that was used in this analysis was the Monte Carlo validation.

Results

Model	Training Error %	Test Error %
Linear Regression	0.073	1.739
KNN, k=1	0.000	1.739
KNN, k=3	1.017	1.449
KNN, k=5	1.236	1.449
KNN, k=7	1.454	1.739
KNN, k=9	1.599	1.739
KNN, k=11	1.599	1.739
KNN, k=13	1.744	2.029
KNN, k=15	1.744	2.029

The table above shows the training and test error percentages from the different model tests. All the model results show training error percentages less than the test error percentages, but despite that, they are close in value to each other, indicating that the models are accurate. In terms of performance, k=3 and k=5 models were the most accurate, while k=13 and k=15 had the highest training and test error percentages. Utilizing higher k values will cause issues on the edges of the data as the outliers could cause a shift in the readings.

Model	Cross-Valid. Test Error %	Cross-Valid. Test Variance
Linear Regression	1.176	2.6*10^-5
KNN, k=1	1.258	2.5*10^-5
KNN, k=3	1.310	2.8*10^-5
KNN, k=5	1.528	4 *10^-5
KNN, k=7	1.673	4.5 *10^-5
KNN, k=9	1.742	4.8 *10^-5
KNN, k=11	1.855	5.2 *10^-5
KNN, k=13	1.947	5.1 *10^-5
KNN, k=15	1.986	5.6 *10^-5

The table above shows the Monte Carlo cross-validation readings. Among the models, linear regression definitely has the least amount of test error, while also having the second least test variance. As for the KNN models, k=1 is the most accurate, having the least amount of test error percentage as well as the least, out of all the tests, cross-validation test variance. Judging by the numbers, the higher the k value is, the higher the error percentage and variance seems to be.

Conclusion

From the analysis and resulting values from the tests, we can conclude that the linear Regression and KNN are both reliable models of analysis. The linear regression tests performed well when performed on the data set. The KNN model showed that the higher values of k you use, the higher the error percentage potentially, as well as increasing variance, meaning that its less accurate. Its important to note that adding in cross-validation reduced the average test error percentage across the board, apart from the higher k values. In conclusion, using low k parameters will yield the most accurate results in this particular data set.

Appendix

Anwar_HW1_cv.py (used for cross-validation):

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

N_ROUNDS = 100
R_SEED = 7406

def read_data() -> tuple:
    ziptrain = pd.read_csv("ISYE7406/HW1/zip.train.csv", header=None)
    ziptest = pd.read_csv("ISYE7406/HW1/zip.test.csv", header=None)
    col_names = ["Y"]
    col_names.extend([str(i) for i in range(ziptrain.shape[1]-1)])
```

```python
    ziptrain.columns = col_names
    ziptrain = ziptrain[ziptrain["Y"].isin([2,7])]
    ziptest.columns = col_names
    ziptest = ziptest[ziptest["Y"].isin([2,7])]
    return (ziptrain, ziptest)

def monteCarlo(df, ntest=-1, rand=None) -> tuple:
    y = df["Y"]
    x = df.drop(columns=["Y"])
    if ntest == -1:
        ntest = np.floor(df.shape[0] * 0.3)
    split = train_test_split(x, y, test_size=ntest, random_state=rand)
    return split

def LR_error(train_x, test_x, train_y, test_y) -> float:
    model = LinearRegression().fit(train_x, train_y)
    preds = model.predict(test_x)
    preds = 2+5*(preds>=4.5)
    error = np.mean(preds != test_y)
    return error

def knn_error(train_x, test_x, train_y, test_y, k=1) -> float:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(train_x, train_y)
    preds = model.predict(test_x)
    error = np.mean(preds != test_y)
    return error

def main():
    rand = np.random.RandomState(R_SEED)
    train, test = read_data()
    ntest = test.shape[0]
    df = pd.concat([train, test])
    col_names = ["LinearReg", "KNN1", "KNN3", "KNN5", "KNN7", "KNN9", "KNN11",
"KNN13", "KNN15"]
    err_test = pd.DataFrame(None, columns=col_names)

    for i in range(N_ROUNDS):
        current_row = []
        train_x, test_x, train_y, test_y = monteCarlo(df, ntest, rand)
        lin_error = LR_error(train_x, test_x, train_y, test_y)
        current_row.append(lin_error)
        k_choice = list(range(1, 16, 2))
```

```python
        for k in k_choice:
            knn_err = knn_error(train_x, test_x, train_y, test_y, k)
            current_row.append(knn_err)
        current_row = pd.DataFrame([current_row], columns=col_names)
        err_test = pd.concat([err_test, current_row], ignore_index=True)

    summary = err_test.describe()
    summary = summary.apply(lambda x: np.square(x) if x.name == 'std' else x)
    summary = summary.rename(index={"std":"var"})
    print(summary)

if __name__ == "__main__":
    main()
```

Anwar_Shehzad_HW1.ipynb (used for Exploratory Data Analysis, Training and Test Error):

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
```

[80] Python

Read and Train Data

```python
ziptrain = pd.read_csv('zip.train.csv', header=None)
ziptest = pd.read_csv('zip.test.csv', header=None)
col_names = ["Y"]
col_names.extend(["X%d" % i for i in range(ziptrain.shape[1]-1)])
ziptrain.columns = col_names
ziptest.columns = col_names
ziptrain27 = ziptrain[ziptrain["Y"].isin([2, 7])]
ziptest27 = ziptest[ziptest["Y"].isin([2, 7])]
ziptrain27.head()
```

[81] Python

	Y	X0	X1	X2	X3	X4	X5	X6	X7	X8	...	X246	X247	X248	X249	X250	X251	X252	X253	X25
3	7	-1.0	-1.0	-1.0	-1.000	-1.000	-0.273	0.684	0.960	0.450	...	-0.318	1.000	0.536	-0.987	-1.0	-1.0	-1.0	-1.0	-1
10	7	-1.0	-1.0	-1.0	-1.000	-1.000	-0.596	0.912	1.000	0.290	...	-1.000	-0.795	0.663	-0.074	-1.0	-1.0	-1.0	-1.0	-1
14	7	-1.0	-1.0	-1.0	-1.000	-1.000	-1.000	-1.000	-1.000	-0.632	...	-1.000	-0.967	0.866	-0.001	-1.0	-1.0	-1.0	-1.0	-1
15	7	-1.0	-1.0	-1.0	-0.929	0.351	0.798	0.806	0.114	0.015	...	0.835	-0.086	-0.991	-1.000	-1.0	-1.0	-1.0	-1.0	-1
22	7	-1.0	-1.0	-1.0	-1.000	-0.869	0.777	-0.007	-0.697	-1.000	...	-0.933	0.667	-0.315	-1.000	-1.0	-1.0	-1.0	-1.0	-1

5 rows × 257 columns

1. Exploratory Data Analysis of Training Data

```python
summary = ziptrain27.describe()
summary
```

[82] Python

	Y	X0	X1	X2	X3	X4	X5	X6	X7	X8
count	1376.00000	1376.000000	1376.000000	1376.000000	1376.000000	1376.000000	1376.000000	1376.000000	1376.000000	1376.00000
mean	4.34375	-0.984251	-0.922622	-0.837593	-0.707725	-0.516504	-0.337713	-0.215398	-0.177012	-0.23698
std	2.49602	0.105798	0.307493	0.439262	0.549117	0.656989	0.719407	0.749200	0.768320	0.75291
min	2.00000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.00000
25%	2.00000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.00000
50%	2.00000	-1.000000	-1.000000	-1.000000	-1.000000	-0.979000	-0.638000	-0.344000	-0.281500	-0.43200
75%	7.00000	-1.000000	-1.000000	-1.000000	-0.684000	-0.047250	0.280500	0.494500	0.555000	0.49325
max	7.00000	0.412000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

8 rows × 257 columns

```python
ziptrain.shape
```
[83] Python

```
(7291, 257)
```

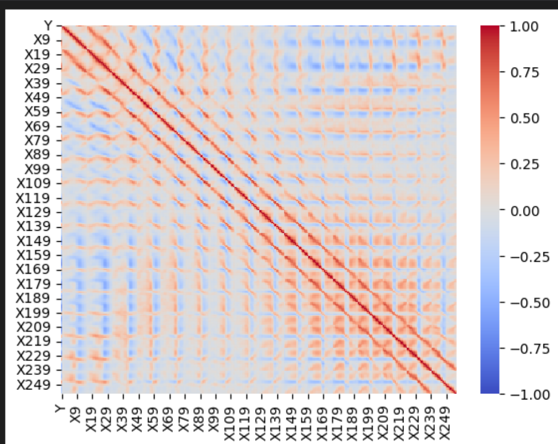
```python
ziptrain[ziptrain["Y"] == 2].shape
```
[84] Python

```
(731, 257)
```

```python
ziptrain[ziptrain["Y"] == 7].shape
```
[85] Python

```
(645, 257)
```

```python
sns.heatmap(ziptrain27.corr(), vmin = -1, vmax = 1, cmap="coolwarm")
plt.show()
```
[86] Python



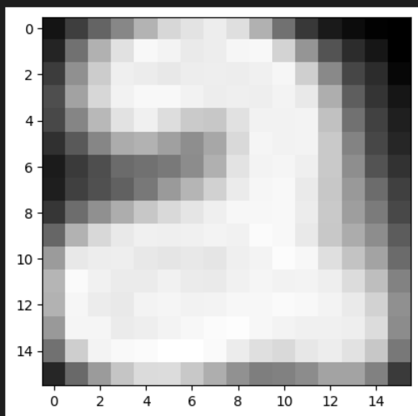

```python
image_row = 1
image_arr = ziptrain27.iloc[image_row, 1:]
image_arr = image_arr.values.reshape((16, 16))
plt.gray()
plt.imshow(image_arr)
print(ziptrain27["Y"].iloc[image_row])
```
[87] Python

```
7
```

```python
image_row = 2
twoSum = ziptrain27[ziptrain27["Y"]== 2].describe()
image_arr3 = twoSum.iloc[image_row, 1:]
image_arr3 = image_arr3.values.reshape((16, 16))
plt.gray()
plt.imshow(image_arr3)
```

[88] Python

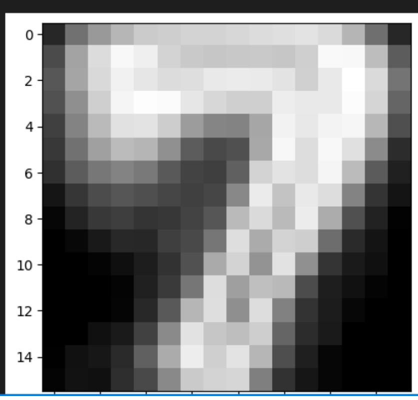
```
<matplotlib.image.AxesImage at 0x21a090b6990>
```




```python
image_row = 2
sevSum = ziptrain27[ziptrain27["Y"]== 7].describe()
image_arr4 = sevSum.iloc[image_row, 1:]
image_arr4 = image_arr4.values.reshape((16, 16))
plt.gray()
plt.imshow(image_arr4)
```

[89] Python

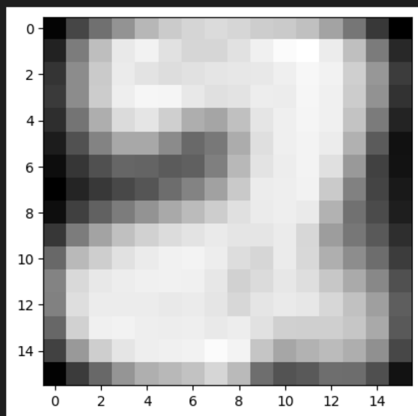
```
<matplotlib.image.AxesImage at 0x21a09846850>
```



```python
image_row = 2
image_arr2 = summary.iloc[image_row, 1:]
image_arr2 = image_arr2.values.reshape((16, 16))
plt.gray()
plt.imshow(image_arr2)
```

[90] Python

```
<matplotlib.image.AxesImage at 0x21a09746490>
```



Training Errors

Linear Regression

```python
y = ziptrain27["Y"]
x = ziptrain27.drop(columns=["Y"])
mod1 = LinearRegression().fit(x,y)
pred1 = mod1.predict(x)
y1pred = 2 + 5 * (pred1 >= 4.5)
train_err = np.mean(y1pred != y)
print("Training Error for Linear Regression = %0.05f"%(train_err))
```

[91] Python

```
Training Error for Linear Regression = 0.00073
```

KNN

```python
kk = list(range(1,16,2))
ypred2 = []
knn_mod = []
for k in kk:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x,y)
    kpred = knn.predict(x)
    t_err = np.mean(kpred != y)
```

[92]

KNN

```python
kk = list(range(1,16,2))
ypred2 = []
knn_mod = []
for k in kk:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x,y)
    kpred = knn.predict(x)
    t_err = np.mean(kpred != y)
    ypred2.append(t_err)
    knn_mod.append(knn)

knn_names = ["KNN%d"%(k) for k in kk]
ypred2 = pd.DataFrame(ypred2, index=knn_names, columns=["Training Error"])
ypred2
```

[92] Python

	Training Error
KNN1	0.000000
KNN3	0.010174
KNN5	0.012355
KNN7	0.014535
KNN9	0.015988
KNN11	0.015988
KNN13	0.017442
KNN15	0.017442

Testing Errors

Linear Regression

```python
test_err_y = ziptest27["Y"]
test_err_x = ziptest27.drop(columns=["Y"])
test_pred1 = mod1.predict(test_err_x)
test_y1pred = 2 + 5 * (test_pred1 >= 4.5)
test_err = np.mean(test_y1pred != test_err_y)
print("Test Error for Linear Regression = %0.05f"%(test_err))
```

[93] Python

```
Test Error for Linear Regression = 0.01739
```

KNN

```python
test_err_knn = []
for knn in knn_mod:
    tknn_pred = knn.predict(test_err_x)
    tknn_err = np.mean(tknn_pred != test_err_y)
    test_err_knn.append(tknn_err)

test_err_knn = pd.DataFrame(test_err_knn, index=knn_names, columns=["Test Error"])
test_err_knn
```

[94] Python

Testing Errors

Linear Regression

```python
test_err_y = ziptest27["Y"]
test_err_x = ziptest27.drop(columns=["Y"])
test_pred1 = mod1.predict(test_err_x)
test_y1pred = 2 + 5 * (test_pred1 >= 4.5)
test_err = np.mean(test_y1pred != test_err_y)
print("Test Error for Linear Regression = %0.05f"%(test_err))
```

[93] Python

```
Test Error for Linear Regression = 0.01739
```

KNN

```python
test_err_knn = []
for knn in knn_mod:
    tknn_pred = knn.predict(test_err_x)
    tknn_err = np.mean(tknn_pred != test_err_y)
    test_err_knn.append(tknn_err)

test_err_knn = pd.DataFrame(test_err_knn, index=knn_names, columns=["Test Error"])
test_err_knn
```

[94] Python

	Test Error
KNN1	0.017391
KNN3	0.014493
KNN5	0.014493
KNN7	0.017391
KNN9	0.017391
KNN11	0.017391
KNN13	0.020290
KNN15	0.020290