Robert Heeter
3 March 2023
ELEC 378

Problem Set #7

Question 1

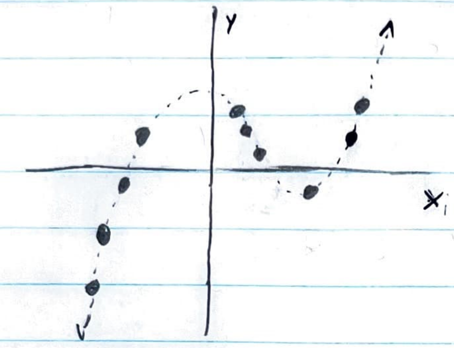
The Crusade Against Multiple Regression Analysis by Richard Nisbett

In this article, Richard Nisbett discusses the pitfalls of “multiple regression analysis,” and how correlational statistical analysis often yields conclusions that are completely unsupported by actual experimental evidence, though are often confused to be well-supported. He offers many examples of situations in which correlation is confused with causation. For example, many people may believe that consuming Vitamin E reduces one’s risk of prostate cancer, while in reality an enormous number of uncontrolled/confounding variables may affect one’s probability of getting prostate cancer, including diet, exercise habits, etc. A second example that Nisbett discusses is the evaluation of car safety; the frequency of deadly accidents between Volvos and Ford F-150 pickups is not necessarily indicative of the models’ safety ratings, since the types of drivers for both vehicles (i.e. more reckless or more careful) also affect the chances of an accident. More broadly, cognitive processes represent a “black box,” and Nisbett warns of the danger of making assumptions and drawing conclusions about cognition purely from observing behavior in correlational studies. In our work with linear regression, we determine the optimal weight parameters that, when applied to a data set, give the most accurate prediction of the actual associated value of each data point. However, the set of computed weighting parameters purely indicates the correlation “strength” of a particular feature on the expected value, not if the feature actually causes (i.e. directly affects) the expected value. In short, multiple linear regression analysis is useful for identifying impactful features, but cannot replace true experimentation for assessing causation.

2. Set of training data $\{x_i \in \mathbb{R}, y_i \in \mathbb{R}\}_{i=1}^{n}$

a. Cubic function: $f(x_i) = ax_i^3 + bx_i^2 + cx_i + d$, where a, b, c, d are constants.

b.



c. $\underset{\text{prediction}}{y} = \underset{\text{data matrix}}{X}\ \underset{\text{weights}}{w} + \underset{\text{error}}{e}$ $\longrightarrow$ goal is to optimize w to minimize strength of error $\|e\|^2$

optimal least-squares predictor w^*

The data matrix has the form: $X = \begin{array}{cccc} & & & \text{column of 1s} \\ \end{array}$ $\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n^3 & x_n^2 & x_n & 1 \end{bmatrix}$, y has form: $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

The weight w have the form: $w = [a, b, c, d]^T$

Find optimizer w: $\min_{w \in \mathbb{R}^2} \|y - Xw\|^2 = \min_{w \in \mathbb{R}^2} \mathcal{L}(w)$ where $\mathcal{L}(w) \triangleq \|y - Xw\|^2$

$$\mathcal{L}(w) = (y - Xw)^T (y - Xw) = y^T y - 2w^T X^T y + w^T X^T X w$$

Minimum where $\nabla \mathcal{L}(w^*) = 0 \longrightarrow \nabla \mathcal{L}(w^*) = -2X^T y + 2X^T X w^* = 0$

$$\therefore X^T y = X^T X w^*$$

$$w^* = \underbrace{(X^T X)^{-1} X^T}_{= X^+ \text{ (moore-penrose pseudoinverse)}} y$$

Given w^*, optimal solution is $\hat{y} = Xw^* = w^*[1]x_i^3 + w^*[2]x_i^2 + w^*[3]x_i + w^*[4]$

for $i = 1 \dots n$

(2)(c) Thus, the optimal least-squares predictor is determined with:

$$w^* = (X^TX)^{-1}X^Ty = X^+y$$, where X and y are the data matrix and vector of training data

d. ① Outliers in the data set can have a significant impact on the accuracy of the fit by skewing the weightings in the w^* vector. Inspecting the data set before running regression can help identify outliers (i.e. graphically or comparing the data point values along one dimension). Outliers can be removed from the data set to improve the prediction accuracy.

