Robert Heeter
ELEC 378
2/24/23

Problem Set #6

1. $\underline{X} \rightarrow n \times p$ data matrix

$0 < q < p$

constraint

Prove PCA objective $\max_{u_1, \dots u_q} \sum_{i=1}^{q} \underline{u}_i^T \underline{X}^T \underline{X} \underline{u}_i$ subject to $\{\|\underline{u}_i\|_2^2 = 1\}_{i=1}^{q}$ is maximized when $\underline{u}_i$ is chosen to be the i^{th} eigenvector of the covariance matrix $\underline{X}^T \underline{X}$.

Lagrange multipliers λ_i: $\max_w \mathcal{L}(w)$ such that $g(w) = 0$, $h(w) = 0$

$$\downarrow$$

$$\max_{w, \lambda_1, \lambda_2} \mathcal{L}(w) - \lambda_1 g(w) - \lambda_2 h(w)$$ ← subtract constraints to find max

Here: $\max_{u_1 .. u_q} \sum_{i=1}^{q} u_i^T x^T x u_i$ such that $\{\|u_i\|_2^2 = 1\}_{i=1}^{q}$

$$\downarrow$$

$$\max_{u_1 .. u_q, \lambda_1 .. \lambda_q} \sum_{i=1}^{q} u_i^T x^T x u_i - \lambda_i \underbrace{(\|u_i\|_2^2 - 1)}_{\text{constraint function must equal zero}}$$

To find max, use gradient with respect to u_k:

From lecture notes $\nabla_z z^T A z = 2Az$ given $A^T = A$.

① $\therefore \nabla_{u_k} (\|u_i\|_2^2 - 1) = \nabla_{u_k} (u_i^T u_i - 1) = \nabla_{u_k} (u_i^T I u_i - 1) = 2Iu_k = 2u_k$

and

② $\therefore \nabla_{u_k} (u_i^T x^T x u_i) = 2x^T x u_k$

So: $\nabla_{u_k} \left(\sum_{i=1}^{q} u_i^T x^T x u_i - \lambda_i (\|u_i\|_2^2 - 1) \right) = \underset{②}{2x^T x u_k} - \underset{①}{2\lambda_k u_k}$

At maximum, $\nabla u_k = 0$:

$$2x^T x u_k - 2\lambda_k u_k = 0 \rightarrow x^T x u_k - \lambda_k u_k = 0 \rightarrow x^T x u_k = \lambda_k u_k$$

This takes the form $Av = \lambda v$, where λ is an eigenvalue of A and v is an eigenvector of A. Here $A = x^T x$ (the covariance matrix), λ_k is the kth eigenvalue of A, and u_k is the kth eigenvector of A.

(1). Therefore, $\max_{u_1 \dots u_q} \sum_{i=1}^{q} u_i^T X^T X u_i$ such that $\{\|u_i\|_2^2 = 1\}_{i=1}^{q}$ is where u_i is the i_{th} eigenvector of the covariance matrix $X^T X$, when $\nabla u_k = 0$.

2. $X^+ = (X^H X)^{-1} X^H$

Prove when $X = U\Sigma V^H$, $X^+ = V\Sigma^+ U^H$

$X^H = V\Sigma^H U^H$

$X^H X = V\Sigma^H \cancel{U^H} \overset{I}{U} \Sigma V^H \leftarrow \boxed{U^H U = I}$

$= V\Sigma^H \Sigma V^H$

$(X^H X)^{-1} = (V\Sigma^H \Sigma V^H)^{-1}$

$= (V^H)^{-1} (\Sigma^H \Sigma)^{-1} (V)^{-1}$ $\boxed{V^{-1}V = I}$

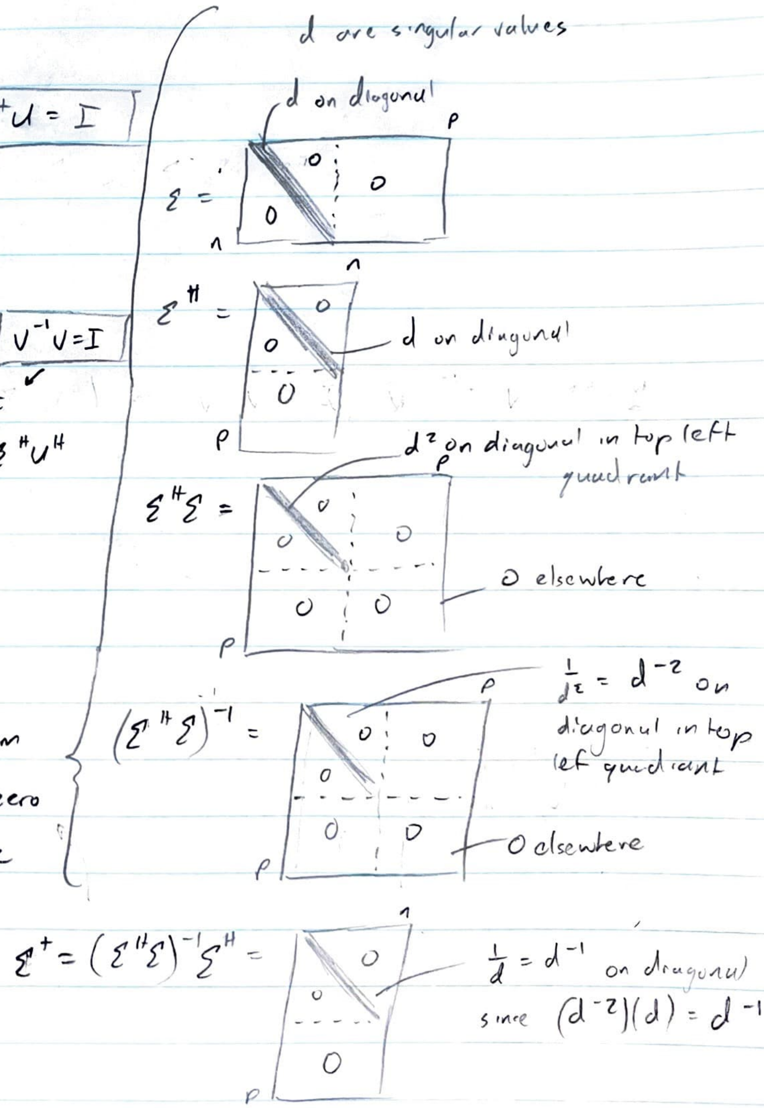
$\boxed{(V^H)^{-1} = V}$

$(X^H X)^{-1} X^H = \cancel{(V^H)}^{-1} (\Sigma^H \Sigma)^{-1} \overset{I}{\cancel{(V)^{-1} V}} \Sigma^H U^H$

$= (V)\underbrace{(\Sigma^H \Sigma)^{-1} \Sigma^H}_{=\Sigma^+} U^H$

Define $\Sigma^+ = \Sigma^H \Sigma^{-1} \Sigma^H$

$X^+ = V\Sigma^+ U^H$, where Σ^+ is found from the reciprocal of the non-zero entries of Σ^H, leaving the zeros in place



3. Next page

3. a. Only numerical features (i.e. features that are scored on a numerical scale) can be used to predict the sale price using linear regression, since regression relies on numerical data to compute the weightings. Thus, the columns (features) that are descriptive (non-numerical) should be removed. The "index" (column 0) and "MSSubClass" (column 1) should also be removed; while both are numerical, they are not meaningful for determining home value — "MSSubClass" is the building class, which is a categorical variable.

See code below for constructing the data matrix X and vector y. The dimensions of X are n = rows = 1460 homes vectors and p = columns / features = 33.

Note that the first column of X is a column of 1s to account for the data being off-center. The features in X were normalized by subtracting the feature minimum value and dividing by the feature range to produce more meaningful feature weights in linear regression.

b. See code below and metrics for accuracy. The average percent difference of the predicted home value from the actual home value (in vector y) was used to quantify the accuracy of the fit. This gave a 12.8% error, which is relatively low considering the variability of home prices. Sklearn was also used to check the fit, giving very similar results.

The graph shows all of the homes, comparing their actual sale price to the scratch implementation of linear regression's prediction. Most of the homes' actual and predicted prices are similar (near the red line indicating 100% accuracy — i.e. actual price = predicted price), but there are a few outliers.

c. The magnitude of the absolute value of the weighting parameters w indicates the importance of the corresponding features on sale price. By ordering the abs. value weights and their corresponding features from largest to smallest, the most to least important features for predicting sale price are found: OverallQual, 1stFlrSF, GrLivArea, ..., OpenPorchSF, BsmtHalfBath, MoSold

most important

least important

(3)(C). See the code output below for the full ranking. Note that a large positive or negative w (large $|w|$) is "impactful" on sale price, since it weights the corresponding features more heavily compared to a smaller $|w|$ which would correspond to a less "impactful" feature.

It makes sense that overall home quality, square footage of the 1st floor, and above grade ground living area are very important factors for home value, while porch square footage, basement half-baths (?), and month sold are relatively unimportant.

ELEC378-HW6

February 24, 2023

[1]:
```
# ROBERT HEETER
# ELEC 378 Machine Learning
# 24 February 2023

# PROBLEM SET 6
```

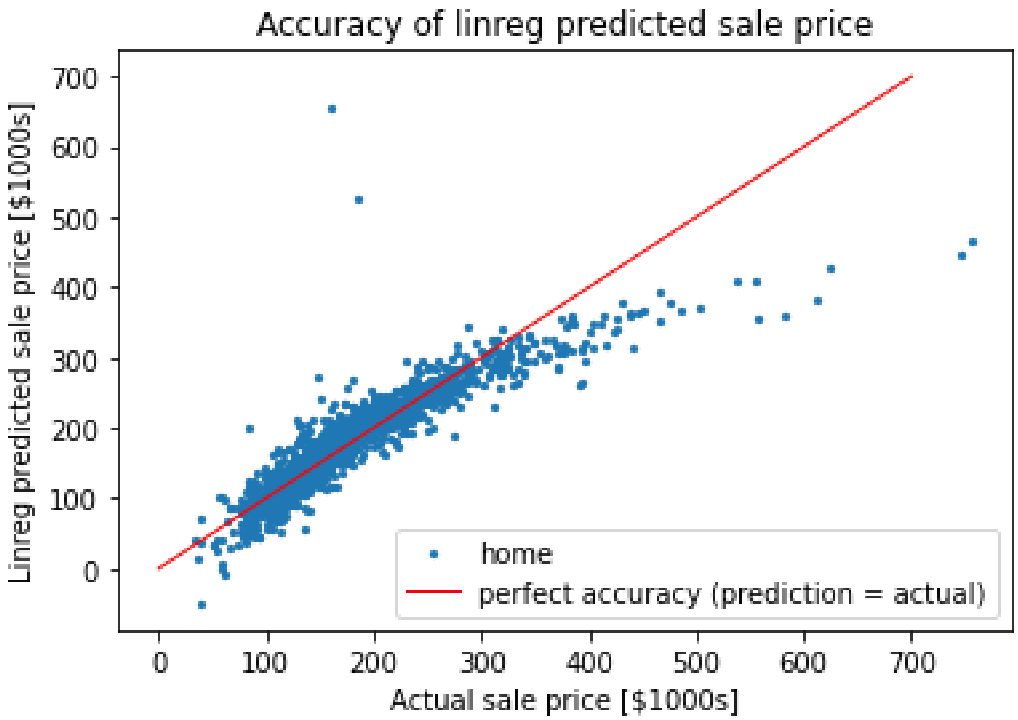
[2]:
```
# PROBLEM 1

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

# PART A

df_train = pd.read_csv('train.csv')
train_nums = df_train.select_dtypes(include='number') # select only numerical␣
↪data types
del train_nums['MSSubClass'] # the MSSubClass column is not actually a␣
↪numerical feature, so remove it

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)

m = X.min(axis=0) # minimum value in each feature
r = np.ptp(X,axis=0) # range of values for each feature

X = (X-m)/r # normalize data using minimum value and range of values

X[:,0] = np.ones([len(X)]) # set first column to 1's to account for the center␣
↪of the data; this replaces the first column of house indexes in the dataset,␣
↪which is not informative

print('dataset for inspection:')
print(X)
print(np.shape(X))
```

```
dataset for inspection:
[[1.         0.0334198  0.66666667 … 0.         0.09090909 0.5       ]
 [1.         0.03879502 0.55555556 … 0.         0.36363636 0.25      ]
 [1.         0.04650728 0.66666667 … 0.         0.72727273 0.5       ]
 …
 [1.         0.03618687 0.66666667 … 0.16129032 0.36363636 1.        ]
 [1.         0.03934189 0.44444444 … 0.         0.27272727 1.        ]
 [1.         0.04037019 0.44444444 … 0.         0.45454545 0.5       ]]
(1460, 33)
```

