Week 35

Exercise 1: Analytical exercises

Show that

$$\frac{\partial \mathbf{b}^T\mathbf{a}}{\partial \mathbf{a}} = \mathbf{b}$$

$$\frac{\partial \mathbf{b}^T\mathbf{a}}{\partial \mathbf{a}} = \frac{\partial (b_0a_0 + b_1a_1 + \ldots + b_na_n)}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial (b_0a_0 + b_1a_1 + \ldots + b_na_n)}{\partial a_0} \\ \frac{\partial (b_0a_0 + b_1a_1 + \ldots + b_na_n)}{\partial a_1} \\ \vdots \\ \frac{\partial (b_0a_0 + b_1a_1 + \ldots + b_na_n)}{\partial a_n} \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} = \mathbf{b}$$

Then show that:

$$\frac{\partial \mathbf{a}^{\mathbf{T}}\mathbf{A}\mathbf{a}}{\partial \mathbf{a}} = \mathbf{a}^T(\mathbf{A} + \mathbf{A}^T)$$

$$\mathbf{a}^T\mathbf{A}\mathbf{a} = [a_0, a_1, \ldots, a_n] \begin{bmatrix} A_{00} & A_{01} & \ldots & A_{0n} \\ A_{10} & A_{11} & \ldots & A_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ A_{n0} & A_{n1} & \ldots & A_{nn} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}$$

$$= [a_0, a_1, \ldots, a_n] \begin{bmatrix} \sum_i A_{0i}a_i \\ \sum_i A_{1i}a_i \\ \vdots \\ \sum_i A_{ni}a_i \end{bmatrix} = \sum_{ij} \left(a_jA_{ji}a_i\right)$$

$$\frac{\partial}{\partial a_k} \sum_{ij} \left(a_jA_{ji}a_i\right) = \sum_j A_{kj}a_j + \sum_i Akia_i = \mathbf{a}^T(\mathbf{A} + \mathbf{A}^T)$$

Lastly, show that:

$$\frac{\partial (\mathbf{x} - \mathbf{A}\mathbf{s})^T(\mathbf{x} - \mathbf{A}\mathbf{s})}{\partial \mathbf{s}} = -2(\mathbf{x} - \mathbf{A}\mathbf{s})^T\mathbf{A}$$

To show that

$$\frac{\partial (\mathbf{x} - \mathbf{A}\mathbf{s})^T(\mathbf{x} - \mathbf{A}\mathbf{s})}{\partial \mathbf{s}} = -2(\mathbf{x} - \mathbf{A}\mathbf{s})^T\mathbf{A}$$

Let's start by expanding the expression on the left-hand side:

$$\frac{\partial(\mathbf{x}-\mathbf{A}\mathbf{s})^T(\mathbf{x}-\mathbf{A}\mathbf{s})}{\partial\mathbf{s}} = \frac{\partial(\mathbf{x}^T\mathbf{x}-\mathbf{x}^T\mathbf{A}\mathbf{s}-(\mathbf{A}\mathbf{s})^T\mathbf{x}+(\mathbf{A}\mathbf{s})^T\mathbf{A}\mathbf{s})}{\partial\mathbf{s}}$$

1. $\frac{\partial\mathbf{x}^T\mathbf{x}}{\partial\mathbf{s}} = 0$ since $\mathbf{x}$ is not a function of $\mathbf{s}$.
2. $\mathbf{x}^T\mathbf{A}\mathbf{s} = (\mathbf{A}\mathbf{s})^T\mathbf{x}$, since both are scalars, so

$$\frac{\partial - \mathbf{x}^T\mathbf{A}\mathbf{s} - (\mathbf{A}\mathbf{s})^T\mathbf{x}}{\partial\mathbf{s}} = -2\frac{\partial\mathbf{x}^T\mathbf{A}\mathbf{s}}{\partial\mathbf{s}} = -2\mathbf{A}^T\mathbf{x}$$

3. $\frac{\partial(\mathbf{A}\mathbf{s})^T\mathbf{A}\mathbf{s}}{\partial\mathbf{s}} = \frac{\partial\mathbf{s}^T\mathbf{A}^T\mathbf{A}\mathbf{s}}{\partial\mathbf{s}} = \sum_j \frac{\partial s_j}{\partial s_k}\sum_i (A^TA)_{ij}s_i + \sum_j s_j \sum_i (A^TA)_{ij}\frac{\partial s_i}{\partial s_k}$

   Since (A^TA) is symmetric, we can combine the two sums:

$$\frac{\partial(\mathbf{A}\mathbf{s})^T\mathbf{A}\mathbf{s}}{\partial\mathbf{s}} = 2\sum_i (A^TA)_{ki}s_i = 2(A^TA)_{k:}\mathbf{s} = 2\mathbf{A}^T\mathbf{A}\mathbf{s}$$

4. Now we can put it all together:

$$\frac{\partial(\mathbf{x}-\mathbf{A}\mathbf{s})^T(\mathbf{x}-\mathbf{A}\mathbf{s})}{\partial\mathbf{s}} = -2\mathbf{A}^T\mathbf{x} + 2\mathbf{A}^T\mathbf{A}\mathbf{s} = -2(\mathbf{x}-\mathbf{A}\mathbf{s})^T\mathbf{A}$$

Exercise 2: making your own data and exploring scikit-learn

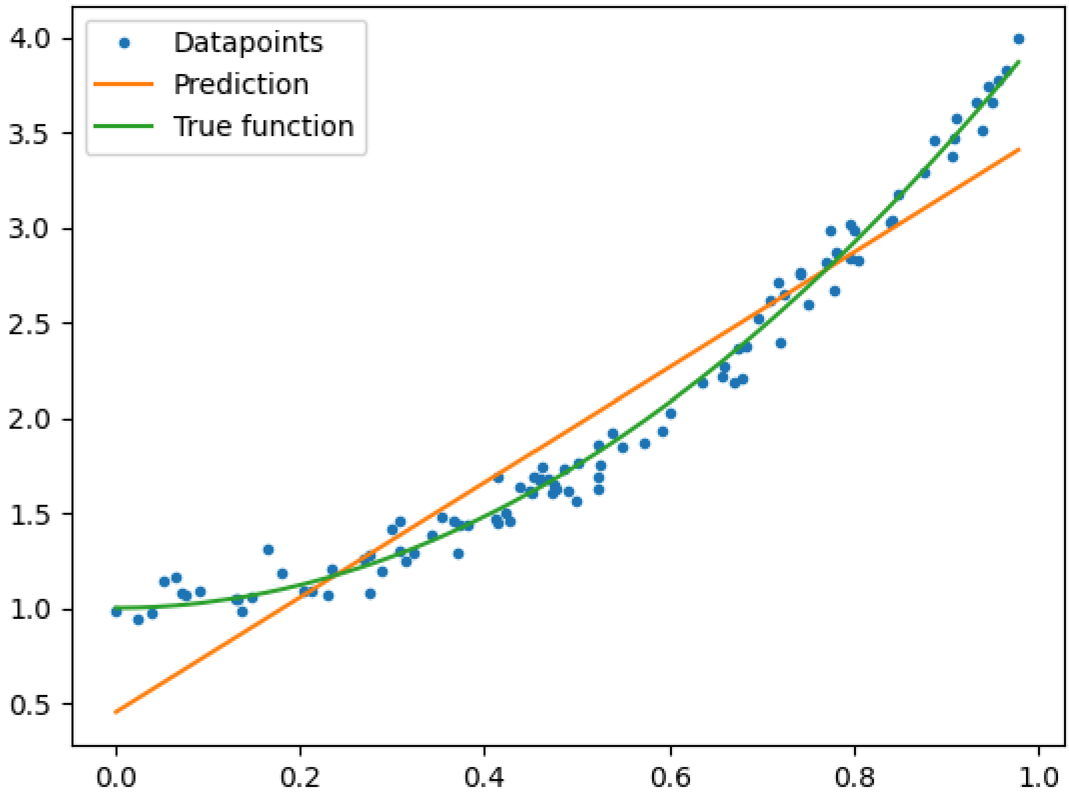
In [ ]:
```
#imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
```

In [ ]:
```
np.random.seed(7)

n = 100
x = np.random.rand(n,1)
x = np.sort(x,axis=0)
y = 1.0 + 3*x**2 + 0.1*np.random.randn(n,1) #noise added to true function
yt = 1.0 + 3*x**2 #true function
```

In [ ]:
```
#first order for fun
X = np.ones((n,2))
X[:,1] = x[:,0]

beta = np.linalg.inv(X.T@X)@X.T@y
ytilde = X@beta
```

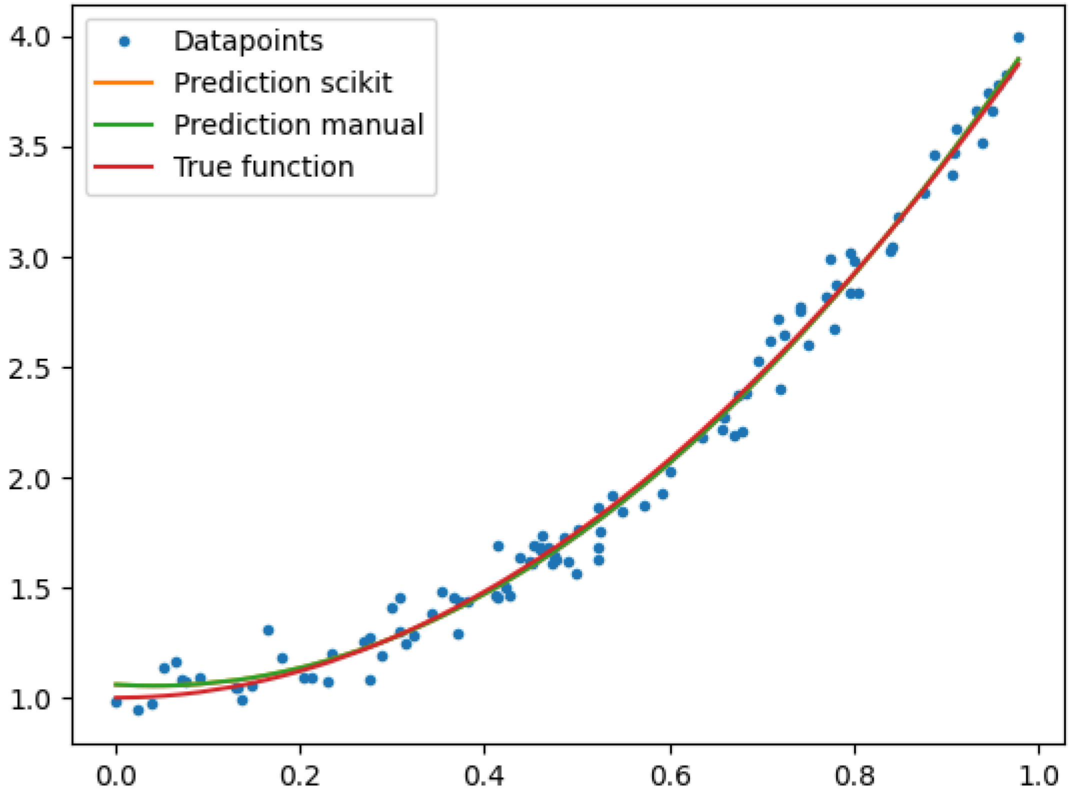
In [ ]:
```
plt.plot(x,y,'o',label='Datapoints',markersize=3)
plt.plot(x,ytilde,label='Prediction')
```

```
plt.plot(x,yt,label='True function')
plt.legend()
plt.show()
```



In [ ]:
```
#second order, with sklearn and without
poly2 = PolynomialFeatures(degree=2)
X = poly2.fit_transform(x)
linreg2 = LinearRegression()
linreg2.fit(X,y)
y_pred2 = linreg2.predict(X)

X = np.ones((n,3))
X[:,1] = x[:,0]
X[:,2] = x[:,0]**2

beta = np.linalg.inv(X.T@X)@X.T@y
ytilde2 = X@beta
```

In [ ]:
```
plt.plot(x,y,'o',label='Datapoints',markersize=3)
plt.plot(x,y_pred2,label='Prediction scikit')
plt.plot(x,ytilde2,label='Prediction manual')
plt.plot(x,yt,label='True function')
plt.legend()
plt.show()
```



In [ ]:
```python
#errors on the sklearn model
mse = mean_squared_error(yt,ytilde)
msem2 = mean_squared_error(yt,ytilde2)
mses2 = mean_squared_error(yt,y_pred2)
print(f'MSE linear: {mse:.5f}')
print(f'MSE poly manual:{msem2:.5f}')
print(f'MSE poly sklearn:{mses2:.5f}')


r2 = r2_score(yt,ytilde)
rm22 = r2_score(yt,ytilde2)
rs22 = r2_score(yt,y_pred2)
print(f'R2 linear:{r2:.5f}')
print(f'R2 poly manual:{rm22:.5f}')
print(f'R2 poly sklearn:{rs22:.5f}')
```

```
MSE linear: 0.04541
MSE poly manual:0.00035
MSE poly sklearn:0.00035
R2 linear:0.93525
R2 poly manual:0.99950
R2 poly sklearn:0.99950
```

Noise coeff- 0.1:

MSE's:

MSE linear: 0.04541

MSE poly manual:0.00035

MSE poly sklearn:0.00035

R2's:

R2 linear:0.93525

R2 poly manual:0.99950

R2 poly sklearn:0.99950

The second order polynomial has great values. The manual method provides the exact same values as the scikit-learn, which is very good to see. We see very low MSE score and a very high R2 score for the polynomial functions. Which indicates that is stays almost identical to the true function. The linear function did much worse, which is expected.

Noise coeff- 0.2:

MSE's:

MSE linear: 0.04557

MSE poly manual:0.00139

MSE poly sklearn:0.00139

R2's:

R2 linear:0.93503

R2 poly manual:0.99801

R2 poly sklearn:0.99801

I doubled the noise coefficient. The results are a lower MSE and R2 score, which is to be expected.

Exercise 3: Split data in test and training data

Design matrix and split data

In [ ]:
```
#import
from sklearn.model_selection import train_test_split
```

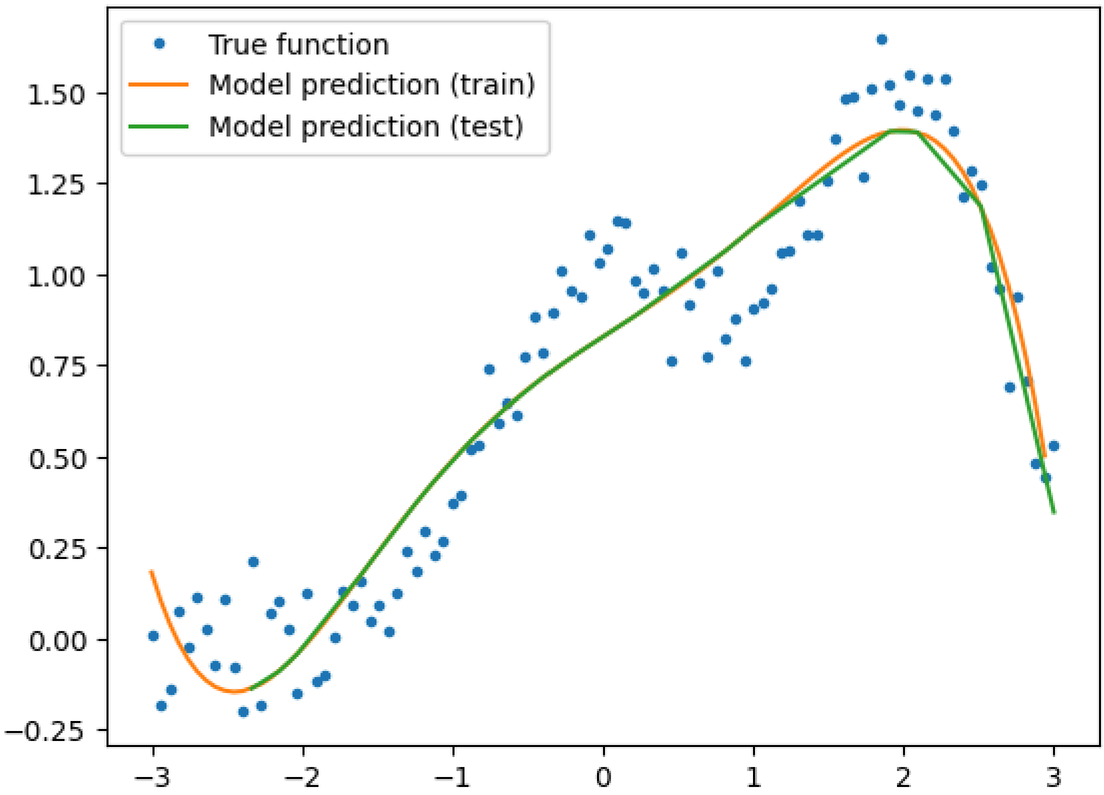
In [ ]:
```
n = 100

# Generate data
x = np.linspace(-3, 3, n).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2) + np.random.normal(0, 0.1, x.
```

In [ ]:
```
X = np.ones((n,6))
X[:,1] = x.flatten()
X[:,2] = x.flatten()**2
X[:,3] = x.flatten()**3
X[:,4] = x.flatten()**4
X[:,5] = x.flatten()**5
```

```
#split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Fit the model

In [ ]:
```
#sort data
sort = np.argsort(X_train[:,1])
X_train = X_train[sort]
y_train = y_train[sort]

sort = np.argsort(X_test[:,1])
X_test = X_test[sort]
y_test = y_test[sort]

# matrix inversion to find beta
beta = np.linalg.inv(X.T @ X) @ X.T @ y
# and then make the prediction
ytilde = X_train @ beta
ypred = X_test @ beta

#plot
plt.plot(x, y,'o', label='True function',markersize=3)
plt.plot(X_train[:,1], ytilde, label='Model prediction (train)')
plt.plot(X_test[:,1], ypred, label='Model prediction (test)')
plt.legend()
plt.show()
```

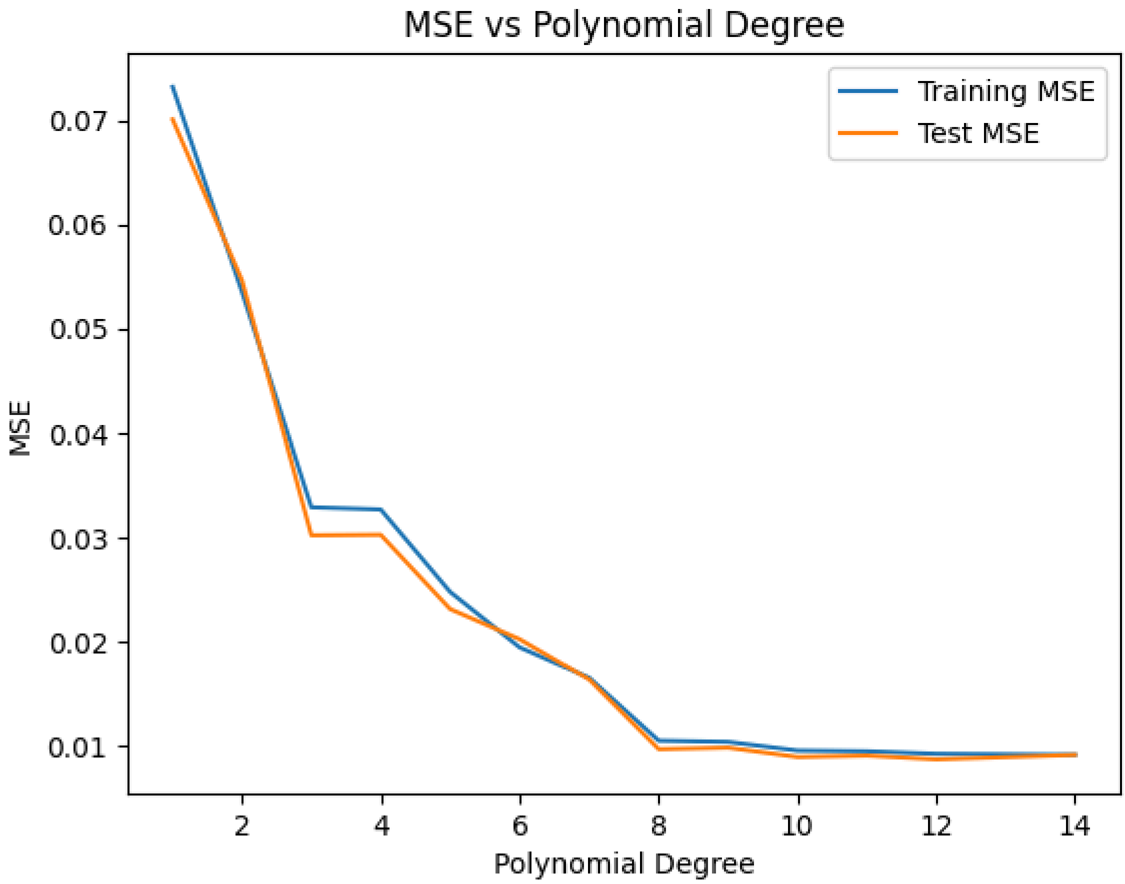


In [ ]:
```
mse_train = mean_squared_error(y_train, ytilde)
mse_test = mean_squared_error(y_test, ypred)

print(f'MSE for training data: {mse_train:.2e}')
print(f'MSE for test data: {mse_test:.2e}')
```

```
MSE for training data: 2.39e-02
MSE for test data: 2.93e-02
```

15 degree polynomial

In [ ]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt

mse_train = []
mse_test = []
degrees = range(1,15)

x = np.linspace(-3, 3, 500).reshape(-1, 1)
y = np.exp(-x**2) + 1.5 * np.exp(-(x-2)**2)+ np.random.normal(0, 0.1, x.s

for i in degrees:
    poly = PolynomialFeatures(degree=i)
    X = poly.fit_transform(x)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0

    beta = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
    ytilde = X_train @ beta
    ypredict = X_test @ beta

    mse_train.append(mean_squared_error(y_train, ytilde))
    mse_test.append(mean_squared_error(y_test, ypredict))



# Plotting the MSE values
plt.plot(degrees, mse_train, label='Training MSE')
plt.plot(degrees, mse_test, label='Test MSE')
plt.xlabel('Polynomial Degree')
plt.ylabel('MSE')
plt.title('MSE vs Polynomial Degree')
plt.legend()
plt.show()

np.where(mse_test == np.min(mse_test)) #finds the minimum test error
```



Out[ ]: (array([11]),)

My plot here does not really match well with Figure 2.11 of Hastie et al. Here the test and train samples mostly have the same MSE values, and does not have a split where test MSE increases and test sample continues to decrease. The 10th degree polynomial gave the lowest MSE value and therefor perfomed the best.