ANN Homework 1 - Sarah Brown

February 23, 2021

1 Question 1

For a square matrix A, prove that: $(A^{-1})^T = (A^T)^{-1}$

1.1 My Answer

Assume that A is invertible as it is a square matrix. This means that there exists A and its inverse A^{-1}. This means that we can rewrite like this:

$$
\begin{gathered}
A * A^{-1} = I \\
(A * A^{-1})^T = I^T \\
((A^{-1})^T * (A)^T) = I \\
((A^{-1})^T * (A)^T) * (A^T)^{-1} = I * (A^T)^{-1} \\
((A^{-1})^T) * I = (A^T)^{-1} \\
((A^{-1})^T) = (A^T)^{-1}
\end{gathered}
$$

Which shows that for a square matrix A: $(A^{-1})^T = (A^T)^{-1}$

2 Question 2

Find the gradient of $f(A) = X\ AX$ with respect to A, where X is a column vector and A is a matrix. Note that A is the variable here, rather than X as discussed in class.

2.1 My Answer

$X^T AT$ can be rewritten as:

$$y(A) = X^T AT = \begin{vmatrix} x_1 & \dots & x_n \end{vmatrix} \begin{vmatrix} a_{11} & \dots & a_{n1} \\ \vdots & \vdots & \vdots \\ a_{1n} & \dots & a_{nn} \end{vmatrix} \begin{vmatrix} x_1 \\ \vdots \\ x_n \end{vmatrix} = \Sigma_{i=1}^n \Sigma_{j=1}^n x_i a_{ij} x_j$$

This can then be used to take the gradient of f(A)

$$y(A) = X^T * A * X \nabla_A y(A) = \frac{\partial y(A)}{\partial A} = \frac{\partial}{\partial A}(\Sigma_{i=1}^n \Sigma_{j=1}^n x_i a_{ij} x_j) = \Sigma_{i=1}^n \Sigma_{j=1}^n x_i x_j = \begin{vmatrix} x_1 x_1 & \dots & x_n x_1 \\ \vdots & \vdots & \vdots \\ x_1 x_n & \dots & x_n x_n \end{vmatrix} = XX^T$$

So $\nabla_A y(A) = XX^T$

3 Question 3

[1]:
```python
import yfinance as yf
import pandas as pd

def get_price(tick,start='2020-10-01',end=None):
    return yf.Ticker(tick).history(start=start,end=end)['Close']

def get_prices(tickers,start='2020-10-01',end=None):
    df=pd.DataFrame()
    for s in tickers:
        df[s]=get_price(s,start,end)
    return df
```

[2]:
```python
feature_stocks=['tsla','fb','twtr','amzn','nflx','gbtc','gdx','intc','dal','c']
predict_stock='msft'

# training set
start_date_train='2020-10-01'
end_date_train='2020-12-31'

X_train=get_prices(feature_stocks,start=start_date_train,end=end_date_train)
y_train=get_prices([predict_stock],start=start_date_train,end=end_date_train)

# testing set
start_date_test='2021-01-01' # end date omit, default is doday
X_test=get_prices(feature_stocks,start=start_date_test)
y_test=get_prices([predict_stock],start=start_date_test)
```

[3]:
```python
X_train
```

[3]:
```
                  tsla          fb       twtr         amzn        nflx  \
Date
2020-10-01  448.160004  266.630005  46.700001  3221.260010  527.510010
2020-10-02  415.089996  259.940002  46.119999  3125.000000  503.059998
2020-10-05  425.679993  264.649994  47.310001  3199.199951  520.650024
2020-10-06  413.980011  258.660004  45.599998  3099.959961  505.869995
2020-10-07  425.299988  258.119995  45.869999  3195.689941  534.659973
…                    …           …          …            …           …
2020-12-23  645.979980  268.109985  54.299999  3185.270020  514.479980
2020-12-24  661.770020  267.399994  53.970001  3172.689941  513.969971
2020-12-28  663.690002  277.000000  54.430000  3283.959961  519.119995
2020-12-29  665.989990  276.779999  54.360001  3322.000000  530.869995
2020-12-30  694.780029  271.869995  54.330002  3285.850098  524.590027

                 gbtc         gdx        intc          dal           c
Date
```

```
2020-10-01  10.870000  39.364471  51.549873  31.100000  42.545544
2020-10-02  10.860000  38.767590  50.336117  31.750000  42.761013
2020-10-05  11.280000  39.364471  51.007137  32.000000  43.985275
2020-10-06  10.845000  37.912056  50.691364  31.059999  43.495571
2020-10-07  10.970000  38.150806  51.974190  32.150002  43.916718
…                 …          …          …          …          …
2020-12-23  28.879999  35.919998  46.289028  40.240002  60.266277
2020-12-24  27.350000  36.029999  46.786011  39.730000  60.058056
2020-12-28  30.450001  35.689999  46.786011  40.150002  60.613323
2020-12-29  30.080000  35.740002  49.092014  40.029999  60.395180
2020-12-30  32.900002  36.560001  48.455875  40.560001  60.345604

[63 rows x 10 columns]
```

