CS178 Homework 2

Due Friday, October 20th, 11:59pm

Instructions

This homework (and subsequent ones) will involve data analysis and reporting on methods and results using Python code. You will submit a **single PDF file** that contains everything to Gradescope. This includes any text you wish to include to describe your results, the complete code snippets of how you attempted each problem, any figures that were generated, and scans of any work on paper that you wish to include. It is important that you include enough detail that we know how you solved the problem, since otherwise we will be unable to grade it.

Your homeworks will be given to you as Jupyter notebooks containing the problem descriptions and some template code that will help you get started. You are encouraged to use these starter Jupyter notebooks to complete your assignment and to write your report. This will help you not only ensure that all of the code for the solutions is included, but also will provide an easy way to export your results to a PDF file (for example, doing *print preview* and *printing to pdf*). I recommend liberal use of Markdown cells to create headers for each problem and sub-problem, explaining your implementation/answers, and including any mathematical equations. For parts of the homework you do on paper, scan it in such that it is legible (there are a number of free Android/iOS scanning apps, if you do not have access to a scanner), and include it as an image in the Jupyter notebook.

Double check that all of your answers are legible on Gradescope, e.g. make sure any text you have written does not get cut off.

If you have any questions/concerns about using Jupyter notebooks, ask us on EdD. If you decide not to use Jupyter notebooks, but go with Microsoft Word or LaTeX to create your PDF file, make sure that all of the answers can be generated from the code snippets included in the document.

Summary of Assignment: 100 total points

- Problem 1: k-Nearest Neighbors (25 points)
    - Problem 1.1: Splitting data into training & test sets (10 points)
    - Problem 1.2: Plot predictions for different values of k (10 points)

    - Problem 1.3: Display performance as a function of k & select best (5 points)
- Problem 2: Linear Regression (25 points)
    - Problem 2.1: Train the model and plot the data along with its predictions (15 points)
    - Problem 2.2: Compute the MSE loss for the training and evaluation data (10 points)
- Problem 3: Feature transformations (25 points)
    - Problem 3.1: Train & display polynomial regression models using feature transforms (10 points)
    - Problem 3.2: Plot the training & evaluation error as a function of degree (10 points)
    - Problem 3.3: Select the best degree for these data (5 points)
- Problem 4: Cross-Validation (20 points)
    - Problem 4.1: Plot the five-fold cross validation error (10 points)
    - Problem 4.2: Select the best degree using cross-validation (5 points)
    - Problem 4.3: Compare cross-validation model selection to that in Problem 2 (5 points)
- Statement of Collaboration (5 points)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import zero_one_loss
from sklearn.metrics import mean_squared_error as mse

from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.linear_model import LinearRegression    # Basic Linear Regressi
from sklearn.linear_model import Ridge               # Linear Regression wit

from sklearn.model_selection import KFold            # Cross-validation tool

from sklearn.preprocessing import PolynomialFeatures # Feature transformatic
from sklearn.preprocessing import StandardScaler

import requests                                      # reading data
from io import StringIO

seed = 1234
```

Training / Test Splits

As we've seen in lecture, it is difficult to tell how accurate our model is from only the data on which it has been trained. For this reason, we usually reserve some data for evaluation, often called "validation" data. We'll start by loading a one-dimensional regression data set to use in the rest of the homework. We will divide this data set into 75% training data, and 25% evaluation data:

In [2]:
```
url = 'https://www.ics.uci.edu/~ihler/classes/cs178/data/curve80.txt'

with requests.get(url) as link: curve = np.genfromtxt(StringIO(link.text),de

X = curve[:,0:-1]      # extract features
Y = curve[:,-1]        # extract target values

# split into training and evaluation data
Xt, Xe, Yt, Ye = train_test_split(X, Y, test_size=0.25, random_state=seed)
```

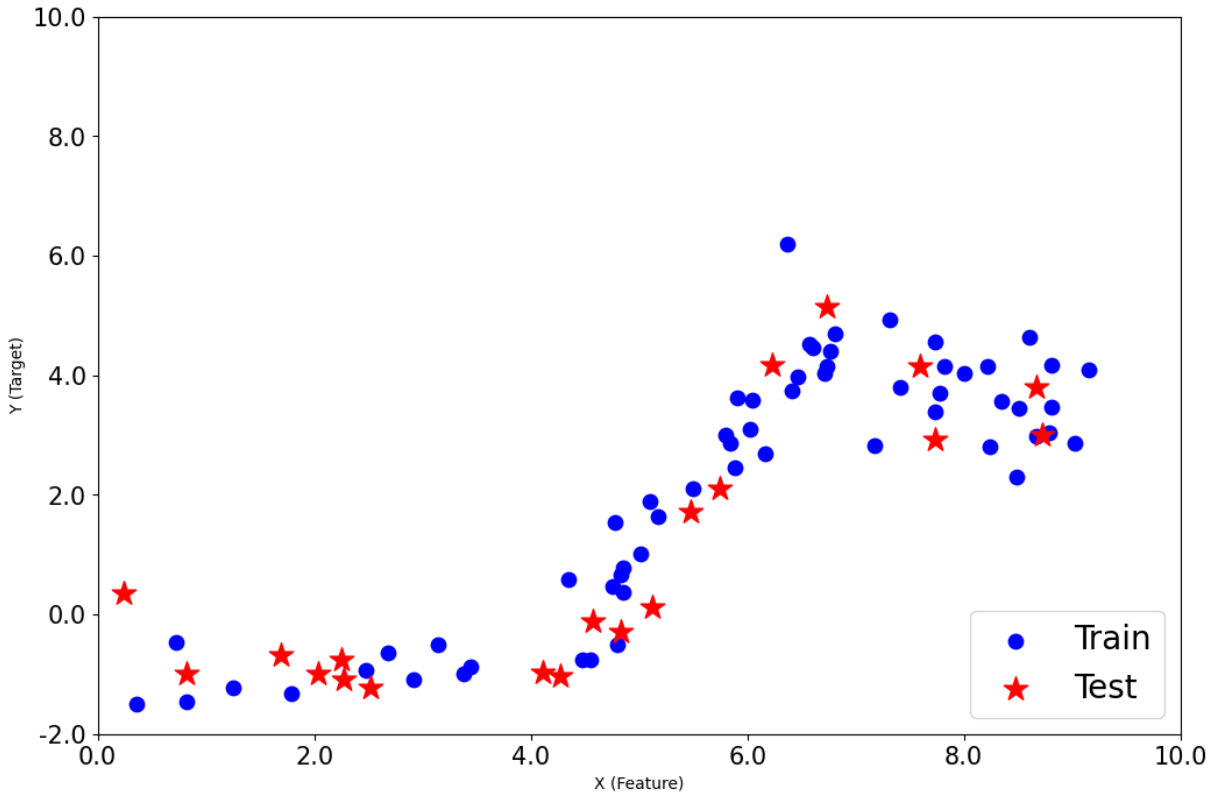
P1: K-Nearest Neighbor Regression

P1.1: Visualizing the Data Splits

Plot the data for this regression problem, with the (scalar) feature x along the horizontal axis, and the real-valued target y as the vertical axis. Plot all the data, displaying the training data X_t in one color, and the evaluation data X_e in a different color.

In [29]:
```
# From Discussion 2

# Plotting the data

f, ax = plt.subplots(1, 1, figsize=(12, 8))
ax.scatter(Xt, Yt, s=80, color='blue', label='Train')
ax.scatter(Xe, Ye, s=240, marker='*', color='red', label='Test')
ax.set_xlim(0, 10)
ax.set_ylim(-2, 10)

plt.xlabel('X (Feature)')
plt.ylabel('Y (Target)')

# ax.set_xticks( ax.get_xticks(), labels=ax.get_xticks(), fontsize=20)
# ax.set_yticks( ax.get_yticks(), labels=ax.get_yticks(), fontsize=20)

ax.set_xticks( ax.get_xticks())
ax.set_yticks( ax.get_yticks())
ax.set_xticklabels(ax.get_xticks(), fontsize=15)
ax.set_yticklabels(ax.get_yticks(), fontsize=15)
```

```
# Controlling the size of the legend and the location.
ax.legend(fontsize=20, loc='lower right')
plt.show()
```

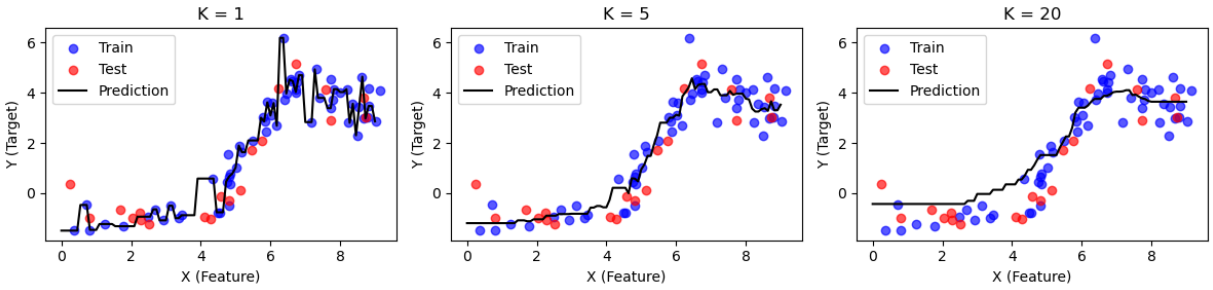


P1.2 Visualizing KNN Regression Predictions

Now use `sklearn`'s `KNeighborsRegressor` class to build a nearest neighbor regression model on your training data. Build three models, using $k=1$, $k=5$, and $k=20$, and for each one display the training data, test data, and prediction function. (Note: you can evaluate the prediction function of your learner by predicting at a dense collection of locations `xs` along the x-axis, and then predicting at these points and connecting them using `plot`.)

In [32]:
```
# Create a figure with 1 row and 3 columns
fig, axes = plt.subplots(1, 3, figsize=(12, 3))

xs = np.linspace(0,9,100).reshape(-1,1)  # get a collection of x-locations a

### YOUR CODE STARTS HERE ###

i = 0
for k in [1,5,20]:
    knn = KNeighborsRegressor(n_neighbors=k) # fitting the k nearest neighbo
    knn.fit(Xt, Yt) # model on the training data

    ys = knn.predict(xs) # Make predictions

    # Plot for the training and evaluation data
    axes[i].scatter(Xt, Yt, color='blue', alpha=0.65, label='Train')
```

```
    axes[i].scatter(Xe, Ye, color='red', alpha=0.65, label='Test')

    # Prediction plot
    axes[i].plot(xs, ys, color='black', label='Prediction')

    # Setting labels
    axes[i].set_xlabel('X (Feature)')
    axes[i].set_ylabel('Y (Target)')
    axes[i].set_title(f'K = {k}')
    axes[i].legend()

    i += 1

###  YOUR CODE ENDS HERE  ###

fig.tight_layout()
```

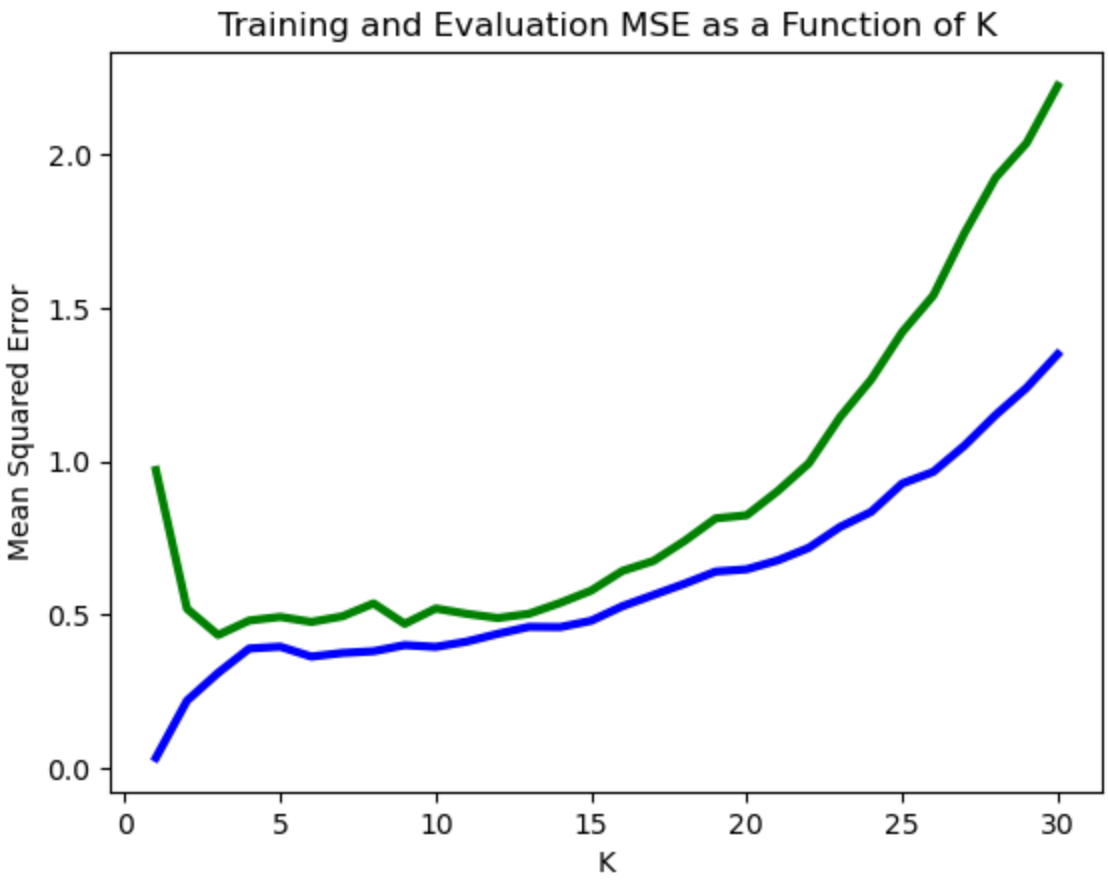


P1.3: KNN Model Selection

Train a model for each k in $1 \leq k \leq 30$, and compute their training and test MSE. Plot these values as a function of k. What is the best value of k for your model?

In [37]:
```
k_values = list(range(1,31))
mse_train = []
mse_eval = []

for i,k in enumerate(k_values):

    ### YOUR CODE STARTS HERE ###
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(Xt, Yt) # training model

    # MSE for the training and evaluation data
    mse_train.append(mse(Yt, knn.predict(Xt)))
    mse_eval.append(mse(Ye, knn.predict(Xe)))

    # labels
    plt.xlabel('K')
    plt.ylabel('Mean Squared Error')
    plt.title('Training and Evaluation MSE as a Function of K')


    ###  YOUR CODE ENDS HERE  ###
```

```
plt.plot(k_values,mse_train,'b-', k_values,mse_eval,'g-', lw=3);
```



The blue line represents the training MSE, which generally increases as K increases. This is expected because with a larger K, the model becomes simpler and may underfit the training data.

The green line represents the evaluation MSE, which decreases initially and then starts to increase after a certain point. This behavior indicates that there is an optimal complexity (or K value) where the model generalizes well to unseen data.

In [36]:
```
# Best value of K
best_k = k_values[np.argmin(mse_eval)]
print(best_k)
print(mse_eval[best_k])
```

```
3
0.4812137825025656
```

The best value of K for this model, which minimizes the evaluation MSE, is K = 3. At K = 3, the MSE is 0.4812137825025656.

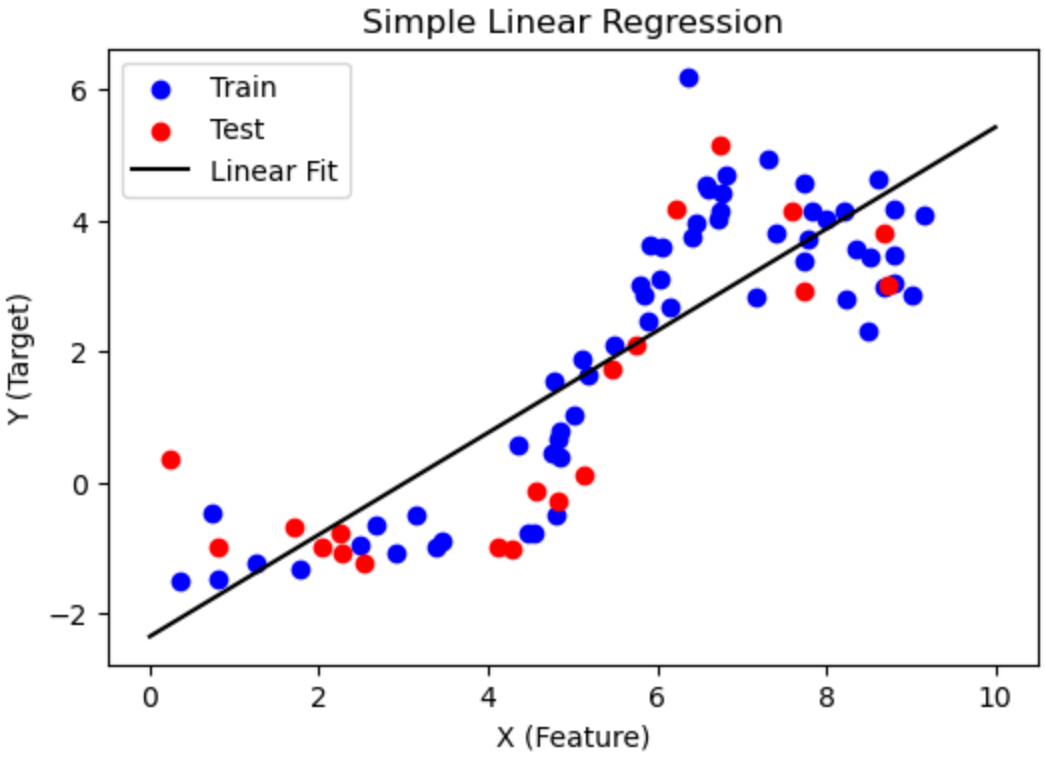
P2: Linear Regression

P2.1: Train linear regression model

Now, let's train a simple linear regression model on the training data. After training the model, plot the training data (colored blue), evaluation data (colored red), and our linear fit (a line) together on a single plot. Also print out the coefficients (slope, `lr.coef_`, and intercept, `lr.intercept_`) of your model after fitting.

In [39]:
```
plt.figure(figsize=(6,4))

    ### YOUR CODE STARTS HERE ###

lr = LinearRegression().fit(Xt, Yt) # create and fit model to training data

# to plot the prediction, we'll evaluate our model at a dense set of locatio
xs = np.linspace(0,10,200).reshape(-1,1)   # data points should be shape (m,
ys = lr.predict(xs)

# plt.figure(figsize=(6,4))
plt.scatter(Xt, Yt, color='blue', label='Train')
plt.scatter(Xe, Ye, color='red', label='Test')
plt.plot(xs, ys, color='black', label='Linear Fit') # plot the data and line
plt.xlabel('X (Feature)')
plt.ylabel('Y (Target)')
plt.title('Simple Linear Regression')
plt.legend()

# slope & intercept of your fit model
print(f'Slope(Coefficient): {lr.coef_[0]}')
print(f'Intercept: {lr.intercept_}')

    ###  YOUR CODE ENDS HERE  ###
```

```
Slope(Coefficient): 0.7768472052927541
Intercept: -2.3463013180118275
```



P2.2 Evaluate your model's fit

Compute the mean squared error of your trained model on the training data (the data it was fit on) and the held-out evaluation data.

In [40]:
```
# training data
mse_train = mse(Yt, lr.predict(Xt))
print(f'MSE of Training data: {mse_train}')

# evaluation data
mse_test = mse(Ye, lr.predict(Xe))
print(f'MSE of Evaluation data: {mse_test}')
```

```
MSE of Training data: 1.270893125474928
MSE of Evaluation data: 1.6723519225582435
```

Problem 3: Feature Transformations

Often we will want to transform our data (as we saw in class). A very simple version of this transformation is "normalizing" the data, in which we shift and scale the feature values to a desirable range; typically, zero mean and unit variance, for example. The `StandardScaler()` object in scikit-learn implements such a transformation.

Typically, a pre-processing transformation works in a similar way to training a model: we `fit` the object to our training data (in this case, computing the empirical mean and variance of the data), and save the parameters of the transformation (the shift and scale

values) so that we can apply exactly the same transformation to subsequent data, for example when asked to predict on a new value of x.

So, for example:

In [41]:
```
scale = StandardScaler().fit(Xt)     # find the desired transformation
X_transformed = scale.transform(Xt)  # & apply it to the training data

# Now, we can train our model on X_transformed...
# lr = LinearRegression()...

# Before we predict, we also need to transform the test point's values:
ys = lr.predict(scale.transform(xs))
```

If you like (and as described in the Discussion code), you can use `sklearn`'s `Pipeline` object to simplify the process of sequentially applying transformations before a predictor.

P3.1: Train polynomial regression models

As mentioned in the homework, you can create additional features manually, e.g.,

In [42]:
```
m,n = Xt.shape            # rest of this cell assumes n=1 feature
Xt2 = np.zeros((m,2))
Xt2[:,0] = Xt[:,0]
Xt2[:,1] = Xt[:,0]**2
print (Xt.shape)
print (Xt2.shape)
print (Xt2[0:6,:])   # look at a few data points to check:
```

```
(60, 1)
(60, 2)
[[ 0.72580645  0.526795  ]
 [ 2.4769585   6.13532341]
 [ 7.7304147  59.75931143]
 [ 9.0207373  81.37370144]
 [ 8.6751152  75.25762373]
 [ 6.4631336  41.77209593]]
```

or, you can create them using SciKit's PolynomialFeatures transform object:

In [43]:
```
Phi = PolynomialFeatures(degree=2,include_bias=False).fit(Xt)
Xt2 = Phi.transform(Xt)
print (Xt2[0:6,:])   # look at the same data points -- same values
```

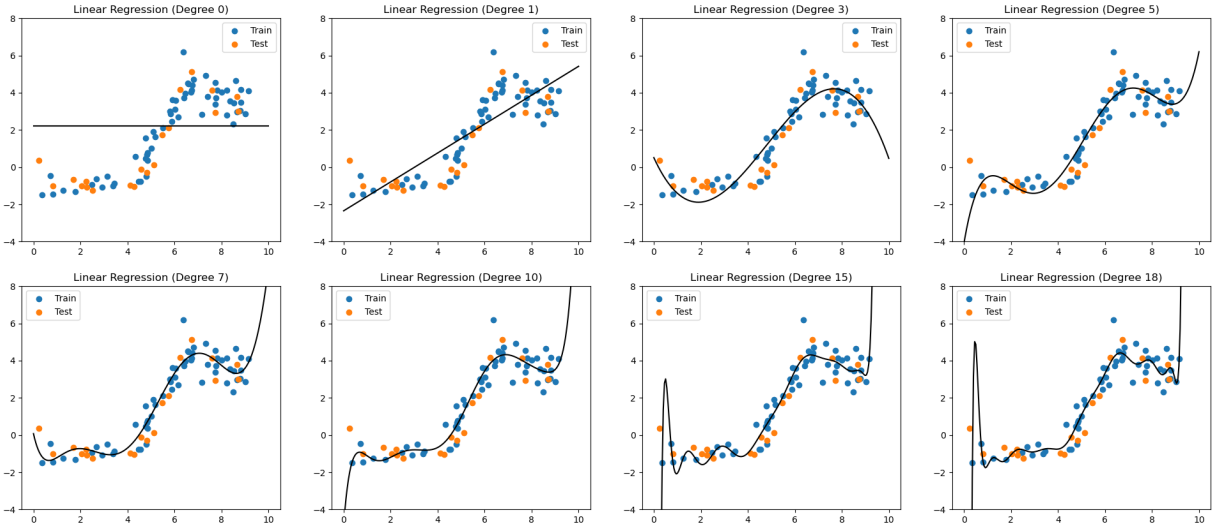
```
[[ 0.72580645  0.526795  ]
 [ 2.4769585   6.13532341]
 [ 7.7304147  59.75931143]
 [ 9.0207373  81.37370144]
 [ 8.6751152  75.25762373]
 [ 6.4631336  41.77209593]]
```

Now, try fitting a linear regression model using different numbers of polynomial features of x.

For each degree $d \in \{0,1,3,5,7,10,15,18\}$:

- Fit a linear regression model using features consisting of all powers of x up to degree d
  - Make sure you apply `StandardScaler` to the transformed data before training
- Plot the resulting prediction function $f(x)$, along with the training and validation data as before

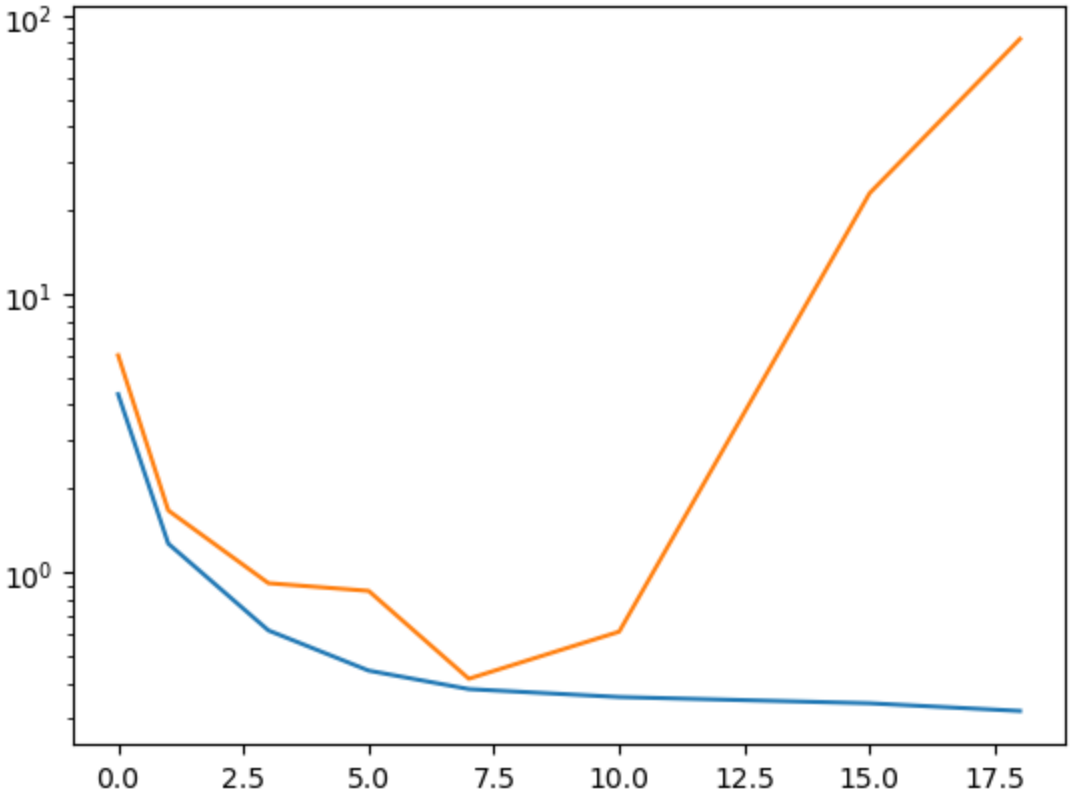
In [53]:
```
from sklearn.pipeline import Pipeline
degrees = [0,1,3,5,7,10,15,18]
# learners = [ [] ]*len(degrees)

fig, ax = plt.subplots(2,4, figsize=(24,10))

train = []
test = []

for i,degree in enumerate(degrees):

    ### YOUR CODE STARTS HERE ###

    # Create a polynomial feature expansion of degree d
    learner = Pipeline([('poly', PolynomialFeatures(degree=degree, include_b
    ('regressor', LinearRegression())])

    # Fit your linear regression and save it to "learners"
    learner.fit(Xt, Yt)
    ys = learner.predict(xs)

    train.append(mse(Yt, learner.predict(Xt))) # training mse
    test.append(mse(Ye, learner.predict(Xe))) # testing mse

    axi = ax[i//4,i%4]
    axi.scatter(Xt, Yt, label='Train')
    axi.scatter(Xe, Ye, label='Test')
    axi.plot(xs, ys, color='black')  # plot the data and your prediction fur
    axi.set_ylim(-4, 8)         # you'll want to set a consistent y-scale fo
    axi.set_title(f'Linear Regression (Degree {degree})')     # don't forget
    axi.legend()

    ###  YOUR CODE ENDS HERE  ###
```



P3.2 Model Performance

Compute the mean squared error (MSE) loss of each of your trained models on both the training data and the evaluation data. Plot these errors as a function of degree (so, degree along the horizontal axis, MSE loss as the vertical axis).

In [55]:
```
# mse_train = [0]*len(degrees)
# mse_test = [0]*len(degrees)

# for i,degree in enumerate(degrees):
#     # Recompute the degree-d poly transform if you didn't save it!
#     mse_train[i] = ...
#     mse_test[i] = ...

plt.semilogy(degrees, train, degrees, test);  # plot mse_train and mse_test
```



P3.3 Model Selection

Which degree would you select to use?

Approximately **Degree 7** as it has the smallest gap between the two lines of training and testing.

P4: Cross-validation

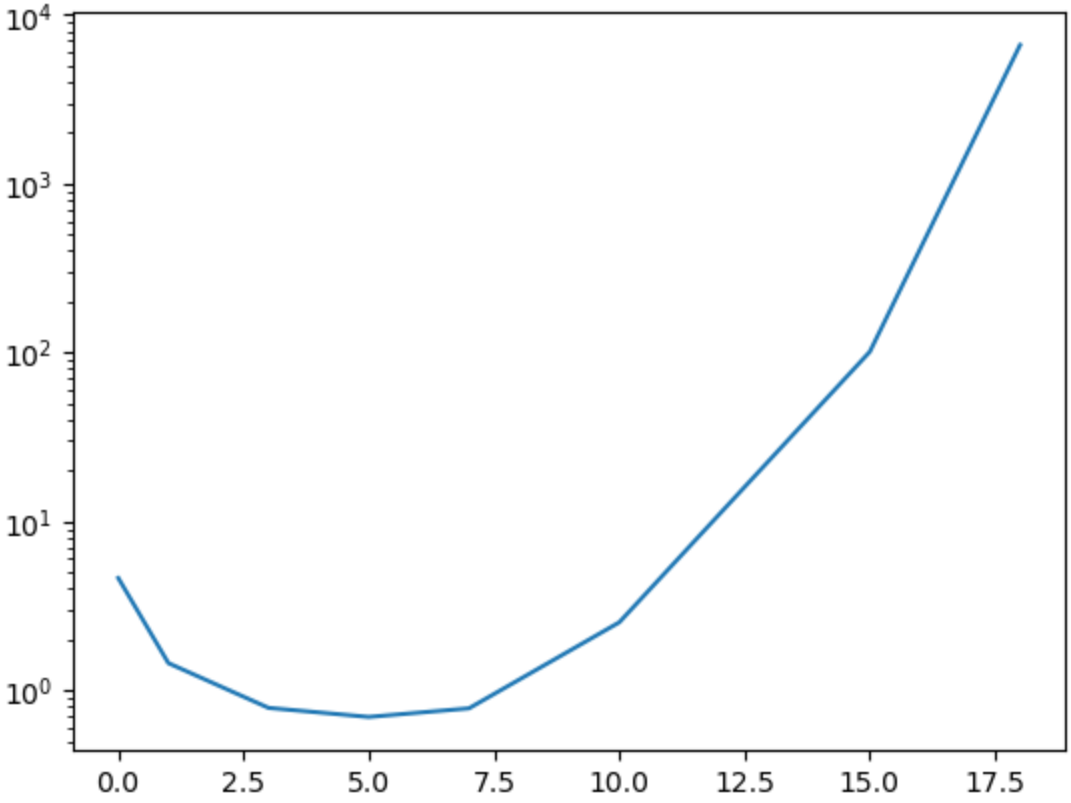
Cross validation is another method of model complexity assessment. We use it only to determine the correct setting of complexity parameters ("hyperparameters"), such as how many and which features to use, or parameters like "k" in KNN, for which training error alone provides little information. In particular, cross validation will not produce a model, only a setting of the hyperparameter values that cross-validation thinks will lead to a model with low test error.

P4.1: 5-Fold Cross-validation

In the previous problem, we decided what degree of polynomial fit to use based on the performance on a held-out set of test data. Now suppose that we do not have access to the target values of those data. How can we determine the best degree?

We could perform another split; but since this is reducing the number of data available, let us instead use cross-validation to evaluate the degrees. Cross-validation works by splitting the training data X_T multiple times, one for eack of the K partitions (`n_splits` in the code), and repeat our entire training and evaluation procedure on each split:

In [61]:
```
mse_xval = [ 0. ]*len(degrees)

i = 0

for degree in degrees:

    xval = KFold(n_splits = 5)
    mse_avg = 0

    for train_index, val_index in xval.split(Xt):
        # Extract the ith cross-validation fold (training/validation split)
        Xti,Xvi,Yti,Yvi = Xt[train_index],Xt[val_index],Yt[train_index],Yt[v

        # Now, build the model:
        # Create a polynomial feature expansion
        # Create a StandardScaler
        learner = Pipeline([('poly', PolynomialFeatures(degree=degree, inclu
        ('regressor', LinearRegression())])
        learner.fit(Xti, Yti) # Fit the linear regression model on the trair
        mse_avg += mse(Yvi, learner.predict(Xvi)) # Compute the MSE on the e

    mse_xval[i] = mse_avg/nfolds
    i += 1

    # Evaluate by averaging the MSE across the five folds


# Plot the estimated MSE from cross-validation as a function of the degree
plt.semilogy(degrees, mse_xval);
```



P4.2: Cross-validation model selection

What degree would you choose based on the cross validation performance?

Degree 5 as it seems to have the lowest average MSE.

P4.3 Comparison to test performance

How do the MSE estimates from 5-fold cross-validation compare to the estimated test performance you found from your held-out data, X_E? Explain briefly.

For the first few degrees, the MSE estimates from 5-fold cross-validation and the estimated test performance were similar after which the differnce increased significantly. The MSE estimates from 5-fold cross-validation were much higher compared to the estimated test performance from the held-out data.

Statement of Collaboration (5 points)

It is **mandatory** to include a Statement of Collaboration in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using EdD) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content before they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to EdD, etc.). Especially after you have started working on the assignment, try to restrict the discussion to EdD as much as possible, so that there is no doubt as to the extent of your collaboration.

I did not discuss this assignemnt with any class mate or friend. I used the ED discussions to look for any doubts.