[3]:

```python
# PART B

# scratch linear regression implementation
psuedo_X = np.linalg.pinv(X) # 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, w) # find calculated y (sale price) from regression,
↪y_approx = X*w (+ error)

n = len(y)
error = np.mean((np.abs(y-y_approx))/y)*100
# error = np.linalg.norm(y-y_approx,ord=1)/n # use 1-norm to calculate average
↪deviation from

print('scratch linreg predicted price (y_approx) average percent difference
↪from actual price (y):')
print(f'{round(error, 3)}%')

plt.figure(0)
plt.scatter(y/1000, y_approx/1000, s=5)
plt.plot([0,700], [0,700], 'r-', linewidth=1)
plt.legend(['home','perfect accuracy (prediction = actual)'])
plt.xlabel('Actual sale price [$1000s]')
plt.ylabel('Linreg predicted sale price [$1000s]')
plt.title('Accuracy of linreg predicted sale price')

# check prediction using sklearn
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X, y)
sk_y_approx = reg.predict(X)
sk_diff = np.mean((np.abs(sk_y_approx-y_approx))/y_approx)*100

print('\nsklearn predicted price (sk_y_approx) average percent difference from
↪scratch linreg implementation (y_approx):')
print(f'{round(sk_diff, 16)}%')
```

```
scratch linreg predicted price (y_approx) average percent difference from actual
price (y):
12.766%

sklearn predicted price (sk_y_approx) average percent difference from scratch
linreg implementation (y_approx):
2.957e-13%
```



[4]:
```
# PART C

w_i = np.flip(np.argsort(abs(w)))

train_nums = train_nums.rename(columns = {'Id':'~'}) # rename first column to␣
 ↪'~' since it was used for centering
labels = train_nums.columns[~np.isnan(data).any(axis=0)] # remove all columns␣
 ↪that were ignored for linear regression

print('most to least impactful features for determining sale price:')
print(labels[w_i])

print('\nmost to least impactful feature weights for determining sale price:')
print(w[w_i])
```

```
most to least impactful features for determining sale price:
```

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

most to least impactful feature weights for determining sale price:
[154168.62961313 127415.33632592 121300.68131278  96394.7595521
 -72982.35282851 -72630.4515861   71890.45327379  70525.57951422
  66073.84453326 -53933.23250924  46319.20148209  44029.17168872
  41375.35420552  37591.32253671 -33227.28966056  27214.69665354
  21863.81734127  21062.47562701  12465.98680795  10336.96711351
  -9329.88763581   8889.68290503   8165.73484158   6962.49937493
   6195.98076262   4235.05261852  -4077.37558977  -2868.56417699
  -2400.39913546  -2176.36568614  -1935.37804652   1637.53001584
   -288.69917314]
```

[ ]: