*You are currently looking at **version 0.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

In [82]:
```
# Assignment 2

#In this assignment you'll explore the relationship between model complexity and generalization performance, by adjusting key para

## Part 1 - Regression
```

In [72]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


np.random.seed(0)
n = 15
x = np.linspace(0,10,n) + np.random.randn(n)/5
y = np.sin(x)+x/6 + np.random.randn(n)/10


X_train, X_test, y_train, y_test = train_test_split(x, y, random_state=0)

def intro():
    %matplotlib notebook

    plt.figure()
    plt.scatter(X_train, y_train, label='training data')
    plt.scatter(X_test, y_test, label='test data')
    plt.legend(loc=4);

intro()
```

Question 1

Write a function that fits a polynomial LinearRegression model on the *training data* `X_train` for degrees 1, 3, 6, and 9. (Use PolynomialFeatures in sklearn.preprocessing to create the polynomial features and then fit a linear regression model) For each model, find 100 predicted values over the interval x = 0 to 10 (e.g. `np.linspace(0,10,100)`) and store this in a numpy array. The first row of this array should correspond to the output from the model trained on degree 1, the second row degree 3, the third row degree 6, and the fourth row degree 9.

The figure above shows the fitted models plotted on top of the original data (using `plot_one()`).

This function should return a numpy array with shape `(4, 100)`

In [73]:
```
def answer_one():
    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures
    # To capture interactions between the original features by adding them as features to the Linear model.

    clf = LinearRegression()
    preds =np.zeros((4,100))
    X_input = np.linspace(0,10,100) #Given requirement
    orders = [1,3,6,9] #Given requirement

    for i in range(len(orders)):
        poly = PolynomialFeatures (orders[i]) # Object to add polynomial features.

        #Add polynomial features to training data and input data:

        #Need to transpose X_train and X_input for poly fit to Mork.

        X_train_poly = poly.fit_transform(X_train[None].T)
        X_input_poly = poly.fit_transform(X_input[None].T)

        #Train Linear regression classifier with training data:
        clf.fit(X_train_poly, y_train)

        #Get predictions from Linear classifier using transformed input data:
        preds[i,:]=clf.predict(X_input_poly)
    return preds
answer_one()
```

```
Out[73]: array([[ 2.53040195e-01,  2.69201547e-01,  2.85362899e-01,
           3.01524251e-01,  3.17685603e-01,  3.33846955e-01,
           3.50008306e-01,  3.66169658e-01,  3.82331010e-01,
           3.98492362e-01,  4.14653714e-01,  4.30815066e-01,
           4.46976417e-01,  4.63137769e-01,  4.79299121e-01,
           4.95460473e-01,  5.11621825e-01,  5.27783177e-01,
           5.43944529e-01,  5.60105880e-01,  5.76267232e-01,
           5.92428584e-01,  6.08589936e-01,  6.24751288e-01,
           6.40912640e-01,  6.57073992e-01,  6.73235343e-01,
           6.89396695e-01,  7.05558047e-01,  7.21719399e-01,
           7.37880751e-01,  7.54042103e-01,  7.70203454e-01,
           7.86364806e-01,  8.02526158e-01,  8.18687510e-01,
           8.34848862e-01,  8.51010214e-01,  8.67171566e-01,
           8.83332917e-01,  8.99494269e-01,  9.15655621e-01,
           9.31816973e-01,  9.47978325e-01,  9.64139677e-01,
           9.80301028e-01,  9.96462380e-01,  1.01262373e+00,
           1.02878508e+00,  1.04494644e+00,  1.06110779e+00,
           1.07726914e+00,  1.09343049e+00,  1.10959184e+00,
           1.12575320e+00,  1.14191455e+00,  1.15807590e+00,
           1.17423725e+00,  1.19039860e+00,  1.20655995e+00,
           1.22272131e+00,  1.23888266e+00,  1.25504401e+00,
           1.27120536e+00,  1.28736671e+00,  1.30352807e+00,
           1.31968942e+00,  1.33585077e+00,  1.35201212e+00,
           1.36817347e+00,  1.38433482e+00,  1.40049618e+00,
           1.41665753e+00,  1.43281888e+00,  1.44898023e+00,
           1.46514158e+00,  1.48130294e+00,  1.49746429e+00,
           1.51362564e+00,  1.52978699e+00,  1.54594834e+00,
           1.56210969e+00,  1.57827105e+00,  1.59443240e+00,
           1.61059375e+00,  1.62675510e+00,  1.64291645e+00,
           1.65907781e+00,  1.67523916e+00,  1.69140051e+00,
           1.70756186e+00,  1.72372321e+00,  1.73988457e+00,
           1.75604592e+00,  1.77220727e+00,  1.78836862e+00,
           1.80452997e+00,  1.82069132e+00,  1.83685268e+00,
           1.85301403e+00],
         [ 1.22989539e+00,  1.15143628e+00,  1.07722393e+00,
           1.00717881e+00,  9.41221419e-01,  8.79272234e-01,
           8.21251741e-01,  7.67080426e-01,  7.16678772e-01,
           6.69967266e-01,  6.26866391e-01,  5.87296632e-01,
           5.51178474e-01,  5.18432402e-01,  4.88978901e-01,
           4.62738455e-01,  4.39631549e-01,  4.19578668e-01,
           4.02500297e-01,  3.88316920e-01,  3.76949022e-01,
           3.68317088e-01,  3.62341603e-01,  3.58943051e-01,
           3.58041918e-01,  3.59558687e-01,  3.63413845e-01,
           3.69527874e-01,  3.77821261e-01,  3.88214491e-01,
           4.00628046e-01,  4.14982414e-01,  4.31198078e-01,
           4.49195522e-01,  4.68895233e-01,  4.90217694e-01,
           5.13083391e-01,  5.37412808e-01,  5.63126429e-01,
         5.90144741e-01,  6.18388226e-01,  6.47777371e-01,
         6.78232660e-01,  7.09674578e-01,  7.42023609e-01,
         7.75200238e-01,  8.09124950e-01,  8.43718230e-01,
         8.78900563e-01,  9.14592432e-01,  9.50714324e-01,
         9.87186723e-01,  1.02393011e+00,  1.06086498e+00,
         1.09791181e+00,  1.13499108e+00,  1.17202328e+00,
         1.20892890e+00,  1.24562842e+00,  1.28204233e+00,
         1.31809110e+00,  1.35369523e+00,  1.38877520e+00,
         1.42325149e+00,  1.45704459e+00,  1.49007498e+00,
         1.52226316e+00,  1.55352959e+00,  1.58379478e+00,
         1.61297919e+00,  1.64100332e+00,  1.66778766e+00,
         1.69325268e+00,  1.71731887e+00,  1.73990672e+00,
         1.76093671e+00,  1.78032933e+00,  1.79800506e+00,
         1.81388438e+00,  1.82788778e+00,  1.83993575e+00,
         1.84994877e+00,  1.85784732e+00,  1.86355189e+00,
         1.86698296e+00,  1.86806103e+00,  1.86670656e+00,
         1.86284006e+00,  1.85638200e+00,  1.84725286e+00,
         1.83537314e+00,  1.82066332e+00,  1.80304388e+00,
         1.78243530e+00,  1.75875808e+00,  1.73193269e+00,
         1.70187963e+00,  1.66851936e+00,  1.63177240e+00,
         1.59155920e+00],
       [-1.99554310e-01, -3.95192721e-03,  1.79851753e-01,
         3.51005136e-01,  5.08831706e-01,  6.52819233e-01,
         7.82609240e-01,  8.97986721e-01,  9.98870117e-01,
         1.08530155e+00,  1.15743729e+00,  1.21553852e+00,
         1.25996233e+00,  1.29115292e+00,  1.30963316e+00,
         1.31599632e+00,  1.31089811e+00,  1.29504889e+00,
         1.26920626e+00,  1.23416782e+00,  1.19076415e+00,
         1.13985218e+00,  1.08230867e+00,  1.01902405e+00,
         9.50896441e-01,  8.78825970e-01,  8.03709344e-01,
         7.26434655e-01,  6.47876457e-01,  5.68891088e-01,
         4.90312256e-01,  4.12946874e-01,  3.37571147e-01,
         2.64926923e-01,  1.95718291e-01,  1.30608438e-01,
         7.02167560e-02,  1.51162118e-02, -3.41690365e-02,
        -7.71657635e-02, -1.13453547e-01, -1.42666382e-01,
        -1.64494044e-01, -1.78683194e-01, -1.85038228e-01,
        -1.83421873e-01, -1.73755533e-01, -1.56019368e-01,
        -1.30252132e-01, -9.65507462e-02, -5.50696231e-02,
        -6.01973195e-03,  5.03325883e-02,  1.13667071e-01,
         1.83611221e-01,  2.59742264e-01,  3.41589357e-01,
         4.28636046e-01,  5.20322987e-01,  6.16050916e-01,
         7.15183874e-01,  8.17052690e-01,  9.20958717e-01,
         1.02617782e+00,  1.13196463e+00,  1.23755703e+00,
         1.34218093e+00,  1.44505526e+00,  1.54539723e+00,
         1.64242789e+00,  1.73537785e+00,  1.82349336e+00,
         1.90604254e+00,  1.98232198e+00,  2.05166348e+00,
         2.11344114e+00,  2.16707864e+00,  2.21205680e+00,
         2.24792141e+00,  2.27429129e+00,  2.29086658e+00,
         2.29743739e+00,  2.29389257e+00,  2.28022881e+00,
         2.25656001e+00,  2.22312684e+00,  2.18030664e+00,
         2.12862347e+00,  2.06875850e+00,  2.00156065e+00,
         1.92805743e+00,  1.84946605e+00,  1.76720485e+00,
         1.68290491e+00,  1.59842194e+00,  1.51584842e+00,
         1.43752602e+00,  1.36605824e+00,  1.30432333e+00,
         1.25548743e+00],
       [ 6.79501877e+00,  4.14319714e+00,  2.23123195e+00,
         9.10495039e-01,  5.49803327e-02, -4.41344174e-01,
        -6.66950030e-01, -6.94942449e-01, -5.85049217e-01,
        -3.85418102e-01, -1.34235851e-01,  1.38818670e-01,
         4.11275215e-01,  6.66715368e-01,  8.93747315e-01,
         1.08510182e+00,  1.23683955e+00,  1.34766042e+00,
         1.41830603e+00,  1.45104693e+00,  1.44924662e+00,
         1.41699501e+00,  1.35880409e+00,  1.27935949e+00,
         1.18332142e+00,  1.07516952e+00,  9.59085935e-01,
         8.38871928e-01,  7.17893068e-01,  5.99048939e-01,
         4.84763319e-01,  3.76991254e-01,  2.77239711e-01,
         1.86598853e-01,  1.05781227e-01,  3.51664578e-02,
        -2.51506700e-02, -7.53106421e-02, -1.15639771e-01,
        -1.46602278e-01, -1.68755085e-01, -1.82706256e-01,
        -1.89077879e-01, -1.88473948e-01, -1.81453660e-01,
        -1.68510358e-01, -1.50056232e-01, -1.26412707e-01,
        -9.78063691e-02, -6.43701362e-02, -2.61492812e-02,
         1.68881553e-02,  6.48371253e-02,  1.17838121e-01,
         1.76057180e-01,  2.39664066e-01,  3.08809354e-01,
         3.83601193e-01,  4.64082501e-01,  5.50209339e-01,
         6.41831222e-01,  7.38674048e-01,  8.40326319e-01,
         9.46229255e-01,  1.05567134e+00,  1.16778775e+00,
         1.28156502e+00,  1.39585128e+00,  1.50937206e+00,
         1.62075184e+00,  1.72854110e+00,  1.83124870e+00,
         1.92737900e+00,  2.01547327e+00,  2.09415450e+00,
         2.16217452e+00,  2.21846241e+00,  2.26217255e+00,
         2.29273075e+00,  2.30987650e+00,  2.31369910e+00,
         2.30466527e+00,  2.28363544e+00,  2.25186569e+00,
         2.21099194e+00,  2.16299281e+00,  2.11012698e+00,
         2.05484079e+00,  1.99964137e+00,  1.94693015e+00,
         1.89879129e+00,  1.85672916e+00,  1.82134864e+00,
         1.79197149e+00,  1.76618168e+00,  1.73929211e+00,
         1.70372475e+00,  1.64829557e+00,  1.55739548e+00,
         1.41005768e+00]])
```

In [74]:

```python
# feel free to use the function plot_one() to replicate the figure
# from the prompt once you have completed question one
def plot_one(degree_predictions):
    plt.figure(figsize=(10,5))
    plt.plot(X_train, y_train, 'o', label='training data', markersize=10)
    plt.plot(X_test, y_test, 'o', label='test data', markersize=10)
    for i,degree in enumerate([1,3,6,9]):
        plt.plot(np.linspace(0,10,100), degree_predictions[i], alpha=0.8, lw=2, label='degree={}'.format(degree))
    plt.ylim(-1,2.5)
    plt.legend(loc=4)

plot_one(answer_one())
```

Question 2

Write a function that fits a polynomial LinearRegression model on the training data `X_train` for degrees 0 through 9. For each model compute the R^2 (coefficient of determination) regression score on the training data as well as the the test data, and return both of these arrays in a tuple.

This function should return a tuple of numpy arrays `(r2_train, r2_test)`*. Both arrays should have shape* `(10,)`

In [75]:
```
def answer_two():
    from sklearn.linear_model import LinearRegression
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.metrics import r2_score

    r2_test, r2_train=[],[]

    for i in range(10):
        poly = PolynomialFeatures(i)
        X_train_poly = poly.fit_transform(X_train[None].T)
        X_test_poly = poly.fit_transform(X_test[None].T)
        linreg = LinearRegression().fit(X_train_poly, y_train)
        r2_train.append(linreg.score(X_train_poly, y_train))
        r2_test.append(linreg.score(X_test_poly, y_test))

    return (r2_train, r2_test)

answer_two()
```

```
Out[75]: ([0.0,
  0.4292457781234663,
  0.45109980444082465,
  0.5871995368779848,
  0.9194194471769315,
  0.9757864143068214,
  0.9901823324795082,
  0.9935250927840571,
  0.9963754538776326,
  0.9980370625642736],
 [-0.4780864173714179,
  -0.45237104233936676,
  -0.06856984149915957,
  0.005331052945772519,
  0.7300494281868388,
  0.8770830091633202,
  0.9214093981555163,
  0.9202150411855775,
  0.6324794161095051,
  -0.645252846658811])
```

```
In [ ]:
```

answer_two()### Question 3

Based on the R^2 scores from question 2 (degree levels 0 through 9), what degree level corresponds to a model that is underfitting? What degree level corresponds to a model that is overfitting? What choice of degree level would provide a model with good generalization performance on this dataset?

(Hint: Try plotting the R^2 scores from question 2 to visualize the relationship)

This function should return a tuple with the degree values in this order: `(Underfitting, Overfitting, Good_Generalization)`

```
In [76]: def answer_three():
             return (0, 9, 7)
         answer_three()
```

```
Out[76]: (0, 9, 7)
```

```
In [ ]:
```

Question 4

Training models on high degree polynomial features can result in overfitting. Train two models: a non-regularized LinearRegression model and a Lasso Regression model (with parameters `alpha=0.01`, `max_iter=10000`, `tol=0.1`) on polynomial features of degree 12. Return the $R^2$ score for LinearRegression and Lasso model's test sets.

This function should return a tuple `(LinearRegression_R2_test_score, Lasso_R2_test_score)`

In [84]:
```python
def answer_four():
    from sklearn.preprocessing import PolynomialFeatures
    from sklearn.linear_model import Lasso, LinearRegression
    from sklearn.metrics import r2_score

    # Constants
    ALPHA = 0.01
    MAX_ITER = 10000
    TOL = 0.1  # Added this
    DEGREE = 12

    # Set up data
    X_tr = X_train.reshape(-1, 1)
    X_tst = X_test.reshape(-1, 1)
    poly = PolynomialFeatures(degree=DEGREE)
    X_tr_ = poly.fit_transform(X_tr)
    X_tst_ = poly.transform(X_tst)

    # Fit OLS
    ols = LinearRegression()
    ols.fit(X_tr_, y_train)
    LinearRegression_R2_test_score = r2_score(y_test, ols.predict(X_tst_))

    # Fit Lasso - ADDED tol=0.1
    linlasso = Lasso(alpha=ALPHA, max_iter=MAX_ITER, tol=TOL).fit(X_tr_, y_train)
    Lasso_R2_test_score = r2_score(y_test, linlasso.predict(X_tst_))

    return LinearRegression_R2_test_score, Lasso_R2_test_score

answer_four()
```

Out[84]:
```
(-4.3119675863308435, 0.6051396919570036)
```

In [ ]:

Part 2 - Classification

For this section of the assignment we will be working with the UCI Mushroom Data Set stored in `mushrooms.csv`. The data will be used to trian a model to predict whether or not a mushroom is poisonous. The following attributes are provided:

Attribute Information:

1. cap-shape: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
2. cap-surface: fibrous=f, grooves=g, scaly=y, smooth=s
3. cap-color: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
4. bruises?: bruises=t, no=f
5. odor: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
6. gill-attachment: attached=a, descending=d, free=f, notched=n
7. gill-spacing: close=c, crowded=w, distant=d
8. gill-size: broad=b, narrow=n
9. gill-color: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
10. stalk-shape: enlarging=e, tapering=t
11. stalk-root: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
12. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
13. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
14. stalk-color-above-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
15. stalk-color-below-ring: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16. veil-type: partial=p, universal=u
17. veil-color: brown=n, orange=o, white=w, yellow=y
18. ring-number: none=n, one=o, two=t
19. ring-type: cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
20. spore-print-color: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
21. population: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
22. habitat: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

The data in the mushrooms dataset is currently encoded with strings. These values will need to be encoded to numeric to work with sklearn. We'll use pd.get_dummies to convert the categorical variables into indicator variables.

In [78]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split


mush_df = pd.read_csv('assets/mushrooms.csv')
mush_df2 = pd.get_dummies(mush_df)
```

```
X_mush = mush_df2.iloc[:,2:]
y_mush = mush_df2.iloc[:,1]


X_train2, X_test2, y_train2, y_test2 = train_test_split(X_mush, y_mush, random_state=0)
```

Question 5

Using `X_train` and `y_train` from the preceeding cell, train a DecisionTreeClassifier with default parameters and random_state=0. What are the 5 most important features found by the decision tree?

This function should return a list of length 5 of the feature names in descending order of importance.

In [79]:
```
def answer_five():
    from sklearn.tree import DecisionTreeClassifier
    clf = DecisionTreeClassifier(random_state=0).fit(X_train2, y_train2)
    df = pd.DataFrame({'feature': X_train2.columns.values, 'importance': clf.feature_importances_})
    return df.sort_values(['importance'], ascending=[0])['feature'].head(5).tolist()
answer_five()
```

Out[79]:
```
['odor_n', 'stalk-root_c', 'stalk-root_r', 'spore-print-color_r', 'odor_l']
```

In [ ]:
```

```

Question 6

For this question, use the `validation_curve` function in `sklearn.model_selection` to determine training and test scores for a Support Vector Classifier (`SVC`) with varying parameter values.

Create an `SVC` with default parameters (i.e. `kernel='rbf', C=1`) and `random_state=0` . Recall that the kernel width of the RBF kernel is controlled using the `gamma` parameter. Explore the effect of `gamma` on classifier accuracy by using the `validation_curve` function to find the training and test scores for 6 values of `gamma` from `0.0001` to `10` (i.e. `np.logspace(-4,1,6)`).

For each level of `gamma` , `validation_curve` will use 3-fold cross validation (use `cv=3, n_jobs=2` as parameters for `validation_curve`), returning two 6x3 (6 levels of gamma x 3 fits per level) arrays of the scores for the training and test sets in each fold.

Find the mean score across the five models for each level of `gamma` for both arrays, creating two arrays of length 6, and return a tuple with the two arrays.

e.g.

if one of your array of scores is

```
array([[ 0.5,  0.4,  0.6],
       [ 0.7,  0.8,  0.7],
       [ 0.9,  0.8,  0.8],
       [ 0.8,  0.7,  0.8],
       [ 0.7,  0.6,  0.6],
       [ 0.4,  0.6,  0.5]])
```

it should then become

```
array([ 0.5,  0.73333333,  0.83333333,  0.76666667,  0.63333333, 0.5])
```

This function should return a tuple of numpy arrays `(training_scores, test_scores)` *where each array in the tuple has shape* `(6,)`.

In [83]:
```
def answer_six():
    from sklearn.svm import SVC
    from sklearn.model_selection import validation_curve

    param_range = np.logspace(-4, 1, 6)

    clf = SVC(kernel='rbf', C=1, random_state=0)

    train_scores, test_scores = validation_curve(clf, X_train2, y_train2,
                                                 param_name='gamma',
                                                 param_range=param_range,
                                                 cv=3,
                                                 n_jobs=2)

    training_scores = np.mean(train_scores, axis=1)
    testing_scores = np.mean(test_scores, axis=1)

    return training_scores, testing_scores

answer_six()
```

Out[83]:
```
(array([0.89069424, 0.97275562, 0.99753816, 1.        , 1.        ,
        1.        ]),
 array([0.88987363, 0.9704579 , 0.99622518, 1.        , 1.        ,
        0.51649434]))
```

In [ ]:

Question 7

Based on the scores from question 6, what gamma value corresponds to a model that is underfitting? What gamma value corresponds to a model that is overfitting? What choice of gamma would provide a model with good generalization performance on this dataset?

(Hint: Try plotting the scores from question 6 to visualize the relationship)

This function should return a tuple with the degree values in this order: `(Underfitting, Overfitting, Good_Generalization)`

```
In [81]: def answer_seven():
             param_range = np.logspace(-4, 1, 6)
             return (param_range[0], param_range[5], param_range[3])
         answer_seven()

Out[81]: (0.0001, 10.0, 0.1)

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```