[4]:
```
y_train
```

[4]:
```
                  msft
Date
2020-10-01  211.418289
2020-10-02  205.179031
2020-10-05  209.348495
2020-10-06  204.900406
2020-10-07  208.801178
…                  …
2020-12-23  220.512131
2020-12-24  222.238144
2020-12-28  224.443069
2020-12-29  223.634918
2020-12-30  221.170593

[63 rows x 1 columns]
```

[5]:
```
import numpy as np

X_train=np.array(X_train)
y_train=np.array(y_train)
X_test=np.array(X_test)
y_test=np.array(y_test)
```

4 Use linear regression to predict msft stock price from the other stocks' prices

4.1 1. Append a dummy feature to both X_train and X_test

[6]:
```
dummyFeatureX_train = np.array(pd.get_dummies(X_train[0]))
dummyFeatureX_test = np.array(pd.get_dummies(X_test[0]))

print("Original:", X_train[0],"Dummy X_train:", dummyFeatureX_train[0])
print("Original:", X_test[0],"Dummy X_test:", dummyFeatureX_test[0])
```

```
Original: [ 448.16000366  266.63000488   46.70000076 3221.26000977  527.51000977
   10.86999989   39.36447144   51.54987335   31.10000038   42.54554367] Dummy
X_train: [0 0 0 0 0 0 0 1 0 0]
Original: [ 729.77001953  268.94000244   54.52999878 3186.62988281  522.85998535
   35.08000183   38.50999832   49.37032318   38.72999954   59.63168716] Dummy
X_test: [0 0 0 0 0 0 0 0 1 0]
```

4.2 2. Find the best linear regression model

Based on your training data ($w = (XX\ )^{-1}Xy$) Note that you may need to transpose the matrices to make things work

(I looked at some examples here that were helpful in figuring this out: https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html)

[7]:
```
from sklearn import linear_model, metrics

linearRegModel = linear_model.LinearRegression() # create linear model object
linearRegModel = linearRegModel.fit(X_train, y_train) # trains the model using␣
↪the training sets
y_predictionsPrice = linearRegModel.predict(X_test) # uses model to make␣
↪predictions
y_predictionsPrice
```

[7]:
```
array([[223.18584796],
       [227.9597718 ],
       [229.55947399],
       [229.82017852],
       [224.70310324],
       [221.32358894],
       [221.7310447 ],
       [223.70921917],
       [226.16984358],
       [218.0501453 ],
       [217.68651618],
       [221.42759589],
       [220.99572843],
```

```
       [219.92681288],
       [217.77071321],
       [218.13340053],
       [212.02005752],
       [215.43376603],
       [214.53834933],
       [220.32163404],
       [221.22779418],
       [221.00221499],
       [221.33191333],
       [223.40260151],
       [226.97801496],
       [226.85129755],
       [225.70983617],
       [224.94476523],
       [225.8098199 ],
       [225.82038667],
       [228.77100907],
       [227.33693303],
       [228.06619271],
       [227.84880384],
       [224.85759493]])
```

4.3 3. Report your training and testing error

How far your prediction from the actual price. Compute the mean square error for both training and testing

[8]:
```python
#compare training data to testing data with mean square error
meanSquareError = metrics.mean_squared_error(y_test, y_predictionsPrice)
r2Score = metrics.r2_score(y_test, y_predictionsPrice)

print("The mean square error is: ", meanSquareError)
print("The coeefficient of determination (with 1 being perfect) is: ", r2Score)
```

```
The mean square error is:  200.3033149699239
The coeefficient of determination (with 1 being perfect) is:
-0.49513546816427767
```