② In some cases, the data correlation matrix X^TX is singular or ill-conditioned, in which case $(X^TX)^{-1}$ does not exist, which makes it impossible to determine a unique w using the Moore penrose pseudoinverse (from the normal equations).

A constraint must be added to w to find a unique solution, which can be done using a lagrange multiplier and a "penalty" on w:

Ridge regression: $$\min_{w \in \mathbb{R}^p} \|y - Xw\|_2^2 + \underbrace{\lambda \|w\|_2^2}_{\text{constraint}}$$

2 norm squared penalty

Lasso regression: $$\min_{w \in \mathbb{R}^p} \|y - Xw\|_2^2 + \overbrace{\lambda \|w\|_2}^{\text{constraint}}$$

1 norm penalty

③ Finally, it is important to ensure that the model that is fitted to the data represents the data trend well, since the model type has an impact on the regression accuracy. For example, both of these data sets produce the same least-squares line fits, but a polynomial (probably quadratic) curve would be better suited for the second:

checking the data visually and using intuitive models can help.

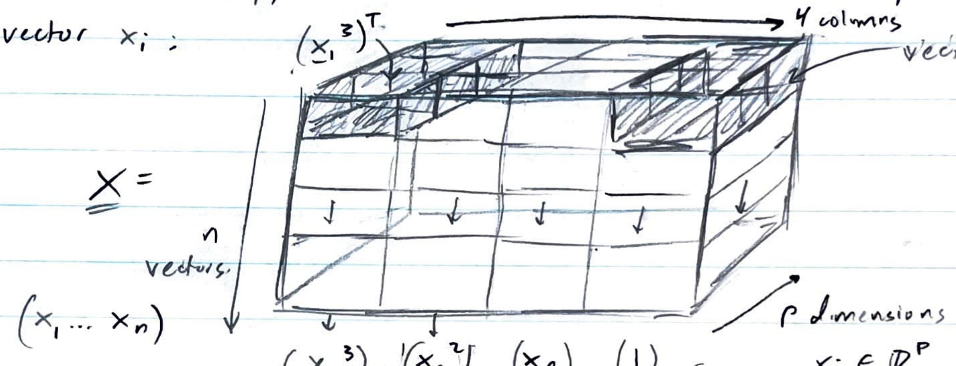
(2).e. The data are now vectors $\underline{x}_i \in \mathbb{R}^p$, $p > 1$

Now, the data matrix has the form : $\underline{\underline{X}} = \begin{array}{cc} & 4 \\ n & \begin{bmatrix} (\underline{x}_1^{\,3})^T & (\underline{x}_1^{\,2})^T & (\underline{x}_1)^T & 1 \\ (\underline{x}_2^{\,3})^T & (\underline{x}_2^{\,2})^T & (\underline{x}_2)^T & 1 \\ \vdots & \vdots & \vdots & \vdots \\ (\underline{x}_n^{\,3})^T & (\underline{x}_n^{\,2})^T & (\underline{x}_n)^T & 1 \end{bmatrix} \end{array}$

Such that $\underline{\underline{X}}$ has dimensions:

n rows × 4 columns × p "layers"

For each dimension in p, an $n \times 4$ matrix is formed with the p^{th} value in each vector x_i :



In this case, if $p > n$ (i.e. the dimension of each vector x_i is greater than the total number of vectors n), then there will be too many dimensions / features and corresponding weight parameters compared to data vectors / samples. As a result, the curve may be overfit to the data, resulting in a worse overall prediction as the regression starts to fit the noise in the data rather than just the underlying trend. A good way around this is component / dimensionality reduction using PCA, which can simplify the regression problem. Constraints with ridge or lasso regression may also help produce an insightful, unique solution.

3. a. See code and output below.

The data matrix is ill-conditioned for any of the 3 reasons:

① $N = 1460$ homes (vectors)
$p = 33$ metrics (features)
} $n > p$, but rank $(X) = 31 < p$, indicating that two columns of X are linearly dependent and X is ill-conditioned

② The singular values of X are found along the diagonal of Σ where $X = U\Sigma V^H$ (SVD of X). Because Σ contains 2 singular values on the diagonal that are close to zero ($10^{-11} \approx 0$), X is ill-conditioned.

③ The condition number of X is $> 10^{16}$, so X is ill-conditioned. To be well-conditioned, the condition number must not be significantly larger than 1.

Because the data matrix is ill-conditioned, its inverse cannot be computed with good accuracy.

b. See code and output below.

The minimum error from ridge regression is 12.13%, which is less than the 12.61% error from unregularized linear regression, using λ (alpha) $= 642$, indicating a better fit for the data / predictive accuracy. In addition, the feature weights from ridge regression are smaller in magnitude than those from unreg. linear regression, which gives better "stability" to the weighting parameters.
↳ i.e. deviations in one metric will not drastically affect prediction.

The graph shows how λ (alpha) affects the ridge regression error and was used to determine the optimal λ value $= 642$ for the smallest error

c. See code and output below.

The minimum error from lasso regression is 12.44%, which is less than the 12.61% error from unregularized linear regression, using λ (alpha) $= 1268$, indicating a better fit for the data / predictive accuracy. In addition, the feature weights from lasso regression are smaller in magnitude than those from unreg. linear regression (and many are zero), which reduces the number of parameters (weights) that impact the prediction and improves the "stability" of the weights.

(3). (c). The graph shows how λ (alpha) affects the lasso regression error and was used to determine the optimal λ value $= 1268$ for the smallest error.

d. Ridge regression with an optimized $\lambda = 642$ produces a lower overall error than lasso regression with an optimized $\lambda = 1268$, but ridge regression also gives somewhat higher parameter weights than lasso regression. Lasso regression's parameter weights are smaller and many are zero, meaning those corresponding features have 0 impact on the home sale price. Thus, lasso regression makes it easier to gauge which features are most important for assessing home value since it reduces the number of impactful features.

i.e lasso regression helps with selecting impactful variables

4. a. See code below.

b. See code below.

c. Both the 1-norm and 2-norm distances produce low (but increasing) misclassification errors with increasing neighbors, but the 2-norm distance error is somewhat lower for corresponding K values. See code below and outputs. Both methods are highly accurate (<5% error) for neighbor sizes <40.

d. Both the ∞-norm and 1-norm distances produce higher error compared to the 2-norm distance across neighbor sizes, especially in the range of 60 to 125 neighbors. See code below and outputs. The ∞-norm is the worst distance metric since it produces the highest misclassification error for many of the neighbor sizes. However, at a low number of neighbors (<20) all 3 distance metrics are highly accurate, giving similar misclassification errors of <3.5%. From these results, it appears that 1-3 neighbors is sufficient for good digit classification.

ELEC378-HW7

March 3, 2023

[1]:
```
# ROBERT HEETER
# ELEC 378 Machine Learning
# 3 March 2023

# PROBLEM SET 7
```

[2]:
```
# PROBLEM 3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

# PART A

df_train = pd.read_csv('train.csv')
train_nums = df_train.select_dtypes(include='number') # select only numerical
 ↪data types

data = train_nums.values
X = data[:,~np.isnan(data).any(axis=0)]
y = X[:,-1] # final column of the data set is the actual sale price
X = np.delete(X,-1,1)

X = X[:,1:] # remove first column of data set since sklearn centers the data
 ↪with fit_intercept

print(f'rank(X) = {np.linalg.matrix_rank(X)}')
print(f'\ncondition(X) = {np.linalg.cond(X)}')
print(f'\ndiagonal matrix of singular values of X from SVD:\n{np.linalg.
 ↪svd(X)[1]}')
```

```
rank(X) = 31

condition(X) = 9.002876317487736e+16
```

```
diagonal matrix of singular values of X from SVD:
[5.67134965e+05 1.05138316e+05 2.25324686e+04 2.23385863e+04
 2.02207361e+04 1.88648770e+04 8.55605120e+03 6.71768936e+03
 6.59551374e+03 4.52726770e+03 2.45552209e+03 2.24042028e+03
 2.11648450e+03 2.02887293e+03 1.49436222e+03 1.47319891e+03
 1.11174224e+03 6.35326262e+02 4.87043831e+02 1.02678447e+02
 3.75103609e+01 3.57223844e+01 3.03596833e+01 2.05915711e+01
 1.88469137e+01 1.60920839e+01 1.44834565e+01 1.23939117e+01
 1.07494666e+01 8.22341553e+00 6.62576816e+00 4.57155422e-11
 4.57155422e-11]
```

[3]:

```
# PART B

# ridge regression
errors = []
alphas = np.arange(1,10000,10)
for a in alphas:
    clf = Ridge(alpha=a, fit_intercept=True).fit(X, y)
    y_approx = clf.predict(X)
    errors.append(np.mean((np.abs(y-y_approx))/y)*100)
plt.figure(0)
plt.plot(alphas,errors)
plt.xlabel('alpha')
plt.ylabel('error (%)')
plt.title('Error of ridge regression predicted sale price')
plt.show()
print(f'\nminimum ridge regression error:\nerror (%) = {errors[(np.
 ↪argsort(errors))[0]]}\nalpha = {alphas[(np.argsort(errors))[0]]}')
print(f'\nridge regression predicted sale prices ($):\n{y_approx}')


# unregularized linear regression (from HW #6)
X_m = np.hstack((np.ones([len(X),1]),X)) # append column of 1's to data matrix␣
 ↪to account for intercept (center of data)
psuedo_X = np.linalg.pinv(X_m) # compute Moore-Penrose pseudoinverse, pseudo_X␣
 ↪= (X^T*X)^-1 * X^T
w = np.matmul(psuedo_X, y) # find optimal weightings, w = pseudo_X*y
y_approx = np.matmul(X_m, w) # find calculated y (sale price) from regression,␣
 ↪y_approx = X*w (+ error)
error = np.mean((np.abs(y-y_approx))/y)*100
print(f'\nunregularized linear regression error = {error}%')
print(f'\nunregularized linear regression predicted sale prices ($):
 ↪\n{y_approx}')
```


```
minimum ridge regression error:
error (%) = 12.126871713425226
alpha = 641

ridge regression predicted sale prices ($):
[219092.52318435 185138.19951186 225390.7898255  … 215073.285221
 137960.3811741  173184.60597563]

unregularized linear regression error = 12.612785356701147%

unregularized linear regression predicted sale prices ($):
[227126.59600561 198031.51742909 222223.03138832 … 232109.41450352
 133868.00040667 160051.06974003]
```

[4]:
```
# PART C

# lasso regression
errors = []
alphas = np.arange(1,10000,10)
for a in alphas:
    clf = Lasso(alpha=a, fit_intercept=True).fit(X, y)
    y_approx = clf.predict(X)
    errors.append(np.mean((np.abs(y-y_approx))/y)*100)
```

```python
plt.figure(1)
plt.plot(alphas,errors)
plt.xlabel('alpha')
plt.ylabel('error (%)')
plt.title('Error of lasso regression predicted sale price')
plt.show()
print(f'\nminimum lasso regression error:\nerror (%) = {errors[(np.argsort(errors))[0]]}\nalpha = {alphas[(np.argsort(errors))[0]]}')
print(f'\nlasso regression predicted sale prices ($):\n{y_approx}')

# unregularized linear regression (from HW #6)
X_m = np.hstack((np.ones([len(X),1]),X)) # append column of 1's to data matrix to account for intercept (center of data)
psuedo_X = np.linalg.pinv(X_m) # compute Moore-Penrose pseudoinverse, pseudo_X = (X^T*X)^-1 * X^T
w = np.matmul(psuedo_X, y) # find optimal weightings, w = pseudo_X*y
y_approx = np.matmul(X_m, w) # find calculated y (sale price) from regression, y_approx = X*w (+ error)
error = np.mean((np.abs(y-y_approx))/y)*100
print(f'\nunregularized linear regression error = {error}%')
print(f'\nunregularized linear regression predicted sale prices ($):\n{y_approx}')
```

```
minimum lasso regression error:
error (%) = 12.443574759801088
alpha = 1261

lasso regression predicted sale prices ($):
[220489.05276195 184600.14161665 225220.20127948 … 216142.69014934
 137401.87572914 171461.8770783 ]

unregularized linear regression error = 12.612785356701147%

unregularized linear regression predicted sale prices ($):
[227126.59600561 198031.51742909 222223.03138832 … 232109.41450352
 133868.00040667 160051.06974003]
```

[5]:

```
# PART D

# assess feature weights between methodologies

clf = Ridge(alpha=642, fit_intercept=True).fit(X, y) # ideal ridge regression␣
 ↪based on part B (alpha = 642)
w_ridge = clf.coef_
print(f'\nridge regression feature weights (w):\n{w_ridge}')

clf = Lasso(alpha=1268, fit_intercept=True).fit(X, y) # ideal lasso regression␣
 ↪based on part C (alpha = 1268)
w_lasso = clf.coef_
print(f'\nlasso regression feature weights (w):\n{w_lasso}')

# unregularized linear regression (from HW #6)
X_m = np.hstack((np.ones([len(X),1]),X)) # append column of 1's to data matrix␣
 ↪to account for intercept (center of data)
psuedo_X = np.linalg.pinv(X_m) # compute Moore-Penrose pseudoinverse, pseudo_X␣
 ↪= (X^T*X)^-1 * X^T
w_unreg = np.matmul(psuedo_X, y) # find optimal weightings, w = pseudo_X*y
print(f'\nunregularized linear regression feature weights (w):\n{w_unreg}')


# assess feature importance ranking between methodologies

labels = train_nums.columns[~np.isnan(data).any(axis=0)] # remove all columns␣
 ↪that were ignored for linear regression
labels_ridge = labels[1:-1] # remove first and last column of indices from␣
 ↪labels
w_i_ridge = np.flip(np.argsort(abs(w_ridge)))
print('\nmost to least impactful features (ridge regression):')
print(labels_ridge[w_i_ridge])
```

```
labels_lasso = labels[1:-1] # remove first and last column of indices from␣
 ↪labels
w_i_lasso = np.flip(np.argsort(abs(w_lasso)))
print('\nmost to least impactful features (lasso regression):')
print(labels_lasso[w_i_lasso])

labels_unreg = labels[1:-1] # remove first and last column of indices from␣
 ↪labels
w_i_unreg = np.flip(np.argsort(abs(w_unreg[1:]))) # ignore intercept (column 0)␣
 ↪for weightings
print('\nmost to least impactful features (unregularized linear regression):')
print(labels_unreg[w_i_unreg])
```

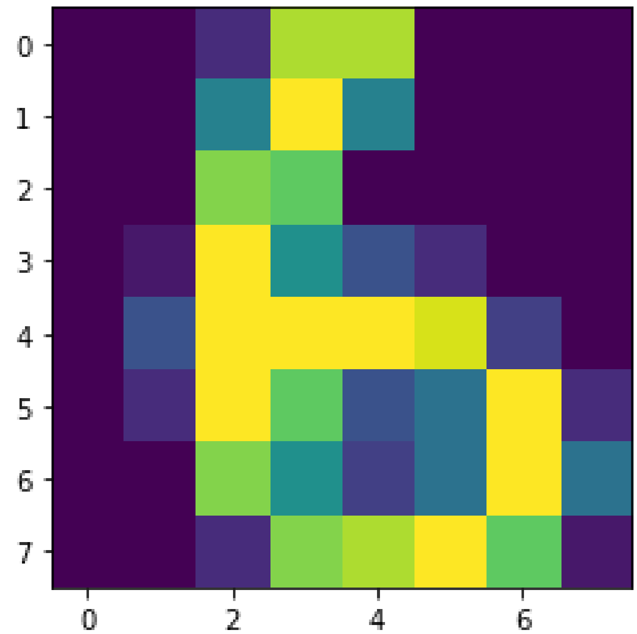
```
ridge regression feature weights (w):
[-1.55552058e+02  3.85229032e-01  1.19969562e+04  3.27696665e+03
  4.44652811e+02  3.13698905e+02  1.55685480e+01 -1.53534158e+00
  5.11712054e-01  1.45449184e+01  2.04304217e+01  2.14781631e+01
 -3.59149263e+00  3.83170922e+01  1.71994517e+03 -1.59123077e+02
  4.51543251e+02 -3.26066953e+02 -4.59738606e+03 -1.30254901e+03
  1.57408927e+03  2.91239548e+03  2.59881660e+03  3.15334669e+01
  3.16958170e+01  3.51476821e-01  2.16943227e+01  2.31244698e+01
  6.94090940e+01 -5.79388835e+01 -1.37711587e+00 -2.60037136e+01
 -6.54030253e+02]

lasso regression feature weights (w):
[-1.64394555e+02  4.41557947e-01  1.83389896e+04  2.60064477e+03
  3.61037733e+02  2.49103897e+02  2.26113866e+01  6.75178573e+00
  5.32070551e+00  3.92082510e+00  5.07011638e+01  4.94994722e+01
  2.32360589e+01  6.47168774e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -5.07074631e+03 -0.00000000e+00
  1.24875991e+03  1.28818776e+03  6.34739681e+00  3.35499017e+01
  3.18126195e+01 -3.26757157e+00  1.11736241e+01  2.03499866e+01
  6.48542981e+01 -5.31664730e+01 -1.22594028e+00 -0.00000000e+00
 -3.35539344e+01]

unregularized linear regression feature weights (w):
[ 5.02595078e+05 -1.62672852e+02  3.96228096e-01  1.79050672e+04
  4.41879480e+03  3.46653503e+02  1.37073924e+02  1.18335979e+01
 -2.72826009e+00  7.87734602e-01  9.89307245e+00  1.88377068e+01
  1.89463690e+01 -6.00030885e+00  3.17837670e+01  8.53489406e+03
  2.46720054e+03  3.57748905e+03 -1.32686163e+03 -1.05307793e+04
 -1.29277699e+04  5.13231805e+03  3.59689511e+03  1.06337499e+04
  1.39621269e+00  2.63726911e+01 -5.61939741e+00  8.72201006e+00
  1.87713841e+01  5.78859914e+01 -4.26136870e+01 -8.91247504e-01
 -1.15348621e+02 -7.57643913e+02]
```

```
most to least impactful features (ridge regression):
Index(['OverallQual', 'BedroomAbvGr', 'OverallCond', 'Fireplaces',
       'GarageCars', 'BsmtFullBath', 'TotRmsAbvGrd', 'KitchenAbvGr', 'YrSold',
       'FullBath', 'YearBuilt', 'HalfBath', 'YearRemodAdd', 'BsmtHalfBath',
       'MSSubClass', 'ScreenPorch', 'PoolArea', 'GrLivArea', 'WoodDeckSF',
       'GarageArea', 'MoSold', '3SsnPorch', 'EnclosedPorch', '2ndFlrSF',
       '1stFlrSF', 'BsmtFinSF1', 'TotalBsmtSF', 'LowQualFinSF', 'BsmtFinSF2',
       'MiscVal', 'BsmtUnfSF', 'LotArea', 'OpenPorchSF'],
      dtype='object')

most to least impactful features (lasso regression):
Index(['OverallQual', 'BedroomAbvGr', 'OverallCond', 'Fireplaces',
       'TotRmsAbvGrd', 'YearBuilt', 'YearRemodAdd', 'MSSubClass',
       'ScreenPorch', 'PoolArea', '1stFlrSF', '2ndFlrSF', 'YrSold',
       'GarageArea', 'WoodDeckSF', 'LowQualFinSF', 'BsmtFinSF1', '3SsnPorch',
       'EnclosedPorch', 'BsmtFinSF2', 'GrLivArea', 'GarageCars', 'BsmtUnfSF',
       'TotalBsmtSF', 'OpenPorchSF', 'MiscVal', 'LotArea', 'KitchenAbvGr',
       'BsmtFullBath', 'BsmtHalfBath', 'MoSold', 'HalfBath', 'FullBath'],
      dtype='object')

most to least impactful features (unregularized linear regression):
Index(['OverallQual', 'KitchenAbvGr', 'GarageCars', 'BedroomAbvGr',
       'BsmtFullBath', 'TotRmsAbvGrd', 'OverallCond', 'Fireplaces', 'FullBath',
       'BsmtHalfBath', 'HalfBath', 'YrSold', 'YearBuilt', 'MSSubClass',
       'YearRemodAdd', 'MoSold', 'ScreenPorch', 'PoolArea', 'GrLivArea',
       'WoodDeckSF', '2ndFlrSF', '1stFlrSF', '3SsnPorch', 'BsmtFinSF1',
       'TotalBsmtSF', 'EnclosedPorch', 'LowQualFinSF', 'OpenPorchSF',
       'BsmtFinSF2', 'GarageArea', 'MiscVal', 'BsmtUnfSF', 'LotArea'],
      dtype='object')
```

[6]:

```python
# PROBLEM 4

import numpy as np
from matplotlib import pyplot as plt

from sklearn.datasets import load_digits
from scipy.spatial.distance import cdist
from scipy.stats import mode
from sklearn.model_selection import train_test_split

# PART A

# load digits
digits = load_digits()

# split digits into training and test data
```

```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,␣
↪test_size=0.2, random_state=0)

# example image
index = 100
plt.imshow(X_train[index].reshape((8,8)))
plt.show()
print(f'Digit: {y_train[index]}')
```




```
Digit: 6
```

[7]:
```
def knn(X_fit, y_fit, X_predict, n_neighbors=5, metric='euclidean'):
    '''
    inputs:
        X_fit - 2D array containing all training data points
        y_fit - 2D array containing all training data labels
        X_predict - 2D array containing all data points to classify
        n_neighbors - K
        metric - see scipy.spatial.distance.cdist:
                 https://docs.scipy.org/doc/scipy/reference/generated/scipy.
↪spatial.distance.cdist.html

    returns: a 1D array of predicted labels for X_predict.
    '''
```

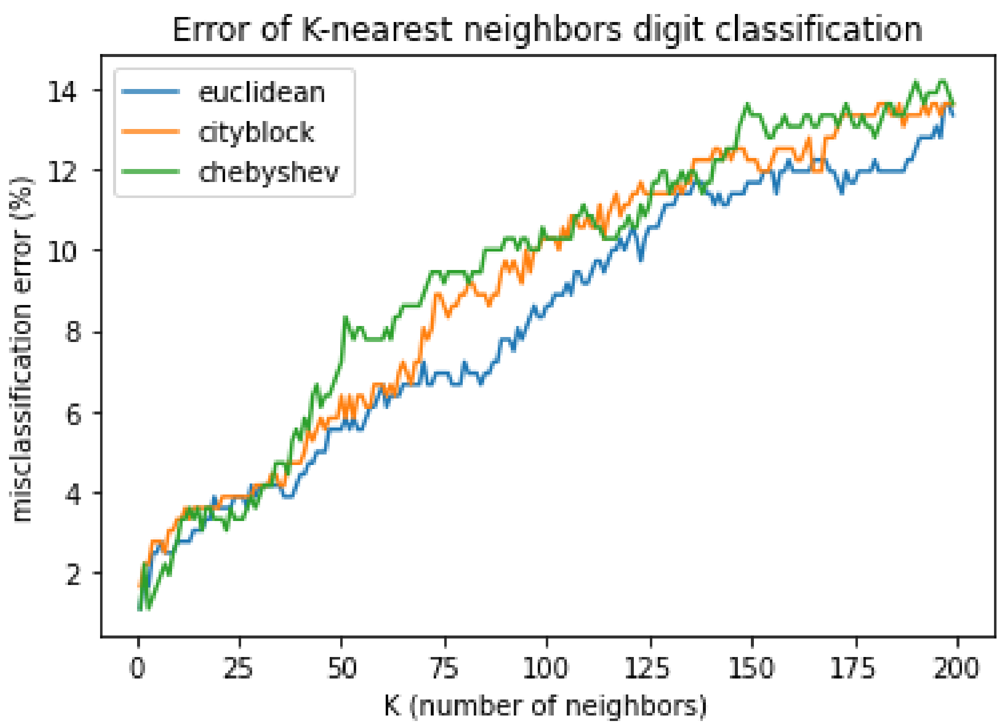
```
    # ensure that X_predict is two dimensional; this only matters if X_predict␣
 ↪is one data point
    if X_predict.ndim < 2:
        X_predict = [X_predict]

    # calculate distances
    distances = cdist(X_predict, X_fit, metric)

    # find the data indices of least distance for each point; keep the closest␣
 ↪n_neighbors data points for each point
    closest = np.argsort(distances,axis=1)[:,0:n_neighbors]

    # find the label of the n_neighbors closest data points
    closest_labels = y_fit[closest]

    # get the mode of each row and return the resulting 1D array of labels
    y_predict = mode(closest_labels, axis=1, keepdims=True)[0][:,0]

    return y_predict
```

[8]:
```
# PART B,C,D

metrics = ['euclidean', 'cityblock', 'chebyshev']
plt.figure(2)

for metric in metrics:
    print(f'\nDISTANCE METRIC = {metric}')

    errors = []
    Ks = range(1,200)

    for K in Ks:

        predictions = knn(X_fit=X_train, y_fit=y_train, X_predict=X_test,␣
 ↪n_neighbors=K, metric=metric)
        error = np.sum((y_test != predictions)*1)/len(predictions)*100

        errors.append(error)

        if K <= 20:
            print(f'K = {K}; misclassification error: {error}%')

    plt.plot(Ks,errors)

plt.legend(metrics)
plt.xlabel('K (number of neighbors)')
plt.ylabel('misclassification error (%)')
```

```python
plt.title('Error of K-nearest neighbors digit classification')
plt.show()
```

```
DISTANCE METRIC = euclidean
K = 1; misclassification error: 1.1111111111111112%
K = 2; misclassification error: 1.9444444444444444%
K = 3; misclassification error: 1.6666666666666667%
K = 4; misclassification error: 2.5%
K = 5; misclassification error: 2.5%
K = 6; misclassification error: 2.7777777777777777%
K = 7; misclassification error: 2.5%
K = 8; misclassification error: 2.5%
K = 9; misclassification error: 2.5%
K = 10; misclassification error: 2.7777777777777777%
K = 11; misclassification error: 2.7777777777777777%
K = 12; misclassification error: 2.7777777777777777%
K = 13; misclassification error: 2.7777777777777777%
K = 14; misclassification error: 3.0555555555555554%
K = 15; misclassification error: 3.0555555555555554%
K = 16; misclassification error: 3.0555555555555554%
K = 17; misclassification error: 3.3333333333333335%
K = 18; misclassification error: 3.3333333333333335%
K = 19; misclassification error: 3.888888888888889%
K = 20; misclassification error: 3.6111111111111107%

DISTANCE METRIC = cityblock
K = 1; misclassification error: 1.6666666666666667%
K = 2; misclassification error: 2.2222222222222223%
K = 3; misclassification error: 2.2222222222222223%
K = 4; misclassification error: 2.7777777777777777%
K = 5; misclassification error: 2.7777777777777777%
K = 6; misclassification error: 2.7777777777777777%
K = 7; misclassification error: 2.5%
K = 8; misclassification error: 3.0555555555555554%
K = 9; misclassification error: 3.0555555555555554%
K = 10; misclassification error: 3.3333333333333335%
K = 11; misclassification error: 3.3333333333333335%
K = 12; misclassification error: 3.6111111111111107%
K = 13; misclassification error: 3.3333333333333335%
K = 14; misclassification error: 3.6111111111111107%
K = 15; misclassification error: 3.6111111111111107%
K = 16; misclassification error: 3.6111111111111107%
K = 17; misclassification error: 3.6111111111111107%
K = 18; misclassification error: 3.6111111111111107%
K = 19; misclassification error: 3.6111111111111107%
K = 20; misclassification error: 3.6111111111111107%
```

```
DISTANCE METRIC = chebyshev
K = 1; misclassification error: 1.1111111111111112%
K = 2; misclassification error: 2.2222222222222223%
K = 3; misclassification error: 1.1111111111111112%
K = 4; misclassification error: 1.3888888888888888%
K = 5; misclassification error: 1.6666666666666667%
K = 6; misclassification error: 1.9444444444444444%
K = 7; misclassification error: 2.2222222222222223%
K = 8; misclassification error: 1.9444444444444444%
K = 9; misclassification error: 2.5%
K = 10; misclassification error: 2.7777777777777777%
K = 11; misclassification error: 3.3333333333333335%
K = 12; misclassification error: 3.3333333333333335%
K = 13; misclassification error: 3.6111111111111107%
K = 14; misclassification error: 3.3333333333333335%
K = 15; misclassification error: 3.6111111111111107%
K = 16; misclassification error: 3.0555555555555554%
K = 17; misclassification error: 3.6111111111111107%
K = 18; misclassification error: 3.6111111111111107%
K = 19; misclassification error: 3.3333333333333335%
K = 20; misclassification error: 3.3333333333333335%
```



[ ]: